

POLITECNICO DI MILANO

SCHOOL OF INDUSTRIAL AND INFORMATION ENGINEERING
DEPARTMENT OF AEROSPACE SCIENCE AND TECHNOLOGY

Master of Science in
SPACE ENGINEERING



Enhancing smallsats lifecycle management by Model-Based Systems Engineering and Decision-Making techniques merging

Supervisor:

Prof. Michèle Lavagna

Author:

Paolo Minacapilli
914961

ACADEMIC YEAR 2020-2021

Abstract

Traditional Systems Engineering approaches highlight some bottlenecks whenever dealing with information exchange among stakeholders, typically producing a large number of documents, difficult to trace and to keep harmonized. This is particularly true for space projects, which entail very complex systems conceivment, design, integration and operation by a number of different players who grow with mission complexity. Model-Based System Engineering (MBSE) is intended to facilitate these activities, providing a common source of truth to the system engineering ecosystem, improving its efficiency and quality by developing a model that evolves along the entire product lifecycle.

This thesis applies an MBSE methodology to the Phase A of the European Space Agency e.Inspector CubeSat mission for all Systems Engineering practices: from the high-level mission objectives definition, through an articulated internal system functional analysis, down to physical architecture and interface engineering, up to mission phases and system modes modeling and concept of operations. Every step is harmonized with requirements definition within the model and their traceability. The approach also proposes the AIV/AIT plan modeling, to be exploited during the operative system development activities. The study is conducted according to the ARCADIA method and adopting the Capella tool, being very effective in mastering different engineering levels with coherence and with an iterative information refinement.

MBSE lacks intelligent support that could strongly help in addressing the most suitable system architecture in line with the high-level mission requirements, speeding up the alternatives selection process. This would be particularly useful during the preliminary design phases, in which the almost infinite design choices are skimmed by the only systems engineers knowledge, who may miss some solutions. A newly approach conceived to make up for this lack is also presented in this thesis, in the form of a decision-making tool prototype, which implements a tailored algorithm that correlates a set of functionalities with a set of available technologies, proposing one or more architectures that are coherent with what the engineers expect from the system behaviour. Such tool is interfaced with the previous MBSE environment, providing as output the modeled elements of the proposed architecture too and a first grid requirements.

Keywords: MBSE, Model-Based Systems Engineering, Small Satellites, Systems Engineering, Decision-Making Methods, CubeSats.

Sommario

Gli approcci tradizionali di ingegneria dei sistemi evidenziano delle difficoltà nello scambio di informazioni tra le parti interessate, con una tipica produzione di una grande quantità di documenti difficili da tracciare e tenere armonizzati. Ciò si verifica in particolare nei progetti spaziali, in cui sistemi molto complessi vengono concepiti, progettati, integrati ed operati da un numero di figure coinvolte che aumentano con la complessità della missione. Per facilitare queste attività si può fare riferimento agli approcci Model-Based Systems Engineering (MBSE), che aumentano l'efficienza e la qualità dello scambio di informazioni, all'interno dell'ecosistema di ingegneria dei sistemi, sviluppando un modello che si evolve lungo l'intero corso di vita del sistema da realizzare.

Questa tesi applica una metodologia MBSE alla Fase A della missione CubeSat e Inspector dell'Agenzia Spaziale Europea, per tutte le pratiche di ingegneria dei sistemi: dalla definizione degli obiettivi principali di missione, passando attraverso una articolata analisi funzionale interna di sistema, fino alla architettura fisica ed alla ingegneria delle interfacce, concludendo con la modellazione delle fasi di missione, i modi di sistema e la sequenza di operazioni del satellite. Ogni passo è armonizzato tramite la definizione, all'interno del modello, dei requisiti e dalla loro tracciabilità. L'approccio propone anche la modellazione del piano di AIV/AIT, da utilizzare durante le attività operative di sviluppo del sistema. Lo studio è condotto utilizzando il metodo ARCADIA ed il software Capella, che risulta essere molto efficace nel padroneggiare i diversi livelli di ingegneria in modo coerente e con un raffinamento iterativo delle informazioni.

MBSE manca di un supporto decisionale intelligente che potrebbe aiutare fortemente nell'identificare l'architettura di sistema che più si addice ai requisiti principali di missione, con una conseguente accelerazione nel processo di selezione delle alternative. Ciò sarebbe particolarmente utile durante le fasi preliminari di progettazione, in cui le scelte progettuali pressoché infinite vengono scremate dalle sole conoscenze dei sistemisti, ai quali possono sfuggire alcune soluzioni. In questa tesi viene anche presentato un nuovo approccio concepito per sopperire a questa mancanza, proponendo un prototipo di strumento decisionale che implementa un algoritmo su misura per la correlazione di un insieme di funzionalità con un insieme di tecnologie disponibili, proponendo una o più architetture coerenti con ciò che gli ingegneri si aspettano dal sistema. Tale strumento si interfaccia con il precedente ambiente MBSE, fornendo in output anche gli elementi modellati dell'architettura proposta ed una prima griglia di requisiti.

Keywords: MBSE, Model-Based Systems Engineering, Piccoli Satelliti, Ingegneria dei Sistemi, Metodi Decisionali, CubeSats.

Acknowledgements

It is worldwide recognized that the most difficult task in any research work is the acknowledgements writing, due to the so numerous feelings and the so little space available to express them all. I will try to do my best.

First, I would like to declare my immense gratitude to my supervisor, Prof. Michèle Lavagna, for letting me find my professional interests thanks to the passion, the enthusiasm and the immense knowledge she transmitted me during any (unfortunately virtual) meeting. I am also grateful for the numerous valuable experiences she offered me, starting from the involvement in the e.Inspector mission, actively joining several reviews with the European Space Agency, up to the technical conferences to which she allowed me to contribute with my work, such as the 5th ESA CubeSat Industry Days and the next 72nd International Astronautical Congress. On this trail, I would also like to thank the whole ASTRA team research group for having welcomed me and for the availability in the last months.

Everything I have done during the master has had the constant presence of my colleague and friend Giacomo, to whom I am grateful for the mutual charge we gave each other during exam sessions. I also thank the *Moscova guys* for the long conversations had during lunch breaks at university and around Milan.

The last year of master has been full of new people that I (virtually) met and that I will surely remember in the future when thinking about university years. I want to thank Stefano, Lorenzo and the whole *CRETULA Team*. I also thank PoliSpace, the space association I joined at Politecnico di Milano, and the whole team I am working with. An implicit thanks goes to all the people I got to know inside and outside university.

A few lines are not enough to express the importance and the centrality my parents have had during my study path. During moments like this, in which I force myself trying to find the right words to describe a too complex baggage of feelings, I retrace all the positive and negative experiences of the last years, and the first images or sounds that come to my mind are linked to my mom and dad, always there to encourage me. For this and much more, thank you. I also wish to dedicate spacial thanks to my sister Giuliana and to my grandmothers, *grazie Nonne*.

I cannot conclude without mentioning my historical friends, the *Artisti Uniti*, for having lightened my mind whenever needed with funny moments and a lot of sport. You are a certainty in my life. Thank you Giorgio for having always been interested about my work and for your LaTeX skills, this is for you.

I want to thank the person with whom I shared every little moment in the last years, Alice, who provided an immense support to me and considerable advice. I do not know what the future foresees, in the meanwhile thank you for having been stand next to me in any occasion.

Lastly, I recognize how different I am with respect to the young freshman of few years ago. For having prepared me to new adventures with polytechnic values, thank you PoliMi.

Contents

List of Figures	xiv
List of Tables	xv
List of Acronyms	xvii
1 Introduction	1
1.1 Space Systems Engineering Background	1
1.2 Towards MBSE for Small Satellites Design	6
1.2.1 Small Satellites Design Context	6
1.2.2 Literature Review of MBSE Applied to Space Missions	7
1.3 Thesis Motivation and Objectives	9
1.4 Thesis Outline	11
2 Model-Based Systems Engineering Concepts	13
2.1 What is MBSE?	13
2.2 MBSE Ingredients	16
2.3 MBSE State of the Art	17
2.4 The ARCADIA Method in a Nutshell	20
2.4.1 Users Needs Understanding	20
2.4.2 Solution Architectural Design	23
2.4.3 ARCADIA Diagrams	25
2.4.4 The Capella Tool	26
3 Case Study: e.Inspector	27
3.1 The e.Inspector Mission	27
3.2 Model Implementation	29
3.2.1 Requirements Management	29
3.2.2 Operational Analysis	33
3.2.3 System Analysis	36
3.2.4 Logical Architecture	42
3.2.5 Physical Architecture	56
3.2.6 System and Subsystems Modes	72
3.2.7 Concept of Operations using Scenario Diagrams	79
3.2.8 AIV/AIT Plan Definition with Capella	87
4 Decision Making Tool for Small Satellites Architectures Generation	93

4.1	Statement of the Problem and Methodology	93
4.1.1	Inputs from the User	94
4.1.2	Tool Embedded Decision Tree	95
4.2	The Algorithm Explained	97
4.2.1	Some Ingredients of the Algorithm: MCDM Methods	98
4.2.2	Level 1 Architectures Selection	101
4.2.3	Level 2 Architectures Selection	106
4.3	Simulations and Validation	110
4.3.1	Subsystems Decision Tree	110
4.3.2	Results and Discussion	116
5	Conclusions and Future Work	123
5.1	Summary of the Results	123
5.2	Limitations of the Study and Future Developments	124
5.3	Final Thoughts	125
A	Appendix - Capella Diagrams	127
A.1	Requirements Trees	127
A.2	System Analysis Diagrams	131
A.3	Logical Architecture Diagrams	132
A.4	AIV/AIT Diagrams	135
B	Appendix - Decision-Making Tool	139
B.1	Functionalities and Alternatives Markers	139
B.2	Algorithms	145
	References	149

List of Figures

- 1.1 The systems engineering engine [36] 3
- 1.2 ESA Project Lifecycle [66] 4
- 1.3 V-Model [25] 4
- 1.4 Life cycle costs against time [60] 5
- 1.5 CubeSats launches trend in the last two decades [43] 6
- 1.6 Layout of the ESA/ESTEC Concurrent Design Facility [7] 10

- 2.1 The Input-Output iterative SE process [81] 14
- 2.2 MBSE factors related to investments and gains [45] 15
- 2.3 Document-based VS Model-Based information exchange. Credit: ESA . . . 16
- 2.4 The PMTE elements and interactions with technology and people [25] . . . 17
- 2.5 SysML Diagram Types [1] 18
- 2.6 ARCADIA method phases [54] 20
- 2.7 Concepts and relations concerning the Operational Analysis [75] 21
- 2.8 ARCADIA Flow Control Functions 23
- 2.9 Concepts and relations concerning the System Analysis [75] 23

- 3.1 e.Inspector mission scheme 28
- 3.2 e.Inspector 12U CubeSat deployed configuration [21] 29
- 3.3 Requirements - Capella Types Folder 30
- 3.4 Requirements - Mass Editing View 31
- 3.5 Requirements - Capella Module 31
- 3.6 Requirement Example 31
- 3.7 [OAB] Requirements - GLOBAL MISSION 32
- 3.8 [OAB] Requirements - TT&C 32
- 3.9 [OCB] Operational Capabilities 34
- 3.10 [OES] Mission Operations Commands Management 34
- 3.11 [OAB] e.Inspector GLOBAL 35
- 3.12 [MCB] Mission & Capabilities 37
- 3.13 [SDFB] Provide Power Supply 38
- 3.14 [SDFB] Provide Communication Services 38
- 3.15 [SDFB] Provide On-board Data Handling 39
- 3.16 [SAB] e.Inspector GLOBAL 40
- 3.17 [SFBD] Root System Functions 41

3.18	[LAB] EPS SS	44
3.19	[LFCD] EPS Initialization and Solar Arrays Deployment	45
3.20	[LFCD] Battery Recharging from Solar Arrays Power	45
3.21	[LAB] OBDH SS	47
3.22	[LAB] GNC SS	49
3.23	[LAB] TT&C SS	50
3.24	[LAB] PROPULSION SS	51
3.25	[LAB] TCS SS	52
3.26	[LAB] STRUCTURES & MECHANISMS SS	52
3.27	[LAB] e.Inspector GLOBAL	54
3.28	[LFBD] Execute close-up visual inspection of a space debris	55
3.29	Physical Links Legend	56
3.30	[PAB] EPS SS Internal Physical Links	57
3.31	PDU's Replicas	58
3.32	[PAB] EPS SS - Power Distribution Unit 1	59
3.33	[PAB] EPS SS - Power Distribution Unit 2	60
3.34	[PAB] OBDH SS Internal Physical Links	61
3.35	[PDFB] Execute Actuator Control	61
3.36	[PAB] OBDH SS - GNC	62
3.37	[PAB] OBDH SS - Telemetry Reading	63
3.38	[PAB] OBDH SS - GNC Telemetry Reading	64
3.39	[PAB] OBDH SS - Telemetry and Data Downstream	64
3.40	[PAB] GNC SS Components	65
3.41	[PAB] GNC SS - DOCK-GNC - Sensors	66
3.42	[PAB] GNC SS - DOCK-GNC - Actuators	66
3.43	[PAB] TT&C SS Components	67
3.44	[PAB] PROPULSION SS	68
3.45	[PAB] STRUCTURE&MECHANISM SS Components	69
3.46	[PAB] EPS SS Mass	70
3.47	[PAB] ALL SUBSYSTEMS Mass	70
3.48	[PCBD] e.Inspector Product Tree	71
3.49	[M&S] GNC Modes	74
3.50	Expanded view of GNC Detumbling Mode	75
3.51	[M&S] PROPULSION Modes	75
3.52	[M&S] TT&C Modes	76
3.53	[M&S] SYSTEM Modes - LEOP Phase (1)	77
3.54	Expanded view of SYSTEM Deployment Mode	77
3.55	[M&S] SYSTEM Modes - TRANSFER Phase (2)	77
3.56	[M&S] SYSTEM Modes - INSPECT Phase (3)	78
3.57	[M&S] SYSTEM Modes - DISPOSAL Phase (4)	78
3.58	[PES] ConOps - LEOP Phase (1)	81
3.59	[PES] ConOps - TRANSFER Phase (2)	82

3.60	[PES] ConOps - Relative Operations	83
3.61	[PES] ConOps - INSPECT Phase (3)	84
3.62	[PES] ConOps - DISPOSAL Phase (4)	85
3.63	[PES] Provide Communication Services	85
3.64	[PES] Provide Power Supply	86
3.65	[PAB] AIV/AIT EPS - Overall Plan	89
3.66	Link to [PDFB] - Functional Tests of SA	89
3.67	[PDFB] AIV/AIT Procedures - Functional Tests of SA	90
3.68	AIV/AIT Functions - Progress Status Sheet	90
3.69	[PFCD] AIV/AIT - EPS testing activities	91
3.70	AIV/AIT EPS DOCK - ACUs Interface Tests Expanded View	92
4.1	Decision Tree Structure	97
4.2	Flow chart of the decision-making algorithm. Light green is used for the steps belonging to the first level, light orange for the second level ones.	99
4.3	Example of Components Modeling in Capella	109
4.4	Decision Tree of ADCS	111
4.5	Decision Tree of Propulsion Subsystem	111
4.6	Decision Tree of EPS	115
4.7	Decision Tree of TT&C	116
4.8	Simulation 1 - L1 Architectures	118
4.9	Simulation 1 - L2 Architectures related to the 2-2-2-2 L1 Architecture	119
4.10	Simulation 2 - L1 Architectures	120
4.11	Simulation 2 - L2 Architectures related to the 1-1-2-1 L1 Architecture	121
4.12	Reaction Wheel Node Component for Decision-Making Tool	122
5.1	INCOSE MBSE Roadmap [71]	124
A.1	[OAB] Requirements - EPS	127
A.2	[OAB] Requirements - TCS	127
A.3	[OAB] Requirements - GS	128
A.4	[OAB] Requirements - MOC	128
A.5	[OAB] Requirements - OBDH	128
A.6	[OAB] Requirements - PL	129
A.7	[OAB] Requirements - PROP	129
A.8	[OAB] Requirements - STR&MECH	129
A.9	[OAB] Requirements - GNC	130
A.10	[SDFB] Approach Target Debris	131
A.11	[SDFB] Deorbit at EOL	131
A.12	[SDFB] Provide Propulsion	131
A.13	[SDFB] Provide Protection against Mechanical Loads	132
A.14	[SDFB] Provide Protection against Temperature	132
A.15	[LFCD] GNC SS - Close Proximity Relative Navigation during Eclipse	133

A.16 [LFCD] OBDH SS - Payload Data Acquisition, Storage and Transmission .	133
A.17 [LFCD] OBDH SS - System Initialization	133
A.18 [LFCD] OBDH SS - System Passivation	133
A.19 [LFCD] PROP SS - Thrusting to Drifting Orbit Initialization	134
A.20 [LFCD] TT&C SS - Acquired Target Images Downlink Line	134
A.21 [LFCD] TT&C SS - Telecommands Uplink Line	134
A.22 [LFCD] TT&C SS - Telemetry Downlink Line	134
A.23 [PAB] AIV/AIT GNC - Overall Plan	135
A.24 [PAB] AIV/AIT PROP - Overall Plan	135
A.25 [PAB] AIV/AIT OBDH - Overall Plan	136
A.26 [PAB] AIV/AIT TCS - Overall Plan	136
A.27 [PAB] AIV/AIT STR&MECH - Overall Plan	137
A.28 [PAB] AIV/AIT TT&C - Overall Plan	137

List of Tables

- 3.1 State Machine model elements 73
- 3.2 Type of messages in Scenario Diagrams 80

- 4.1 Indexes involved in the decision-making problem 97
- 4.2 Scale of relative importance in AHP 100
- 4.3 Example of Pairwise Matrix in AHP 100
- 4.4 Common actuators for small satellites [80] 112
- 4.5 Common sensors for small satellites [80] 112
- 4.6 Common chemical propulsion technologies for small satellites [80] 113
- 4.7 Common electric propulsion technologies for small satellites [80] 114
- 4.8 Common secondary batteries for small satellites [80] 114
- 4.9 Common high gain antennas for small satellites 115
- 4.10 Common low gain antennas for small satellites 116
- 4.11 Simulation 1 - Overall Architecture related to the 2-2-2-2 L1 Architecture 118
- 4.12 Simulation 2 - Overall Architecture related to the 1-1-2-1 L1 Architecture 121

- B.1 Functionalities Markers of Simulation 1 139
- B.2 Functionalities Markers of Simulation 2 140
- B.3 ADCS L1 and L2 Alternatives Markers 141
- B.4 Propulsion Subsystem L1 and L2 Alternatives Markers 142
- B.5 EPS L1 and L2 Alternatives Markers 143
- B.6 TT&C L1 and L2 Alternatives Markers 144

List of Acronyms

A1	Alternative 1
A2	Alternative 2
ACU	Antenna Control Unit
ADCS	Attitude Determination And Control System
ADN	Ammonium dinitramide
ADR	Active Debris Removal
AFM	Alternatives Functionalities Matrix
AHP	Analytic Hierarchy Process
AIT	Assembly, Integration and Test
AIV	Assembly, Integration and Verification
ALT	Alternative
ARCADIA	Architecture Analysis & Design Integrated Approach
ASTRA	Advanced Space Technologies for Robotics and Astrodynamics
CAM	Collision Avoidance Maneuver
CDF	Concurrent Design Facility
COTS	Components Off The Shelf
D1	Decision Level 1
D2	Decision Level 2
DSML	Domain Specific Modelling Language
EOL	End Of Life
EPS	Electrical Power System
ESA	European Space Agency
ESTEC	European Space Research and Technology Centre
FPA	Flat Panel Antenna
GNC	Guidance, Navigation and Control
GNSS	Global Navigation Satellite System
GS	Ground Station
HAN	Hydroxylammonium nitrate
HET	Hall-Effect Thruster
HGA	High-Gain Antenna
IBM	International Business Machine
IEC	International Electro-technical Commission
IFM	Input Functionalities Matrix
IMU	Inertial Measurement Unit
INCOSE	International Council On Systems Engineering

ISO	International Organization for Standardization
L1	Level 1
L2	Level 2
LA	Logical Architecture
LAB	Logical Architecture Blank
LEO	Low Earth Orbit
LEOP	Launch and Early Orbit Phase
LFBD	Logical Functional Breakdown
LFCD	Logical Functional Chain Description
LGA	Low-Gain Antenna
M&S	Mode & State
MarCO	Mars Cube One
M-ARGO	Miniaturised Asteroid Remote Geophysical Observers
MB4SE	Model-Based For Systems Engineering
MBSE	Model-Based Systems Engineering
MCB	Mission Capabilities Blank
MCDM	Multi-Criteria Decision Making
MMH	Monomethylhydrazine
MOC	Mission Operations Center
NASA	National Aeronautics and Space Administration
NATO	North Atlantic Treaty Organization
NB	Non Beneficial
OA	Operational Analysis
OAB	Operational Architecture Blank
OBC	On-Board Computer
OBDH	On-Board Data Handling
OCB	Operational Capabilities Blank
OES	Operational Exchange Scenario
OFM	Output Functionality Matrix
OFV	Output Functionality Vector
OHB	Orbitale Hochtechnologie Bremen
OMG	Object Management Group
PA	Physical Architecture
PAB	Physical Architecture Blank
PCBD	Physical Components Breakdown
PDFB	Physical Data Flow Blank
PDU	Power Distribution Unit
PES	Physical Exchange Scenario
PFCD	Physical Functional Chain Description
PFM	Proto-Flight Model
<i>PL</i>	Payload
PLATO	PLANetary Transits and Oscillations of stars

PMTE	Process Method Tool Environment
POLIMI	Politecnico di Milano
RAX	Radio Aurora eXplorer
REC	Replicable Elements Collection
ReqIF	Requirements Interchange Format
RPL	Replica
SA	System Analysis
SAB	System Architecture Blank
SDFB	System Data Flow Blank
SE	System Engineering
SFBD	System Functional Breakdown
SP	Solar Panels
SS	Sun Sensor
SS	Subsystem
SysML	Systems Modelling Language
TCS	Telecommunication System
TT&C	Telemetry, Tracking and Command
UHF	Ultra High Frequency
UML	Unified Modeling Language
VHF	Very High Frequency

Chapter 1

Introduction

The whole is more than the sum of its parts.

ARISTOTLE

The purpose of this chapter is to introduce the reader to the vast discipline of Systems Engineering, with emphasis on space systems. The aim is not to provide a thoroughly description of all systems engineering practices, as it would not be possible to do in a few pages, but to highlight some key features of this broad field of knowledge, in order to better catch the context of applicability of the thesis. A discussion on Model-Based Systems Engineering follows, in order to define the open points found in literature to which the thesis aims to provide an answer. Lastly, the work outline is introduced.

1.1 Space Systems Engineering Background

Space Industry is a very dynamic and challenging environment in which the most (apparently) inhomogeneous companies concur and collaborate toward a shared objective, that is to furnish a certain kind of product/service. The word inhomogeneous refers to the great variety of disciplines that a space item asks for its realization, be it a satellite, a launcher or whatever complex system intended as a combination of elements that provide a capability not attainable by single parts alone. Dealing with such complexity has forced organizations to formalize the concepts related to the design, testing and operation of systems, leading to the definition of the Systems Engineering discipline.

The modern origins of Systems Engineering can be traced to the 1930's [30] and the roots of its formalization date back to 1962, when Arthur David Hall defined it in *A Methodology for Systems Engineering* [29] as a process with five phases [33]:

1. Perform system studies (program planning);
2. Define the problem definition, select the objectives, perform systems analysis, select the best system;

3. Repeat phase 2 in more detail (development planning);
4. Develop parts of the system, integrate and test them;
5. Operate the system (current engineering).

Such list is quite in line with current SE practices, which refer to the international standard ISO/IEC 15288 introduced in 2002, when the systems engineering discipline was officially recognized as the *preferred mechanism* to establish a collaborative environment between acquirers and suppliers in the realization of a product/service [30]. Many definitions of SE followed that date, some of which are here provided with a focus on those coming from space agencies:

- “Systems engineering is an interdisciplinary approach and means to enable the realization of successful systems” - *International Council on Systems Engineering (INCOSE)* [30].
- “Systems engineering is a methodical, multi-disciplinary approach for the design, realization, technical management, operations, and retirement of a system” - *NASA Systems Engineering Handbook* [36].
- “System engineering is an interdisciplinary approach governing the total technical effort to transform requirements into a system solution” - *ESA ECSS-E-ST-10C Rev.1* [65].

In other words, systems engineering is the process which ensures that the customer’s needs are satisfied throughout the entire product life cycle; to accomplish this task, systems engineers require an holistic knowledge of all the involved technical disciplines used to define requirements, evaluate design tradeoffs, analyse technical risks and many others, at the same time being able to manage the project, defined as a temporary endeavors undertaken to create a unique product or service [35], providing programmatic, cost and schedule inputs. The way SE supports the development and realization of end products is summarized in Figure 1.1.

Concerning space projects, their life cycle is typically divided into 7 phases, here presented according to the ESA standards ECSS-E-ST-10C Rev.1 [65]:

- Phase 0 - Mission analysis/needs identification: understand customer needs, propose possible mission/system concepts, draft system-level requirements.
- Phase A - Feasibility: finalize the expression of the needs, propose system solutions to meet the customer expectations.
- Phase B - Preliminary Definition: preliminary define the system solution, demonstrate that the solution meets the technical requirements according to the schedule, the target cost and the customer requirements.

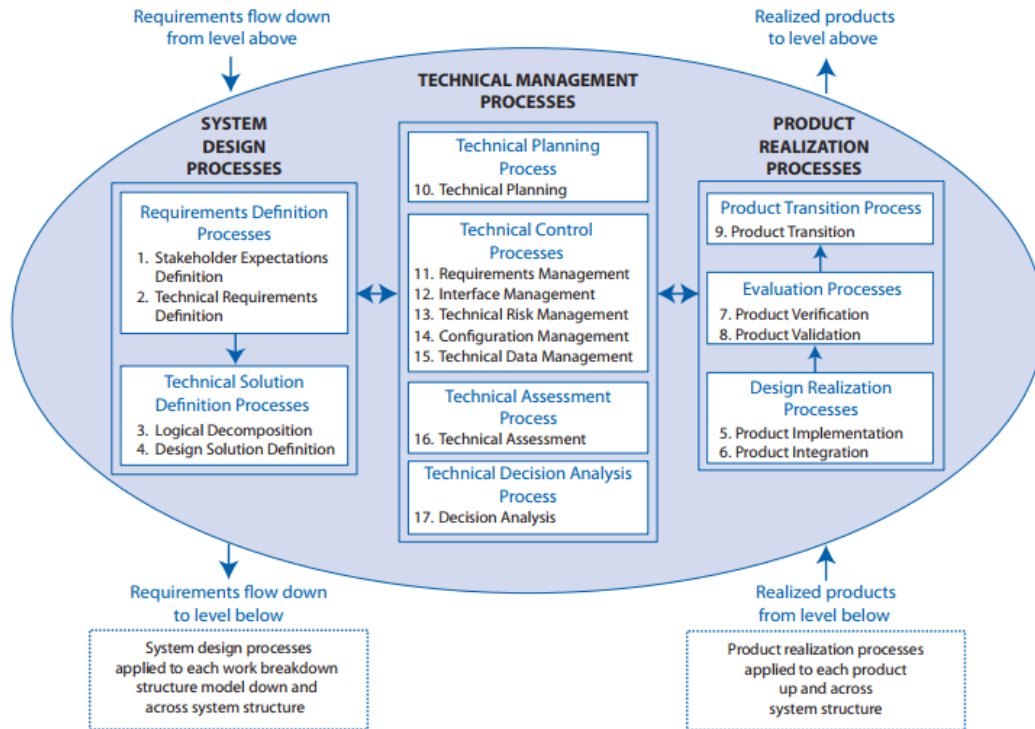


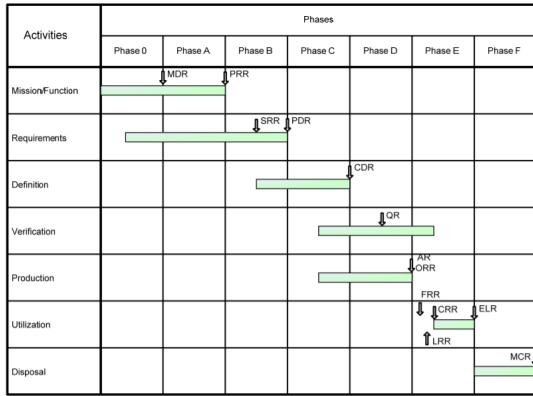
Figure 1.1: The systems engineering engine [36]

- Phase C - Detailed Definition: establish the system detailed definition.
- Phase D - Qualification and Production: finalizes the development of the system by qualification and acceptance, finalize the preparation for operations and utilization.
- Phase E –Utilization: supports the launch campaign, supports the entity in charge of the operations, provide documents in support to anomaly investigations and resolutions.
- Phase F – Disposal: supports the entity in charge of the disposal to safely dispose all products launched into space as well as ground segment.

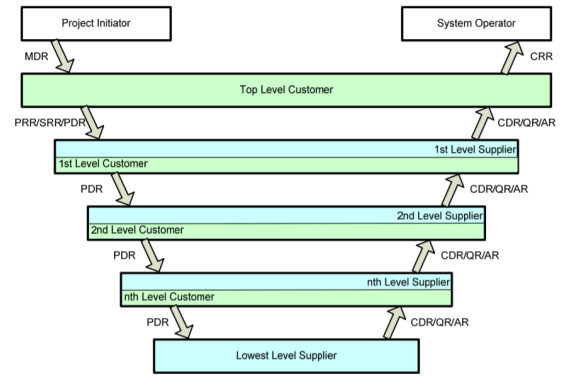
Each phase includes end milestones in the form of project reviews, the outcome of which determines readiness of the project to move forward to the successive phase, as shown in Figure 1.2.

The need of an effective approach to manage SE practices across life cycle stages led to the creation of different models in the last decades, mostly inherited from software development procedures, which are applied depending on the kind of product to be realized. A brief description of the most popular ones is here reported:

- Waterfall (1970) [56]: sequential process flowing downwards through all the required stages. For projects in which requirements are well understood from the beginning and not likely to change during the execution of a project.



(a) ESA Project Phases and Reviews



(b) ESA V-Model

Figure 1.2: ESA Project Lifecycle [66]

- Spiral (1986) [9]: it is actually a class of incremental and iterative methods, useful when requirements are unclear or uncertain at the start of a project. If the design is acceptable, additional level of detail is developed; if not, the process iterates up to the obtainment of a design that better matches stakeholder needs.
- V-model (1991) [26]: the design evolves over time from stakeholder requirements, to the system concept and finally, the elements of design (left branch of the V, top-down). The integration and testing activities follow in the right branch of the V (bottom-up). This method is sequential, however it allows iteration and recursion since the two branches of the V are linked by means of verification and validation bridges (Figure 1.3).

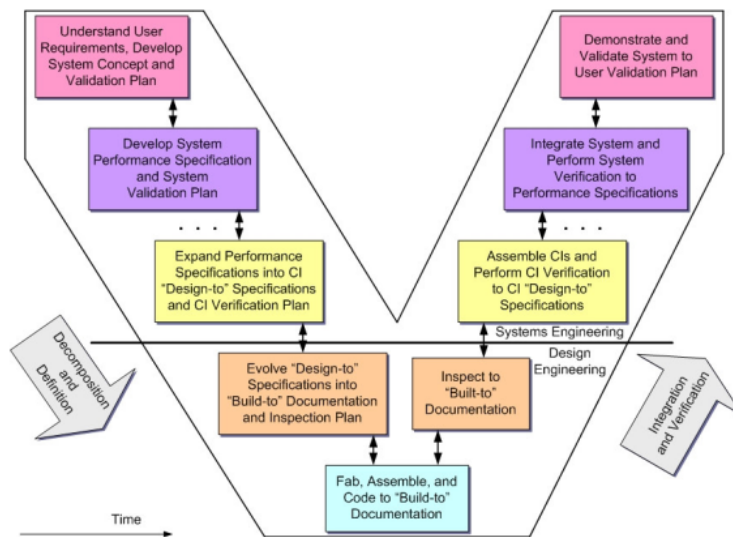


Figure 1.3: V-Model [25]

The most widely accepted approach is the V-Model, as it embeds the transformation of broad mission objectives into quantifiable design parameters constantly assessing the

user expectations. To this aim, *requirements* represent a key concept and pillars of SE, as they allow to define formally, correctly and in a quantifiable way the expected needs of the final users. A better characterization of requirements, with particular focus on space-related ones, is reported in Section 3.2.1. Requirements must be continuously critically assessed as they play the role of driving the engineering process and they are used to verify and validate whether the designed system will meet the customer’s actual needs. *Verification* is the determination that each element of the system meets the requirements of a documented specification; it ensures you are building the system right. *Validation* is the determination that the entire system meets the high level objectives; it ensures you are building the right system.

As products are steadily growing in complexity, physical and functional connections between components are becoming increasingly important, implying an overall increased probability of committing errors. A direct consequence is the centrality of *risk management* within projects in order to reduce the probability of late design changes. As the project proceeds, costs to make corrective actions increase, therefore one of the primary goals of systems engineers is to avoid late modifications which may drastically impact the project in terms of costs, and consequently time. This concept is presented in Figure 1.4, where the actual life cycle costs (green rectangles) are reported together with foreseen costs (committed costs curve). The blue arrow indicates that errors are less expensive to remove early in the life cycle [30].

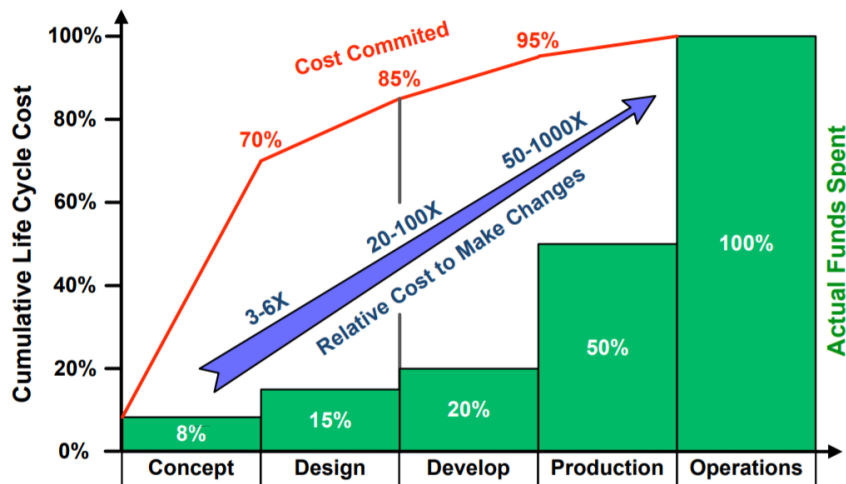


Figure 1.4: Life cycle costs against time [60]

It is then fundamental to verify, even in early design phases, the adequacy of the solution in relation to the needs and constraints, reducing the risk of re-evaluating the architecture in advanced development phases. To do that, information should be exchanged in the easiest way among engineers and shared with stakeholders. This is often not so trivial in large projects, as all the SE practices currently rely on a complex exchange of written documents, which are prone to inconsistencies and inherently suffer of problems related to the timeliness of information [44]. In particular, space systems ex-

hibit a wide number of subsystems interactions, involving a lot of expertise coming from different domains, therefore the enormous quantity of exchanged information and data requires a method and a “place” where to orchestrate the systems engineering practices and guarantee the traceability between the problem statement, the solution definition and the system verification/validation. To cope with such aspects, Model-Based Systems Engineering approaches provide a better quality interactions among engineers and with the involved stakeholders, facilitating the early recognition of errors and supporting complex systems management, as discussed in the next section.

1.2 Towards MBSE for Small Satellites Design

1.2.1 Small Satellites Design Context

Over the last decades the space sector is experiencing a fast growing thanks to the advancements in miniaturization of electronics, that allow the development of smaller platforms if compared to traditional ones. This fits into the space economy context, in which small platforms such as CubeSats are acquiring a significant importance due to their reduced mass, volume and consequently cost, the latter being one of the largest barriers to satellites development. Such revolutionary design philosophy is making small satellites a success story, confirmed by the diagram in Figure 1.5 which shows the total number of small satellites launched in the last two decades.

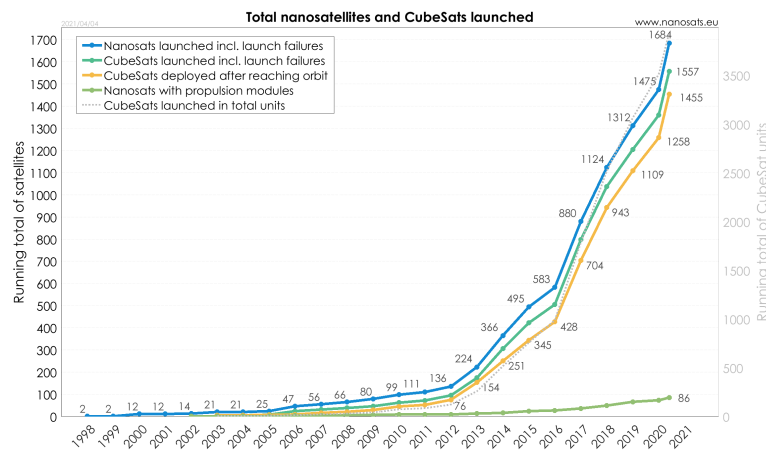


Figure 1.5: CubeSats launches trend in the last two decades [43]

Originally, the CubeSat standard was created by Stanford and California Polytechnic State Universities in 1999, specifying that a standard 1U unit is a 10 cm cube [52] with a mass of up to 2 kg [62]. Such standard is maintained whenever different form factors are needed, with configurations that can be made of several units. The original intention of CubeSats was to provide an educational experience to university students, allowing them to develop a real space project with affordable costs. However, the low cost, low development time, and the diffusion of commercial-off-the-shelf (COTS) components con-

tributed to a recent awareness spread among the space community, that is recognizing the value of such miniaturized platforms for a wide variety of mission scenarios: Earth remote sensing, telecommunications, astronomy, technology-demonstrator [13]. Nowadays, most of CubeSats missions are directed toward low Earth orbits (LEO), however in the last years a remarkable interest toward their exploitation for deep space exploration is being born, as in the case of the NASA's Mars Cube One (MarCO) mission launched on 5 May 2018 and headed for Mars [61]. Brand new missions are under development at ESA too (Hera, LUCE, M-ARGO and others), which are all going to write a new chapter in the solar system exploration using small platforms [77].

It is clear that the reduced costs and the miniaturization do not imply a reduced complexity, therefore it is important to not underestimate the engineering effort required in the design of small spacecrafts, particularly challenging due to their limited resources which have to cope at the same time with the inherent complexity of a space system. From the systems engineering methodology point of view, small satellites still rely on document-based approaches inherited from the traditional space industry. Without an alignment between emergent technologies and SE approaches, the risk is to postpone the further advancement of small satellites. In this framework, the steadily increasing use of Model-Based Systems Engineering in the space community perfectly matches the need of having a more clear and consistent way of doing systems engineering, improving the overall efficiency within organizations in order to better ride the wave of the incoming space exploration challenges, which ask for shorter design cycles and above all the need of an excellent understanding of customers' aspirations and goals [75].

1.2.2 Literature Review of MBSE Applied to Space Missions

Several developments in the last decades have significantly pushed forward the adoption and deployment of MBSE solutions in space programs [22], in order to streamline their systems engineering process, which ask for an increased efficiency in the design, development, deployment and verification. An example is the Model Based For System Engineering (MB4SE) initiative [48], a platform promoted by ESA where technical discussions about the deployment of MBSE in space projects take place. A literature research has been conducted for this work to assess the level of advancement of MBSE practices for space missions, not only CubeSat ones, with the aim of understanding lessons learned and open works.

The benefits of MBSE is being demonstrated across programs, such as the NASA Europa Clipper currently in Phase C, which demonstrated higher level thinking among engineers, improved access to information for new team members, saved time, prevented errors and minimized drudge work [8]. In ESA, an MBSE approach to the e.Deorbit mission for its Phase A to Phase B1 [57] was tested to assess the benefits and the implications with respect to the work of three contractors (Airbus Defence and Space, OHB

and Thales Alenia Space), who adopted their own methodology and tools [24]. The work was effective in maintaining the system complexity providing a holistic and collaborative view of the project; the application of MBSE represented a significant increase in the usage of SE practices with respect to previous ESA missions [23]. However the activity showed a limited success in performing reviews using models due to the lack of knowledge of non-practitioners who were not comfortable with the articulated diagrams [44]; the key lesson learned was then to provide tailored views of the models for people who want to judge the engineering work in specific domains.

The Euclid mission is the first ESA's attempt to apply a complete MBSE concept for a major project [4], as the sophisticated spacecraft to be launched in 2022 will study the geometry of the Universe by measuring the dark matter and dark energy distribution. Among the lessons learned there is a net benefit in terms of completeness of verification by full coverage check of requirements and a successful exploitation of model for mission reviews purposes, with a simpler identification of all interfaces and a coherent view of functions and allocation [44]. However the group also highlighted a delayed return-on-investment due to absence of a modeling background among engineers and managers, who still often rejected the concept. This was actually something expected since the Euclid mission started the adoption of MBSE during Phase B1, while it is important to smoothly introduce the community to MBSE starting from early phases and at the same time identifying the correct level of modeling details to avoid overload. The ESA Planetary Transit and Oscillations (PLATO) mission is also adopting a similar approach to Euclid, using as guidelines the numerous lessons learned. The work done by Chhaniyara et al. applied MBSE to space robotics systems [15], highlighting the need of MBSE for such complex project but also the necessity of having some modeling guidelines for large projects, not provided by the System Modeling Language (SysML).

Concerning CubeSats, the literature research performed for this thesis ran into a considerable work conducted by the Space Systems Working Group (SSWG) lead by David Kaslow and established by the INCOSE, with the purpose of promoting systems engineering principles and techniques, in particular model-based ones [34]. One of the main outputs obtained between 2012 and 2018 is the CubeSat Reference Model [38], a set of more than fifteen papers with the scope of proving the applicability of MBSE practices for designing CubeSats. The first phase of the project consisted in the application of an MBSE approach to a 3U CubeSat mission called RAX (Radio Aurora Explorer) [67, 68], demonstrating the applicability of MBSE for space missions. An integrated modeling approach was also proved in the second phase of the project bridging the MBSE model with ModelCenter, allowing to perform engineering analysis and simulations [39]. In the third and last phase the working group developed a CubeSat Enterprise model to capture cost and product life cycle aspects [5, 37].

Another MBSE study in support of nano-satellites development can be consulted in

[28], where the DelFFi mission of the Delft University of Technology has been used as case study. The study focused on requirements development in an MBSE environment, which revealed to be very effective for their traceability, and in the development of a collaborative workspace with proper construction of Simulink models to verify the design and to link MBSE with the onboard software, increasing the quality of the product.

Lastly, it is worth mentioning the application of MBSE for the AeroCube-10 mission developed by the Aerospace Corporation of El Segundo, composed by two 1.5U CubeSats to demonstrate satellite-to-satellite pointing operations and other technologies [17]. The whole life cycle has been explored using MBSE in order to cope with the mission complexity: requirements, concept of operations, verification activities, physical architecture and system-level analyses were modeled to provide the team a single source of truth. The team assessed that modeling forces conversations among engineers helping to early detect design errors; re-usability of model elements allowed to save time in future projects; the interfaces description was at a different level with respect to static document-centric approach; sudden changes of mission requirements were quickly captured [16].

1.3 Thesis Motivation and Objectives

Downstream the presented literature research it is possible to state that almost all projects benefit from model-based approaches. However, there is still a sort of repulsion by engineers who feel comfortable with text-based procedures, successful over the years. Also, the apparent complexity of models and languages discourages the community. As it is recognized the need of collecting more demonstrative applications of MBSE to small satellites design, given their fast growing in the space sector, this thesis work provides a complete modeling of a complex CubeSat mission in order to investigate which are the benefits in implementing MBSE for the whole life cycle of space systems and to address key engineering issues related to the approach. In order to demonstrate MBSE potential and gaps, the study passes through all the design phases, from high-level mission objectives definition and requirements modeling to functional analysis, physical architecture and interface engineering, concept of operations and modes definition, ending up with a newly approach for embedding AIV/AIT plan into the model. A real CubeSat development scenario is engaged to build the approach, namely the ESA e.Inspector mission, for which the study will serve as basis in future mission phases.

It is recalled that MBSE does not automate the design of a system, but represents a support to systems engineering practices which still have to be practiced by engineers. When a space mission is conceived, a large number of variables have to be faced by systems engineers who have to perform accurate subsystems sizing and consequent device selection in order to accomplish the mission objectives. In Phase 0/A, an elevated number of feasible architectures has to be reduced to no more than two consistent solutions. To do that, ESA performs such assessment studies in the Concurrent Design Facility (Figure

1.6), a place where a group of experts from several disciplines work together using a variety of sophisticated tools to assess new missions feasibility ensuring consistent and high-quality results in a much shorter time with respect to a sequential or centralised approach (before CDF a pre-phase A lasted 6-9 months and up to few weeks nowadays [7]).

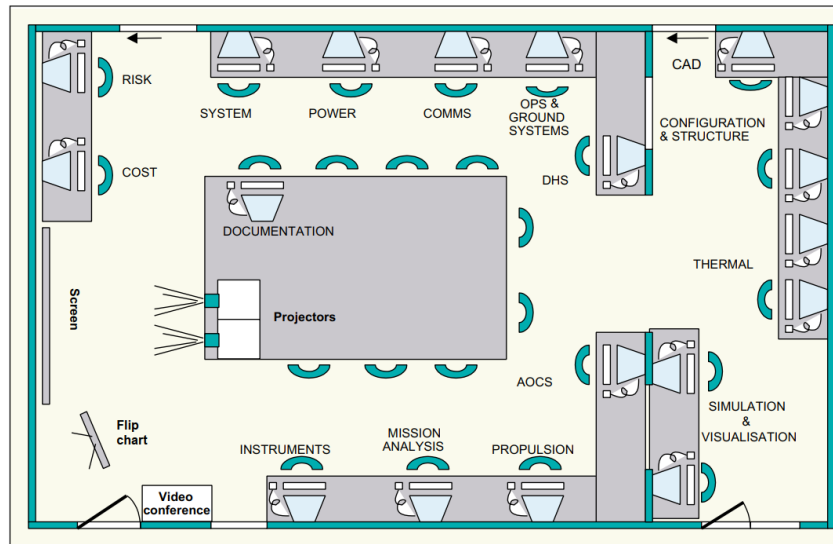


Figure 1.6: Layout of the ESA/ESTEC Concurrent Design Facility [7]

Concurrent engineering therefore deals with an important branch of SE, that is Decision Analysis, comprehensively described in [36] and defined as the “framework within which analyses of diverse types are applied to the formulation and characterization of decision alternatives that best implement the decision-maker’s priorities”. It is not so straightforward to skim the almost infinite design choices and decisions just relying on systems engineers knowledge, since such human brain-based method can miss innovative and still feasible solutions. Therefore, this thesis work also presents a newly approach conceived to extend the space engineers capabilities by developing a tailored decision-making tool that correlates a set of functionalities with a set of available technologies, proposing one or more architectures that are coherent with what the engineers expect from the system behaviour. The purpose is to assess the feasibility and the functioning of the developed algorithm in its preliminary prototype version, defining a virtual limited warehouse of small satellites technologies/components, which need less customization with respect to bigger spacecrafts, that can be automatically selected by the tool in order to obtain an architecture compliant with users needs. If used as support for an MBSE methodology, such as ARCADIA, the tool can overcome one of MBSE limits, that is the lack of intelligent capabilities which can accompany the modeler, enhancing the overall solution.

1.4 Thesis Outline

The thesis is organized into 5 chapters. **Chapter 1** introduces the systems engineering discipline and small satellites design context, presents a literature review of MBSE applied to space missions and defines the scope of the study. In **Chapter 2** an overview of MBSE languages, tools and methodologies is assessed, followed by a description of those adopted for this work. In **Chapter 3** an application of MBSE to a real CubeSat mission is presented in details, responding to the first thesis objective. **Chapter 4** illustrates the method and the algorithms behind the decision-making tool, followed by some simulations to validate it, responding to the second objective of the thesis. The conclusions to the research work are presented in **Chapter 5**, together with limitations of the study and future developments. Additionally, **Appendix A** adds some other diagrams related to the MBSE case study and **Appendix B** reports the algorithms and some tables developed for the decision-making tool.

Chapter 2

Model-Based Systems Engineering Concepts

All models are wrong, but some are useful

GEORGE E. P. BOX

In Section 1.2, some key principles of model-based systems engineering have been introduced. The purpose of this chapter is to define what MBSE is, its benefits and limits, and to present a state of the art review of the ingredients needed to formally apply it, selecting some of them to conduct the case study of Chapter 3.

2.1 What is MBSE?

Model-based engineering has been a topic of discussion for over twenty years, when the document-based SE approach disclosed its limits in terms of waste of time in writing and in consulting them. The need of a more direct source of information was already highlighted by the engineering community in late 1990s, however first MBSE approaches started to become popular in 2007, when the INCOSE published the *Systems Engineering Vision 2020* defining MBSE as “the formalized application of modeling 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” [71], with the main goal of increasing the productivity by minimizing the unnecessary manual transcription of concepts and having a single “source of truth” in large teams.

As discussed in Section 1.1, systems engineering is matter of iterative process between stakeholders and engineers until the design specification is matured. Figure 2.1 illustrates the continuous looping between systems engineering activities, called *SE process*. When all such activities are spread across written documents in a paper-based approach, the following issues arise:

- *Information Management*: in a document-based approach the information is mostly

textual and an immediate drawback is the waste of time related to documents writing. As documents are written by a number of people that increase with the project complexity, it often happens to have a repeated information in multiple documents which are loosely coupled. Such an approach is prone to misinterpretation as it depends on the “writer’s pen”, causing a further non-optimal use of time and team effort. The organization of reviews with stakeholders involvement also relies on lengthy procedures based of multiple documents exchange with an inherent difficulty of information traceability.

- *Requirements Traceability*: the main cause of failure in projects stays on requirements unclearness. Textual requirements are often difficult to interpret since they may appear as stand-alone sentences without a clear link to the system design quantities and qualities, with the consequent risk of generating ambiguities both internally to an organization and with respect to stakeholders.
- *Design Changes*: current practices foresee a consistent manual work to process a change of the design, even during early design phases. It is indeed required to span across multiple documents to ensure end-to-end traceability and consistency, slowing down the project being such activities time-consuming.

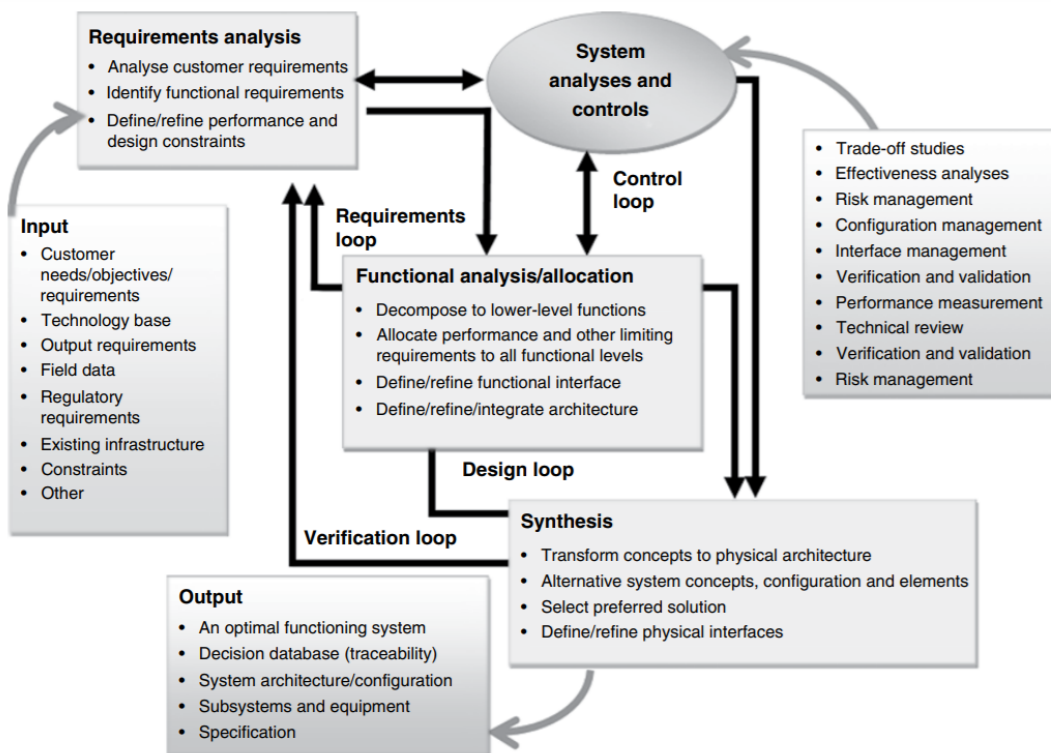


Figure 2.1: The Input-Output iterative SE process [81]

All the listed difficulties, merged with the current wave of digitalization which asks for an improved representation of systems development in order to optimize the overall product life cycle, provide an interesting research thread that can be identified with the

diffusion of Model-Based Systems Engineering. The very first advantage of MBSE is that the information is both visual and textual since it is contained in a *model*, defined in [27] as “a representation of one or more concepts that may be realized in the physical world” or “an abstraction of a system, aimed at understanding, communicating, explaining, or designing aspects of interest of that system” in [18]. MBSE is then about elevating models in the engineering process to a central role, to ensure specification completeness and consistency, traceability of requirements and design choices, reuse of design patterns and specifications which positively impact successive projects, and a shared understanding of the designs among users and designers [4].

Actually, all systems engineering has always been model-based; what changes with MBSE is the shift of the models repository from the lead engineers minds, who try to communicate it in order to align the team and ensure a shared understanding, to a digital representation accessible by any member of the working group. As the model becomes the single source of truth across multiple domains of an organization and with stakeholders, the information becomes unambiguous, accessible and intuitive, with a direct improved design team communication throughout the whole life cycle and a consequent improved product quality. A change in the design can be managed more easily with respect to a classical approach, due to the relationships across model elements such as requirements, functionalities, components etc. which define traceability paths and also allow early detection of errors. Figure 2.2 shows a comparison of system life cycle costs between traditional SE and MBSE approaches, highlighting that the main investment in MBSE is related to the infrastructure building and training of the personnel about the modeling language, the method and the adopted tool. These are actually the main cultural roadblocks that still prevent from a widespread awareness of how MBSE can enhance the SE practices.

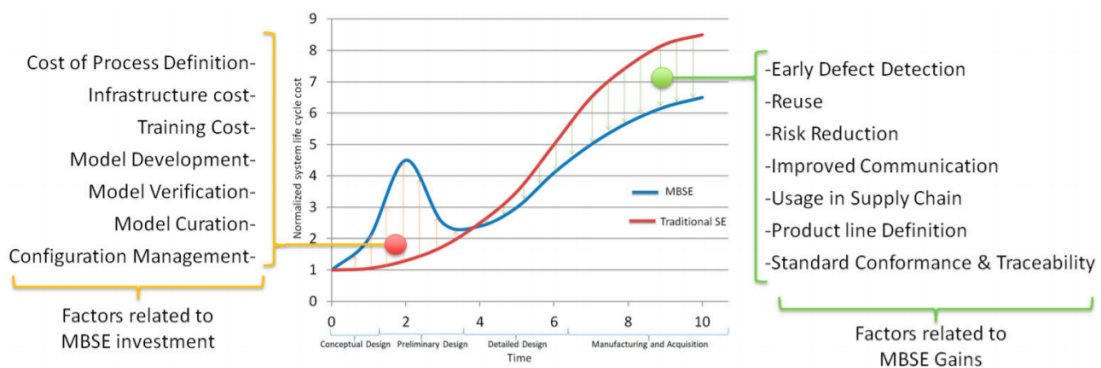


Figure 2.2: MBSE factors related to investments and gains [45]

2.2 MBSE Ingredients

Traditional systems engineering focuses on creating and managing documentation about a system among the engineering teams and the stakeholders. In MBSE a central system model is used to develop, manage and control relevant systems engineering information (Figure 2.3).

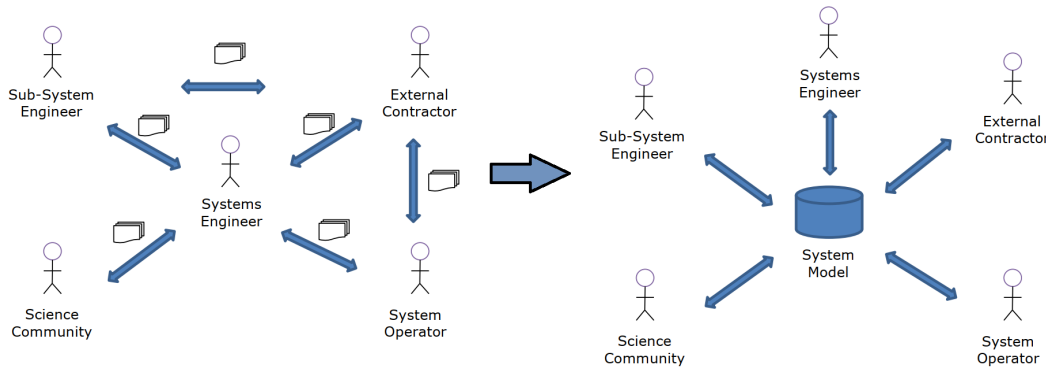


Figure 2.3: Document-based VS Model-Based information exchange. Credit: ESA

MBSE is not just a matter of doing diagrams to represent results, but it represents a support to systems engineering activities through modeling. Therefore, it requires a clear *methodology*, which is a collection of related processes, methods, and tools [47], define as follows:

- A *Process* 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* 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. Each method is also a process itself, with a sequence of tasks to be performed for it. The “how” at one level of abstraction becomes the “what” at the next lower level [25].
- A *Tool* is an instrument that, when applied to a particular method, facilitate the accomplishment of the tasks, provided it is applied properly and by somebody with proper skills and training. Most tools used to support systems engineering are computer- or software-based, which also known as Computer Aided Engineering (CAE) tools [25].

Those listed are the key ingredients of any MBSE approach, surrounded by the *Environment*, which consists in the external objects, conditions, or factors that influence the actions of an object, individual person or group [47]. Figure 2.4 shows the presented concepts and their relations.

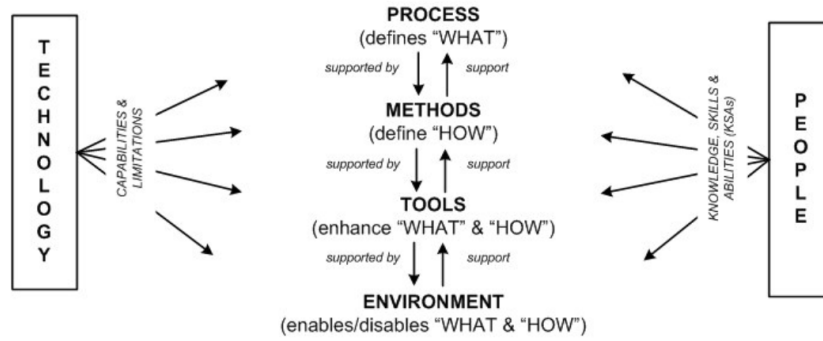


Figure 2.4: The PMTE elements and interactions with technology and people [25]

The main purpose of an MBSE approach is then to be able to integrate all these aspects, using a common terminology to clearly communicate what the model wants to capture. Therefore, a *modeling language* must also be introduced, each one characterized by its own syntax and semantics. *Syntax* refers to the structure of the language and can be abstract or concrete; the first one is related to constructs and rules for building the model, the second one is the set of symbols used to express the constructs. *Semantics* provide meaning for the constructs, therefore are the meanings associated to the constructs of a language.

2.3 MBSE State of the Art

Currently, one of the most common languages adopted in MBSE is the OMG Systems Modelling Language (SysML), initially developed to close the communication gap between systems engineers and software engineers [27]; its roots come from the Unified Modeling Language (UML), adding some concepts and removing others. Since it is just a language and not a methodology, it helps communicating amongst those trained with its notation without imposing a specified method on the MBSE approach.

SysML offers nine diagrams types to model a system, reported in Figure 2.5. Four of them are focused on *behavior*:

- The *Activity Diagram* is used to represent system behavior through a controlled sequence of actions that transform inputs to outputs;
- The *Sequence Diagram* provides representations of message based behavior. For example, interactions between parts and the flow of control, including the time variable;
- The *State Machine Diagram* is used to represent the life cycle of a block in response to event occurrences;
- The *Use Case Diagram* is adopted to describe basic system functionalities and the actors that invoke them.

Other four SysML diagrams are devoted to the *structure*:

- The *Block Definition Diagram* displays the system hierarchy and relationships between blocks to outline the system architecture;
- The *Internal Block Diagram* specifies the internal structure of a single block;
- The *Parametric Diagram* supports engineering analysis as it expresses constraints or equations that relate value properties;
- The *Package Diagram* displays how the model is organized.

Lastly, the *Requirement Diagram* is used to build text-based requirements trees and their relationships with other system elements. Each of the nine representations actually shows something different, therefore a critical task of systems engineer is to guarantee connectivity and coherence between the views.

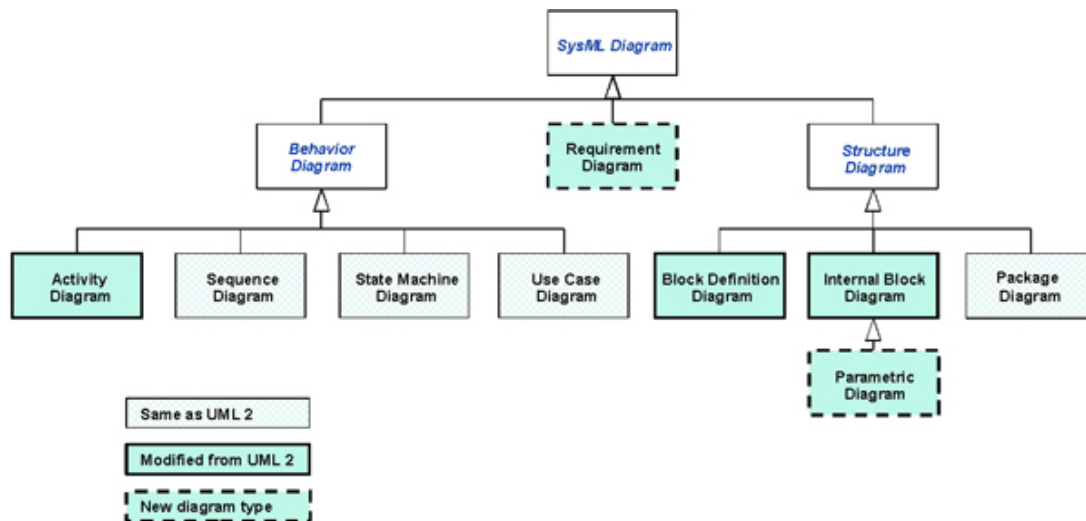


Figure 2.5: SysML Diagram Types [1]

SysML is just a language and in order to be adopted it needs a tool (some examples are IBM Rhapsody [32], No Magic’s Cameo Systems Modeler [69], Enterprise Architect [70] etc.) and, most important, a methodology behind. The most widespread MBSE methodologies which use SysML are the INCOSE Object-Oriented Systems Engineering Method (OOSEM), the IBM Telelogic Harmony-SE and the IBM Rational Unified Process [25].

Among other most used modeling languages there is the Object Process Language (OPL), particularly attractive since it also embeds a methodology, called Object Process Methodology (OPM), and a dedicated tool, OPCAT [19]. The work in [14] adopts it for a small satellite application. Other languages are Architecture Analysis & Design Language (AADL) [74], Modelica [6] and others.

One of the lack of the above cited languages and methodologies resides in their object-oriented nature, proved to be difficult to understand by non-software background engineers, who require appropriate training with highly qualified personnel. Focusing just on SysML, the work done in [3] furnishes some considerations about its applicability using the OOSEM methodology. The author highlighted SysML inability to provide constructs that support a tight integration of a system's structural and behavioral aspects, moreover functions have to be modeled using activities or blocks which result semantically confusing and ineffective.

A newly emerging MBSE solution is the ARCADIA (ARChitecture Analysis & Design Integrated Approach) methodology&language developed by Thales [72], a Domain Specific Modeling Language (DSML) inspired by UML/SysML and NATO Architectural Framework (NAF) standards [54]. One of the great power of ARCADIA is the tailored open source tool, called Capella, which perfectly catches the method and the language.

With respect to SysML, ARCADIA and Capella focuses on the method, therefore systems engineers are not required to be modeling experts. Moreover, they support functional analysis and functional flows in multiple diagrams; this is a great benefit in particular in the context of a space mission design, where one of the pillar systems engineering practice is the understanding of the system and subsystems from a functional viewpoint, which then drive requirements definition. Dedicated model elements distinguish functions from components in Capella, while SysML uses the same blocks leading to the loss of the conceptual difference between structural element and functions [76].

A summary of the Capella benefits, in addition to those previously mentioned, is here reported:

- it unifies the three basic ingredients of an MBSE approach: tool, language and method;
- it has been successfully deployed in a wide variety of industrial contexts [54] and developed by a big space company;
- it is open-source and equipped with a number of free add-ons, also customizable;
- it is intuitive and the website is plenty of material to become acquainted with the method and the tool.

According to the mentioned advantages, ARCADIA and Capella are selected as MBSE solution for this work; a better presentation of their key principles is furnished in the next section.

2.4 The ARCADIA Method in a Nutshell

ARCADIA consists of iterative processes based on three mandatory interrelated activities to be performed within a system design: need analysis and modeling, architecture building and validation, requirements engineering [54]. Four different working levels are defined for the architecture process, reported in Figure 2.6 and detailed in the following sections. Each level is modeled in Capella using as starting point the outputs coming from the level that precedes it, thanks to the automatic transition of elements. This way, coherence is maintained since lower level elements are realizations of upper level ones.

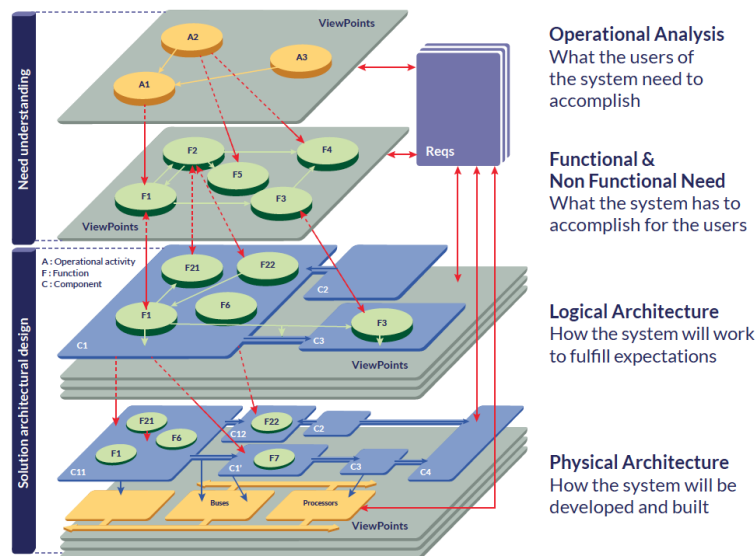


Figure 2.6: ARCADIA method phases [54]

2.4.1 Users Needs Understanding

Operational Analysis

The OA defines the needs and objectives of future users of the system, far beyond system requirements and independently of the future system to be realized. The concept of system, indeed, does not appear in this level which instead focuses on the working environment in which it will be designed, therefore on actors and their responsibilities. This level can be treated as a model of the jobs of future users: what are their activities, which roles must they fulfill and under which operational scenarios. The main concepts encountered in this level are here listed; the definitions are taken from [55]:

- *Operational Capability*: capability of an organization to provide a high level service leading to an operational objective being reached;
- *Operational Entity*: entity belonging to the real world (organization, existing system, etc.) whose role is to interact with the system being studied or with its users;

- *Operational Actor*: particular case of a (human) non-decomposable Operational Entity;
- *Operational Activity* (orange colored): process step carried out in order to reach a precise objective by an Operational Entity, which might need to use the future system in order to do so;
- *Operational Interaction*: exchange of information or of unidirectional matter between Operational Activities;
- *Operational Process*: series of activities and of interactions that contribute toward an Operational Capability;
- *Operational Scenario*: scenario that describes the behavior of Entities and and/or Operational Activities in the context of an Operational Capability. It is commonly represented as a sequence diagram, with the vertical axis representing time.

The most adopted operational diagrams within this work are the Operational Capabilities Blank, Operational Activity Scenario and Operational Architecture Blank diagrams. Figure 2.7 shows how OA concepts are interconnected.

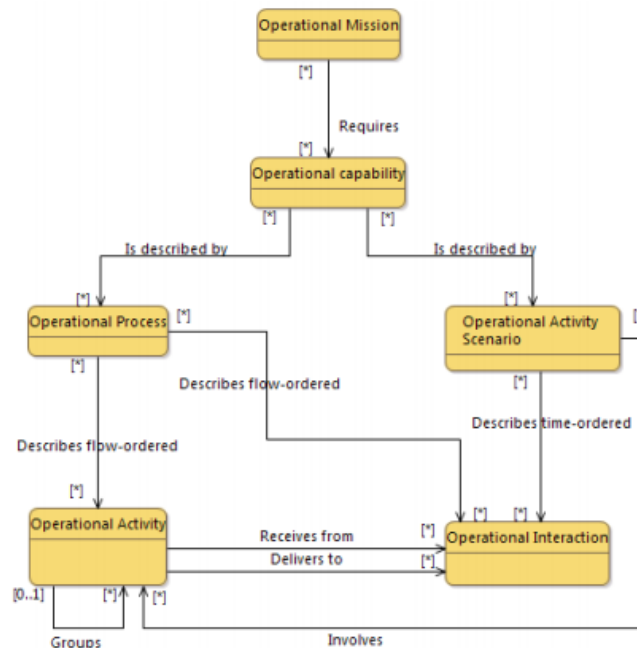


Figure 2.7: Concepts and relations concerning the Operational Analysis [75]

System Analysis

Also called Functional & Non Functional Need analysis, this level introduces the concept of system and defines how it can satisfy the former operational needs. This process helps to determine the functions that are needed by the system, in terms of what it has to do and

not how, being compliant with non-functional properties asked for. A capability trade-off analysis takes place here and system requirements are consolidated. First functional interfaces with the Entities introduced in the OA are also modeled. The main concepts encountered in this level are here listed [55]:

- *System*: organized group of elements that function as a unit (black box) and respond to the needs of the users. The System owns Component Ports that allow it to interact with the external Actors;
- *Actor* (light blue colored): any element that is external to the System (human or nonhuman) that interacts with it;
- *System Capability*: capability of the System to provide a high-level service allowing it to carry out an operational objective;
- *Mission*: high-level need/service which exploits System Capabilities.
- *Function*: behavior or service provided by the System or by an Actor. A Function owns Function Ports that allow it to communicate with the other Functions. A Function can be split into sub-functions and are green colored;
- *Functional Exchange*: unidirectional (green colored) exchange of information or of matter between two Functions, linking two Function Ports (green for output ports, red for input ports);
- *Component Exchange*: connection between the System and one of its external Actors, allowing circulation of Functional Exchanges allocated to it;
- *Component Port*: Component Exchanges link the System to Actors, via Component Ports (white squares), which can be uni- or bidirectional;
- *Functional Chain*: element of the model that enables a specific path to be designated among all possible paths (using certain Functions and Functional Exchanges).

ARCADIA also proposes a set of “path conditions”, modeled as predetermined functions, which result to be very powerful in expressing data flows. They are present at any level from SA on and here listed (Figure 2.8 shows their graphical representation):

- *Duplicate Function* transmits the same exchange to all recipients;
- to specify the combination of items of several exchanges issued from different sources, a *Gather Function* to constitute a single Exchange fusing those received from different sources is used;
- to specify the selection of one among several potential recipients, a *Route Function* is used;
- a *Select Function* is defined to specify the selection of one source among several;

- *Split Function* is used to specify the broadcasting of some exchanges to each recipient selectively.

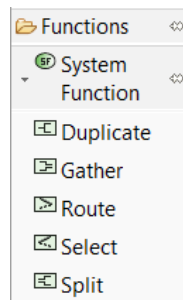


Figure 2.8: ARCADIA Flow Control Functions

Among SA diagrams, the following ones are the mostly used: Mission Capabilities Blank, System Data Flow Blank, System Architecture Blank, System Functional Break-down. Figure 2.9 illustrates the connections between SA concepts.

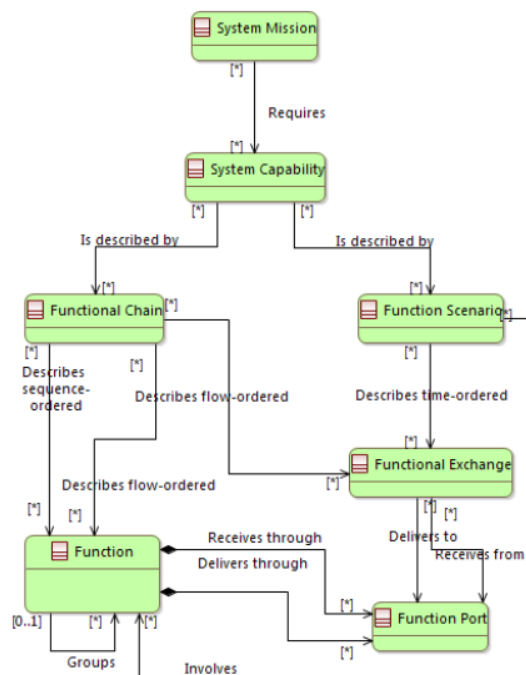


Figure 2.9: Concepts and relations concerning the System Analysis [75]

2.4.2 Solution Architectural Design

Logical Architecture

The basic functional analysis of the SA is here articulated through an internal functional analysis in order to understand how the system will have to work to achieve the required performance. First architectural solutions and engineering decisions are here introduced,

which are unlikely to be challenged later in the development process. Several decompositions of the system into logical components is performed and each function is allocated to one component. The output of this level is a logical solution, that is the best compromise architecture functionally described, that responds to the needs defined in the OA and SA. ARCADIA proposes the following concepts [55]:

- *Logical Component* (blue colored): structural element within the System, with structural Ports to interact with the other Logical Components and the external Actors. A Logical Component can have one or more Logical Functions. It can also be subdivided into Logical subcomponents;
- *Logical Actor*: any element that is external to the System (human or non-human) and that interacts with it;
- *Logical Function*: behavior or service provided by a Logical Component or by a Logical Actor. A Logical Function has Function Ports that allow it to communicate with the other Logical Functions. A Logical Function can be subdivided into Logical subfunctions;
- *Functional Exchange*: a unidirectional exchange of information or matter between two Logical Functions, linking two Function Ports;
- *Component Exchange*: connection between the Logical Components and/or the Logical Actors, allowing circulation of the Functional Exchanges;

Some LA diagrams used in this thesis are: Logical Architecture Blank, Logical Exchange Scenario, Logical Functional Chain Diagram, Logical Functional Breakdown.

Physical Architecture

Real components that will constitute the system are formalized in the PA, each one carrying its own sub-components and functions. This level defines the components to be produced introducing, with respect to the LA, design decisions, rationalization and architectural patterns. Physical interfaces are also defined. In this level, it is important to distinguish between two types of components [55]:

- *Behavior Physical Component* (blue colored): Physical Component tasked with Physical Functions and carrying out part of the behavior of the System;
- *Node (or Implementation) Physical Component* (yellow colored): Physical Component that provides the material resources needed for one or several Behavior Components. It represents a real component that will be integrated in the system.

All the concepts presented in the LA are also present here, therefore just the new ones introduced in the PA are here reported:

- *Physical Port* (yellow squares): non-oriented port that belongs to an Implementation Component (or Node). The Component Port, on the other hand, has to belong to a Behavior Component;
- *Physical Link* (red colored by default): non-oriented material connection between Implementation Components (or Nodes). The Component Exchange remains a connection between Behavior Components. A Physical Link allows one or several Component Exchanges to take place;
- *Physical Path*: organized succession of Physical Links enabling a Component Exchange to go through several Implementation Components (or Nodes).

An interesting feature of Capella, applicable at any level but mostly exploited in the PA, consists in the creation of reusable model elements, such as complete physical components with ports, functions, etc. A *Replicable Elements Collection (REC)* is a definition of an element which can be reused in multiple contexts/models. A *Replica (RPL)* is an instantiation of a REC. RECs can also be packaged in external libraries, which can be shared among several projects [54].

The most adopted PA diagrams are the Physical Architecture Blank and Physical Exchange Scenarios.

EPBS (End Product Breakdown Structure)

This level responds to the question “What is expected from the provider of each component?”, deducing from the PA the conditions that each component must fulfill to satisfy the architecture design constraints and limitations. As this level will not be treated for this thesis, it is not further described here. Readers can refer to [54] for more details.

2.4.3 ARCADIA Diagrams

An overview of the main types of diagrams, also mentioned in the previous sections, with focus on those adopted in this work, is here provided. All the definitions are taken from [55]:

- *Data Flow diagrams*: represent the information dependency network between Functions. The Functional Chains can be represented as highlighted paths;
- *Architecture Blank diagrams*: their main goal is to show the allocation of Functions to Components, as well as Functional Exchanges, Component Exchanges etc.;
- *Scenario diagrams*: they show the vertical sequence of the messages passed between elements (lifelines), inspired by the UML/SysML sequence diagrams. A lifeline (Instance Role, in Capella) is the representation of the existence of a model element that participates in the scenario involved (i.e. Functions, Components, Actors,

System). It has a name that reflects the name of the model element referenced and is represented graphically by a dotted vertical line. A Message is a unidirectional communication item between lifelines that triggers a behavior in the receiver;

- *Mode and State diagrams*: they are graphical representations of state machines inspired by UML/SysML. A state machine is a set of Mode/States linked together by Transitions. A Transition describes the reaction of a structural item when an event takes place. More details about this kind of diagrams are provided in Section 3.2.6;
- *Breakdown diagrams*: represent hierarchies of either Functions or Components at all levels of engineering;
- *Capability diagrams*: particularly useful in Operational Analysis and System Analysis, they can highlight the relations between Missions, Capabilities and Actors in order to catch the high level objectives of the mission/system.

Other diagrams are available in Capella, such as Class diagrams used to model data structures and Exchange Items. As these concepts are not exploited for this thesis, the reader can refer to [54] for more details about them.

2.4.4 The Capella Tool

Capella is an Eclipse application implementing the ARCADIA method and its methodological guidance through a browser which proposes all the previously mentioned diagrams and model elements. As graphical representations of elements play a key role in communication, Capella relies on a consistent color scheme. In particular, all function-related elements are green, and all component related elements are blue [54] (except Node Components which are yellow colored). Customization is also admitted.

Among the most powerful and helpful capabilities of Capella, it is worth mentioning the automatic computation of diagrams according to model elements defined in some other diagrams, so that the integrity, traceability and coherence are maintained; this way, model elements are uniquely defined and can have multiple graphical representations depending of the diagram they appear. Furthermore, the presence of *filters* simplifies views in case the user desires to visualize just a subset of elements. Many add-ons can be installed and customized to extend the tool capabilities. These and many other properties make Capella an ideal tool for team communication and single source of truth, reducing the possibility of late changes by early detection of errors and anticipating problem solving. For this work, the version 5.0 of Capella is adopted.

Chapter 3

Case Study: e.Inspector

I paint from the top down. From the sky, then the mountains, then the hills, then the houses, then the cattle, and then the people.

GRANDMA MOSES

This chapter presents the extended MBSE approach developed using the ARCADIA method and the Capella tool for the Phase-A of the e.Inspector CubeSat. A summary of the mission is provided at first, followed by the implementation of all systems engineering practices in an MBSE environment. The way requirements are managed is presented and the four ARCADIA levels are explored: Operational Analysis, System Analysis, Logical Architecture and Physical Architecture. Then, the focus is shifted toward Modes management, mission Phases definition and Concept of Operations. Lastly, an approach for AIV/AIT plan development within the tool is proposed.

3.1 The e.Inspector Mission

An overview of the e.Inspector mission is given in this section to frame the context in which the MBSE approach is developed and tested. A detailed description of the mission can be found in the Mission Description Document [21]; just some important aspects, functional to the work presented in the following, are here presented. The e.Inspector high level mission goal is to carry out a close-up visual inspection of a European space debris, with the scope of improving the understanding of its status at the time of flight, validating GNC sensors to be used for a next capture of the debris and to reduce risks of future Active Debris Removal (ADR) missions. e.Inspector is a *European Space Agency (ESA)* mission led by *Politecnico di Milano* for the systems engineering part, mission analysis and relative dynamics. Two main partners contribute: *Leonardo* furnishes the payloads and *Leaf Space* provides the ground segment and so downlink/uplink support. One of the first analysis conducted for the mission has been the target selection, which had to face two main programmatic constraints: the requirements on the image acquisition phase to be completed by 2025 and on the re-enter in Earth atmosphere within 25 years from the mission start. According to them and other criteria, several targets were

identified; the VESPA upper part, proposed as baseline by ESA, is one of them. Currently, the project is at the end of the Phase A studies, having concluded the Preliminary Requirements Review and moving into the Phase B.

In order to better catch the following analysis about systems engineering aspects managed in a MBSE environment, it is important to provide a focus on the mission timeline, which is constrained by the maximum duration of 2 years because of the incoming ADR mission, the ClearSpace-1. The scheme in Figure 3.1 provides a high level description of mission phases and the main requirements driving the design. During the *Launch and Early Orbit Phase (LEOP)*, the 12U CubeSat is activated to detumble and deploy appendages, after which the beacon telecommunication is established and the platform commissioning performed. As the CubeSat will exploit a piggyback launch, it will not be released in the exact orbit of the selected target. Therefore, a *Transfer Phase* is foreseen, in which a natural drift is firstly exploited, followed by a low thrust propulsion unit to finalize the arrival to the target orbit. Due to the high flexibility and robustness required with respect to the range of targets, a study has been performed to prove the transfer feasibility according to the available budget (of about 300 m/s) and the different identified launches. The core of the mission is the *Inspect Phase*, in which the relative dynamics with respect to the target is done to acquire scientific data and match the aforementioned mission objectives. The inspection strategy, which begins at a distance of 20 km from the target up to 100 m, alternates some so-called Holding Orbits, in which commissioning is performed and data are downloaded, to ballistic Inspection Orbits, where data of the target are acquired and nominal science is performed. After some trade-offs analysis, the selected payloads on board are two cameras, one working in the visible band and the other one in the infrared, in order to ensure science data acquisition even in eclipse conditions. Lastly, in the *Dispose Phase* a disposal maneuver is performed and the CubeSat is passivated after having moved in total safety away from the target. Figure 3.2 shows the CubeSat configuration and some highlighted components, all presented within the following MBSE approach developed for this mission.

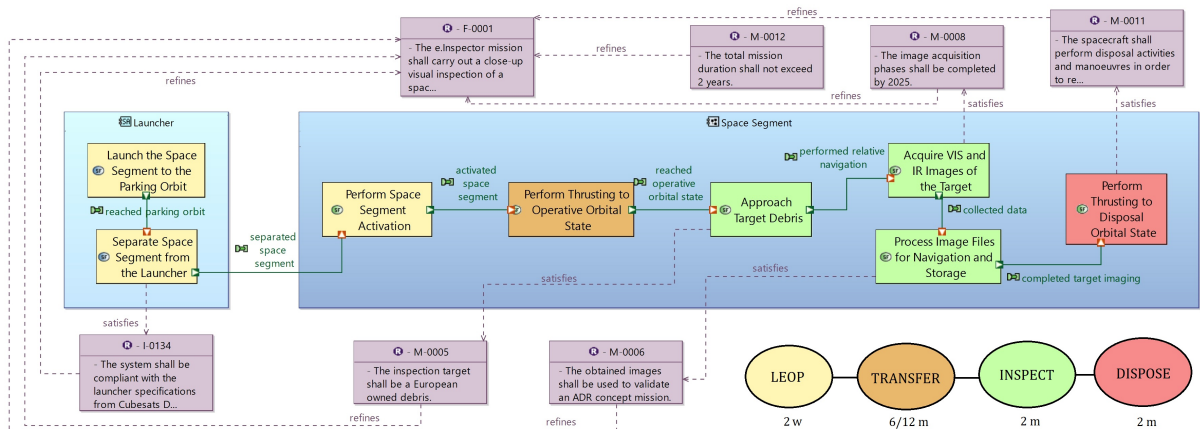


Figure 3.1: e.Inspector mission scheme

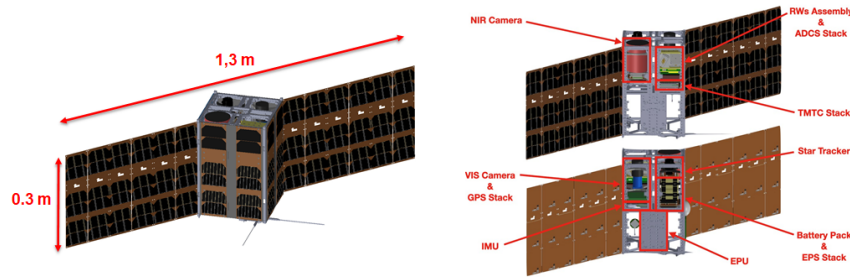


Figure 3.2: e.Inspector 12U CubeSat deployed configuration [21]

3.2 Model Implementation

3.2.1 Requirements Management

Current systems engineering practices, especially space related ones, reckon on requirements as communication means among engineers and as the main vector to ensure correct design of the system, providing at the same time a description of the product architecture. A requirement can be defined as a statement that captures system functional and performance aspects or sets constraints. Any space system is typically described by hundreds of requirements, therefore a method to organize and easily check them shall be identified.

The most widespread requirement-based engineering approach adopts textual requirements, which are traced within system functionalities. It is often difficult to conduct such traceability study using a document-centric approach, since jumping from a document to another increases the possibility of generating misinterpretation events, particularly true as the number of requirements increase due to the system complexity; it is well known that the main cause of risk in projects is due to unclear requirements writing. The work done in the paper by Bonnet et al. [10] proposes the concept of *model requirements*, which are basically model elements such as Functions, Functional Exchanges, Components, Component Exchanges, etc. encountered in all MBSE approaches. They are conceived as “smart” requirements which provide a well defined information with a strict syntax and precise semantics. However, the authors of the cited paper also recognize that textual requirements are still needed within a project since they better catch some aspects, sometimes in a easier and more complete way. Therefore, the work presented in the following will also include textual requirements, which can be linked to the mentioned *model requirements* to ease their traceability, completing each other. This section presents the way requirements are organized and modeled to ease the establishment of links with other model elements. The *Requirements Viewpoint* add-on is adopted since it allows to deal with requirements in a very effective way using Capella, as here presented. It is pointed out that for this work a set of requirements was already available from an in-house MBSE tool developed by the e.Inspector working group at Politecnico di Milano, the ASTRA team, therefore they were manually imported in Capella. However, the add-on not only allows to create new requirements in the model, but also to automatically import them from a

Requirement Interchange Format (ReqIF), Object Management Group (OMG) Standard, whenever available.

Each requirement is defined by a unique identification code, reporting the category, the subsystem acronym and a four-digit number, and a text which explicitly states its content. Then, a number of properties further characterize it as a model element. Figure 3.3 reports the list of properties, defined for this project, as they appear in the *Capella Types Folder*. They include some *Enumeration Data Types* such as Importance (Mandatory/Nice to have) and Progress Status (Rework Necessary, To Be Reviewed, etc.) and the *Requirement Types* (Functional, Mission, Interface etc.); the categories description is not here reported and can be consulted in ECSS-E-ST-10-06C [64], followed as guideline for the requirements classification. Another critical aspect related to requirements is their verification, to check that the system is compliant with what they state. Therefore the Data Type *Verification Method* is introduced, having as items the verification methods defined in ECSS-E-ST-10-02 [63]: Test, Analysis, Review of Design and Inspection. Lastly, the list also reports two *Relation Types*: the *satisfies* one is an incoming link used to assert that a specific model element related to the system architecture covers an aspect of the requirement, the *refines* one is an outgoing link used to establish internal relationships between requirements, basically decomposing parents into children, such that the trees can be generated. Any Requirement Type has a number of Enumeration Data Types, called *Attribute Definitions*, which items can be selected using the Capella *Mass Editing View* (Figure 3.4). To enrich the requirements description, additional notes can be added to their property sheets.

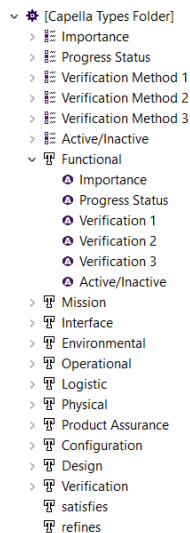


Figure 3.3: Requirements - Capella Types Folder

Requirements are grouped into folders according to the subsystem they belong. Figure 3.5 shows this way of organizing them and Figure 3.6 presents an example of requirement. The presented organization of requirements is a first important plus provided by the MBSE approach, since they are not simple sentences as in a document-based organization but actually represent concrete model elements.

Capella does not provide a dedicated requirements diagram to build trees; however,

	ReqIFName	ReqIFText	Importance	Verification 1	Verification 2	Verification 3	Progress Status
	M-0005	M-0005 The ins...	Mandatory	Review of De...			
	M-0006	M-0006 The ob...	Mandatory	Review of De...			Rework Necessary
	M-0007	M-0007 The tot...	Mandatory	Review of De...	Analysis		To Be Reviewed
	M-0008	M-0008 The im...	Mandatory	Review of De...			To Be Discussed
							Reviewed OK
							Draft

Figure 3.4: Requirements - Mass Editing View

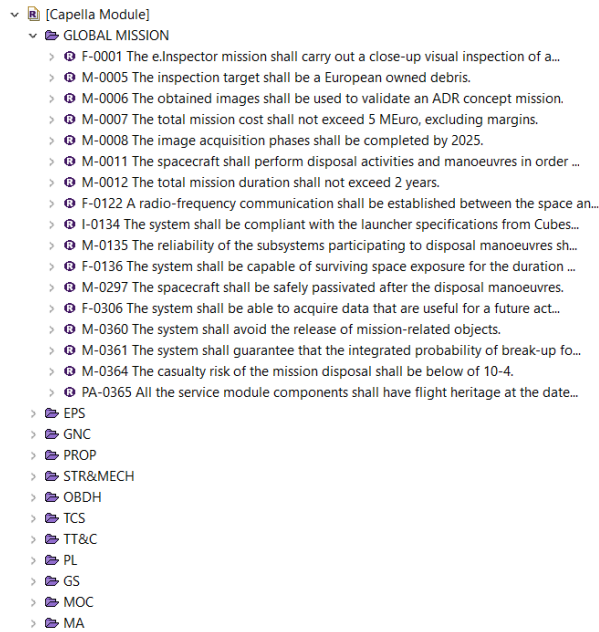


Figure 3.5: Requirements - Capella Module

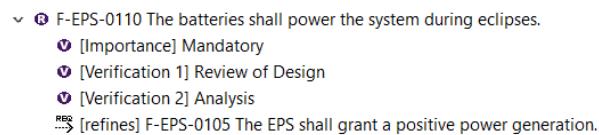


Figure 3.6: Requirement Example

since they can be reported in any diagram thanks to the Capella transverse modeling, for this work some initially empty Operational Architecture Blank diagrams are exploited to overcome this lack. Once the *refines* relations are defined, requirements can be added to these empty diagrams and Capella automatically generates the trees, which results to be very intuitive to trace backwards each low level requirement, ensuring their consistency and completeness.

Concerning the e.Inspector mission, a sort of Level 0 of requirements is firstly defined, called *Global Mission*, in which mostly high level ones are introduced. Figure 3.7 shows the related tree, with one main parent refined by some daughters. Once the top-level requirements are defined, system and subsystems requirements are derived in order to accomplish what the mission has to carry out, providing engineering specifications. Therefore, each

branch of 3.7 is further refined by subsystems requirements, per which a tree like the one in Figure 3.8 showing the TT&C subsystem is created for all subsystems. Since the aim of this section is not to catch and describe the requirements themselves, but to present the way they are managed within this project, the remaining requirement trees are not presented here and can be found in Appendix A.1. It is just reminded that, as requirements can be traced by any model element at any level, they can be graphically found in successive diagrams to highlight certain aspects or to make remarks. Of course, as the design progresses, requirements will evolve and mature; managing their updates and revisions through a MBSE approach, as experienced for this work, reveals to be very effective with respect to other approaches since the traceability can be easily caught, drastically reducing the time and the effort spent for such activity.

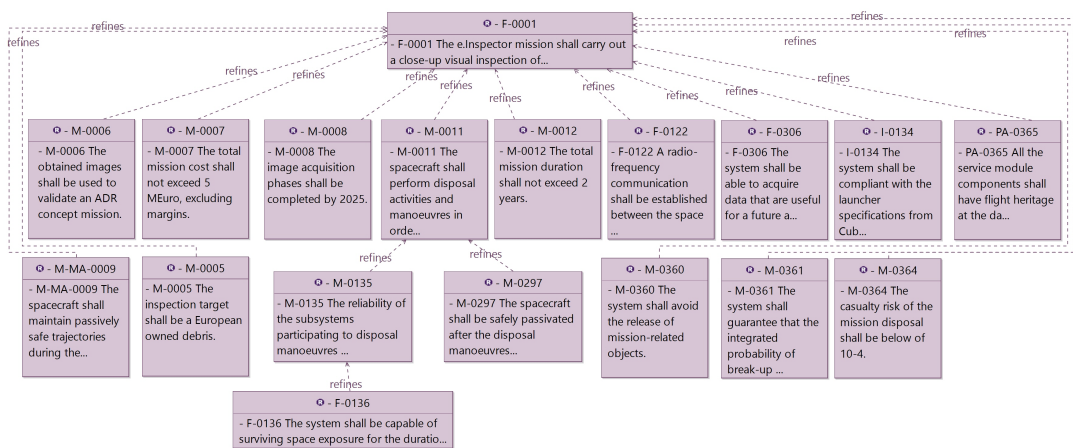


Figure 3.7: [OAB] Requirements - GLOBAL MISSION

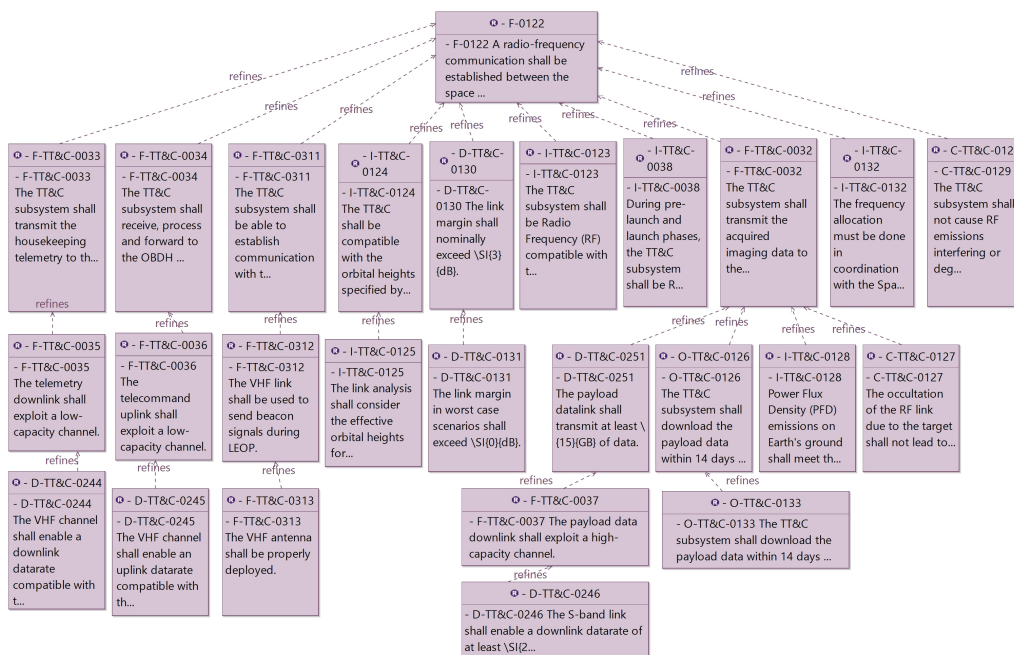


Figure 3.8: [OAB] Requirements - TT&C

3.2.2 Operational Analysis

Whatever project and system design requires an initial definition of the high level objectives which are declined into drivers and constraints for the alternatives selection, identifying at the same time the stakeholders and their responsibilities. The ARCADIA method perfectly catches this concept throughout the Operational Analysis that focuses on what the involved entities are looking to accomplish, despite the concept of system is still not introduced. ARCADIA does not rigorously impose to follow all the design steps in a dictated order, but puts the systems engineer in a position per which he/she can decide whether to carry out certain methodological activities or not and in any order. It is good practice to model the Operational Analysis in the context of a CubeSat design, and so for the e.Inspector mission, due to the complexity of such space systems, which are the result of a collaboration and information exchange among a multidisciplinary set of entities, for the entire mission lifecycle.

The first step consists in fixing certain high level services, called Capabilities, at this stage independent on the system that is going to be realized and further detailed in successive design levels. The diagram devoted to this kind of description is called Operational Capabilities Blank, reported in Figure 3.9, which simply highlights the involved Entities and the related Capabilities, graphically represented respectively by gray rectangles and bronze medallions. It is important to remember that each graphical elements is a manifestation of a model element, the latter having multiple types of graphical representations. All connections here depart from an Operational Capability and are directed toward an Entity or an Actor (these concepts were presented in Section 2.4.1), purely representing a relationship without any kind of temporal sequence. It can be noted that some Entities share certain Capabilities, meaning that a collaboration between them is expected. As an example, the Entity *POLIMI* shares most of its Capabilities, due to its centrality in this project.

An Entity is not necessarily a company or an institution, but can also be something abstract whose role is to interact with the system being studied. In this sense, the *Environment* is modeled as an Entity due to its relevant role in providing constraints on the future system. The *Space Debris Expert*, instead, is modeled as an Operational Actor, as its icon suggests. For this diagram, a modeling choice consists in reporting only those stakeholders that in some way interact with the system to be designed, excluding from this analysis the full set of involved entities such as suppliers, sponsors, testers and others, that are still unknown in the very first mission phases, so their modeling at this stage would not add any value. It is here reminded that the Entities *Payload Provider* and *Ground Segment Provider* are actually the two main contributors to the mission, respectively *Leonardo* and *Leaf Space*, together with *Politecnico di Milano*.

Each Operational Capability is further described by a number of *Operational Activities*

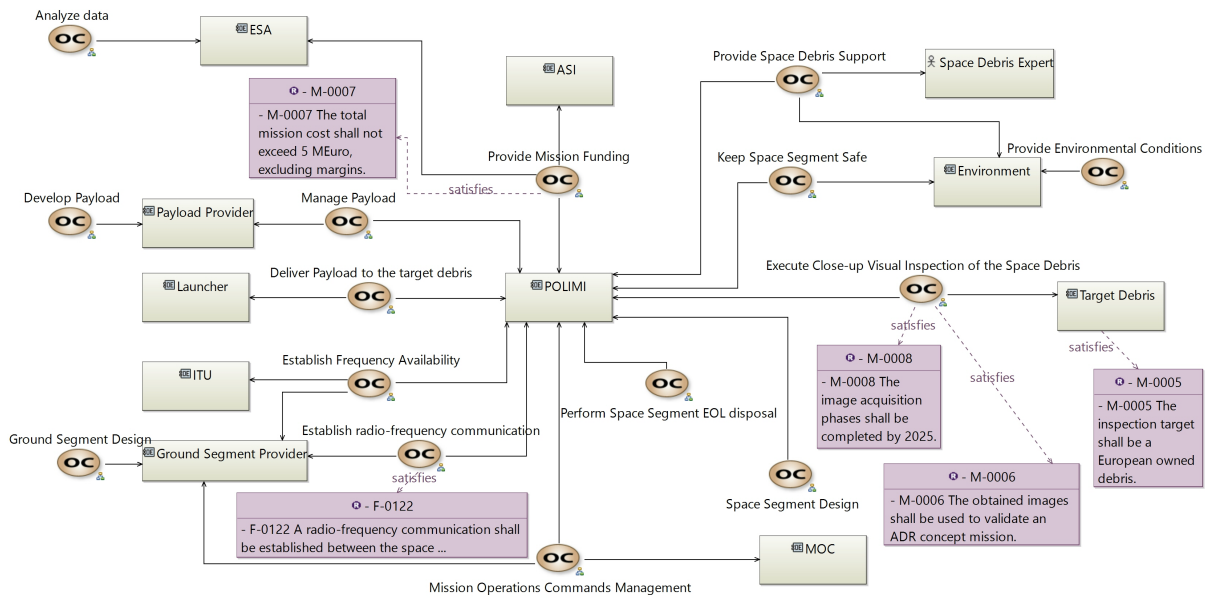


Figure 3.9: [OCB] Operational Capabilities

allocated to Entities and Actors. Here the simple Operational Entity Scenario diagrams are adopted since they also introduce the time dimension. An example is shown in Figure 3.10 describing the Capability *Mission Operations Commands Management*, where three Entities are involved and their high level Activities represented in orange exchange information by means of Interactions model elements. The activity *Receive commands* is allocated to *POLIMI*, actually meaning that such Entity has to design a system able to perform that Activity (it is recalled that the concept of system is still absent at this level).

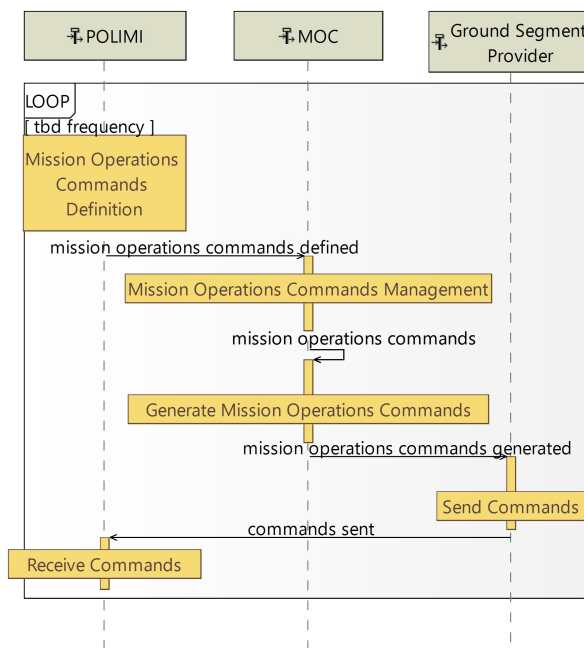


Figure 3.10: [OES] Mission Operations Commands Management

The last and most significant diagram belonging to the Operational Analysis is called Operational Architecture Blank (OAB) (Figure 3.11) here realized with the scope of showing the full set of Operational Activities previously defined within Scenario Diagrams, their allocations to respective entities and their interactions. For the e.Inspector mission, the *POLIMI* carries out the most of activities since it has a central role in the system development. The diagram also highlights a blue colored line, called Operational Process, used to highlight a particular logical series of Activities which contribute toward an objective, here the entire *Mission Lifecycle*. The first constraints and mission objectives are identified starting from the Operational Analysis, therefore some high-level requirements are traced in the diagrams by the model elements to which they are related. It is noted that the requirement *M-0005* in the OAB is also reported in the OCB (Figure 3.9) even though it is satisfied by a different model element. This is not an anomaly but represents one of the advantages of dealing with requirements in a model-based environment, that is the easiness and the consistency of their tracing. Despite it is still a very high level representation, the OAB is useful to provide a global vision of what the main system-interacting Entities have to realize for the project, regardless of any technical solution. It is the main output of the Operational Analysis and the final deliverable for the next modeling phase: the System Analysis.

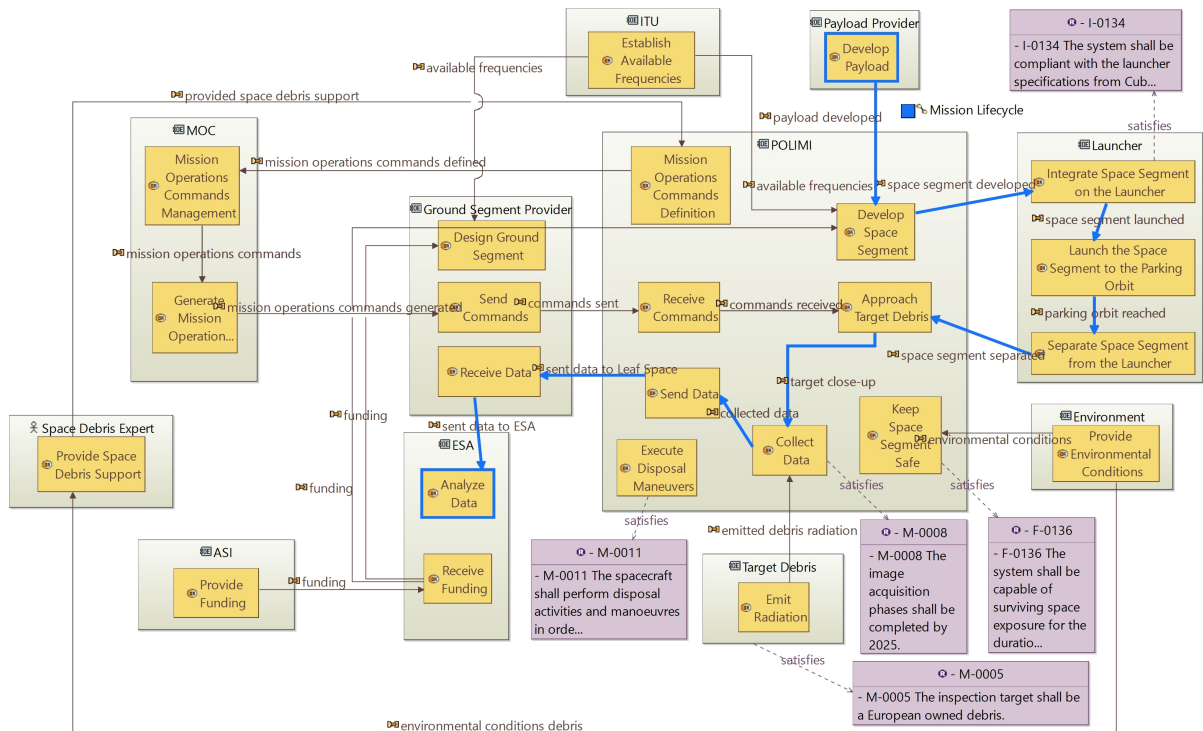


Figure 3.11: [OAB] e.Inspector GLOBAL

3.2.3 System Analysis

ARCADIA's strength is to be an iterative method that, for each modeling level, exploits the results obtained from the previous one. This is done in the System Analysis, where the stakeholders needs previously examined are translated into "what the system has to accomplish for the users" [50]. The concept of system is here introduced and systems engineers can start asking whether the Activities reported in the Operational Analysis, now called System Functions and here transitioned thanks to the Capella Transition functionality, will be realized by the system or left to the stakeholders. New Functions will be introduced in order to cover the System Analysis aspects and to lead the way to a complete description of the CubeSat, in full compliance with the mission requirements.

Despite the total flexibility and breadth of the method could distract from the real objectives of the System Analysis, it is reminded that this level should not provide a deep description of the system but to frame its essential functioning. In order to accomplish this task, the Mission Capabilities Blank diagram is firstly exploited, with the scope of accompanying the modeler toward system Functions definition. As Figure 3.12 shows, four Missions are introduced, each one described by a number of System Capabilities by means of the Capability Exploitation relation. Both Missions and Capabilities are linked to System Actors, that from this level on incorporate both the meanings of Entities and Actors met in Section 3.2.2. These relations are called respectively Mission Involvements and Capability Involvements; for graphical reasons, the former are indicated by light blue lines. It is highlighted that this diagram only presents those System Actors that directly influence the behavior of the system.

The four system Missions are here briefly discussed. The idea is to categorize the Capabilities into four blocks, each one providing an essential high level service furnished by the system, that is the CubeSat (or Space Segment) itself. The *Keep Space Segment Safe* Mission refers to the fact that the system shall survive to the space environment; the *Provide Support* one is related to the presence of subsystems and their tasks, despite the concept of subsystem is still not present in the System Analysis but will be introduced in the Logical Architecture, Section 3.2.4; the *Execute Close-up Visual Inspection of the Space Debris* one is the core of the e.Inspector mission as the two daughter Capabilities indicate. One can notice that not all subsystems are explicitly reported in the *Provide Support* mission, such as the GNC one that instead is represented by the *Approach Target Debris* Capability, as discussed in the following. This modeling approach avoids the creation of redundant Functions, since the main concern related to the GNC subsystem at this stage is related that aforementioned Capability. Finally the *Provide Passivation EOL* is introduced in order to highlight its belonging to a different phase of the mission. The diagram of Figure 3.12 is very simple and, most important, can be replicated in a future CubeSat project as long as some small changes are made.

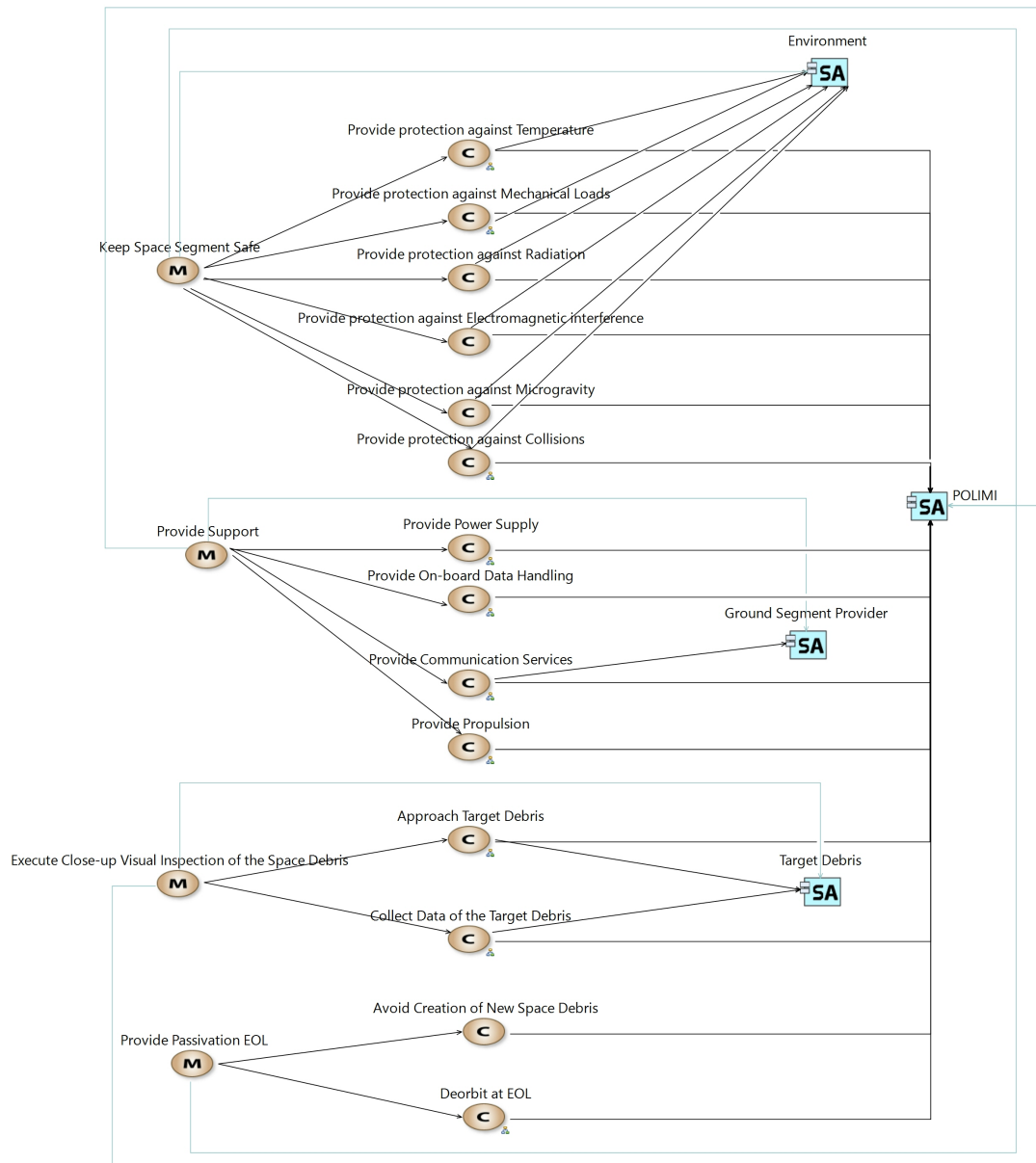


Figure 3.12: [MCB] Mission & Capabilities

The last comment regarding Figure 3.12 concerns the little colored icon that appears in the bottom-right of almost all system Capabilities. This is a recurrent icon in Capella, indicating that the model element is further described in one or more other diagrams. It is good practice to detail all Capabilities with proper Functions, however one can notice that such icon is not present in some of them, mostly linked to the *Keep Space Segment Safe*. This is another precise modeling choice, since these Capabilities are basically related to the space-compliant components selection and it would be an unnecessary burden of the model to describe them. All the others, on the other hand, hatch up into a set of Functions reported in some dedicated System Data Flow Blank diagrams. An example is given by the *Provide Power Supply* Capability, which conducts to the simple diagram in Figure 3.13, where four Functions resume what the power subsystem has to do. Some links, called Functional Exchanges, logically connect them; a green port indicates an

outflow while the red one an inflow. The father functionality *Provide Power Supply* is also reported, carrying the same name of the capability it describes. The coding appearing in some functions should not cause concerns, since it is a legacy from an inherited functional analysis performed by the working team at Politecnico di Milano.

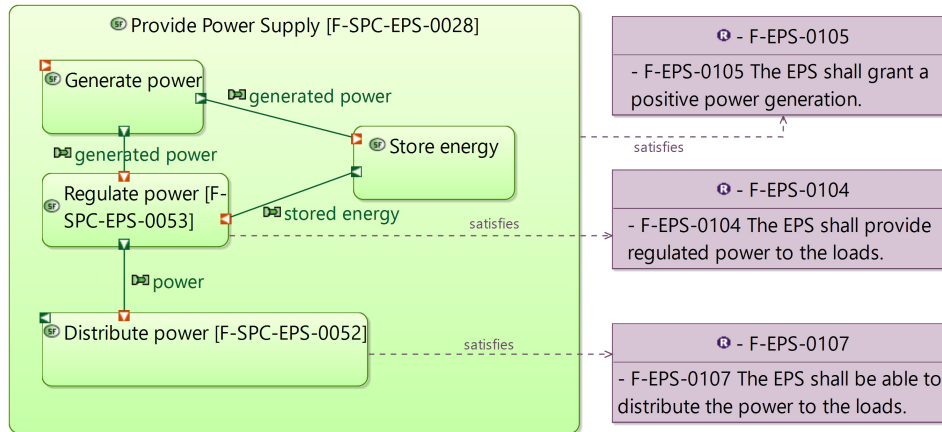


Figure 3.13: [SDFB] Provide Power Supply

It would be too page-consuming without adding value to the discussion to show here all System Data Flow Blank diagrams, reported in Appendix A.2, therefore just the *Provide Communication Services* and the *Provide On-board Data Handling* ones are here presented in Figures 3.14 and 3.15 since they introduce something new with respect to the previous one. In the former, the white color is used to indicate Functions that realize certain Operational Activities, defined in the previous level. The diagram in Figure 3.15 instead better presents the concept of father functions and the important condition of Capella consisting in the possibility to introduce Exchanges only between leaf Functions.

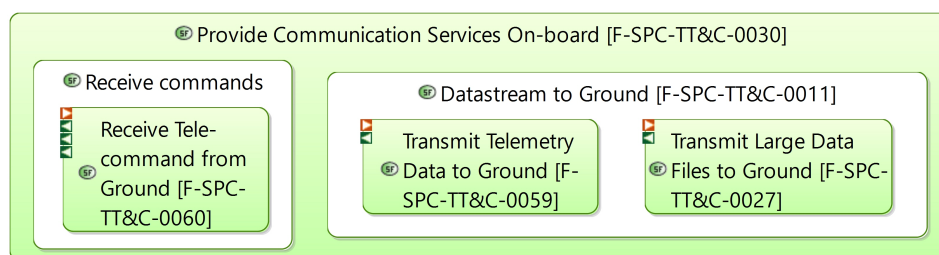


Figure 3.14: [SDFB] Provide Communication Services

Due to the not so high number of Functions, it is still possible to visualize all of them in one single System Architecture Blank diagram, reported in Figure 3.16. This diagram shows the allocation of leaf Functions to the system, in dark blue, and to the Actors that interact with it, in light blue. Actors are transitioned from the Operational Analysis and their Functions are realizations of the previous Operational Activities; whenever needed, Functions were added to them in order to guarantee a satisfactory interface description

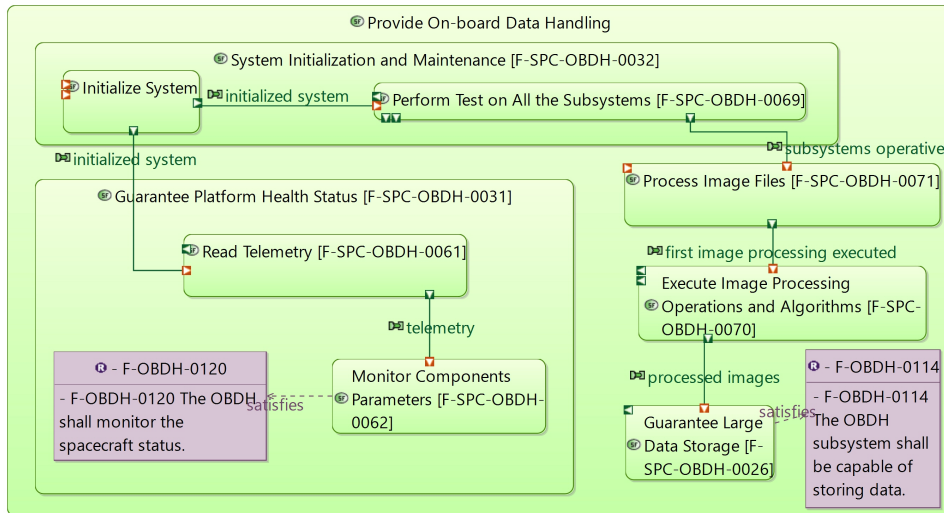


Figure 3.15: [SDFB] Provide On-board Data Handling

with the space segment. The SAB diagram also introduces the concept of Component Exchange, which meaning and importance is explained in Section 2.4.1. Any Functional Exchange that crosses the system or an Actor boundary shall be allocated to a Component Exchange, according to the ARCADIA method. As already mentioned, the System Analysis does not carry the concept of subsystems; however, a graphical organization of the Functions that belong to each subsystem could be noticed. Moreover, to facilitate the SAB diagram reading, it was decided to apply the yellow color to those Functional Exchanges connecting Functions of different subsystems; the classical Capella green is kept for the remaining ones.

Lastly, the concept of Functional Chains is here put in practice. Their aim is to provide the description of a certain behavior, making use of the available system and Actors Functions. In example, the blue line connects Functions that describe the *Data Collection and Download* operation while the red one refers to the *System Initialization* one. Regarding the latter, it may seem a too much generic description since it just mentions that the power, generated or stored, somehow has to be used for the system initialization. However, this level of detail is sufficient to provide a description of what the system has to do, without technical solutions involvement, as the System Analysis foresees. Many Functional Chains can be created if needed, useful to check the expected system behavior in different contexts.

In order to visualize the complete set of system Functions, both leafs and parents, the System Function Breakdown of Figure 3.17 can be consulted.

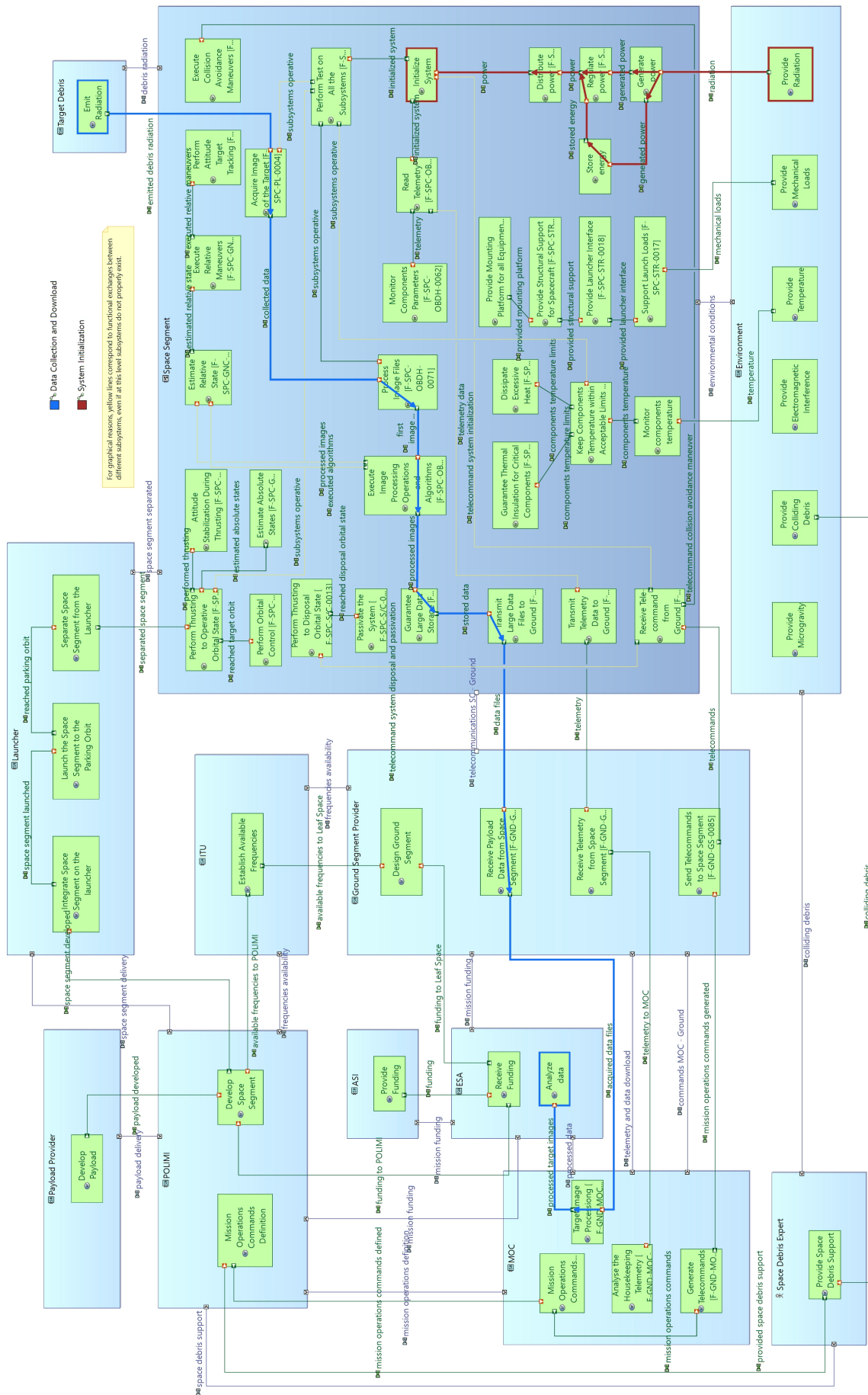


Figure 3.16: [SAB] e.Inspector GLOBAL

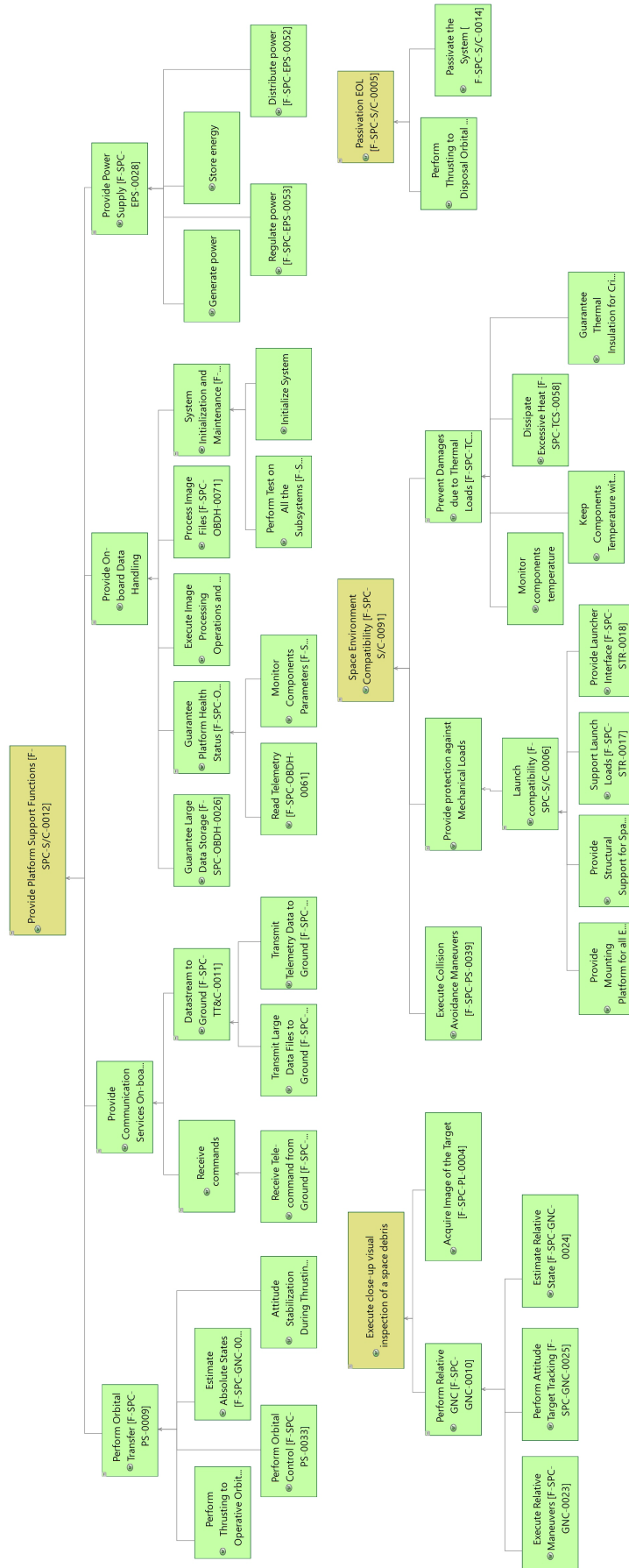


Figure 3.17: [SFBD] Root System Functions

3.2.4 Logical Architecture

Following the system design process adopted for the e.Inspector mission, the next step consists in opening the System Analysis black-box in order to set up a new functional analysis, whose foundations are inherited from the previous design level, that aims to define how the system should work to meet the system requirements. This is a delicate step forward in the design since the expected output is the final system logical architecture, properly selected on the basis of a trade-off analysis among all the possible solutions that satisfy the requirements. Big decisions driving the project and influencing the future Physical Architecture are taken here, being careful to leave a certain degree of freedom for the latter, otherwise construction choices would be too much constrained.

The Logical Architecture allows to introduce the concept of subsystems into the model, fundamental in the context of a MBSE approach developed for a CubeSat mission. They are here defined as Logical Components, which in turn have allocated other sub-components carrying the Logical Functions. Once again, Capella permits to transition all the model elements from the SA, which are here subject to a refinement procedure. It is clarified that the Logical Architecture here focuses on the system, while Logical Actors and their Functions coming from the SA are left unchanged since their detailed modeling is out of this work scope. A complete modeling of each Actor would surely improve the interface details with the CubeSat, however this is something that should be done by the Actors themselves, otherwise the risk of bad modeling due to the lack of information is high, with direct consequences on the overall model. Due to the numerous Logical Functions that are going to be showed, it is no more possible (for graphical reasons) to realize a single diagram including all of them. Therefore, each subsystem will be internally modeled in the following subsections together with their main interactions with other subsystems or external Actors.

An important remark is mandatory to clarify the philosophy behind the results that are going to be presented. The Logical Architecture is the product of complex design processes and decisions involving all subsystems. These aspects are not here exposed, since the aim of this work is to present the way these decisions and design results, coming from the subsystems engineers of the e.Inspector mission, can be managed in the context of a MBSE approach. Moreover, the focus is here on the platform and on the mission rather than on the payload, designed by *Leonardo*, therefore no dedicated diagram about its functioning will be reported. The same applies to whatever product furnished by stakeholders, such as the ground segment or the launcher; their modeling here is limited to the few functions needed to describe the platform interfaces with them. The Mission Description Document [21] can be consulted, with proper authorization, for the system design sizing, payload and ground segment aspects.

Electric Power Subsystem (LA)

The first subsystem here analyzed is the Electric Power Subsystem, that generates, stores, regulates and distributes electric power [79]. An interesting Capella add-on, called *System to Subsystem Transition*, can be used to delegate the modeling of each subsystem to a different team/personnel or to some subcontracting companies [55]. Despite the clear advantage of having multiple models realized with a very high precision by the subsystems experts and then merged in total respect of the interfaces defined at system level, for this work they are developed in one single model and conceived as Logical Components, as previously discussed. This is done because of the absence of multiple contributors to the model creation, that instead are typically present in a collaborative environment.

Figure 3.18 shows the Logical Architecture Blank diagram for the EPS, modeled as a cyan-colored Logical Component to which other sub-components are allocated. Recalling that in the LA the contents have to be defined in terms of how the system has to perform the needs expressed in the SA, the first step here consists in identifying conceptual solutions in line with requirements, and then expressing it in terms of Functions. Starting from the system Function *Generate Power* (Section 3.2.3), the e.Inspector EPS engineer identified the Solar Panels as the best primary power generation for this CubeSat. Despite in the LA no solution in terms of which components and how they are made should be reported, two main Functions describing how the power shall be produced can be identified: *Deploy Solar Panels* and *Solar Radiation to Electric Power Conversion*. One can think that the first one is actually a solution, since it suggests that the solar panels are also in a deployed configuration and not just in a body-mounted one; this is something acceptable whenever a design solution is frozen at the time the LA is conducted. Once the Functions describing how the system will generate power are defined, a dedicated component is created, here called *Power Generation*, and not for example Solar Panels, since it is good practice not to attribute names containing references to a specific technology in the LA, and to name the Logical and Physical Components differently [55]. The modeling approach adopted for the *Power Generation* is extended to the remaining EPS Logical Components: *Electrical Energy Storage*, *EPS DOCK*, *Battery Protection*, *Arrays Power Conditioning* and *Power Distribution*.

Still on the internal EPS functioning, the various components communicate by means of the Functional Exchanges, which are in turn allocated to proper Component Exchanges. It is not worth commenting all of them, since the the diagram reading should provide a self explanation. Only the modeling aspects are here explicitly discussed, such as the adoption of some particular Control Functions used to define more precisely the path conditions [55]. The five types of ARCADIA's flow Control Functions are described in Section 2.4.1; for the EPS modeling just the Split and the Route ones are used. To catch the power of the Split one, it can be noted how effective it is in the description of the power flow coming from the *Distribute Solar Panels Power* Function, that can be directed

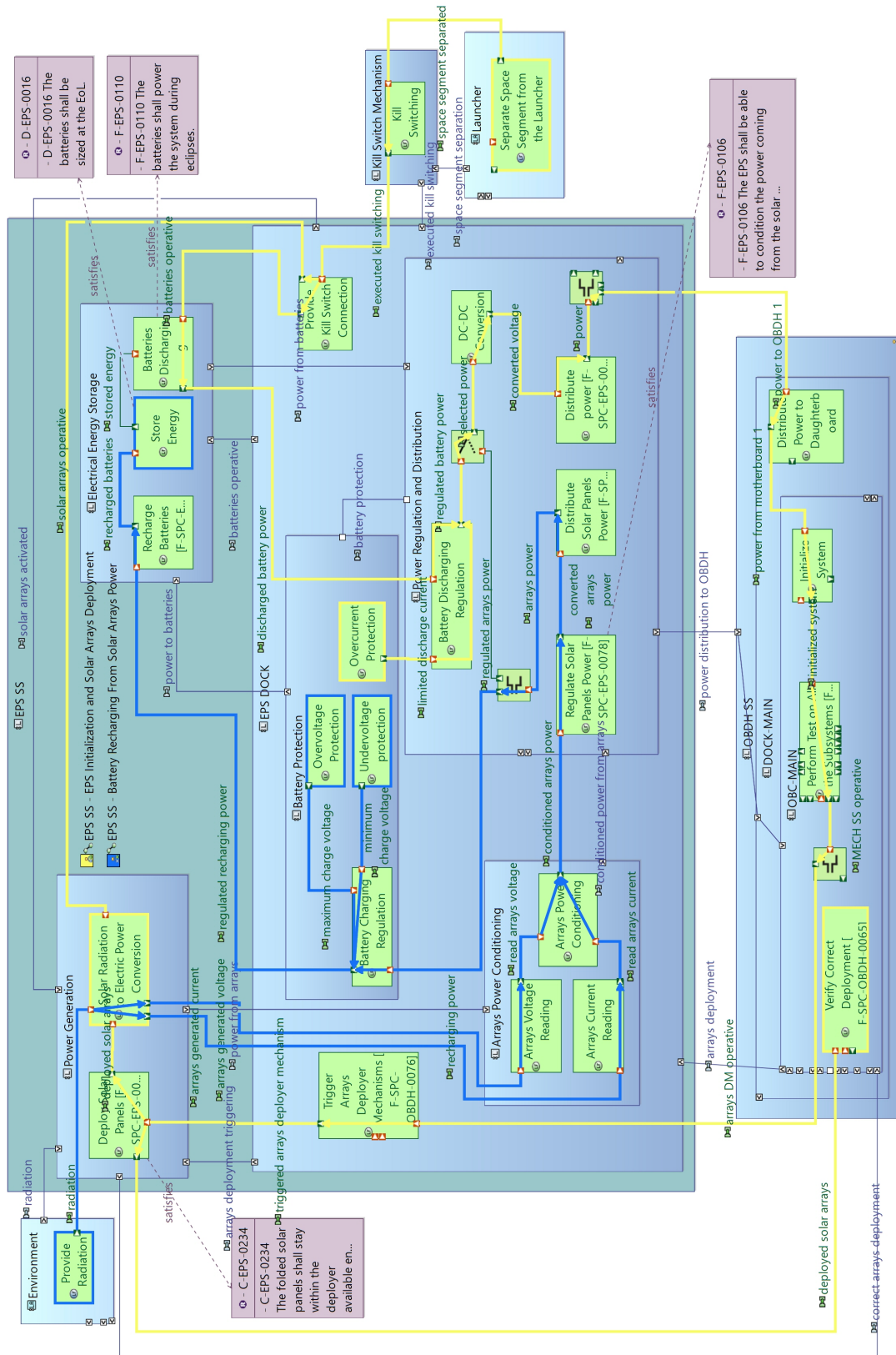


Figure 3.18: [LAB] EPS SS

toward batteries for their recharging or toward the power distribution line. The Route one, instead, is employed to specify the selection of one among several power sources,

that are the batteries and the solar panels. This is a very intuitive way of modeling since in one simple diagram a lot of information can be extracted with little effort, particularly suitable to complex systems such as those belonging to the space industry; the only required competence by team members is the modeling language knowledge and so the semantics.

The diagram in Figure 3.18 also shows interactions of the EPS with Actors and sub-systems. Two Functional Chains, respectively *EPS Initialization and Solar Arrays Deployment* in yellow and *Battery Recharging From Solar Arrays Power* in blue, highlight the way the EPS communicates with such external blocks. The first one begins with the spacecraft separation that activates the kill switching mechanism and so the power circulation, leading to the overall system initialization guided by the *OBC-MAIN* component. The solar panels are consequently deployed and the complete EPS becomes operative. The second Functional Chain focuses on how the solar panels power is managed to recharge the batteries. A malfunction in any of the involved Exchanges means that the system is unable to deliver the overall service. Once created, Functional Chains can be represented in a dedicated Functional Chain Description diagram, as Figure 3.19 and 3.20 show. It is reminded that these logical successions of Functions do not include the temporal variable here.

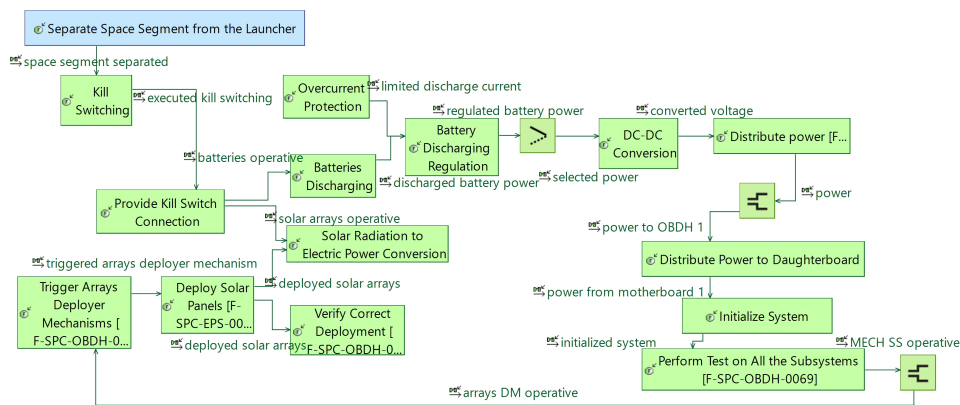


Figure 3.19: [LFCD] EPS Initialization and Solar Arrays Deployment

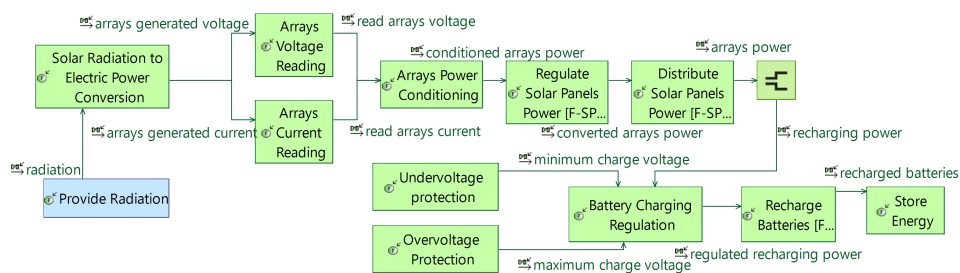


Figure 3.20: [LFCD] Battery Recharging from Solar Arrays Power

On-Board Data Handling Subsystem (LA)

This section presents the Logical Architecture modeling of the subsystem that processes and distributes commands, elaborates, stores and formats data [79]. The OBDH is a very delicate subsystem due to the many interfaces it has with the rest of the platform and the payload too. The here adopted MBSE approach is identical to the EPS one: a functional analysis is conducted, sometimes appearing as a solution coming from the OBDH subsystem engineer, leading to the definition of proper Components that can carry them.

The Logical Architecture Blank diagram in Figure 3.21 presents the overall OBDH modeling; two main Components, coming from engineers design, are present: *DOCK-MAIN* and *DOCK-GNC* boards. They provide interfaces to the allocated *OBCs*, which in turn contains most of the Functions and the related Exchanges. Moving the attention toward the middle-left portion of the diagram, the payload modeling can be noted. As previously stated, it is described in a very synthetic but sufficient way, enough to allow the description of its interfaces with the CubeSat. Three Functional Chains, *System Initialization*, *Payload Data Acquisition, Storage and Transmission* and *System Passivation*, are highlighted respectively in blue, red and green. The black outline in the *Perform Test on All the Subsystems* function indicates that it is part of more than one Functional Chain. Their Logical Functional Chain Description diagrams are reported in Appendix A.3

Guidance, Navigation and Control Subsystem (LA)

The GNC subsystem provides determination and control of attitude and orbit position, plus pointing of the CubeSat [79]; it represents the most important subsystem for the e.Inspector mission due to the delicate proximity operations near the target, better discussed in Section 3.2.7.

The Logical Architecture Blank diagram realized for the GNC subsystem reported in Figure 3.22 shows four main Logical Components: *Navigation Image Processing*, *GNC Algorithms*, *Sensors* and *Actuators*. It is clear that the first two listed should actually be part of the OBDH subsystem, in particular carried by one (or both) of the *OBCs*. However, this is a step further in the modeling process that would unnecessarily constraint the Physical Architecture. For this reason, it was decided to include them inside the GNC subsystem. It is also highlighted that the *Navigation Image Processing* component only contains those functions related to the image processing used for navigation scopes, different from the image processing for science reported in the OBDH diagram in Figure 3.21. Concerning the *Sensors* and *Actuators* blocks, they are voluntarily still very generic to leave a certain degree of flexibility in case of future modifications of the diagram. Moreover, a precise definition of the kind of sensors and actuators would imply a consequent implementation decision that may result too precocious at this stage.

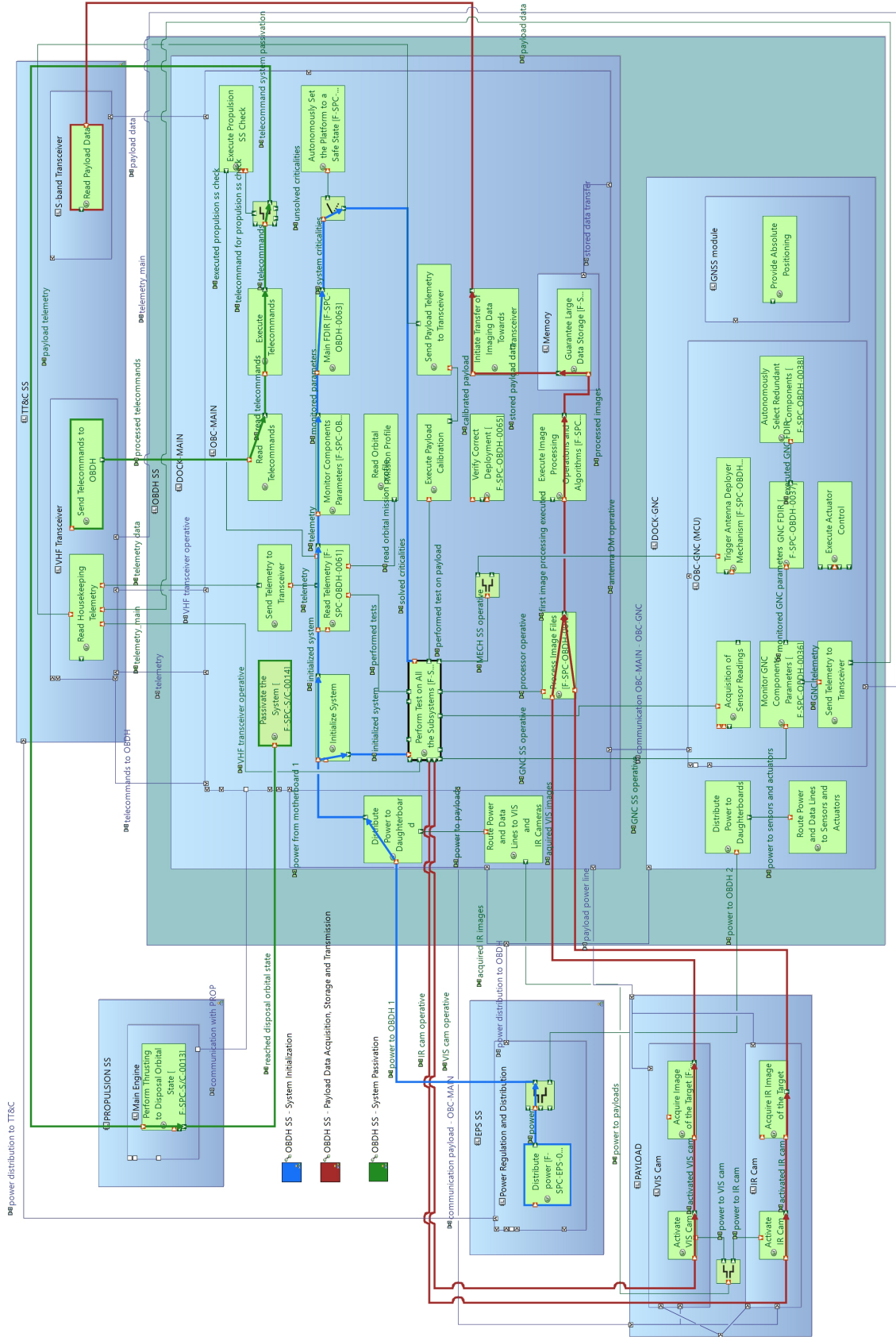


Figure 3.21: [LAB] OBDH SS

The *Close Proximity Relative Navigation during Eclipse* Functional Chain is highlighted in blue in Figure 3.22; the related Logical Functional Chain Description diagram can be found in Appendix A.3. Starting from the infrared images acquisition, it describes the logical sequence of functions that conduct toward the execution of attitude maneuvers aimed at performing target tracking during the eclipse. The purpose of MBSE in this context is not to explain how algorithms will work, but to consolidate the understanding and the presentation of how the different Logical Components with their Functions collaborate toward a unique scope. It is clear that the aforementioned Functional Chain cannot be applied to all mission conditions since, for example, whenever the target is not in eclipse the involved Functions would be different; also the distance from the target influences the component functions selection (this aspect will be better investigated later on this work). Many other Functional Chains can be created according to what the systems engineer wants to analyse.

Telemetry, Tracking & Command Subsystem (LA)

It is here presented the LA model of the subsystem that provides the interface between the CubeSat and ground systems [79], receiving commands and downloading telemetry and scientific data. The VHF band is employed for the telemetry and telecommands links, while the S-band is adopted for the payload data download [21]. The Logical Components devoted to their functioning (*VHF Antenna*, *VHF Transceiver*, *S-band Antenna* and *S-band Transceiver*), reported in Figure 3.23, may appear as implementation solutions, and so in contrast with the LA modelling philosophy. This is actually not true, since their description in terms of Function is voluntarily left wide-ranging. The good practice of naming Logical Components differently from Physical Components is not here respected because of the not so much detail increase that, as lately shown, takes place in the PA, where Logical Components are recycled and simply converted to physical ones. It is not excluded that in future new Functions aimed at better describing the internal functioning of the TT&C subsystem could be introduced.

The modeling of the interfaces between the CubeSat and the ground segment forces to increase the number of Functions of the latter with respect to the System Analysis. In particular, the *Ground Segment Provider* Actor now has a new Function called *Receive Beacon Signal from Space Segment* and two others that better describe its interaction with the *MOC*. The involvement of the *OBC-MAIN* components in Figure 3.23 is mandatory to show how the TT&C subsystem interfaces with the rest of the CubeSat throughout the OBDH subsystem. Three Functional Chains are highlighted: *Telecommands Transmission Line*, *Telemetry Transmission Line* and *Acquired Target Images Transmission Line*. Their *Logical Functional Chain Description* diagrams can be found in Appendix A.3.

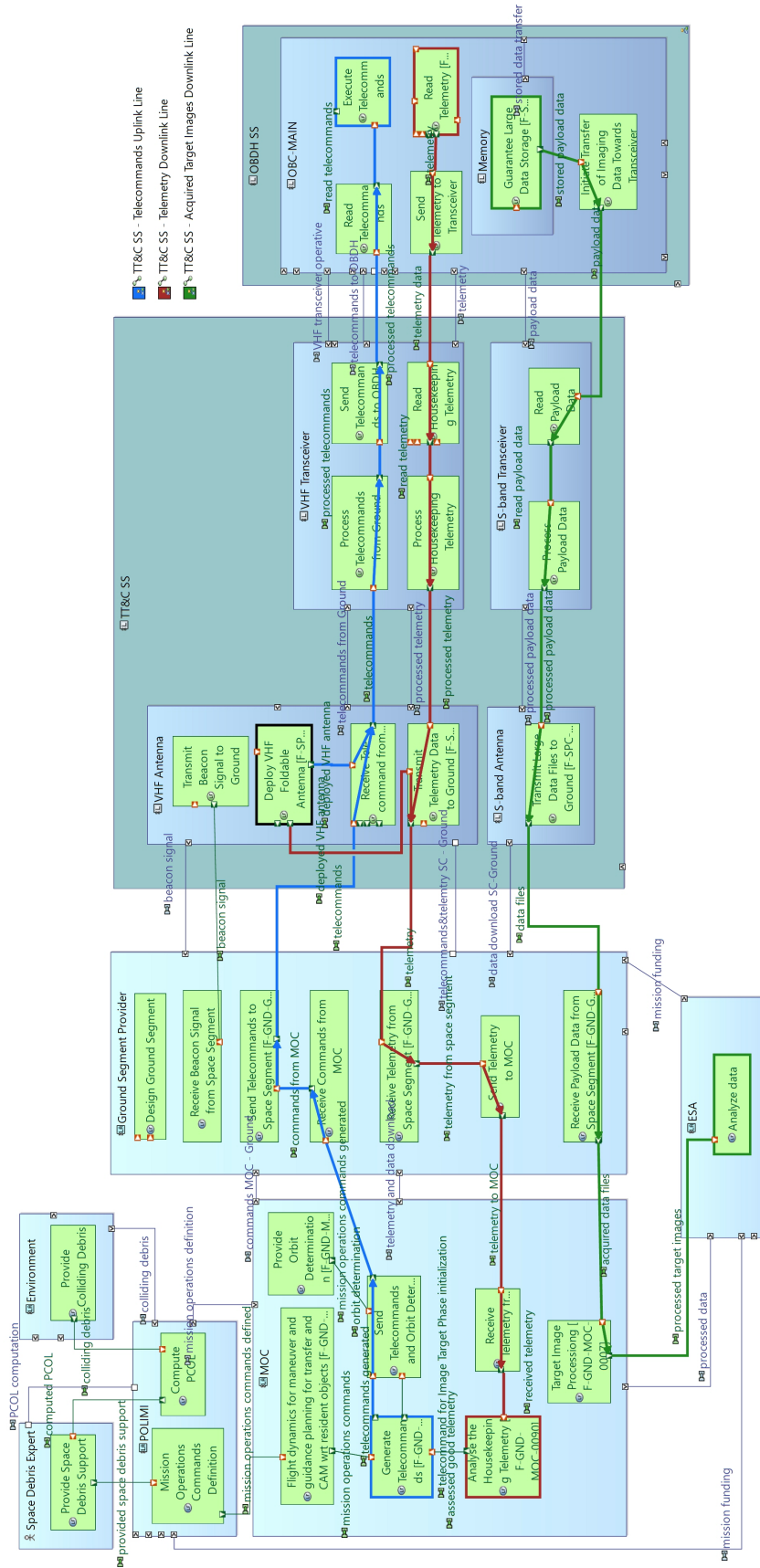


Figure 3.23: [LAB] TT&C SS

Propulsion Subsystem (LA)

The e.Inspector CubeSat is equipped with a Propulsion Subsystem which provides thrust to execute orbital and relative maneuvers for the various mission phases described in Section 3.1. The modeling of this subsystem is strictly related to the Concept of Operations, indeed the Logical Architecture Blank diagram in Figure 3.24 shows seven main Functions which reflect the transfer strategies after commissioning, the imaging phase, the CAM and the disposal phase as the engine has to provide the right level of thrust for all of such different stages. Also, the internal functioning is reported together with the main interfaces with other subsystems. As done for the GNC, just one Functional Chain in blue is reported as example, called *Thrusting to Drifting Orbit Initialization*, describing the activation of the Propulsion subsystem after having received the proper command. Its Logical Functional Chain Description can be consulted in Appendix A.3.

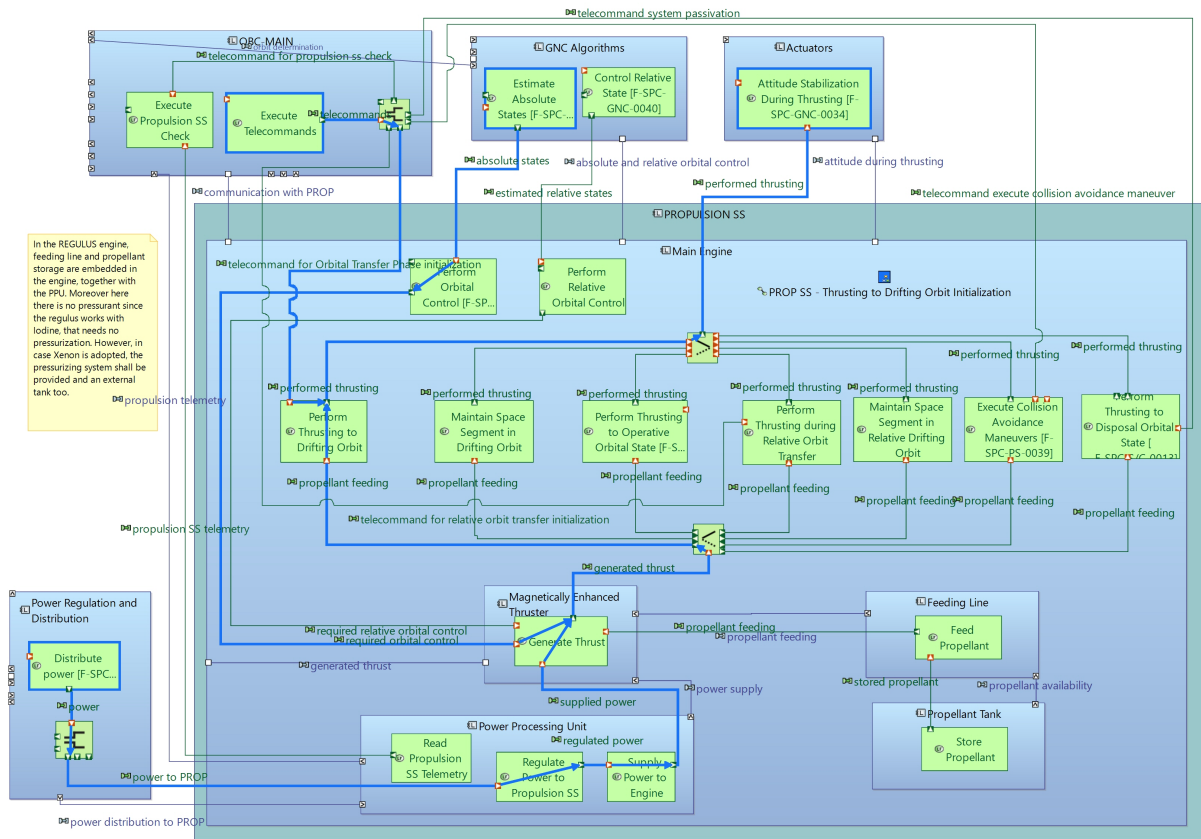


Figure 3.24: [LAB] PROPULSION SS

Thermal Control Subsystem (LA)

The TCS maintains equipment within allowed temperature ranges [79]. An important requirement for the e.Inspector mission states that “the thermal control shall be based solely on passive control techniques”, therefore the absence of active components make the diagram in Figure 3.25 very simple. The *Thermocouples* Component indicate that some of them are needed to *Monitor components temperature*, as the contained Function suggests.

A detailed definition of how thermocouples are distributed among physical components is not here reported since it is out of the LA scope, therefore such vague description is sufficient to understand how temperature measurements are done. Things may change during late design phases, so this diagram may increase its complexity and is open to future updates.

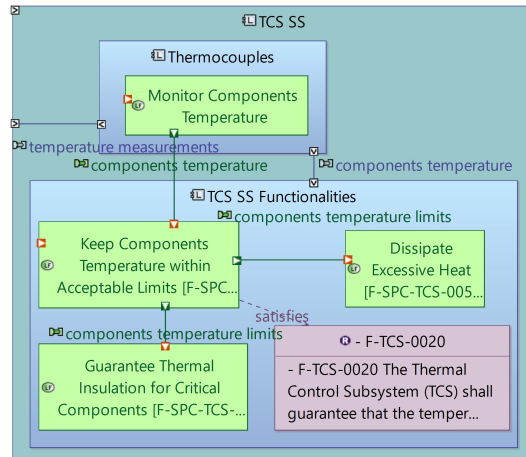


Figure 3.25: [LAB] TCS SS

Structures and Mechanisms Subsystem (LA)

This subsystem provides support structure and moving parts [79]. With respect to the System Analysis two camera supports were added, describing how payloads are integrated with the structure. No details on physical interfaces are reported here, since these aspects are mainly related to the Physical Architecture. Figure 3.26 resumes the Logical Architecture of this subsystem in a synthetic Logical Architecture Blank diagram.

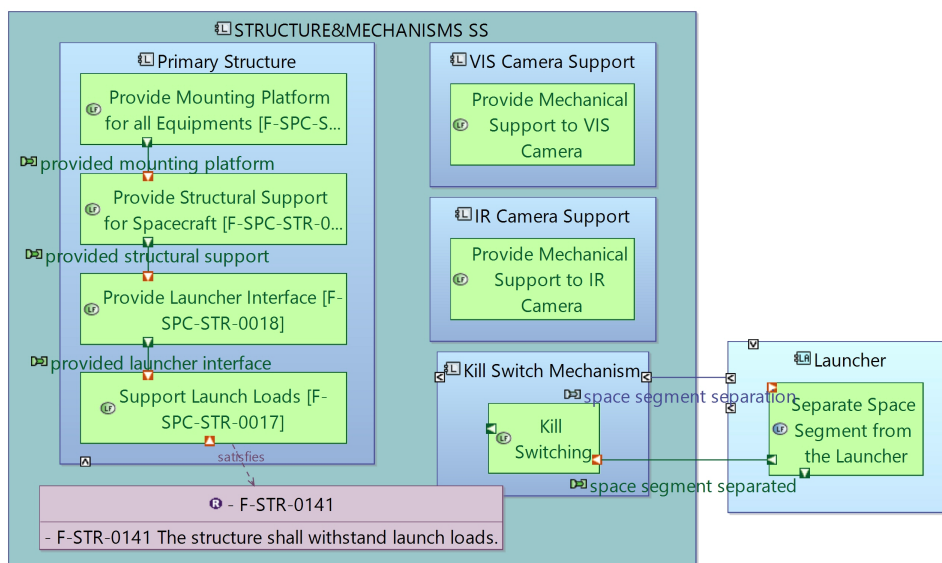


Figure 3.26: [LAB] STRUCTURES & MECHANISMS SS

e. Inspector Global View (LA)

All the previously described subsystems and actors are reported in Figure 3.27, where just the Components are present without the allocated Functions. The Component Exchanges provide a description of the information flows between subsystems, recalling that each of them provides the “transport” to one or more Functional Exchanges. This diagram also allows the model user to navigate through the previous diagrams using this global view as reference. Indeed, the little icon in the bottom-right of Components indicates the existence of one or more diagrams modeling them. Logical Actors are empty since they were not broken down into Components, as previously discussed. For graphical reasons, the *Space Segment* is colored in light grey while its subsystems in cyan. The remaining Logical Components are represented in blue per default Capella color-coding.

It is here clarified that the LA, despite quite detailed due to the system complexity, does not freeze the design. Indeed, the range of feasible alternative solutions can still be explored and modifications can be introduced at this level. Attention must be paid in case of a sudden change in the LA during late design phases, since the successive modeling level, the Physical Architecture, is derived from it and therefore coherence must be carefully guaranteed. The contrary is not true, so a modification done at LA level should not have implications on the higher levels (OA and SA).

It is worth showing how Logical Functional Breakdown diagrams are automatically generated by Capella, downstream to the definition of relationships between Functions. Due to the high number of Logical Functions, it would be very difficult to represent all of them in one single page, therefore just the breakdown of *Execute close-up visual inspection of a space debris* is reported in Figure 3.28 as an example.

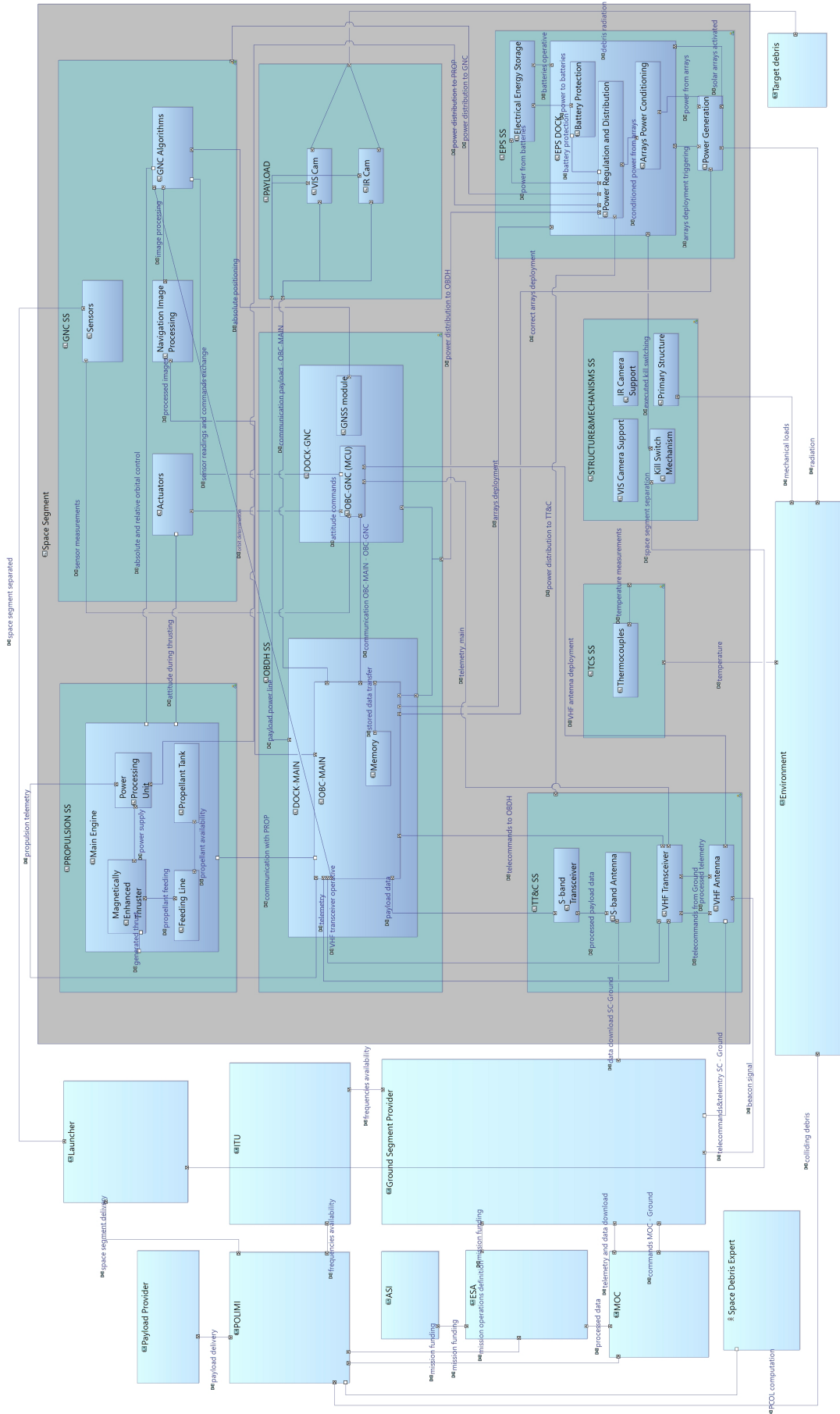


Figure 3.27: [LAB] e.Inspector GLOBAL

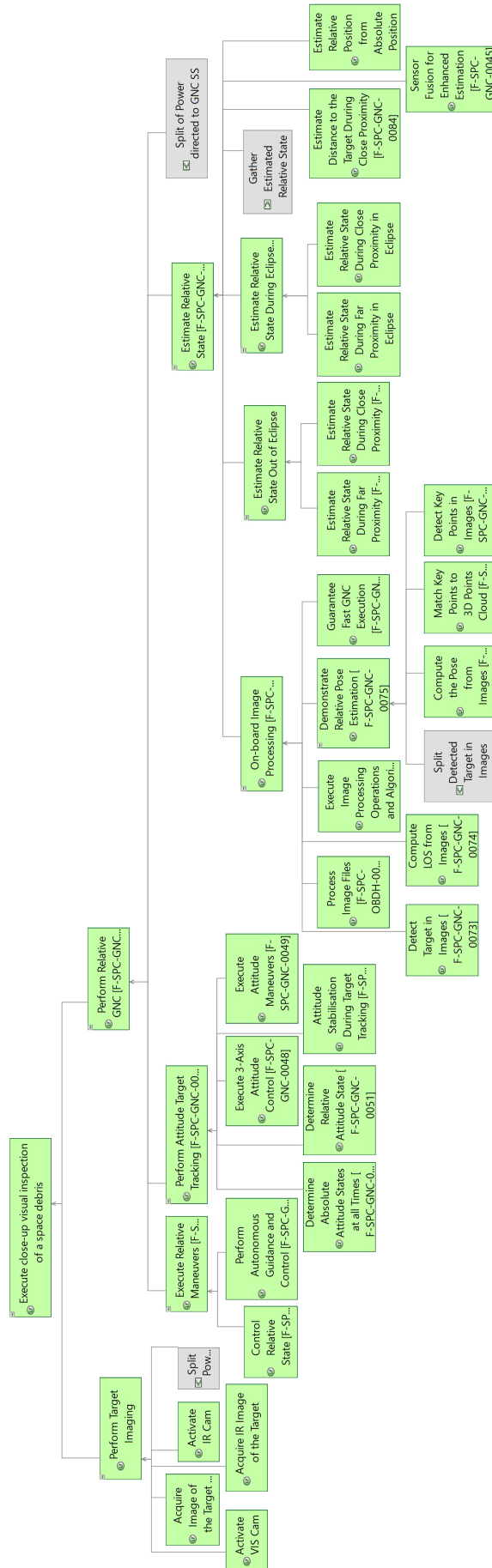


Figure 3.28: [LFBD] Execute close-up visual inspection of a space debris

3.2.5 Physical Architecture

The fourth level in the ARCADIA method is called Physical Architecture. Here the technological choices are modeled and the focus moves toward Physical Components definition that will constitute the real system. It is recommended to develop it once the system alternatives have been narrowed down to a limited number (possibly one) and a trade-off analysis already conducted, otherwise the effort of modeling a lot of architectures becomes considerable. The usual transition of model elements from the LA is performed once again, providing the starting point for a more detailed analysis. All components here presented are not specific since the procurement aspects are not part of the PA; this allows to have a modeling approach quite general and focused on the architecture, that can be then declined into the components selection.

In order to well understand the diagrams that will be presented in the next pages, it is suggested to carefully read the description of the PA in Section 2.4.2, focusing on the difference between Behavior Physical Components (blue boxes as baseline) and Node Physical Components (yellow boxes as baseline). In PA the concept of Physical Link has a central role since it allows to model the real interfaces among components. The default Capella color for these links is red, however a customized palette is adopted for this work due to the different kind of interfaces present in a CubeSat. Figure 3.29 shows the color code: the classical red is used for Data Interfaces (such as data exchanges between OBCs and sensors or actuators, commands distribution, etc.), the orange represents Electrical Interfaces (power lines) and the black color is adopted for Mechanical Interfaces (physical interfaces, mechanical supports, etc.).

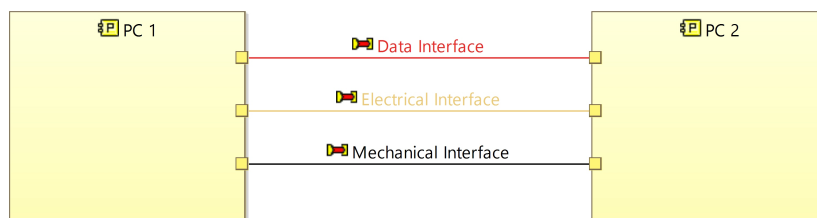


Figure 3.29: Physical Links Legend

As previously done, each subsystem will be analyzed in the following, fixing all the design ambiguities left unsolved in the LA and leading to a much finer level of detail. It is recalled that the trade-off analysis of the different possible system physical architectures, together with the components selection and the overall system sizing, is part of the Mission Description Document [21]; therefore, the following work does not intend to justify such kind of design decisions but aims at modeling the final architecture in a MBSE context, in order to demonstrate the use of the ARCADIA method and the Capella tool applied to a CubeSat design, here the e.Inspector mission.

The Physical Architecture Blank diagrams are the most recurrent for this work, there-

fore they are the baseline for each subsystem modeling unless differently specified. Since it would be quite hard to show the steps involved in the diagrams building, they will be presented in their final form and the rationale behind them will be discussed. For each subsystem a first PAB diagram is presented, with the aim of introducing the internal Physical Node Components and the internal Physical Links. Other diagrams are adopted to show the Physical Behavior Components and their Exchanges with other subsystems. Physical Functions are reported just occasionally, whenever the diagram complexity is not so elevated or when a significant breakdown is introduced with respect to the logical Functions. However it shall be reminded that each Component actually contains a number of Physical Functions which describe them, as well as each Component Exchange contains one or more Functional Exchanges.

Electric Power Subsystem (PA)

Figure 3.30 shows the *Physical Architecture Blank* diagram in which EPS internal links are highlighted. The important information extracted from this diagram is the physical implementation, since each yellow block is a Node Component that will be part of the Product Tree, and so of the system; the cyan is used again to distinguish the EPS component from the others.

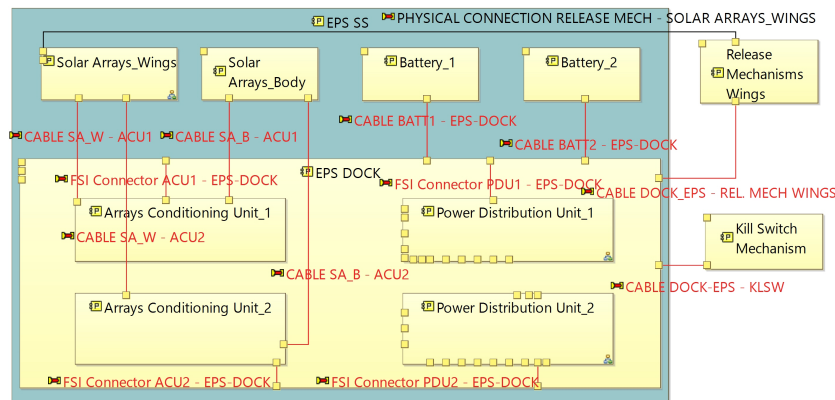


Figure 3.30: [PAB] EPS SS Internal Physical Links

The solar panels are differentiated into *Wings* and *Body-mounted*, two *ACUs* and two *PDU*s are chosen as baseline for allowing redundancy of power lines and limiting the stress on the component [21]. These are implementation choices, absent in the LA where just the conceptual architecture aimed at the system functioning description was required. For the modeling of redundant components, a powerful Capella functionality, called Replicable Element Collection (REC), allows to replicate a set of model elements as a whole avoiding to spend too much effort in modeling multiple times the same element. In example, since the *PDU*s carry the same Behavior Components and Functions, once the first one is modeled, the remaining one can be easily replicated, saving time and effort. Then, the interfaces of each replica with the rest of the system can be specified according to the

system architect needs. Figure 3.31 illustrates this concept applied to the *PDU*s. Each *Power Distribution* Behavior Component has a Split function used to model the ON/OFF switching of each power line.

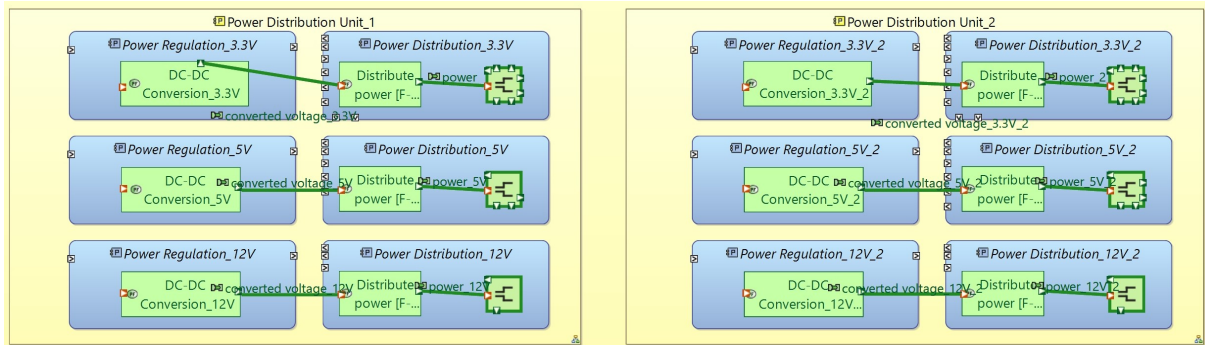


Figure 3.31: PDUs Replicas

The main EPS function is to distribute power to all system components, therefore it is worth to analyze the way it interfaces with the rest of the CubeSat through the *Power Distribution Units*, which power lines are reported in diagrams of Figures 3.32 and 3.33. At the time of this work, the e.Inspector mission is concluding the Phase A design, therefore a detailed interface engineering cannot be provided. Aspects such as the eventual routing of power throughout the boards (such as the *DOCK-GNC* to sensors and actuators) still have to be fixed, therefore all the lines are here modeled by means of a direct feeding from the *PDU*s; future updates in the architecture would not surely represent an obstacle from the modeling point of view, due to the flexibility of the approach to sudden changes. In order to differentiate the main power lines from the backup ones, the Component Exchanges are called differently, using the words *main* and *secondary*. It is recalled that Physical Links have Component Exchanges allocated, while the latter have in turn Physical Functions allocated (not reported in these diagrams for seek of simplicity). Since diagrams are by default *Synchronized*, whenever two Components appear in a diagram, all the Exchanges between them are automatically shown. This is what happens in Figure 3.32, which results to be a messy diagram. Since the numerous connections may distract from the purpose of the diagram, that is to present power distribution lines, Capella allows to set diagrams in *Unsynchronized* state; this is done for the *PDU_2* in Figure 3.33, where also Behavior Components are hidden.

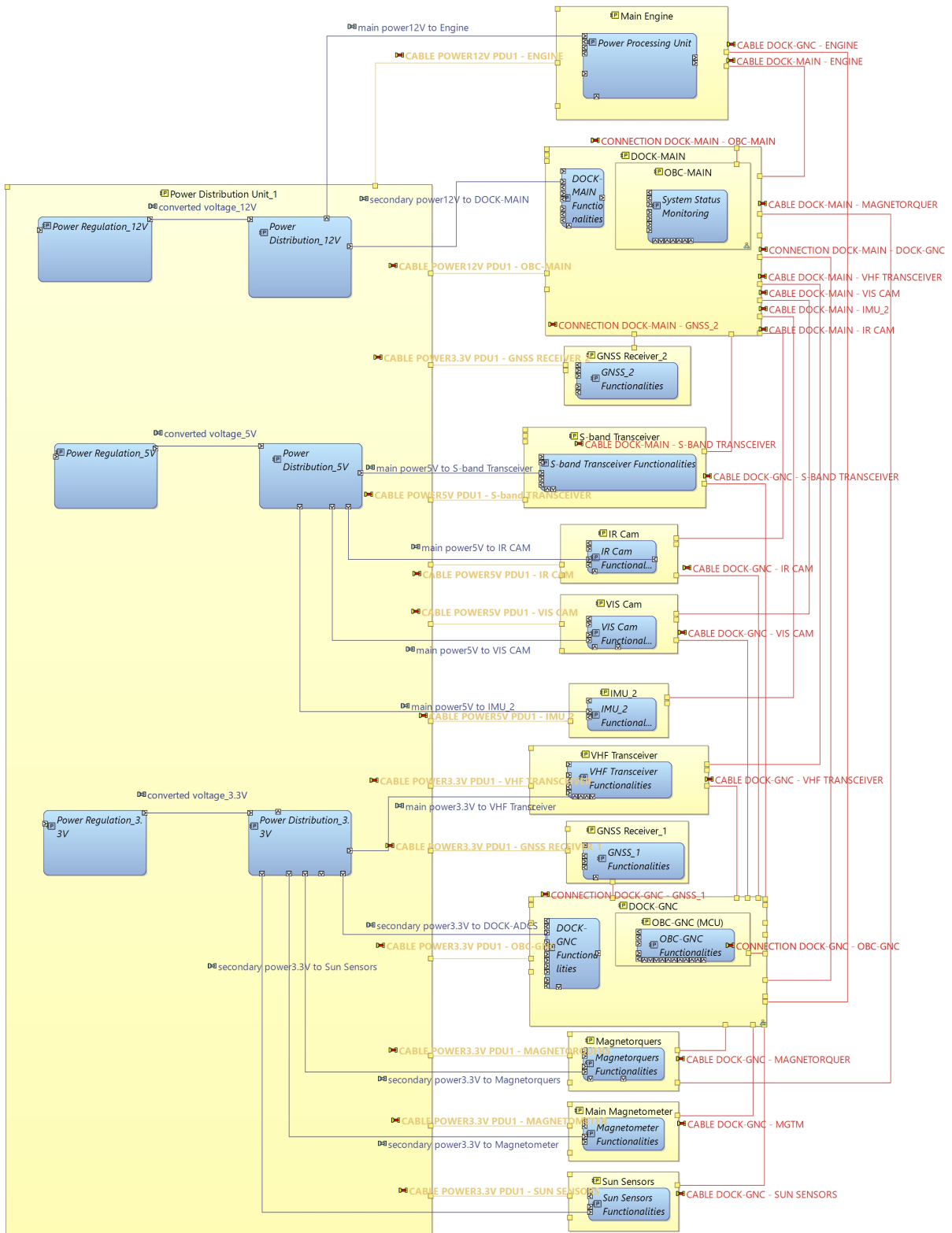


Figure 3.32: [PAB] EPS SS - Power Distribution Unit 1

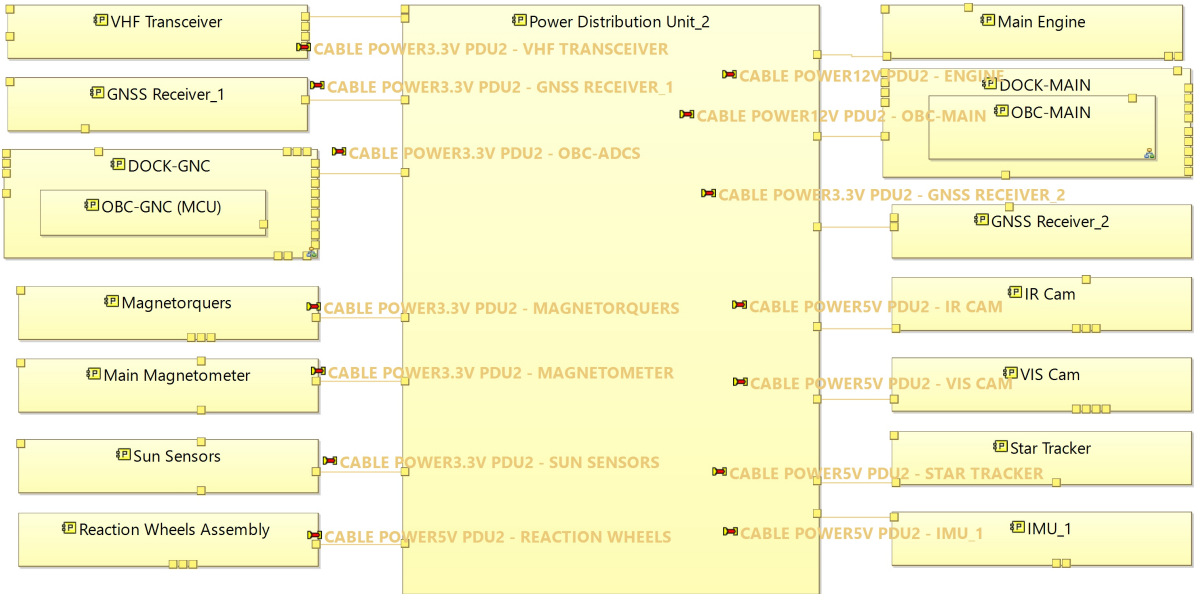


Figure 3.33: [PAB] EPS SS - Power Distribution Unit 2

On-Board Data Handling Subsystem (PA)

The OBDH components and their internal Physical Links are reported in Figure 3.34; the *DOCK-MAIN* and the *DOCK-GNC* are two boards that mount the *OBCs*, all discussed in the next paragraphs. The internal functioning modeling of OBDH is not here presented since the focus is shifted toward the interfaces with the rest of the platform. A certain number of diagrams are proposed to furnish a complete description of such interactions, with particular focus on those with the GNC and the TT&C subsystems. Since some Functions have been broken down into leaf ones in order to support the PA level of detail, an example is reported in Physical Data Flow Blank diagram of Figure 3.35, showing the *Execute Actuator Control* Function breakdown.

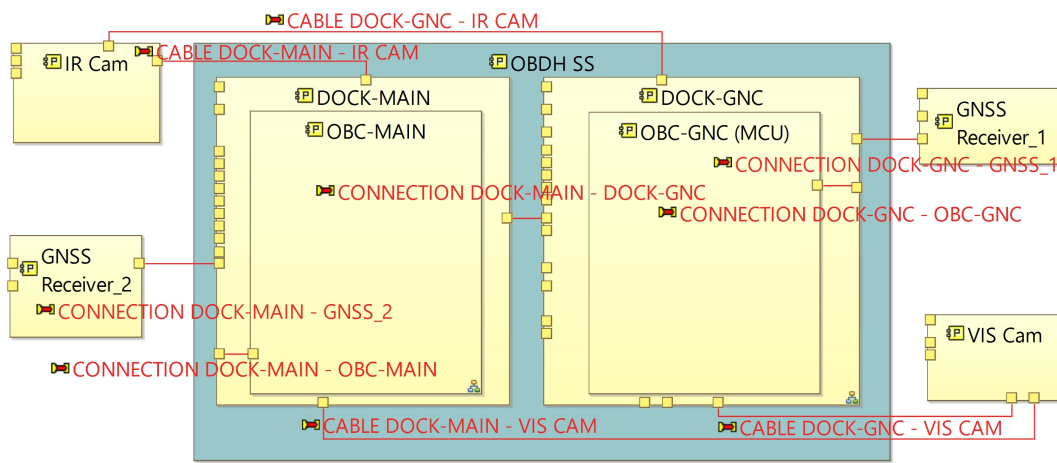


Figure 3.34: [PAB] OBDH SS Internal Physical Links

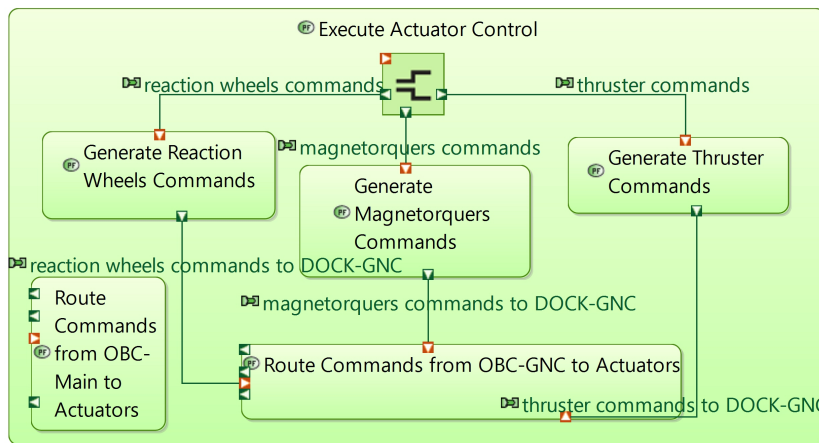


Figure 3.35: [PDFB] Execute Actuator Control

It is firstly presented the diagram in Figure 3.36 in which the two OBDH boards interface with the GNC components. The interfaces design is much more mature here with respect to the LA: the boards provide the routing of signals from sensors and to actuators letting them interface with the *OBCs*, which contain the Component Exchanges related

to the algorithms execution. It is recalled that in the LA these algorithms execution blocks were allocated to the GNC subsystem in a quite generic way; here instead the maturity of the design allows to directly allocate them to precise OBDH components. The definition of these interfaces has been carefully designed by the OBDH engineers and a brief justification of them is given here to better interpret the diagram. The *OBC-GNC* is the baseline computer in charge of attitude and navigation algorithms, while the *OBC-MAIN* is mostly devoted to image processing and system functioning. It also provides redundancy for what concerns GNC algorithms, indeed secondary data routing lines to all actuators and from some sensors are present. The *IMU 1* and the *GNSS 1* are connected to the *OBC-GNC* through the *DOCK-GNC*, the *IMU 2* and the *GNSS 2* to the *OBC-MAIN* by means of the *DOCK-MAIN*. This way, the absolute determination is not lost unless both computers stop working. Concerning the relative attitude, three sensors provide their measures to the *OBC-GNC*, as on Figure 3.36. In case of its loss, the camera can be used as horizon sensor by the *OBC-MAIN* guaranteeing the relative attitude determination at any time together with the *IMU 2* angular velocities measures.

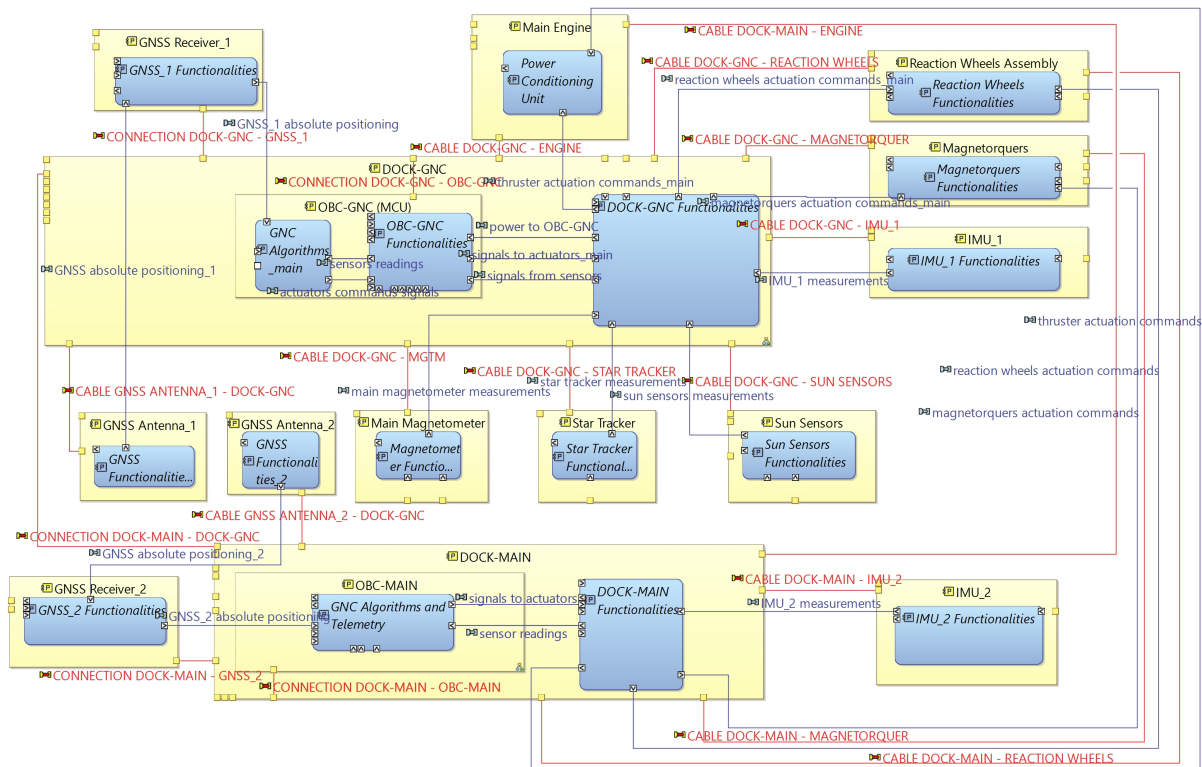


Figure 3.36: [PAB] OBDH SS - GNC

Another important interface of the OBDH subsystem is the telemetry reading and its transmission toward the radios, together with the scientific data. The diagram in Figure 3.37 shows both the *OBC-MAIN* and the *OBC-GNC* collecting telemetry data, being the latter used as baseline (subscript *main* in diagrams) while the former as backup. In order to not have one single messy diagram, the GNC telemetry is presented in a dedicated diagram reported in Figure 3.38. These diagrams provide a synthetic but highly descriptive view of the telemetry routing to the respective computers and allow to rapidly check

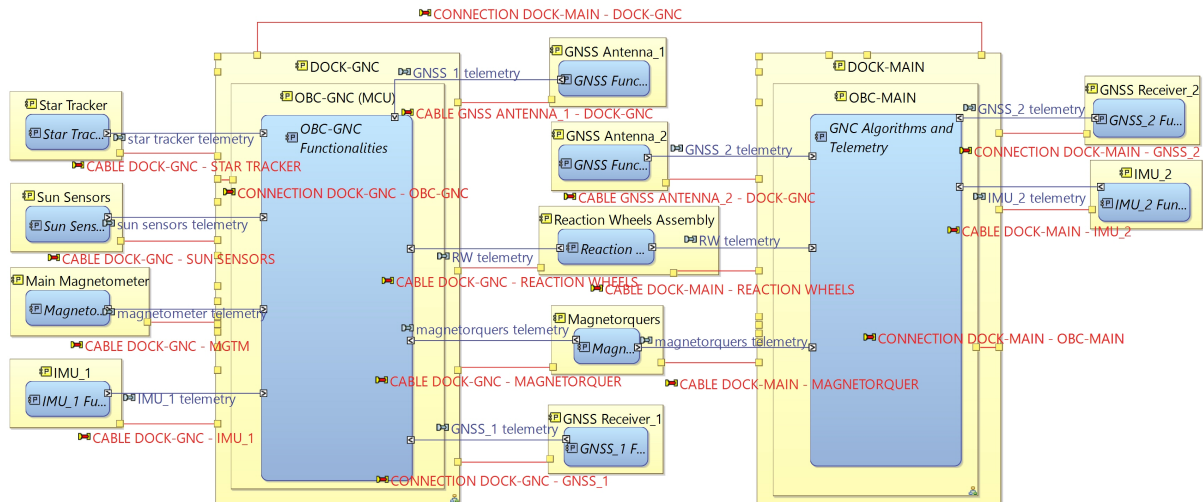


Figure 3.38: [PAB] OBDH SS - GNC Telemetry Reading

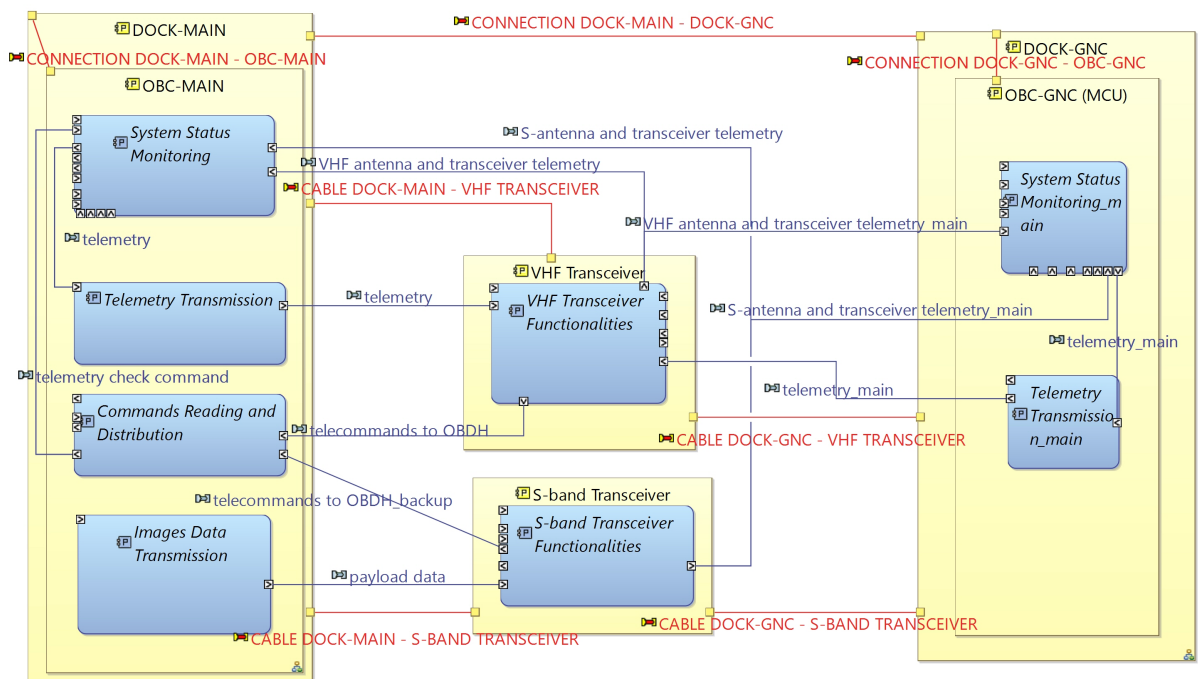


Figure 3.39: [PAB] OBDH SS - Telemetry and Data Downstream

Guidance, Navigation and Control Subsystem (PA)

With respect to the LA, here architectural components are precisely defined after the subsystem sizing. Since the algorithms execution blocks are in charge of the OBDH subsystem, as discussed in the previous section, the description is here limited to sensors and actuators modeling and their interfaces with the boards. Figure 3.40 presents them and together with their Functions.

Interfaces of GNC sensors and actuators with the OBCs were already presented in Section 3.2.5 in terms of Physical Links. Here a functional analysis is also conducted,

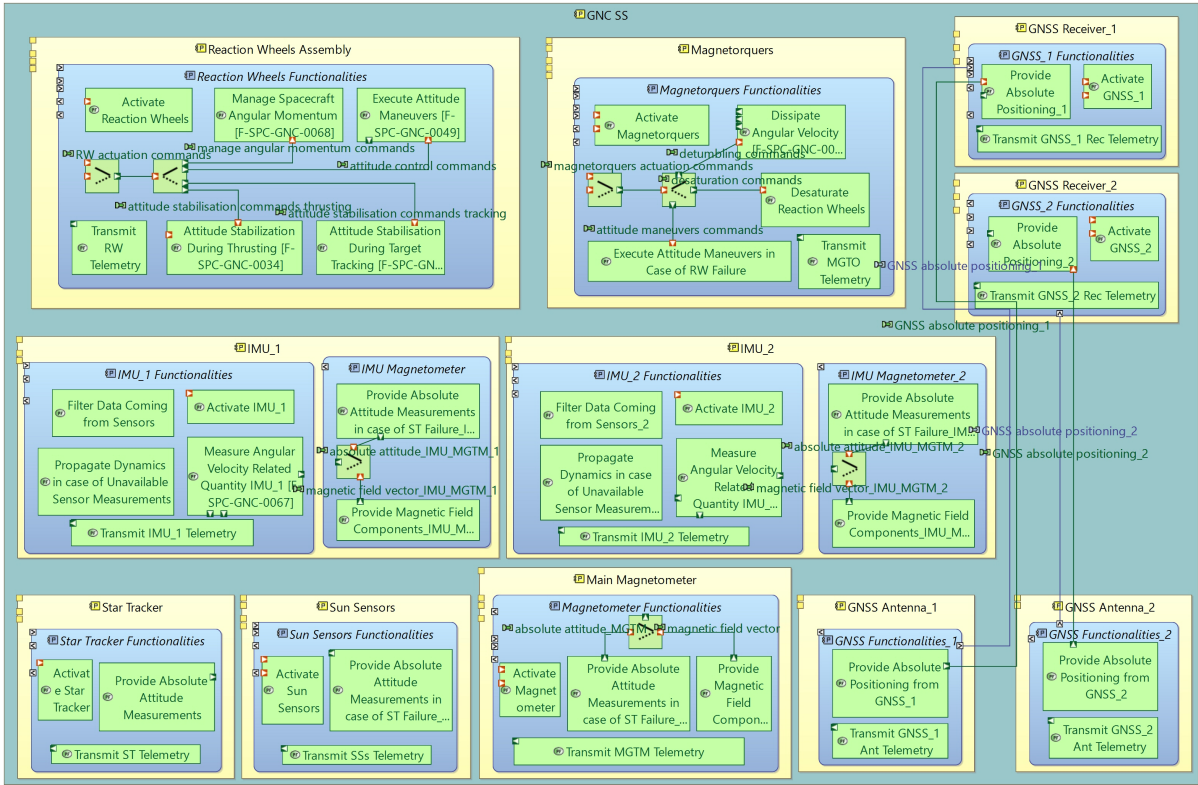


Figure 3.40: [PAB] GNC SS Components

with the aim of analyzing functional flows and data exchanges. Figures 3.41 and 3.42 respectively illustrate how sensors and actuators interface with the baseline computer in charge of attitude algorithms execution, the *OBC-GNC*. It is recalled that the Component *Sun Sensors* actually comprehends 12 sun sensors that are mounted on each of the 6 faces of the CubeSat; however it was decided to model them as a single Physical Component to avoid diagrams congestion. A future application of this model, especially from Phase B on, would certainly benefit from the modeling of each single sun sensor in order to precisely define their interfaces.

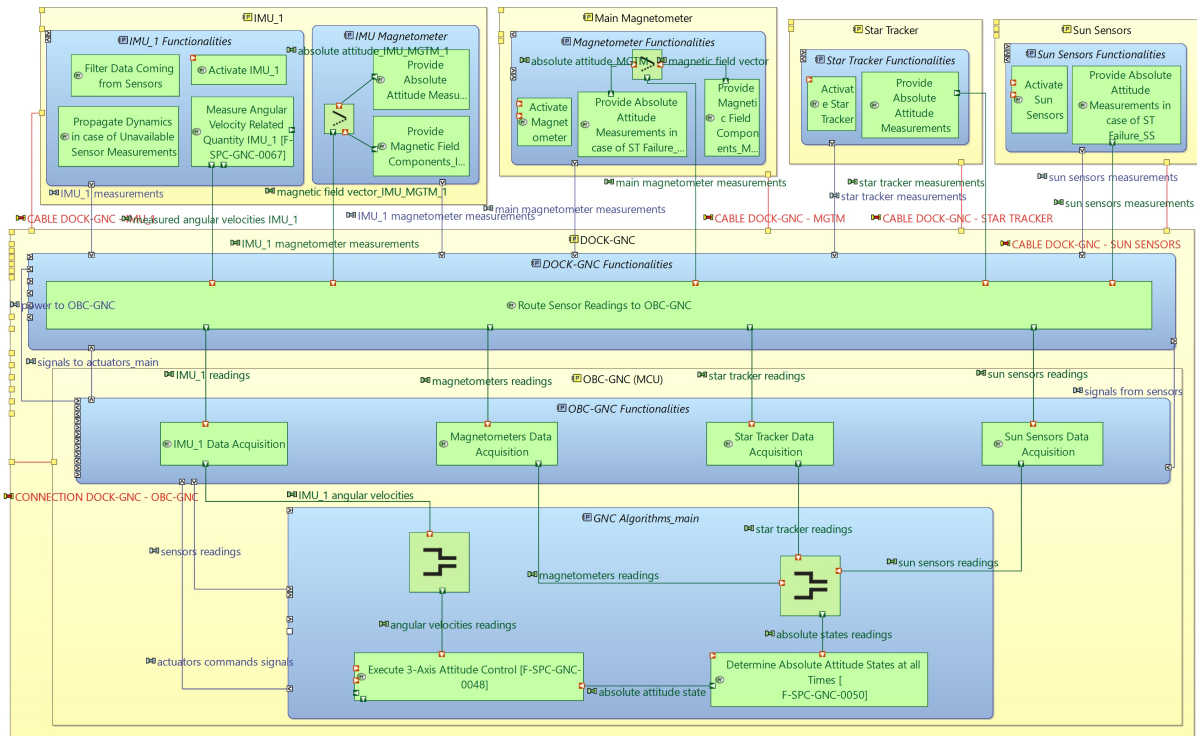


Figure 3.41: [PAB] GNC SS - DOCK-GNC - Sensors

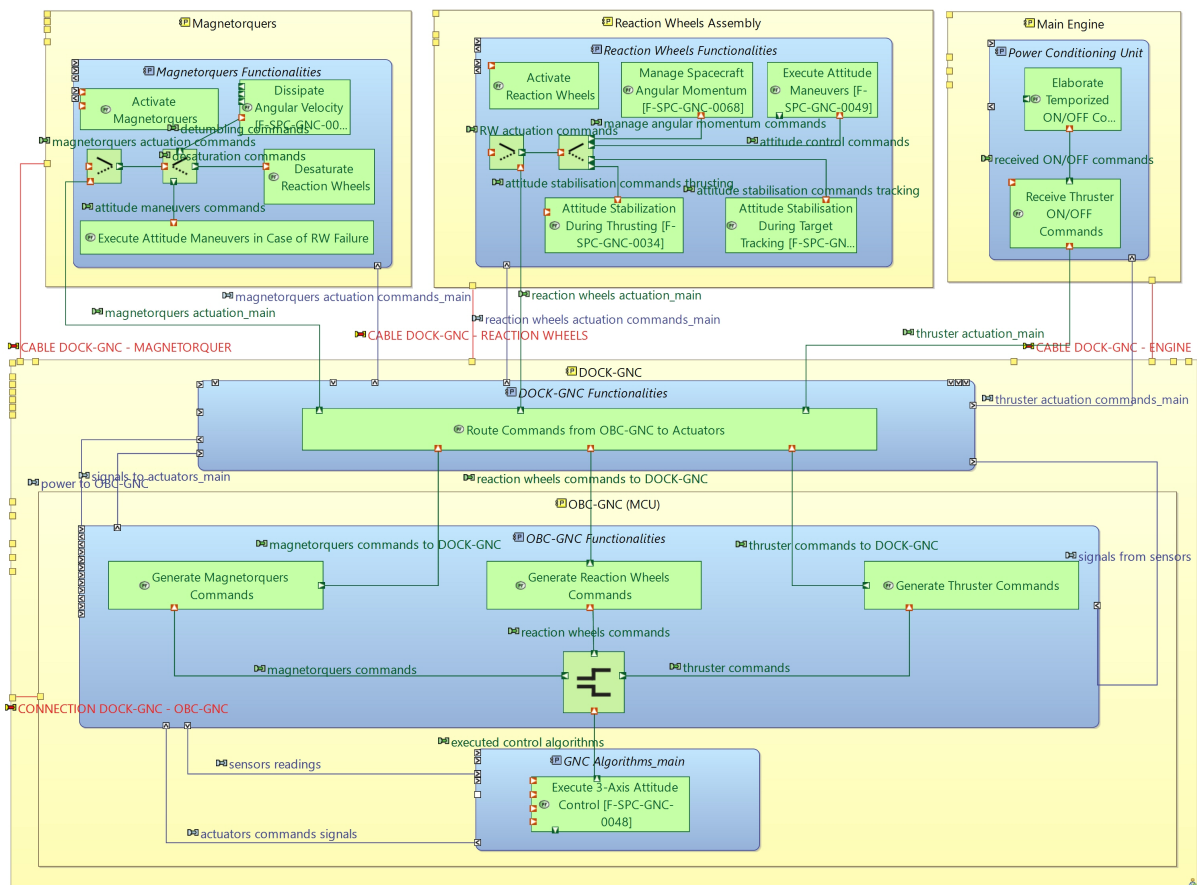


Figure 3.42: [PAB] GNC SS - DOCK-GNC - Actuators

Telemetry, Tracking & Command Subsystem (PA)

For the scopes of this work, the TT&C description does not go much further in the PA with respect to the LA. The only modification consists in the introduction of a new component, that is a second *S-band Antenna* used as backup in case of *VHF Antenna* malfunction for uplink operations. Consequently, the *S-band Transceiver* will have two new functionalities in order to deal with this component. Figure 3.43 shows these new model elements. Another comment concerns the *VHF Antenna Folding* mechanism, here modeled as Behavior Component inside the *VHF Antenna* since it is part of it and not as Node one, unlike the *Release Mechanisms Wings* for the solar panels deployment. The interfaces with ground systems can be recalled from the LA since they are unchanged, while the interfaces with other subsystems (basically the OBDH) can be consulted in Section 3.2.5.

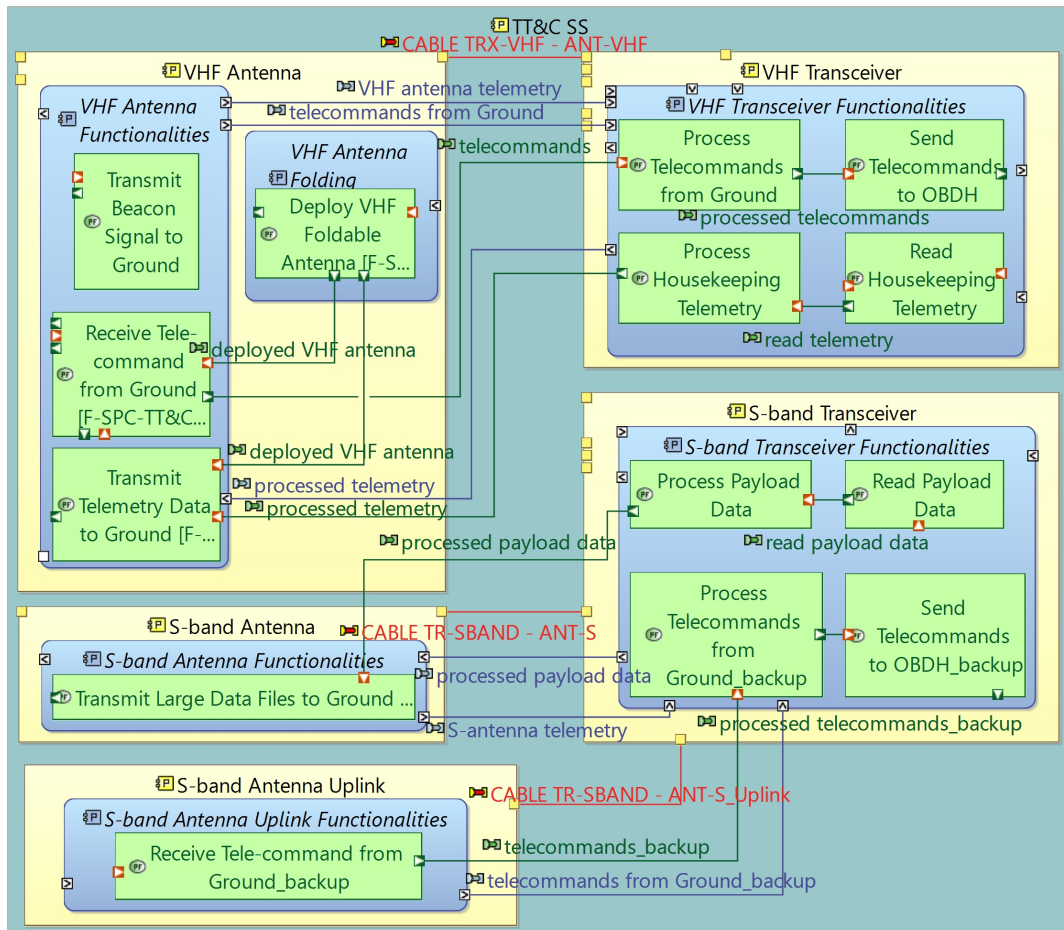


Figure 3.43: [PAB] TT&C SS Components

Propulsion Subsystem (PA)

The PA related to the Propulsion subsystem provides a few more internal functional details with respect to the LA, as Figure 3.44 shows. One single Node Physical Component is present, that is the *Main Engine*, which is composed by a number of sub-components

modeled as Behavioral ones. Therefore, just the engine will appear in the Product Break-down. Since the external interfaces of the engine were already encountered in previous diagrams, this section only focuses on its internal functioning.

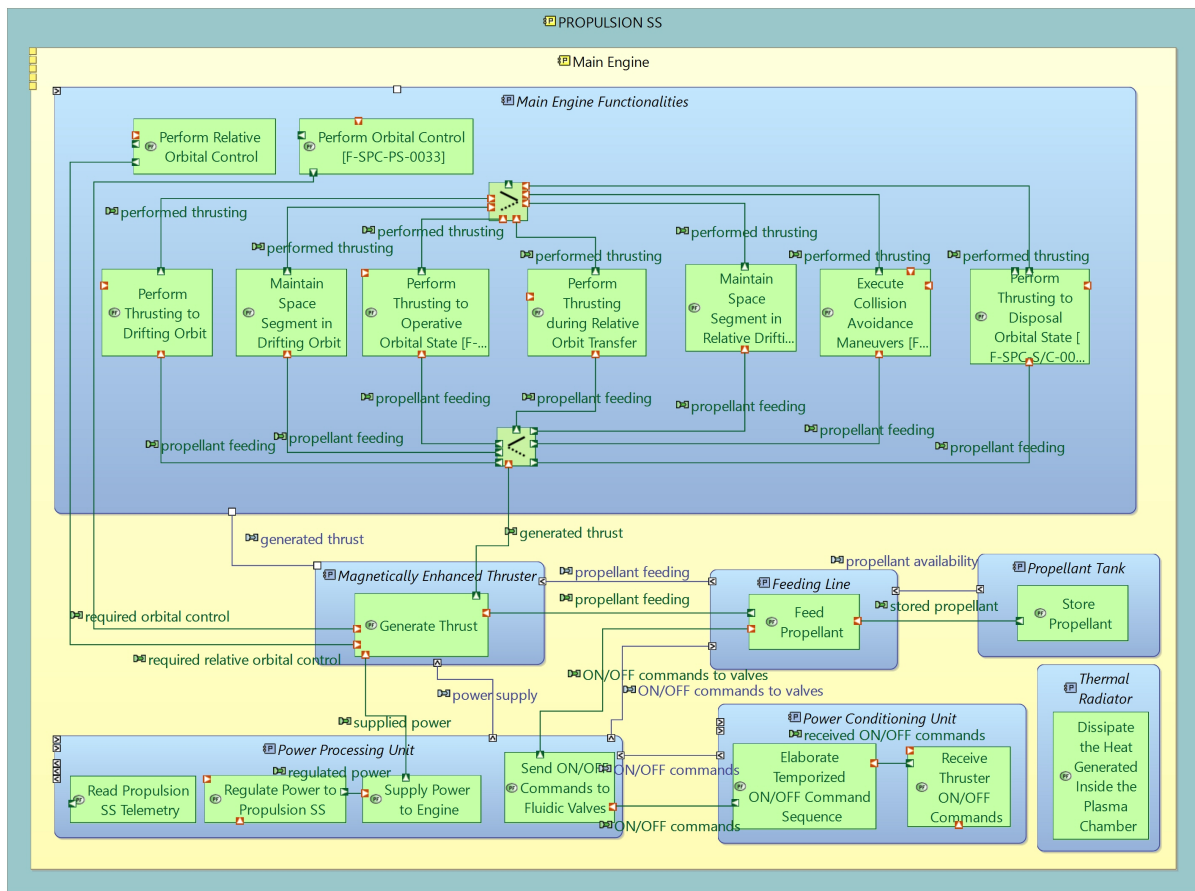


Figure 3.44: [PAB] PROPULSION SS

Thermal Control Subsystem (PA)

The LA modeling of the TCS is enough for the scopes of this work, also due to the absence of active components. Therefore, no diagram is here reported since the only modification is the introduction of *Paint* as a Node Physical Component, also present in the Product Tree. Concerning *Thermocouples*, they are still modeled as a single component, like in the LA, since their functioning is part of the components in which they are distributed and therefore require a step further in their design and modeling that is out of this work scopes. The reading of components temperatures is taken into account considering it as part of the telemetry coming from components, presented in Section 3.2.5.

Structures and Mechanisms Subsystem (PA)

Lastly, the Structures and Mechanisms Subsystem is modeled. Recalling that not all the physical interfaces are frozen at this level of design, some of them may be missing in the diagram of Figure 3.45. Not only physical connections among components and primary

structure are reported, but also some supports that represent mounting points. This diagram is useful for a Phase A since it provides a panoramic view of all the physical connections that shall be accurately designed and can also be exploited as reference for the integration plan development.

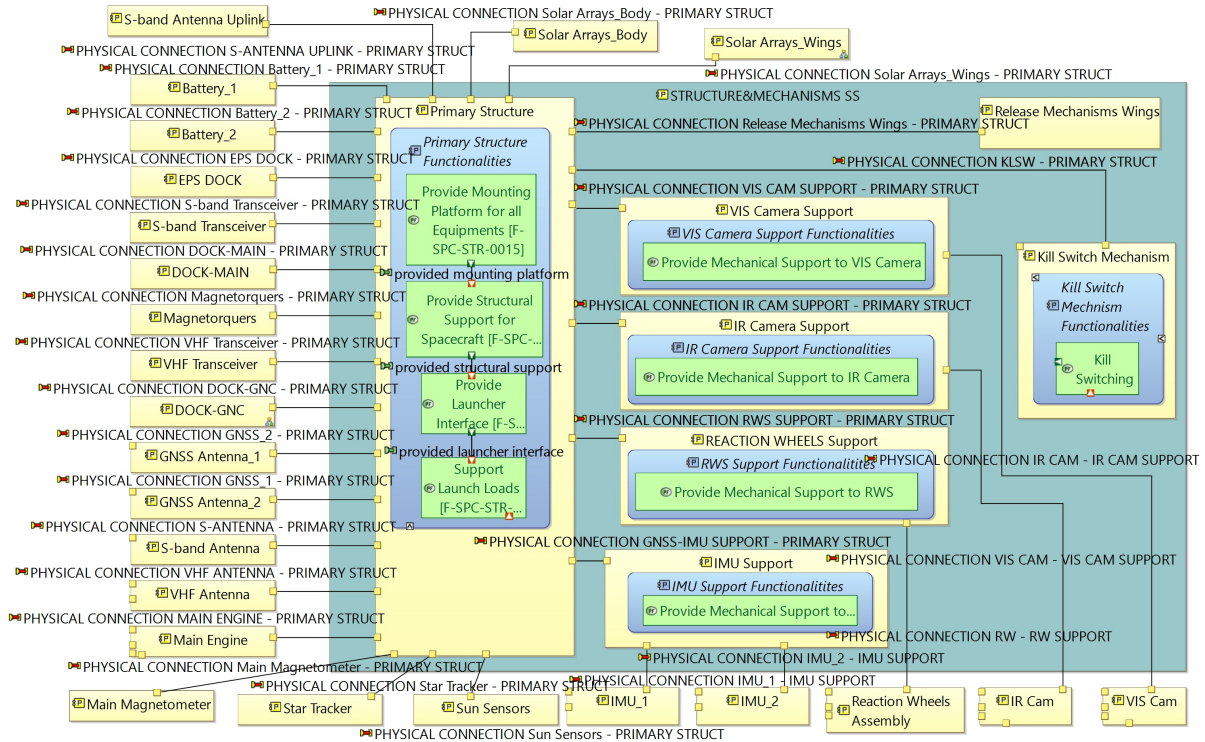


Figure 3.45: [PAB] STRUCTURE&MECHANISM SS Components

e. Inspector Global View (PA)

The presented Physical Architecture modeling demonstrated to be a very powerful support to systems engineering practices for a CubeSat design, in which multiple subsystems carry a number of components that have to communicate among them. Clearly, the precise data exchange modeling and communication protocols require a step further in the design process, not presented in this work, but still feasible adopting the ARCADIA method supported by the Capella tool.

Capella is supported by a number of Add-Ons that support system engineering activities and allow model refinement. One of them was already presented, called *Requirements Viewpoint*. Another one, exploited for the e.Inspector mission, is called *Basic Mass Viewpoint*. Its functioning is very simple: an allowable maximum mass is defined at system and subsystems level, then a mass is assigned to each component and the tool warns the user whenever the sum of components masses exceeds the maximum overall one. Applying this tool to each subsystem, and then at system level, the system engineer not only has a mass database included in the model, but also the possibility to rapidly experiment changes in

the design conducting trade-off analysis. An example is reported in Figure 3.46, showing the tool usage for the EPS. The “Max” mass assigned to each Node Component is the margined one. When a component, such as the *EPS DOCK*, contains sub-components, its total mass is the sum of the sub-components and its mass. Since the tool has been applied to each subsystem, the diagram in Figure 3.47 is realized, representing the mass database at system level. A very similar Add-On is called *Basic Price Viewpoint*, having the same functioning but focused on the cost of components.

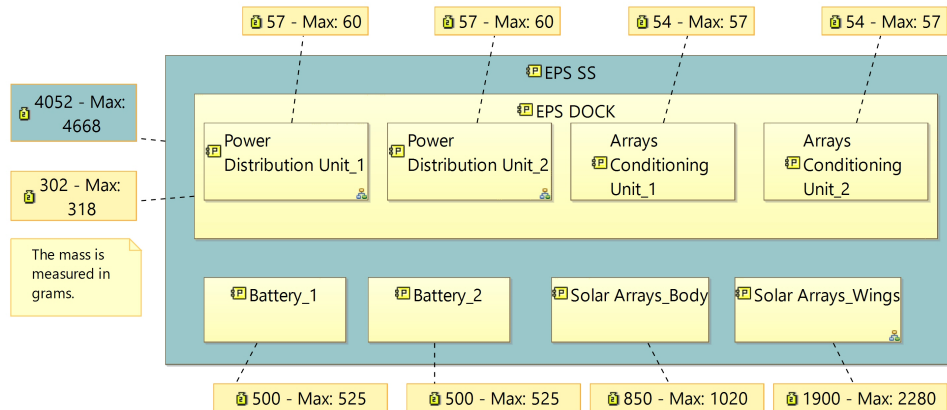


Figure 3.46: [PAB] EPS SS Mass

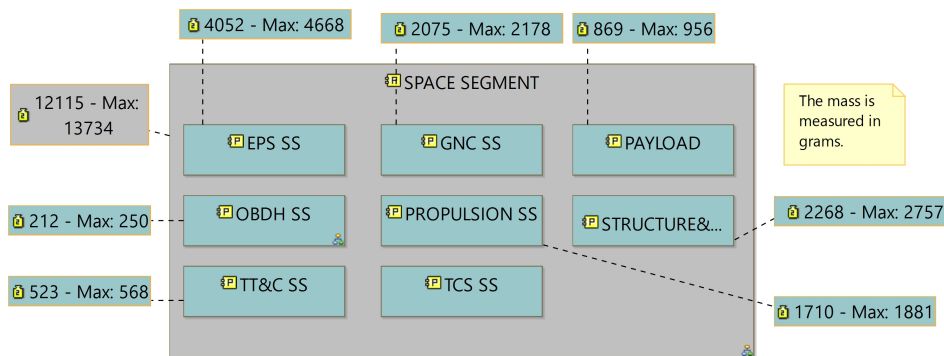


Figure 3.47: [PAB] ALL SUBSYSTEMS Mass

To conclude with the Physical Architecture, one of the outputs is the Product Tree, here automatically generated by the tool in the Physical Component Breakdown diagram, Figure 3.48.

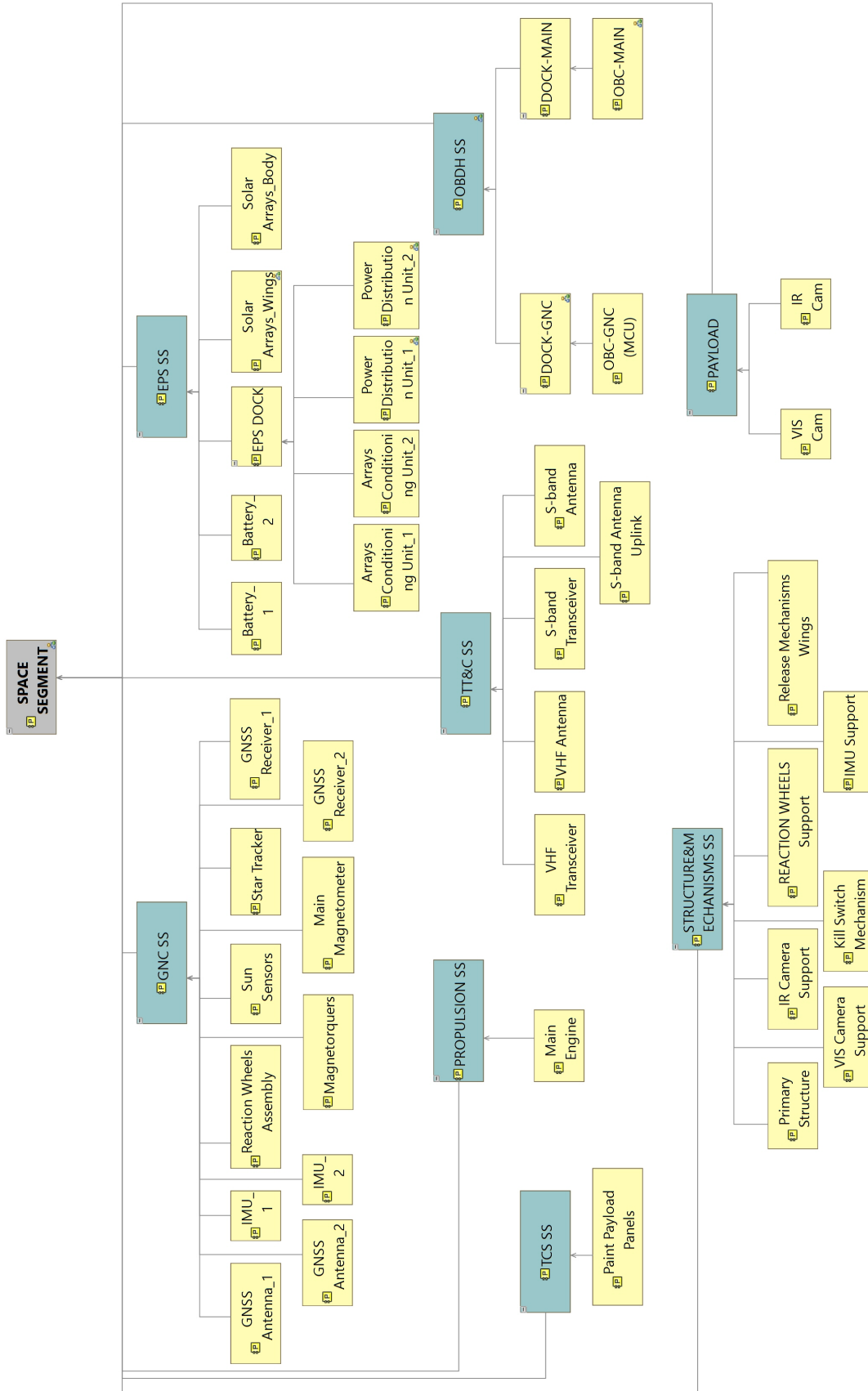


Figure 3.48: [PCBD] e.Inspector Product Tree

3.2.6 System and Subsystems Modes






A space system is conceived and designed having in mind its operative life, punctuated by a number of phases which define the whole mission. Particular attention must be paid while defining which subsystem functionalities are needed in each phase, therefore approaching a vast topic in system engineering that is the Modes and States definition. Many attempts have been done in the last years, as discussed in the works by Olver and Ryan [51] and by Wasson [78], with the aim of establishing a universal definition of states and modes, however some conflicts can still be found in literature proving that the use of one or the other is mostly a methodological choice in the context of a project.

Since the ARCADIA method distinguishes Modes and States, the two concepts cannot be mixed together in the same diagram in the Capella environment. A common definition among systems engineers states that a mode is the result of a design decision, allowing to consciously switch the system from one to another, while a State is the consequence of something that happens to the system, representing an unexpected or even undesired event. Only the concept of Mode is considered for this work. The adopted definition is taken from the work by Bonnet et al. [11], where each Mode is mainly characterized by the intended functional nature of the system at that time under certain conditions. The transition from one Mode to another is usually an explicit decision triggered by a functional event, such as a change in the use of the system to respond to new needs or situations. It is therefore conditioned by choices made by the system or by Actors through the creation of a Functional Exchange or the activation of a particular Function within the transition [75]. Modes are here described making use of Modes & States Machine diagrams in Capella.

The e.Inspector mission is characterized by four Phases, already presented in Section 3.1, each one requiring a number of system Modes. In Capella, Modes are defined by one or more Functions expected to be executed by the system. Recalling that all the Functions are carried out by Behavior Components and, in turn, by Node Components, whenever a Function is present in one Mode it means that the Component containing it is active. In theory, each Component should be characterized by some Modes and the combination of simultaneous Component Modes should identify a subsystem Mode, which then define system ones. This procedure can be very effective in late design phases, however it may result precocious to define all Components Modes during a Phase A. Therefore, the analysis will mostly focus on system Modes and some subsystem ones, those per which it is possible to conduct this kind of analysis at this level. In particular, the studied subsystems are the Propulsion, the TT&C and the GNC. In the following, their Modes are firstly presented, opening the stage to a focus on system ones.

In order to better interpret the diagrams that are going to be presented, an overview of the used model elements is reported in Table 3.1.

Table 3.1: State Machine model elements

	Initial Pseudo State	Used to represent the initial state when entering in a mode or state machine.
	Final Pseudo State	Represents the end of a state machine.
	Choice Pseudo State	It allows splitting of compound transitions into multiple alternative paths.
	Fork Pseudo State	Serves to split an incoming transition into two or more transitions terminating on different modes.
	Terminate Pseudo State	Implies that the execution of the mode is terminated immediately.

Another important concept is the *State Transition*, that defines the condition governing the passage from a Mode to another one. Two concepts are related to it: the *Guard Condition*, a Boolean expression written in squared brackets that must be true when the event takes place for the transition to be triggered, and the *Trigger*, that defines the condition for the transition activation, typically a Functional Exchange already present in the model. A Trigger can also be a *Time Event*, modeled using the keywords “after” and “at”, or a *Change Event*, modeled using the keyword “when” [55]; in these cases the expression that follows the keywords is not necessarily an already existent Functional Exchange.

All Functions and Functional Exchanges used to define the Modes contents and the Transitions belongs to the Physical Architecture. It could also be possible to define Modes using Logical Architecture or other levels elements, however for this work it was decided to proceed with the PA since it allows to provide a more detailed description.

Subsystems Modes

The first subsystem here analyzed is the GNC, whose State Machine diagram is reported in Figure 3.49. Each grey rectangle represents a subsystem mode. For graphical reasons it was decided to not report the Functions that each mode carries out, however it shall be remembered that all Modes are precisely described by the Functions they have allocated. An example is provided in Figure 3.50 for the *GNC Detumbling Mode*; this kind of expanded views is not be presented for the remaining Modes, in order to keep the focus on State Machine diagrams rather that on Modes definition. However, the model can be consulted in the Capella environment, navigating through these information too.

Focusing again on the diagram in Figure 3.49, it is interesting to provide the rationale that stays behind some of the Transitions. In particular, the Fork Pseudo State is adopted to distinguish the Navigation Modes from the Attitude ones. *GNC Absolute Navigation* and *GNC Absolute Attitude* are the baseline GNC Modes, active for the entire mission duration until Change Events happen. Concerning the Navigation, the switch from Absolute to Relative takes place once a well defined distance from the target debris is met; in turn, the distance also governs the Relative Navigation Modes selection, since they involve different GNC algorithms and techniques, therefore different subsystem Functions. Similar considerations are applied to the Relative Attitude Mode activation, as the transition in the diagram suggests. No specific details on the design, reported in the Mission Description Document [21], are intended to be given in this work.

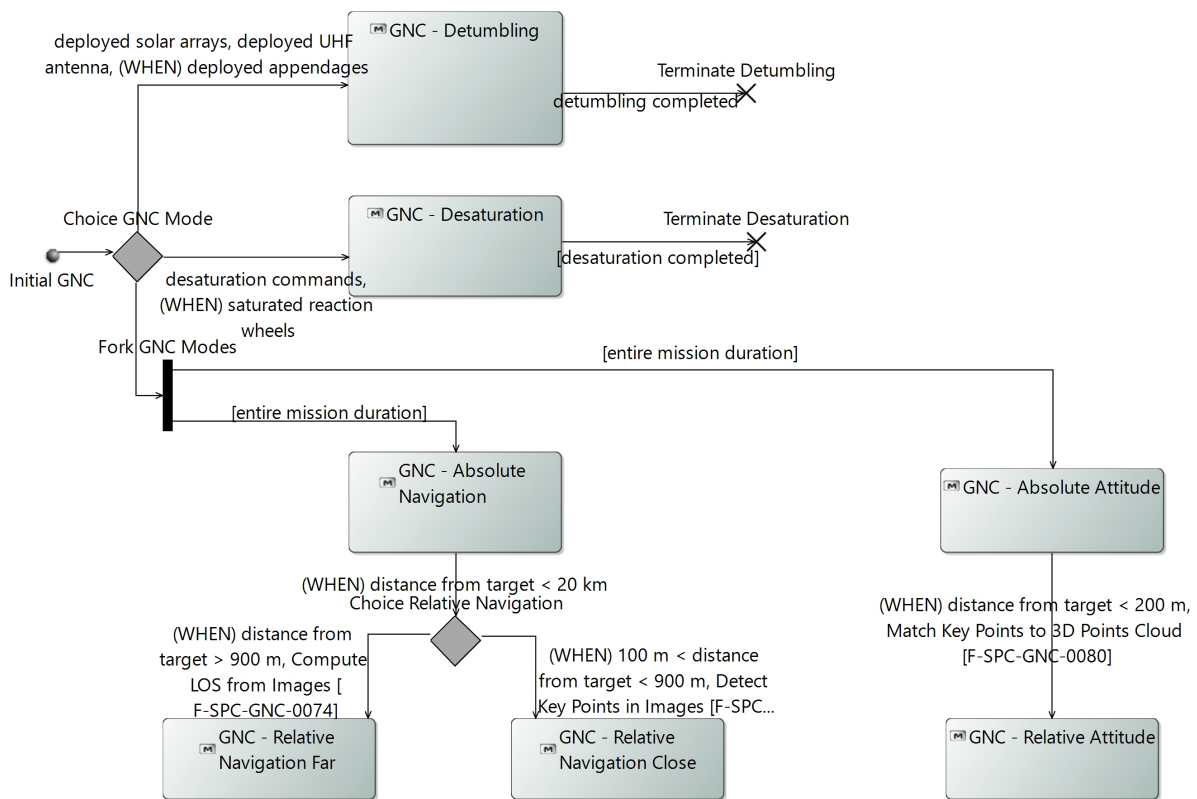


Figure 3.49: [M&S] GNC Modes

The attention is now shifted toward the diagram of Figure 3.51, showing the Propulsion subsystem Modes. The Transitions from one Mode to another reflect the mission phases, that will be better presented in the next section. The engine is firstly turned on to initiate the drifting toward the operative orbit, then the *Relative Orbit Transfer Mode* is used whenever needed during the Inspect Phase. Two other Modes characterize this subsystem, respectively the *Disposal Mode* and the *Collision Avoidance Mode*, the latter followed by a Terminate Pseudo State that is activated once the maneuver is executed.

Lastly, the TT&C subsystem Modes are presented in Figure 3.52. All the Transitions

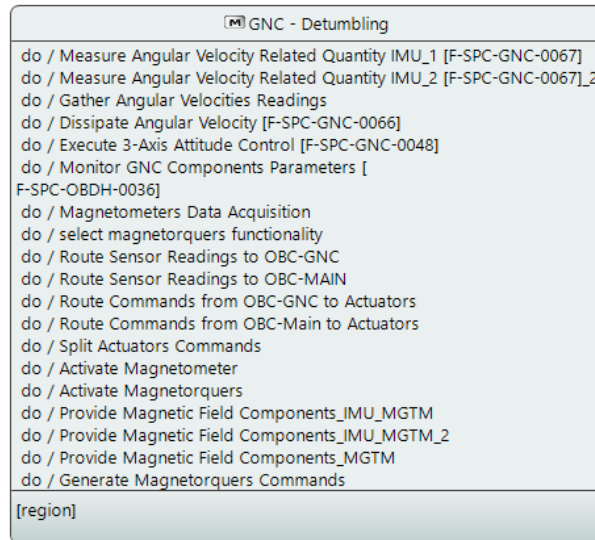


Figure 3.50: Expanded view of GNC Detumbling Mode

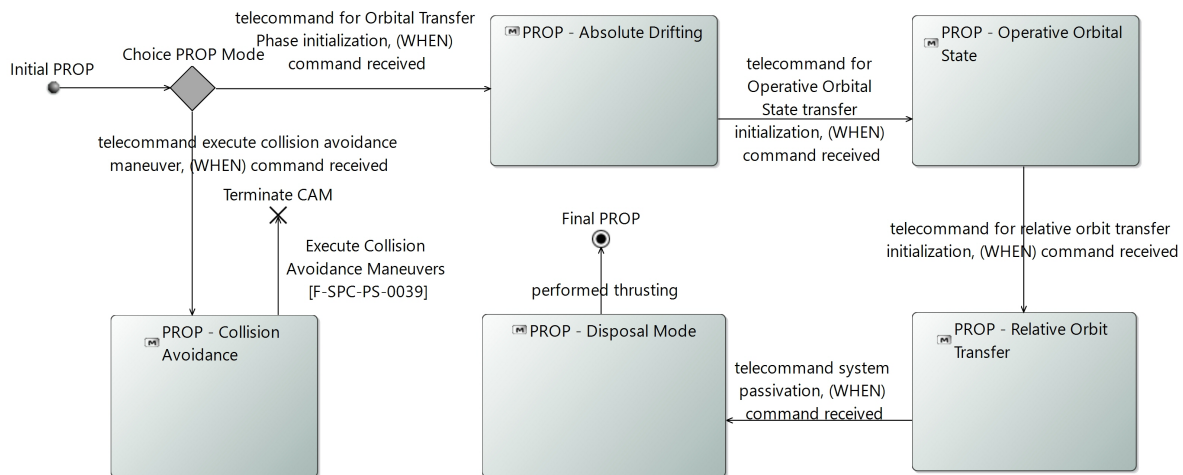


Figure 3.51: [M&S] PROPULSION Modes

make use of Functional Exchanges already presented in the model, indicating that once the expressed data is available, that Mode can be activated. This is a very simple way of modeling, that actually ignores the more complicated internal switches from one mode to the other. However from the modeling point of view this diagram is still very useful due to recurrent involvement of TT&C Modes in almost all system ones, as showed in Section 3.2.6.

No Modes are created for the other subsystems such as the EPS or the OBDR, since almost all their functionalities are activated during each system mode; therefore it was decided to include them in a so-called *System Base Mode* which provides all the basic Functions. The latter is actually a “ghost” Mode from the graphical point of view, indeed it is not reported in the following system Modes diagrams. However, since the System Base Mode provides vital system functionalities, it is always on from mission beginning up to the system passivation.

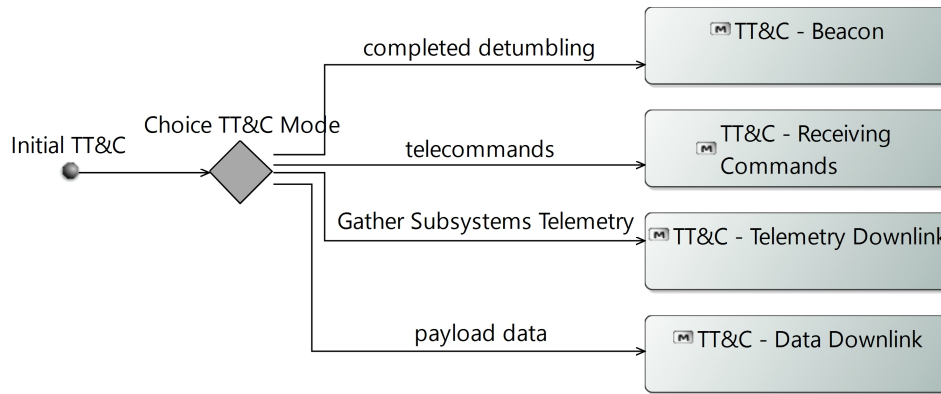


Figure 3.52: [M&S] TT&C Modes

System Modes

In the previous section, *simple modes* were adopted for subsystems, described by a number of Functions and exempt of sub-Modes. The concept of *composite modes* is introduced. These are Modes that contain one or more *regions*, each one having a set of subsystem Modes, called sub-Modes, as well as other Functions. A *region* is a top-level part of a State Machine intended as a container for the other Modes. This approach is very useful in the context of a CubeSat design, since it allows to easily define the system Modes starting from the subsystem ones that are active. It also drastically reduces the modeling time since otherwise, in case of non availability of subsystem Modes, the modeler would have to insert Functions one by one, with the risk of losing some of them. Each mission phase with the associated modes will be analyzed in the following and the exploitation of the work done at subsystem level will be discussed.

Following the chronological succession of mission phases, let's start with the *LEOP* one. Its State Machine diagram, showed in Figure 3.53, presents the Modes the system undergo from the CubeSat separation up to the phase conclusion. Some system Modes possess empty regions, because of the absence of proper subsystem Modes in the model. This is actually a modeling choice, and not a gap, since for such system Modes it is easier to directly define Functions at system level rather that creating non-recurrent subsystem Modes. To better clarify this concept, an example is reported in Figure 3.54, where the *SYSTEM - DEPLOYMENT* Mode contains tailored Functions for it.

The remaining phases with associated Modes are reported in Figures 3.55, 3.56 and 3.57. All of them share the *SAFE* and the *COLLISION AVOIDANCE* Modes, that may be needed in whatever phase of the mission. The latter is directly activated after telecommand reception, however the transition reported in diagrams does not exclude the autonomous CAM activation. It is clearly visible the exploitation of some recurrent subsystem Modes that fill the regions, enhancing an easier description of system Modes.

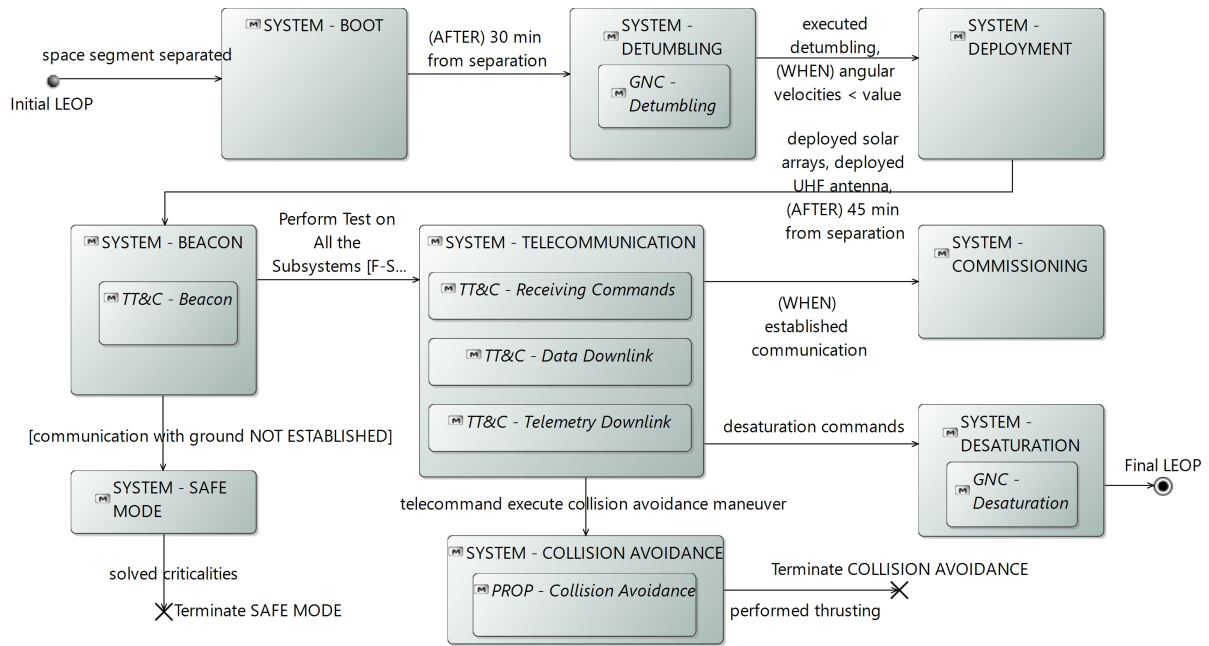


Figure 3.53: [M&S] SYSTEM Modes - LEOP Phase (1)

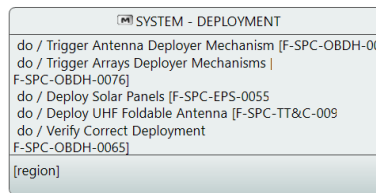


Figure 3.54: Expanded view of SYSTEM Deployment Mode

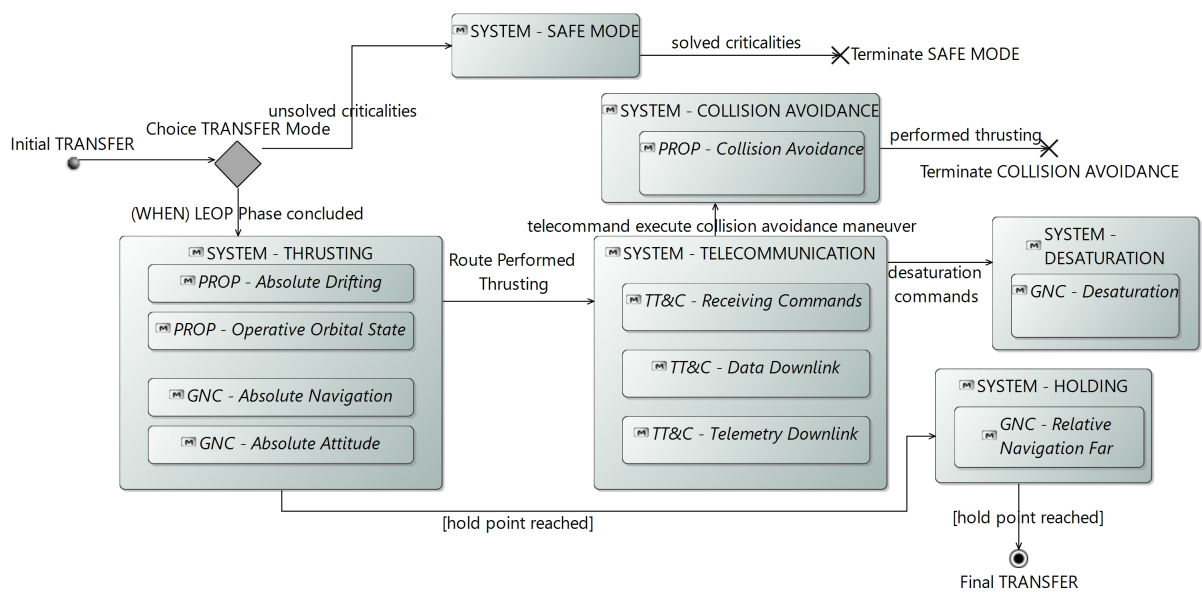


Figure 3.55: [M&S] SYSTEM Modes - TRANSFER Phase (2)

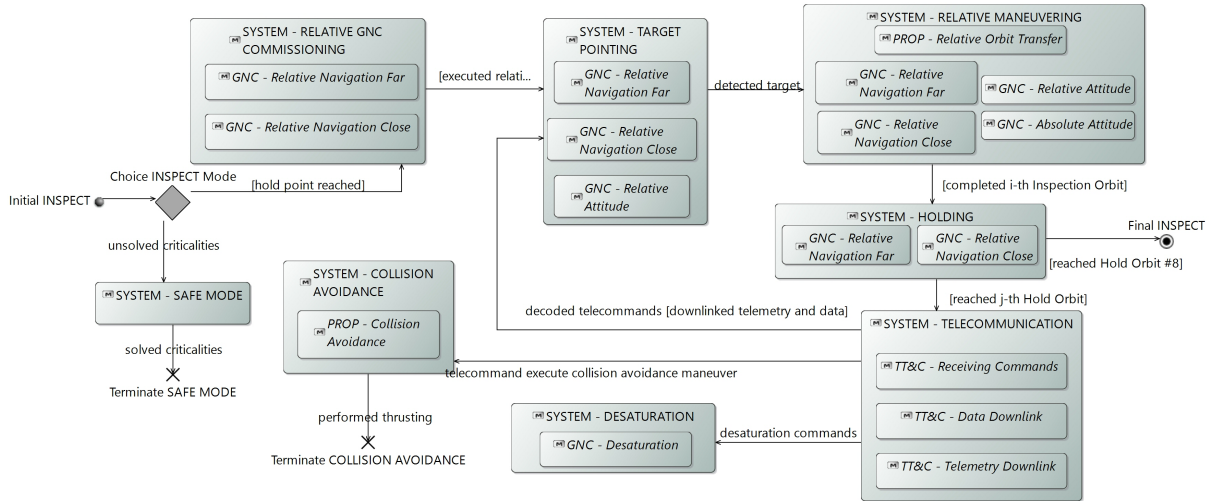


Figure 3.56: [M&S] SYSTEM Modes - INSPECT Phase (3)

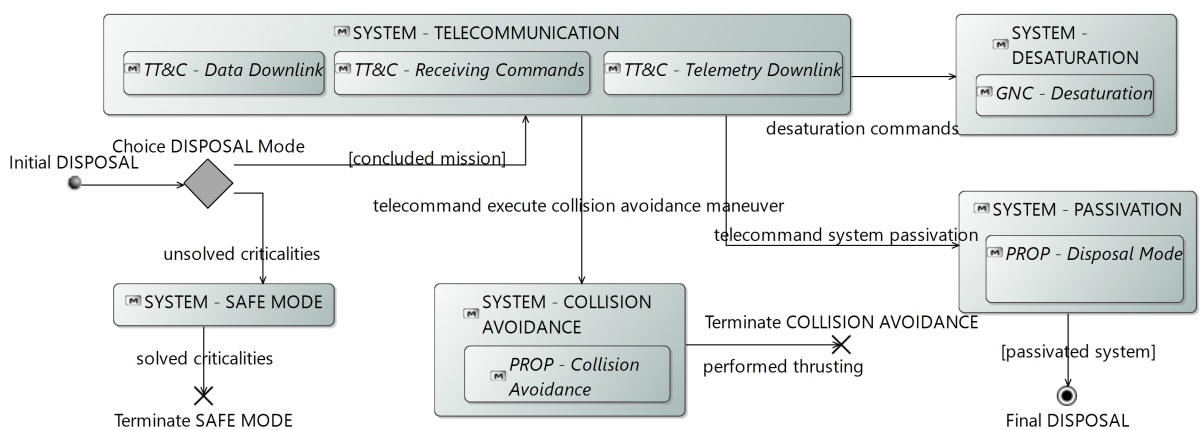


Figure 3.57: [M&S] SYSTEM Modes - DISPOSAL Phase (4)

3.2.7 Concept of Operations using Scenario Diagrams

In this section, all the work previously done related to the system architecture and its Modes is exploited in order to describe how the CubeSat will be operated during the different phases, with the goal of meeting the initial high level objectives. It is important to conduct this kind of analysis since an operational perspective allows to think more deeply about system needs, leading to a check out of the architecture. In a MBSE environment such as Capella, this is much more than effective. Indeed the ConOps are created using already modeled elements, such as Components, Functions and Functional Exchanges; therefore, whenever the modeler cannot find one that is satisfactory for a particular operation to be described, he/she is forced to go back and refine the model. This approach leads the way to a consistency analysis of the system architecture, opening new discussion points on the design. ConOps definition also stimulates requirements development since it forces to think about how the system might be used, understanding whether it can or cannot support the defined operations with the available requirements.

In Capella, the adopted diagrams to conduct such analysis are called *Scenario Diagrams*, inspired by the UML/SysML sequence diagrams. There are several types of Scenario Diagrams, all showing the vertical sequence of exchanged messages between elements, called lifelines. The diagrams available in Capella are differentiated basing on the lifelines, which can be Functions or Components/Actors (here called *Objects*), and the messages, which can be Functional Exchanges, Component Exchanges or Exchange Items. The ones adopted for this work are called Exchange Scenarios, in which the lifelines are Components/Actors and the sequence messages are Functional Exchanges or Component Exchanges. The elements that will appear in lifelines are mainly Functions and Modes.

In the following, a high-level timeline description of the activated Functions and Components for each mission phase will be provided. Some examples will also be presented focusing on certain mission aspects from the system functioning point of view, such as end-to-end communications or inspection maneuvers strategies. All the elements here used are taken from the Physical Architecture.

Before proceeding with the diagrams, a brief explanation of the adopted model elements is here provided. Together with Components/Actors reported as vertical lines (also called *Instance Roles* in Capella scenario diagrams, or *Lifelines* in UNL/SysML), Functions (here called *State Fragments*) and Functional Exchanges (here called *Sequence Messages*), some other concepts are involved, mostly inherited from UML/SysML. Since these diagrams have the time dimension, the *Duration* constraint is introduced, that is the time frame between two messages or Exchanges. Another concept encountered in Scenarios is the *Combined Fragment*, represented by a grey rectangle that covers Instance Roles. Each fragment contains a control structure called *Interaction Operator* that defines the type of logical condition to apply to the elements the fragment contains, typically Functions




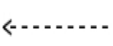
and Functional Exchanges, and can be split into *operands*, separated by dashed horizontal lines. Some operands are characterized by a *Guard*, which defines the condition to “activate” the fragment, reported as text in brackets at the top left of the operand. The adopted operators are here listed:

- LOOP: the fragment can be executed several times, and the guard condition states the iteration frequency (e.g. every day);
- OPT: the fragment is only executed if the condition provided is true (very similar to a “while” cycle);
- ALT: the fragment contains some alternative operands, only executing the one that contains the true condition (very similar to an “if” cycle);
- PAR: the fragment contains a number of operands that are executed in parallel, no Guard condition is present here
- STRICT: this interaction operator requires a strict sequencing (order) of the operands within the combined fragment.

Great use of combined fragments is done for this work since they allow to describe logic structures in a very compact and concise manner.

Some other important concepts concerning the type of messages involved for this work, inherited from UML/SysML sequence diagrams, are the creation of *self messages*, *messages with return* and *synchronous/asynchronous* messages. Table 3.2 provides their meaning and the correspondent graphical representations.

Table 3.2: Type of messages in Scenario Diagrams

	Self Message	A message an object sends to itself.
	Synchronous Message	A message that requires a response before the interaction can continue.
	Asynchronous Message	A messages that do not need a reply for interaction to continue.
	Return Message	Drawn with a dotted line pointing back to the original lifeline.

Mission Phases ConOps

In Capella, a Scenario describes the behavior of the system in the context of a particular Capability [55]. Therefore, a Capability should be present or created in order to attach the desired diagram. Once the diagram is created, Components or Actors are added and

the tool proposes only Functions that are already allocated to them. Therefore, whenever the modeler wants to include a Function that is allocated to a not present Component, the latter shall be added to the diagram to proceed. New Components and new Functions can clearly be created here too.

The first example reported in Figure 3.58 refers to the LEOP Phase and illustrates some Functions associated to the correspondent Components. Clearly, not all the Components actually working during this Phase are illustrated and a much more complicated diagram would be needed to provide a complete description of the contributions provided by each of them. Despite the instruments to do that are available since all system aspects have been modeled in the previous analysis, the aim here is to provide a high level view of the Phases in terms of operations, serving at the same time as demonstrative example of how Scenario Diagrams are used. The adoption of the *Duration* constraint is recursive in this diagram, not only to indicate the estimated duration of the whole phase, but also to highlight some constraints coming from the Cal Poly CubeSat Design Specification document [62], such as the minimum time after which all deployables shall wait to deploy after separation from the launcher (30 minutes), or the constraint on the elapsed time before the transmission of any signal (45 minutes).

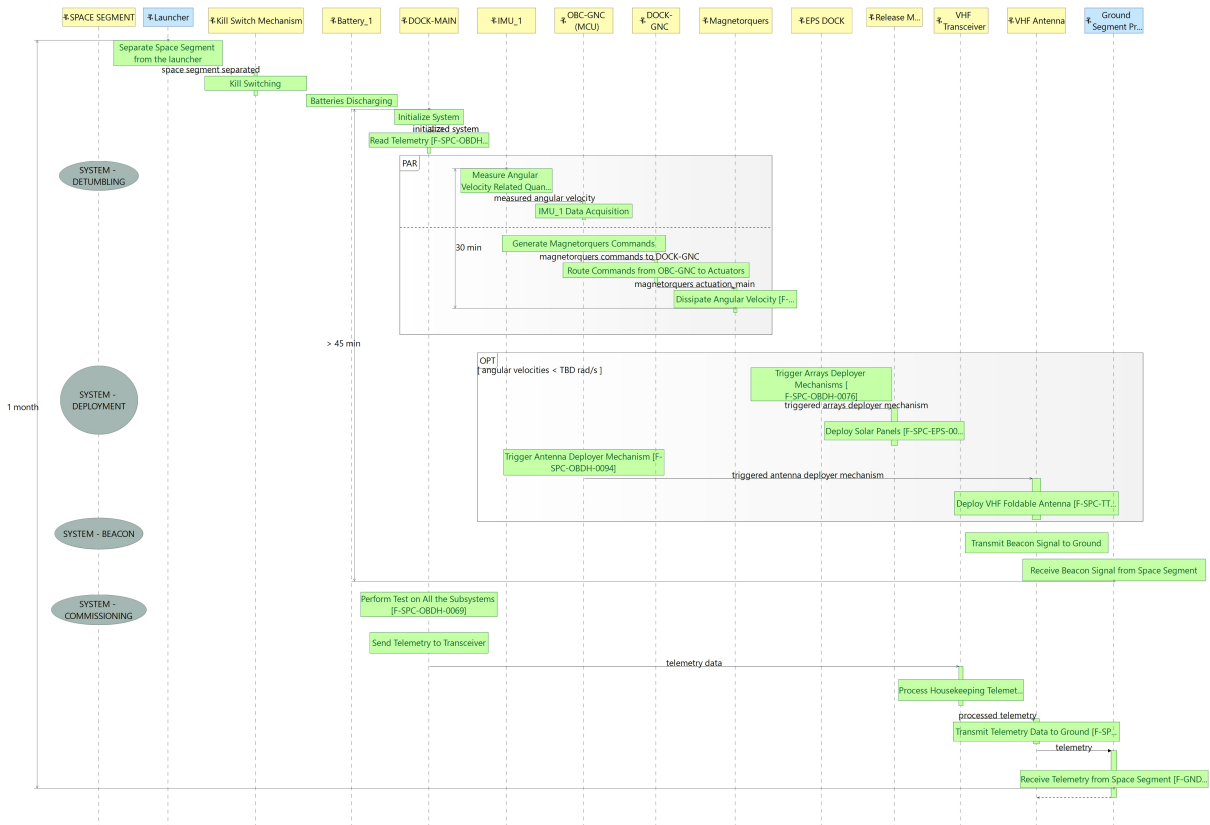


Figure 3.58: [PES] ConOps - LEOP Phase (1)

Figure 3.59 shows the TRANSFER Phase, initiated after the command reception. Requirements can be used to enrich Scenarios, such as the one here reported which refers

to the orbit prediction frequency. The use of synchronous messages with return can be noted in this diagram to ensure correct communication between the space and the ground segment.

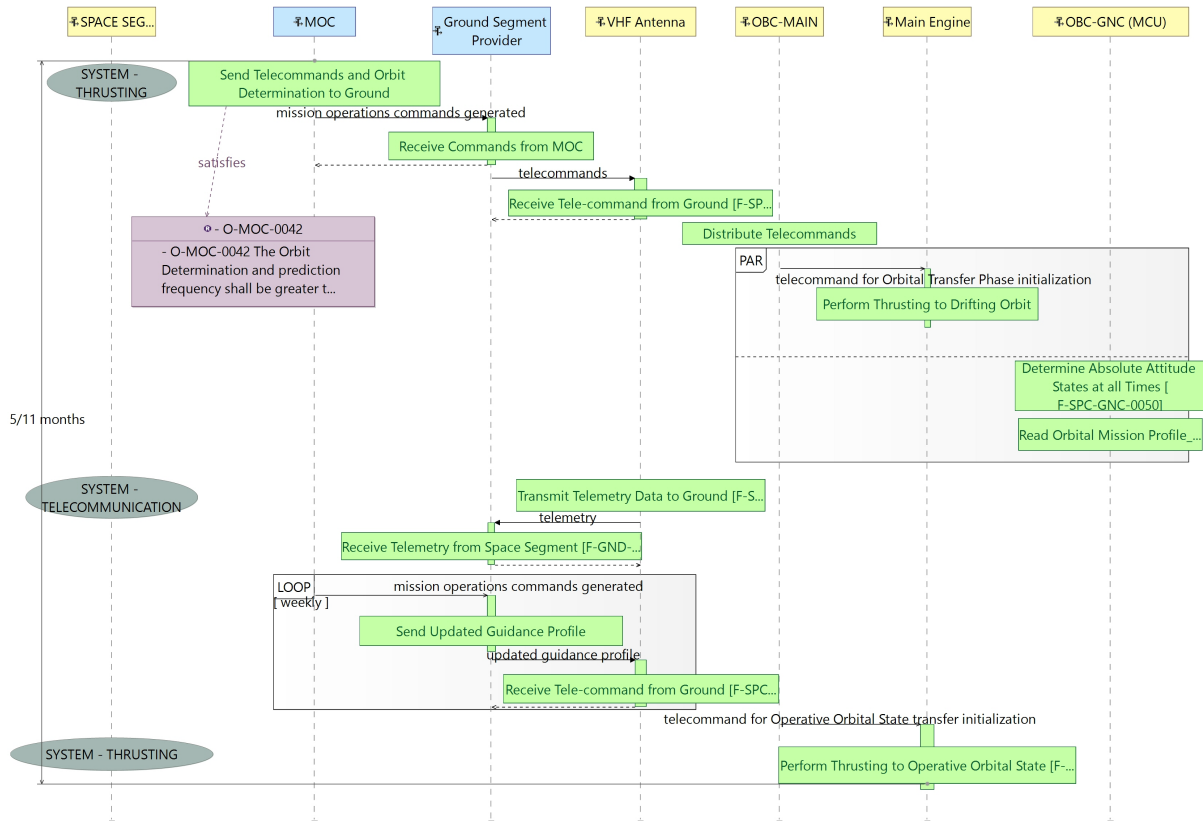


Figure 3.59: [PES] ConOps - TRANSFER Phase (2)

The INSPECT Phase is the most delicate and crucial for this mission, therefore it is important to provide a clear description of it. Section 3.1 briefly introduced the proximity operations as a succession of Inspection Orbits, where the images of the target are acquired, and Hold Orbits, in which the system health status is checked and data are downlinked. The Scenario Diagram in Figure 3.60 presents such high level description, alternating the orbits in a strict sequence. It is voluntarily left incomplete since the purpose here is not to deeply describe such operations, but to demonstrate the utility of Scenarios in the context of a complicated space mission such as the e.Inspector one.

The previous diagram also serves to better interpret, from an external point of view, the diagram in Figure 3.61 related to the INSPECT Phase. Great use of operators is done here to describe the execution of Functions; a big LOOP operator encloses all the lifelines since the contained Functions are repeated for each Inspection and Hold Orbits, the latter indicated by the correspondent System Holding Mode. Recalling the State Machine in Figure 3.49, the activation of GNC Modes, here highlighted in cyan, depends on the distance from the target. Absolute positioning and absolute attitude are the baseline

modes, the relative navigation begins when the distance is lower than 20 km, while the relative attitude and pose estimation at a distance lower than 200 m.

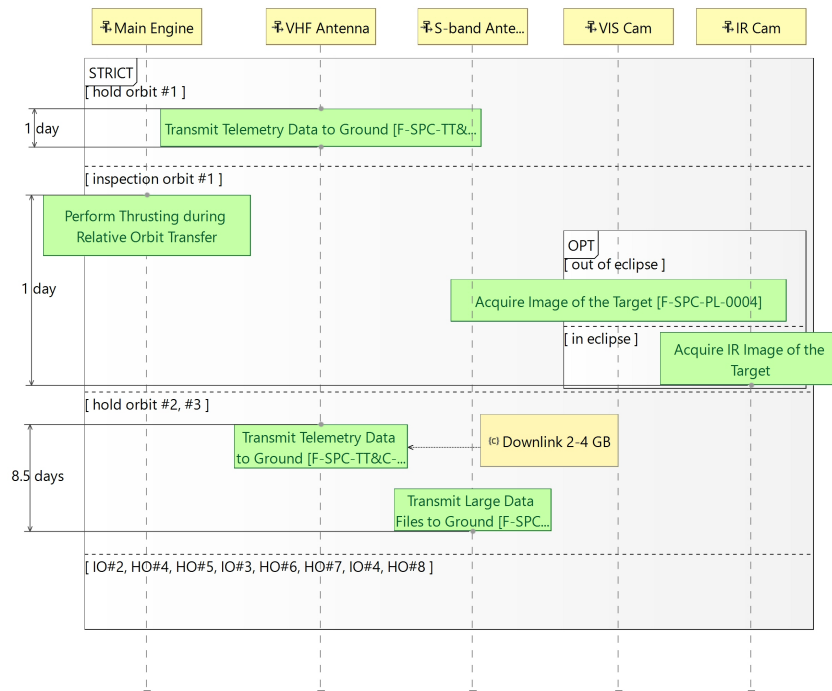


Figure 3.60: [PES] ConOps - Relative Operations

Lastly, the DISPOSAL Phase ConOps are reported in Figure 3.62. Before the disposal maneuver effectively can take place, the main engine shall move the CubeSat from the nearest Hold Orbit to the target, after the completion of the previous phase, up the the farther Holding Point. The OPT operator is used to indicate such condition, in which a telecommand with return message declares the beginning of the disposal maneuver followed by the system passivation.

Another use of Scenario Diagrams

Despite Scenario Diagrams are not actually intended to describe a very generic context, the previous section still results to be very useful in order to show the mission in its completeness and in a summarized way. However, they should be used to describe a particular use of the system that foresee a logic succession of functions and the temporal variable too, enclosing a lot of information that otherwise, in a document-centric approach, would be subject to misinterpretation. Two examples are reported in Figure 3.63, presenting some insights about the TT&C subsystem functioning together with some details such as the frequency at which the telemetry has to be downlinked and the inclusion of a requirement concerning the time window for large data transmission, and in Figure 3.64, about the EPS; the latter includes the time of eclipses and, more important, the way the subsystem jumps from the solar panels power generation to the batteries exploitation. During this work, it was noted how Scenarios development allowed the detection of some

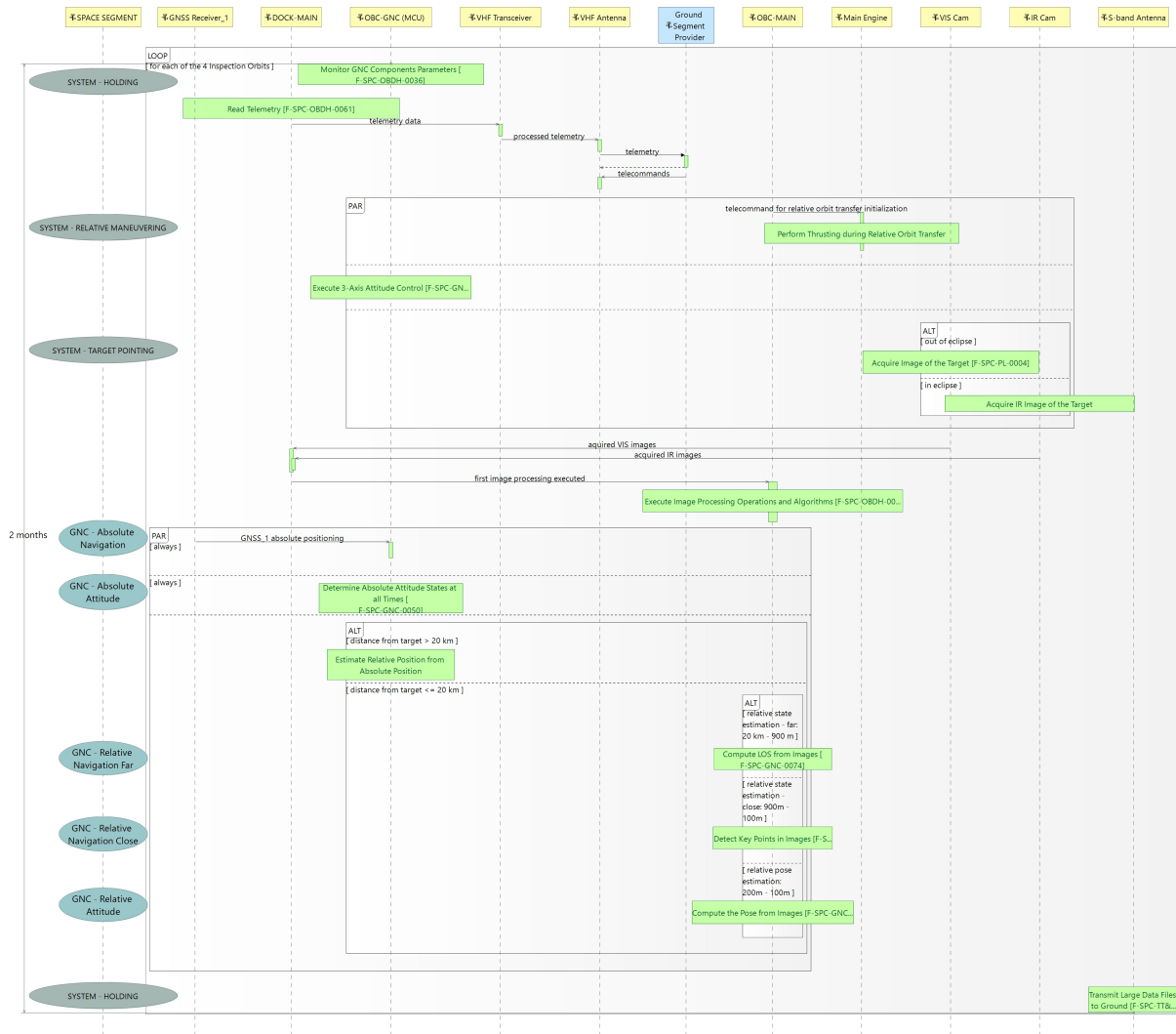


Figure 3.61: [PES] ConOps - INSPECT Phase (3)

“missing” functionalities in the model, forcing to think about solutions using the available architecture and components, from a temporal and logical point of view.

3.2.7. Concept of Operations using Scenario Diagrams

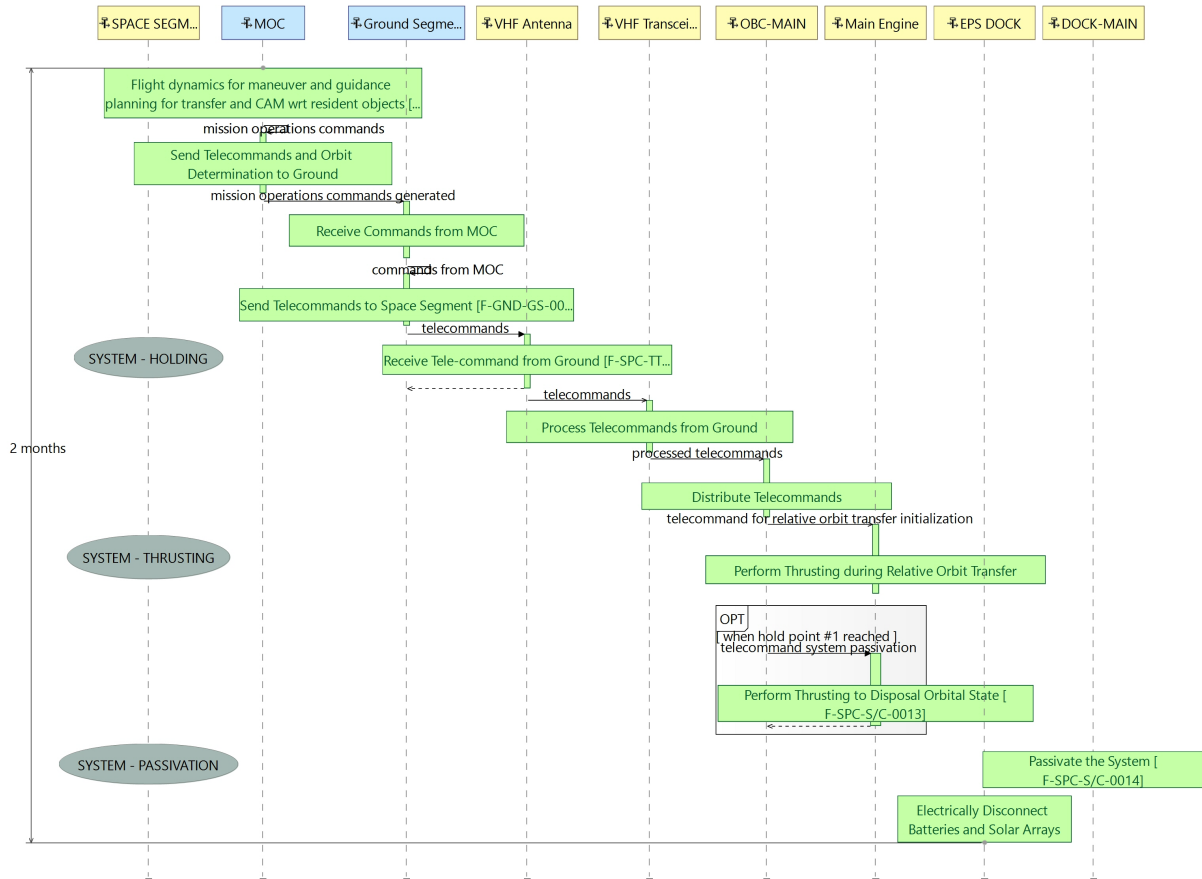


Figure 3.62: [PES] ConOps - DISPOSAL Phase (4)

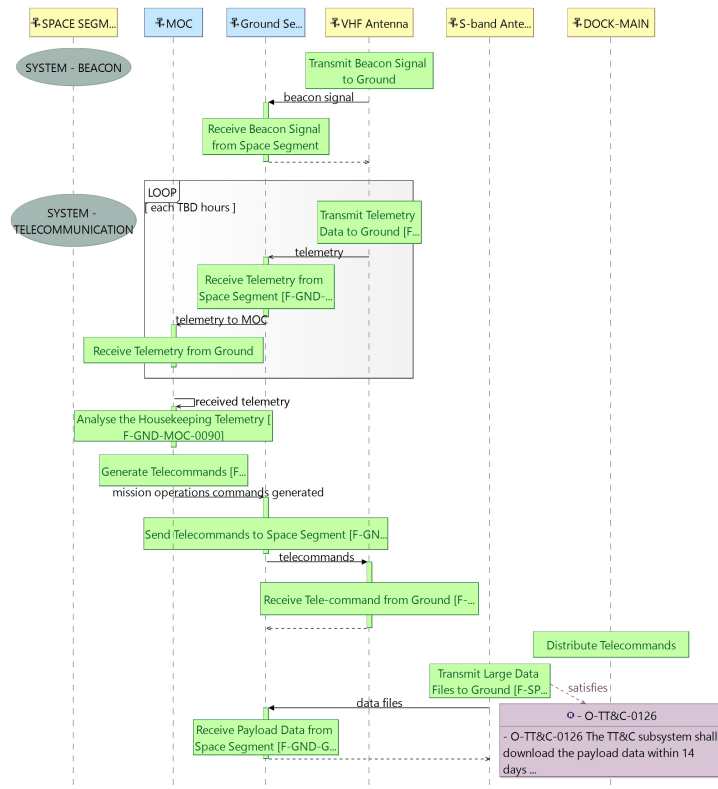


Figure 3.63: [PES] Provide Communication Services

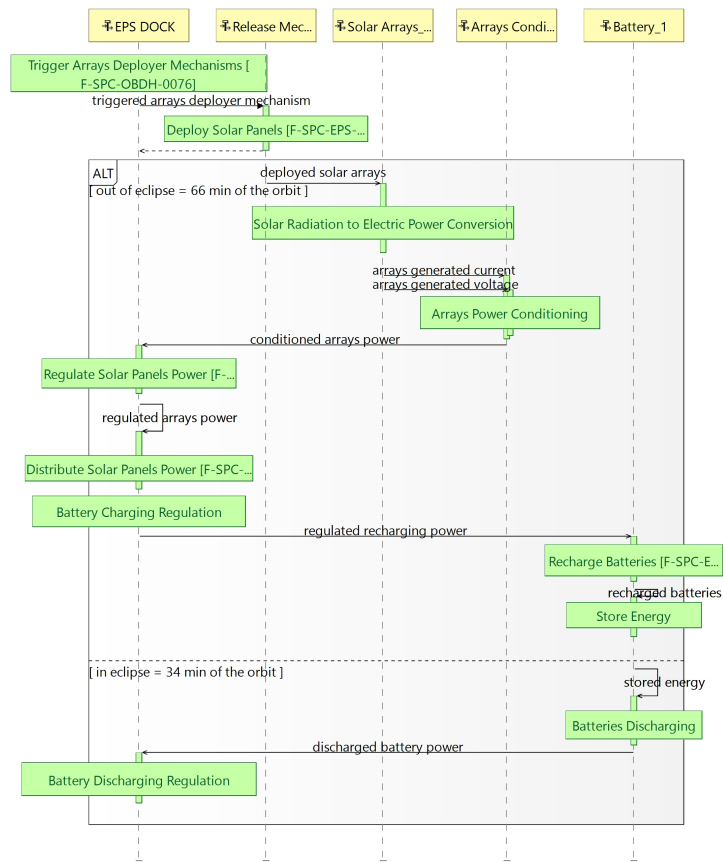


Figure 3.64: [PES] Provide Power Supply

3.2.8 AIV/AIT Plan Definition with Capella

This section proposes an MBSE approach for the Assembly, Integration, Verification and Testing activities in the context of the e.Inspector mission. The plan definition is available in the e.Inspector AIT/AIV Plan [20] and has been exploited for this work. The aim here is not to enter the details and the rationale behind the activities that will be encountered in the following diagrams, but the focus is shifted toward the proposed non document-centric approach using the Capella tool. With respect to the previous analyzed systems engineering practices, the ARCADIA method and the Capella tool do not propose a precise way of dealing with the AIV/AIT plan development. The modeler is free to build his/her own approach depending on the needs, therefore this is the way this section should be interpreted.

Verification and testing activities are defined since the Phase A of a space mission and continue to be refined during the entire product development. The classical approach exploits traceability links between textual requirements and tests procedures. However, relying just on them to derive test campaigns results in a lack of a detailed vision of the needs, also reducing the possibility to identify problems. This is due to the inability of textual requirements to cover all system aspects. As presented in the previous sections, an articulated model has been created with the aim of defining any functional and physical aspect of the system; the power of the here proposed framework, and in general of any MBSE approach which deals with test campaigns, resides in the guidance provided by the same model elements used as source of knowledge in the definition of the AIV/AIT plan.

An optimization of verification and validation strategy using Capella has been found in the paper by Bonnet et al. [10]; here, Functional Chains built within the global architecture are exploited to define some so-called *Requested Versions*, which represent test increments. Some *Developed Versions* are iteratively introduced in order to understand which tests previously defined can be performed according to the components availability at the time of their definition. This approach results very effective in de-stressing the test engineering since it allows to master the functional contributions of each component, providing very precise basis for a test campaign definition. However, it exploits already modeled elements without introducing the real test activities to be done and without entering the test procedures; it mostly represents a reference for them, which should be defined in a separate environment.

Instead of relying just on already modeled elements, the here proposed approach introduces new Functions, Functional Exchanges and Behavior Components which explicitly define the test activities. A practical example will be presented, in which tests conducted for the EPS are exploited as demonstrator. It is reminded that the approach must be intended as a prototype proposal, since it sometimes results to be in contrast with some of the ARCADIA pillars. However, it is recalled that ARCADIA actually does not propose

a method for managing test activities within the model, therefore the aforementioned contrasts can be ignored while focusing on the gained benefits.

AIV/AIT plan: EPS case study

The approach is developed within the Physical Architecture, therefore any model element in the context of the AIV/AIT plan is part of it and cannot be found in upper levels. This is a decision that directly comes from the need of working with elements which represent real physical Components that will constitute the system and that will be integrated and tested, respectively exploiting Physical Links and Physical Functions which describe them in the model.

The first step consists in defining a PAB diagram for the subsystem, here the EPS, such as the one in Figure 3.65 (similar diagrams for the rest of the subsystems are reported in Appendix A.4). The Actor in charge of executing the tests, in this case POLIMI, carries some Behavior Components, each one called with the subsystem name, the type of model used and the name of the Physical Component to be tested (e.g. *EPS PFM – Solar Arrays*, where PFM stands for Proto Flight Model). These Components have allocated a number of Physical Functions, expressly created, which explicitly state the activities to be performed on that Component. For example, a Functional Test shall be performed on the Proto Flight Model of Solar Arrays. These high level test blocks, in the form of Physical Functions, provide a global view of the tests to be performed on the subsystem. They are connected by Functional Exchanges which just indicate their logical sequencing; in order to be coherent with the ARCADIA method, whenever exchanges are established between functions belonging to different Components, Functional Exchanges are allocated to proper Component Exchanges.

The first contradiction can be highlighted, that is the exploitation of model elements which, according to the ARCADIA method, should be part of the system while here are treated as if they are allocated to an external operator. This is the case of the Physical Functions in Figure 3.65: since they are allocated to Behavior Components, Capella automatically assigns them to the system despite the Components are deployed inside the Actor (this is why they have a green stamp and not a blue one). To solve this issue, Behavior Components should be neglected leaving the Functions “fluctuating” inside the Actor; however, at that point, a coding should be defined to distinguish the various Components, weighting down the blocks and making their reading difficult. Therefore, it was decided to ignore this issue, keeping the Functions inside Behavior Components and with the green stamp. In the Functional Breakdown they will easily be distinguished from the remaining system Physical Functions thanks to the creation of a dedicated test activities branch.

The previous diagram is a sort of navigation menu, a starting point linking to other diagrams which better detail the activities. Two links can be noted: the first one is related to the Physical Function *Functional Tests of SA* as the icon in the bottom right of it suggests

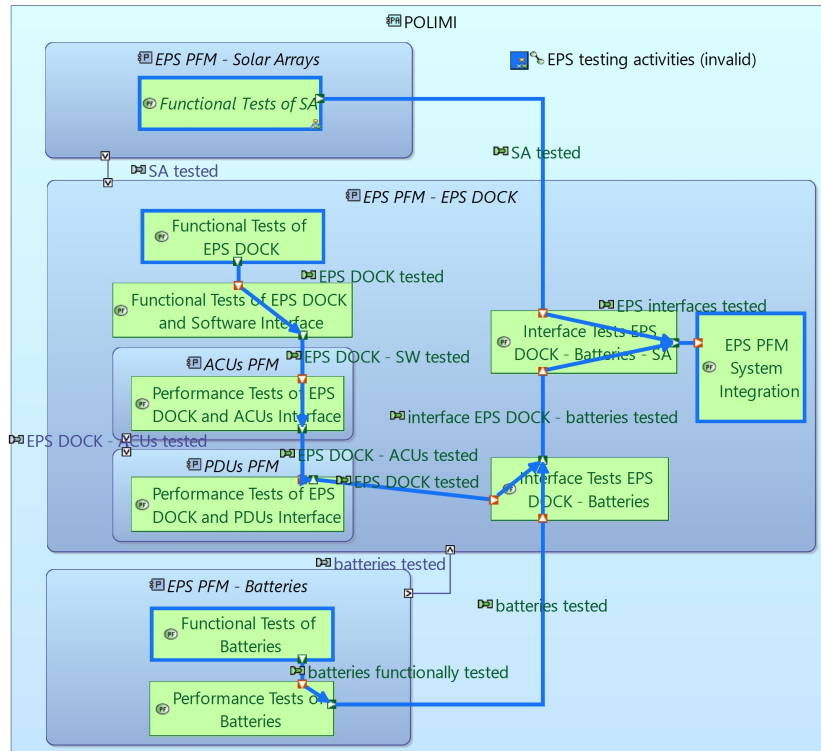


Figure 3.65: [PAB] AIV/AIT EPS - Overall Plan

(the italics is automatically used by Capella whenever a Function hosts sub-Function), the second one is a Functional Chain Description diagram associated to the highlighted chain.

Let's focus on the first link. Right clicking of the Function, the tool proposes to open an existing diagram allocated to that Function, as Figure 3.66 shows.

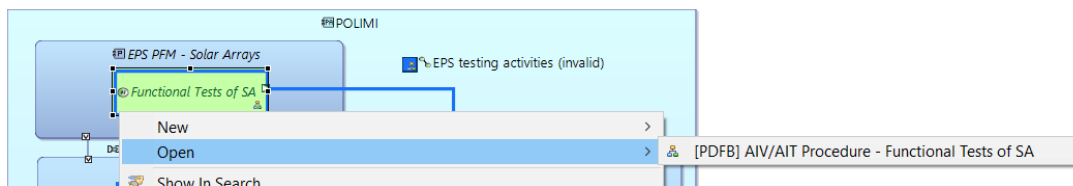


Figure 3.66: Link to [PDFB] - Functional Tests of SA

As the name suggests, it is a diagram showing the procedures that must be done to accomplish the upper activity, reported in Figure 3.67. Having one or more diagrams like that for each activity of the AIV/AIT plan allow systems engineers to have a complete view of all the procedures to be performed, all embedded in the same workspace. The Exchanges here indicate pure logical sequencing, however is clearly possible to report them in a Scenario Diagram to also catch the temporal dimension. Lastly, each Function has a dedicated sheet in which the progress status can be set, as well as comments or open discussions (Figure 3.68); in a team environment it allows to drastically reduce the effort spent in communicating, using these diagrams as single source of truth. Eventually, a graphical coding can be set in a team to distinguish among completed/in progress/not

completed activities.

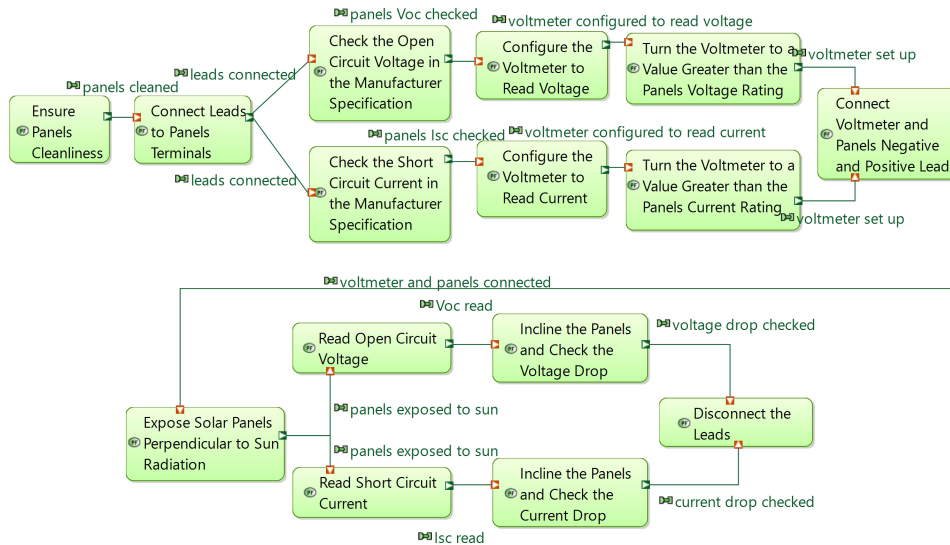


Figure 3.67: [PDFB] AIV/AIT Procedures - Functional Tests of SA

Progress Status :	
Review :	REWORK_NECESSARY TO_BE_REVIEWED TO_BE_DISCUSSED REVIEWED_OK DRAFT UNDER_REWORK

Figure 3.68: AIV/AIT Functions - Progress Status Sheet

Going back to the diagram in Figure 3.65, the second link previously mentioned is analyzed. Right clicking on the blue stamp Functional Chain *EPS testing activities*, it is possible to open a Physical Functional Chain Diagram, Figure 3.69, which reports all the activities that are part of the chain. All Exchanges between Functions are kept unchanged, however here the concept of *Sequence Link* is introduced. It is graphically represented by a dotted line connecting two Functions and is used to introduce the temporal dimension. Whenever a Sequence Link connects a function F1 (source) to a function F2 (sink), it means that F2 starts after F1 does. Here all these links appear together with a Functional Exchange, however it is also possible to have independent Sequence Links. This is an alternative option of dealing with temporal sequences instead of using Scenarios.

Let's focus now on two elements encountered for the first time in this work: the dark green blocks with the Functional Chain icon on the top left and the yellow blocks with the {c} icon. They are respectively Functional Chains expressed in a compact form, here exploited to create a bridge between the test activities and PA elements, and *Constraint* elements, discussed in the following. The proposed approach is very simple: some Functional Chains are already defined within the model, highlighting certain functional aspects

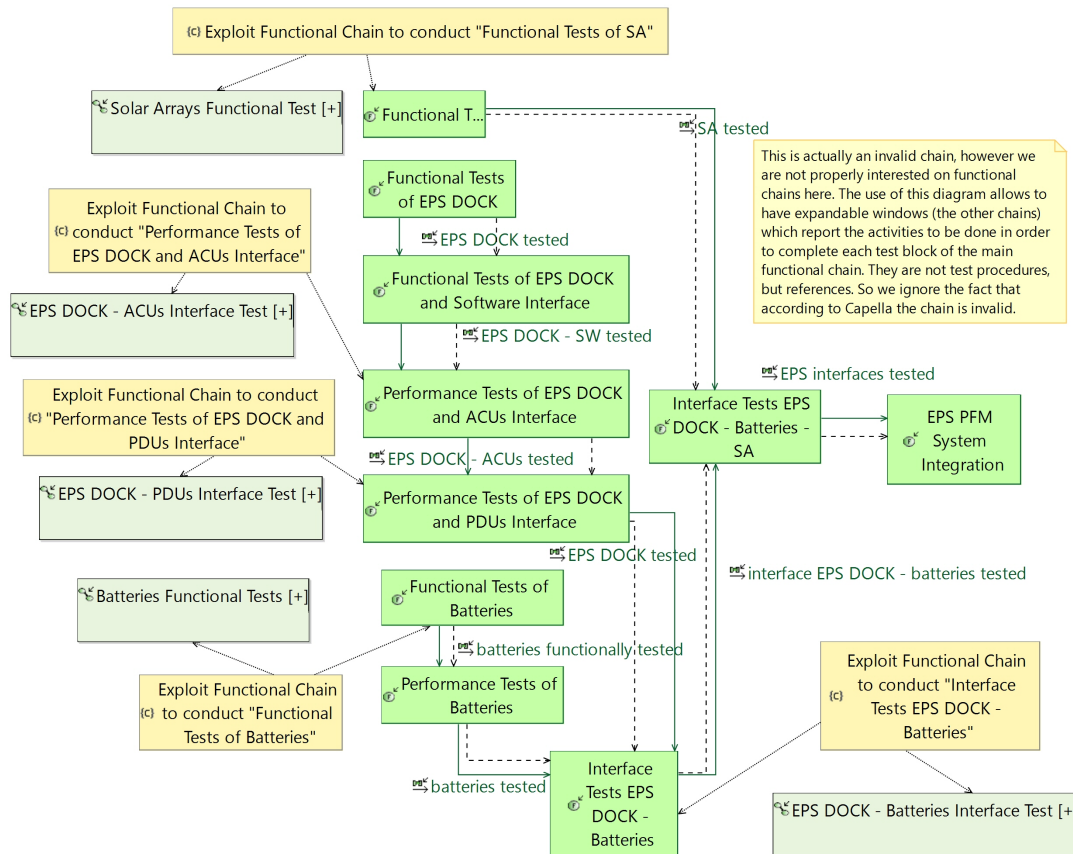


Figure 3.69: [PFCD] AIV/AIT - EPS testing activities

of each subsystem or component. Since all test activities necessarily refer to the system functional or interface analysis, whenever it is decided to conduct a certain test, systems engineers can exploit the chains reported in dark green blocks which contain such functional aspects of the system. Any Functional Chain can be added to this diagram and, assuming that the desired component/subsystem functionalities have been properly modeled and a clear description is present within the model, the probability of committing errors during the test campaign can be drastically reduced. This is also very useful since during the test activities problems typically arise and some changes have to be applied to the system; in this case, engineers can go back to the model, refine the analysis and finally exploit the new Functional Chains for a further check. This is what the green blocks show, a compact form of Functional Chains which serves as reference for each aspect of the campaign; indeed they can be expanded in the same diagram whenever needed for consulting, as reported in the example of Figure 3.70 for the chain *Performance Tests of EPS DOCK and ACUs Interface*. More than one Functional Chain can clearly be associated to a test activity, therefore the Constraint element is used to explicitly “allocate” them to activities. Actually this is not a formal allocation, but more a graphical one used within this approach.

The one presented is a simple way of dealing with AIV/AIT activities exploiting at the same time the already modeled elements. To do that, another rule of Capella has

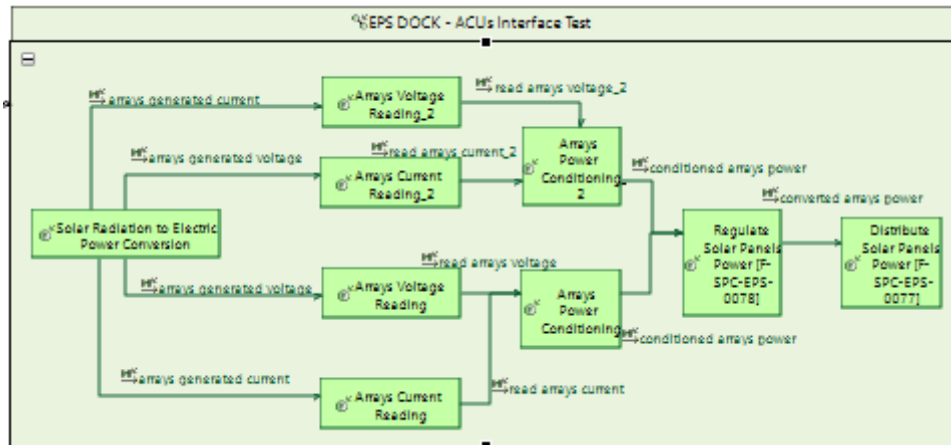


Figure 3.70: AIV/AIT EPS DOCK - ACUs Interface Tests Expanded View

been violated. In the diagram of Figure 3.65, the Functional Chain results to be *invalid*. This happens because in the diagram of Figure 3.69, the chains used as reference are not connected to the Functions representing the test activities, resulting in a broken overall chain. However, being totally aware of that, this aspect is ignored since the benefit of having all these information in one single diagram is worth. Some upgrades can be of course introduced in the future in order to deal with these contradictions, by creating a dedicated “AIV/AIT Viewpoint”.

Summing up, three diagrams have been involved in this approach:

- a PAB diagram (Figure 3.65) for each subsystem to define the test activities related to them;
- a PDFB diagram (Figure 3.67) for each test activity to define the procedures related to them;
- one or more PFCD diagram (Figure 3.69) for each subsystem to have a direct reference to model elements.

The advantage of dealing with testing and verification activities within the same environment in which the system was modeled resides in the possibility to exploit all the knowledge and information embedded in the model. So, for example, in the context of system integration, Physical Links can be consulted to check the correctness of the integration plan serving also as base for its definition. The presented approach is demonstrative and experimental and, as said, does not include the complete set of activities to be conducted, which strongly depend on the next mission design phases.

Chapter 4

Decision Making Tool for Small Satellites Architectures Generation

Nothing is more difficult, and therefore more precious, than to be able to decide.

NAPOLEON BONAPARTE

MBSE lacks intelligent support that could strongly help in addressing the most suitable architecture in line with the system functionalities, speeding up the alternatives selection process in a Phase 0 design. This would be particularly useful during the preliminary design phases, in which the almost infinite design choices are skimmed by the only system engineers knowledge, who may miss some solutions. A newly approach conceived to solve this issue is described in this chapter in the form of a decision-making tool prototype in support of the previously described MBSE approach. The problem and the method are firstly presented in Section 4.1, followed by the algorithm explanation in Section 4.2 and a practical demonstrative simulation in Section 4.3.

4.1 Statement of the Problem and Methodology

The approach starts from the definition of one or more high level *functionalities*, that can be formalized in the Capella environment, describing some expected system behaviors and characterized by a list of items or, as called from here on, *markers*. The tool embeds a number of *decisions* at various levels, each one containing some *alternatives* which are also described by the same aforementioned markers. Decisions are intended as a sort of “level identifiers” or “alternatives containers” and do not have markers. The set of decisions at different levels form a combination tree, where the alternatives selection is driven by their ability to satisfy the functionalities throughout a sort of matching algorithm between the markers, followed by a decision-making problem resolution for the architectures ranking.

4.1.1 Inputs from the User

The method is here formalized introducing m desired functionalities representing the main user input to the tool. As said, each functionality is represented by a vector containing n markers, called *Input Functionality Vector*. The *Input Functionality Matrix* in Equation 4.1.1 is then created placing the *IFVs* in its columns:

$$\mathbf{IFM} = \left[\mathbf{IFV}_1, \dots, \mathbf{IFV}_j, \dots, \mathbf{IFV}_m \right] = \begin{bmatrix} f_{11} & \cdots & f_{1j} & \cdots & f_{1m} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ f_{21} & \cdots & f_{ij} & \cdots & f_{2m} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ f_{n1} & \cdots & f_{nj} & \cdots & f_{nm} \end{bmatrix} \quad (4.1)$$

Input markers values f_{ij} can have boolean values (1 if the functionality is characterized by that marker, 0 if not), or can be assigned a number from 2 to 4 which indicates the importance of that marker for the functionality. Higher the value, more important the i -th marker for the j -th functionality. As presented in the following, the tool autonomously maps these inputs providing as output a coherent architecture.

Another input is asked to the user, called *Functionalities Temporal Concurrency Matrix*. It is an $[m \times m]$ matrix having value 1 if two functionalities are required at the same time, 0 if not. It will be used by the tool to exclude those alternatives which satisfy a functionality but are completely unsuitable for a contemporary one, compromising it. As example, in Equation 4.1.1 the functionalities #1 and #2 are contemporary.

$$\mathbf{F} = \begin{bmatrix} 0 & 1 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix} \quad (4.2)$$

Lastly, as the relative weights assigned to the functionalities are computed using the Analytic Hierarchy Process (Section 4.2.1), a pairwise matrix with the relative importance between functionalities is required. Since it is not so straightforward to get consistent matrices, in particular as the number of functionalities increases, an algorithm has been developed for their automatic generation, allowing to save time in compiling the matrix and at the same time ensuring its consistency. Therefore, the last input asked to the user is not a user-built pairwise matrix, but the following quantities:

- \mathbf{v}_{imp} = *Vector Importance*: row vector $[1 \times m]$ where the m functionalities are ordered from the most important to the least one. Value 0 if i -th and the $(i+1)$ -th functionalities are equally important, 1 if the i -th is more important than the $(i+1)$ -th. Value 0 shall be put in the last cell.

- $s = \text{Sparsity Factor}$, scalar ($0 < s < 1$) typically equal to 1. Higher s higher differences between the criteria (functionalities) will be obtained once the pairwise matrix is given to the AHP. Lower s , lower differences.

The algorithm firstly computes a so-called *jump* value, defined as the minimum difference between two values in the pairwise matrix. Without the jump value, if the number of functionalities given as input is higher than 9, in the pairwise matrix there would be relative importance numbers exceeding the usual scale of the AHP, which goes from 1 to 9. Algorithm 1 shows the steps for the computation of the pairwise matrix. It is reminded that the user is left free to opt for his/her own manually compiled matrix; on that case attention shall be paid toward the Consistency Ratio, which must not exceed the value 0.1.

Algorithm 1: Automatic pairwise matrix building.

Input: v_{imp} = Vector Importance, s = Sparsity Factor
Output: P_{fun} = Pairwise Matrix of Functionalities

```

// Begin
 $N_{fun}$  = Number of Functionalities
// Compute jump value
if  $N_{fun} \leq 9$  then
  |  $jump = s$ 
else
  |  $jump = \frac{8}{N_{fun}-1} \cdot s$ 
end
// Compute Pairwise Matrix
for  $i = 1 \rightarrow N_{fun}$  do
  | for  $j = 1 \rightarrow N_{fun}$  do
    | | if  $i = j$  then
    | | |  $P_{fun}(i, j) = 1$ 
    | | | else if  $i < j$  then
    | | | |  $P_{fun}(i, j) = 1 + (\sum_{k=i}^{j-1} v_{imp}(k)) \cdot jump$ 
    | | | |  $P_{fun}(j, i) = \frac{1}{P_{fun}(i, j)}$ 
    | | end
  | end
end
// End

```

4.1.2 Tool Embedded Decision Tree

A number of decisions have to be “installed” in the tool. Decisions can be hierarchically divided into different levels; an example related to the space field is to consider as first level decision the stabilization technique for the GNC subsystem, while as second level

nested into the upper one the sensors or actuators selection. It is recalled that each decision belonging to whatever level is characterized by a given set of alternatives.

Let consider l decisions belonging to the first level; each w -th decision contains p_w alternatives and each alternative is in turn described by a predetermined vector of n elements. These vectors are embedded in the code (they are a predetermined setting) and compiled according to the following rules:

- value 0 (boolean) if the i -th marker does not characterize the k_w -th alternative;
- value 1 (boolean) if the i -th marker satisfies the k_w -th alternative;
- value 2 (scale) if the i -th marker weakly accomplishes the k_w -th alternative;
- value 3 (scale) if the i -th marker well accomplishes the k_w -th alternative;
- value 4 (scale) if the i -th marker greatly accomplishes the k_w -th alternative.

The presented coding differentiates the boolean values from the scaled ones; the reason behind this choice will be clarified in the Section 4.2.2. Each k_w -th alternative belonging to the w -th first level decision is characterized by the following vector of markers:

$$\mathbf{a}_{k_w} = \begin{bmatrix} m_{1k_w} \\ \vdots \\ m_{ik_w} \\ \vdots \\ m_{nk_w} \end{bmatrix} \quad (4.3)$$

and each of the l first level decision by the following array, in which the number of columns p_w (that is the number of alternatives for that decision) is variable as it depends on the w -th decision:

$$D\mathbf{1}_w = \begin{bmatrix} m_{11_w} & \cdots & m_{1k_w} & \cdots & m_{1p_w} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ m_{i1_w} & \cdots & m_{ik_w} & \cdots & m_{ip_w} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ m_{n1_w} & \cdots & m_{nk_w} & \cdots & m_{np_w} \end{bmatrix} \quad (4.4)$$

The second level of decisions is nested into the first one, meaning that each alternative of each k_w -th first level alternative contains a set of d_{k_w} second level decisions, the latter having in turn their own total number of alternatives that is different for each of them. An array characterizes each second level decision:

$$D\mathbf{2}_{h_{k_w}} = \begin{bmatrix} s_{11_{h_{k_w}}} & \cdots & s_{1g_{h_{k_w}}} & \cdots & s_{1q_{h_{k_w}}} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ s_{i1_{h_{k_w}}} & \cdots & s_{ig_{h_{k_w}}} & \cdots & s_{iq_{h_{k_w}}} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ s_{n1_{h_{k_w}}} & \cdots & s_{ng_{h_{k_w}}} & \cdots & s_{nq_{h_{k_w}}} \end{bmatrix} \quad (4.5)$$

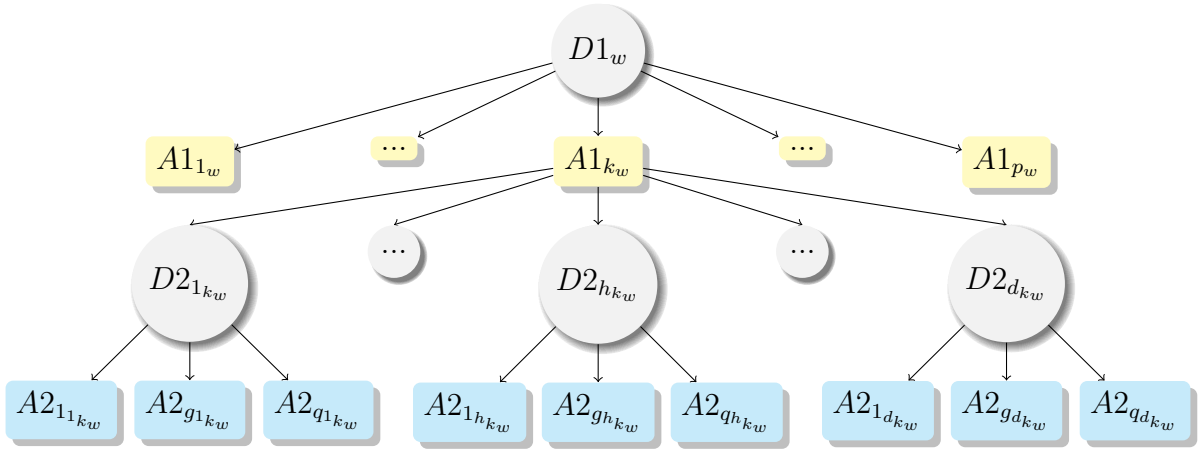


Figure 4.1: Decision Tree Structure

The notation burdening is due to the fact that the total number of first level alternatives is different among the first level decisions, the total number of second level decisions is different among the first level alternatives and the total number of second level alternatives is different among the second level decisions, while the n number of markers is the same for all of them. To better clarify the adopted indexes, Table 4.1 reports a legend of symbols while Figure 4.1 illustrate the structure of the decision tree: gray bubbles are decisions, each one containing a number of alternatives (yellow is used for first level alternatives, blue for second level ones).

Table 4.1: Indexes involved in the decision-making problem

	Index	Total Number
Markers	i	n
Functionalities	j	m
First Level Decisions D1	w	l
First Level Alternatives A1	k_w	p_w
Second Level Decisions D2	h_{k_w}	d_{k_w}
Second Level Alternatives A2	$g_{h_{k_w}}$	$q_{h_{k_w}}$

Having the *IFM* and the set of decisions with their alternatives, the tool has to identify the most suitable architecture, composed by at least one alternative for each decision, which satisfies the functionalities according to the markers mapping. The detailed procedure and the algorithm are presented step by step in the following section.

4.2 The Algorithm Explained

Each alternative of the decision tree has a number of “enlightened” markers. The purpose of the tool is to select the set of alternatives which guarantee the maximum coverage of the

markers asked by the functionalities. Firstly all the combinations of alternatives belonging to the first level decisions are evaluated, leading to the ranking of a number of first level architectures. For each of them, the second level architectures are computed and ranked according to some rules presented in the following sections. A scheme of the algorithm is provided in Figure 4.2. The algorithm is developed using MATLAB and great use of cell arrays is done to agglomerate the alternatives within the different decisions, the latter intended as a sort of “containers” which mark the level they belong.

4.2.1 Some Ingredients of the Algorithm: MCDM Methods

Several Multi-Criteria Decision Making methods are adopted within the tool algorithm, therefore a review of them is furnished in the following. The Analytic Hierarchy Process is exploited to compute the weights of the functionalities, as there can exist some more important or delicate functionalities, typically with a central role in the mission, to which the priority is given in the design context. Then, some other MCDM approaches have been investigated to compute the Performance Scores from a decision matrix.

The Analytic Hierarchy Process

The Analytic Hierarchy Process is a widespread technique proposed in 1980 by Saaty [59] based on decomposing complex MCDM problems into hierarchies which relates the alternatives. For this thesis work the AHP is engaged in the computation of functionalities weights, which are the intended as decision criteria for the alternatives selection.

Once the hierarchical structure is developed and the objectives to be ranked identified, a so-called Pairwise Comparison Matrix is compiled attributing the relative importance values to pairs of objectives. A scale from 1 to 9 is introduced to do that, reported in Table 4.2. An example of pairwise matrix is presented in Table 4.3; according to the scale, the attribute #1 is much more important with respect to the attribute #2. Reciprocal values are adopted to indicate that the second attribute of the pair is more important with respect to the first one, so the attribute #3 is very much more important with respect to the attribute #1 in this case. Once compiled the pairwise matrix, the following steps are followed:

1. The Normalized Pairwise Matrix is obtained dividing each cell per the sum of the column values where the cell belongs;
2. The Criteria Weight is computed by averaging each row;
3. Multiply each value of not normalized matrix columns per the Criteria Weights of the row they belong. They represents the ranking values of the attributes.

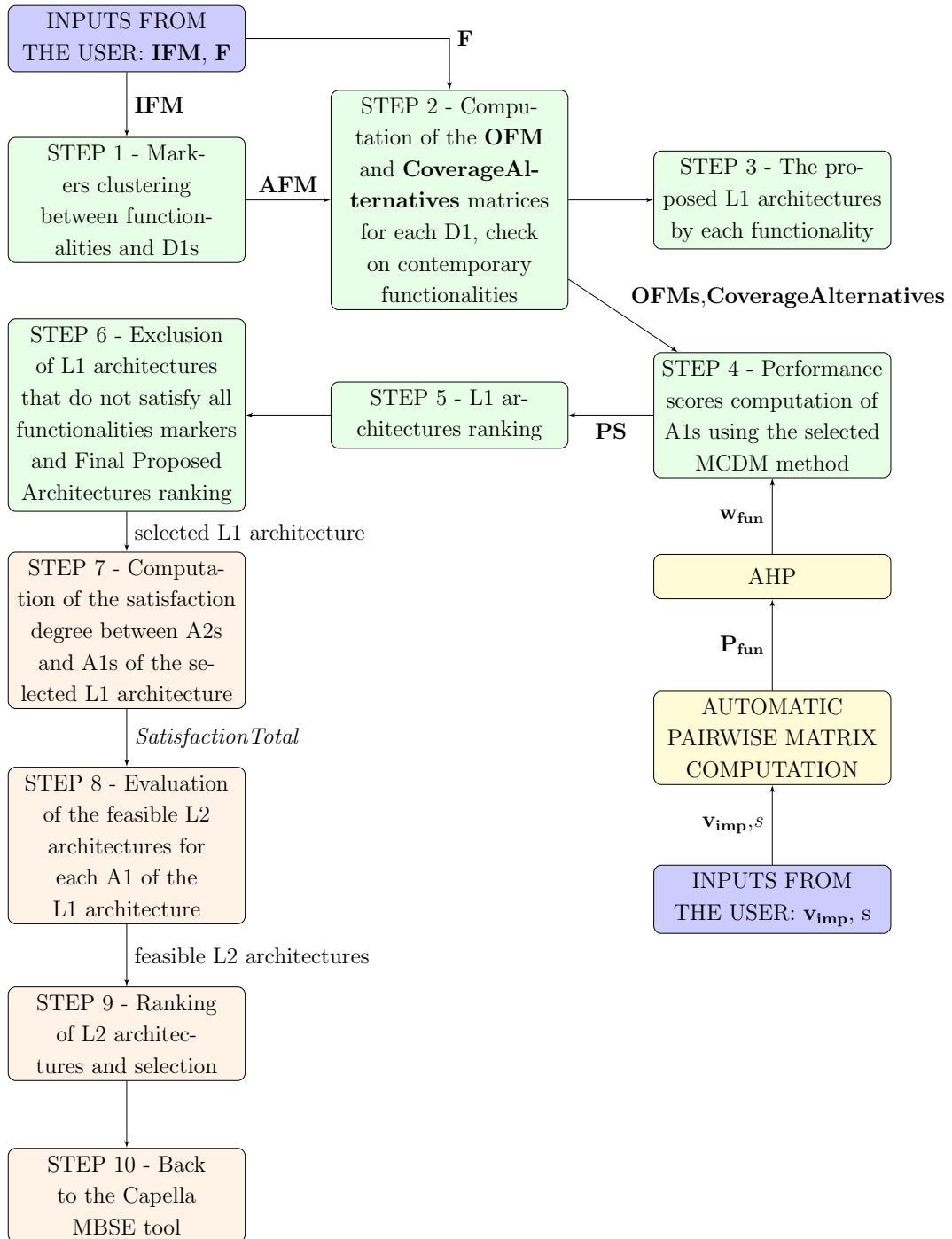


Figure 4.2: Flow chart of the decision-making algorithm. Light green is used for the steps belonging to the first level, light orange for the second level ones.

Table 4.2: Scale of relative importance in AHP

Value	Definition
1	Equal importance
3	Somewhat more important
5	Much more important
7	Very much more important
9	Absolutely more important
2,4,6,8	Intermediate values

Table 4.3: Example of Pairwise Matrix in AHP

	A1	A2	A3
A1	1	5	1/7
A2	1/5	1	1/9
A3	7	9	1

In order to understand the consistency of the original Pairwise Matrix, the Consistency Index is computed as in Equation 4.6:

$$CI = \frac{\lambda_{MAX} - n_{attributes}}{n_{attributes} - 1} \quad (4.6)$$

where λ_{MAX} is the maximum eigenvalue of the matrix and $n_{attributes}$ is the total number of attributes. Then the Consistency Ratio (CR) is computed as the fraction between CI and the Random Index, the latter depending on the number of attributes in the problem (see [58, 41]). If $CR < 0.1$ then the Pairwise Matrix is consistent so the previously computed Criteria Weights can be taken as valid, otherwise something has to be changed in the matrix.

Other MCDM Methods for Decision Matrix Resolution

A number of MCDM methods have been investigated for this work, used to solve the decision matrix of alternatives after having computed the criteria weights (functionalities weights here) from the AHP. Namely, the Weighted Sum Method (WSM), the Weighted Product Method (WPM), the Technique for Order of Preference by Similarity to Ideal Solutions (TOPSIS), the Evaluation Based on Distance from Average Solution (EDAS) and the Preference Ranking Organization Method for Enrichment Evaluation II (PROMETHEE II). Their description is not here provided, however the reader can consult the main references adopted [41, 73, 40, 46].

4.2.2 Level 1 Architectures Selection

Step 1: the Clustering Technique

Recall the scope of the method: m functionalities with n markers have to be mapped into a set of decisions and their alternatives, described by the same markers, in order to extrapolate a quantity that tells how much each alternative is suitable for each functionality. In case of boolean markers values for both functionalities and alternatives, it would be quite straightforward to establish a simple one-to-one correspondence between the set of input markers and the set of code-embedded ones. However, this is not the case since both input and code (alternatives) markers are also classified according to a linear scale as previously discussed. A way to merge the information coming from the **IFM** and the code-embedded information has to be formulated.

A matrix, called *Alternatives-Functionalities Matrix*, similar to those of Equations 4.4 and 4.5, is compiled by the tool for each functionality and for each decision. The elements of these matrices are defined according to the rule in Equation 4.8:

$$AFM_j^w = \begin{bmatrix} x_{11_w} & \cdots & x_{ik_w} & \cdots & x_{1p_w} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ x_{i1_w} & \cdots & x_{ik_w} & \cdots & x_{ip_w} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ x_{n1_w} & \cdots & x_{nk_w} & \cdots & x_{np_w} \end{bmatrix} \quad (4.7)$$

$$x_{ik_w} = \begin{cases} 0 & \text{if } f_{ij} = 0 \vee m_{ik_w} = 0 \\ 1 & \text{if } f_{ij} = 1 \wedge m_{ik_w} = 1 \end{cases} \quad (4.8)$$

Equation 4.8 imposes a simple but very important condition; indeed, whenever the i -th marker of the j -th functionality is null (a marker assumes null value whenever it is not declared in the functionality list of characteristics), the considered marker has nothing in common with that functionality, and so it would not make sense to evaluate an alternative with respect to that functionality taking into account a wrong marker. Instead, if the functionality and the alternative is described by a non-zero marker (equal to 1 in case of boolean values), the value 1 is assigned in the **AFM**.

The one presented above is the case of boolean marker values. However, markers can also be quantified. For example, one can state that a marker for a functionality has a value according to a proper scale (higher the value, more important the marker for that functionality) while the same marker may have a different value for other functionalities. The mapping in this case is quite more complex with respect to the boolean case and a way to correlate the functionalities and the alternatives markers has to be found. The proposed approach consists in adopting a scale that takes values from 2 to 4, preserving the boolean values that can still be assigned, according to the coding in Equation 4.9. If

an alternative marker is boolean, also the functionality one shall be boolean.

$$f_{ij} = \begin{cases} 0 & \text{the } i\text{-th marker does not characterize the } j\text{-th functionality (boolean)} \\ 1 & \text{the } i\text{-th marker simply characterizes the } j\text{-th functionality (boolean)} \\ 2 & \text{the } j\text{-th functionality is weakly characterized by the } i\text{-th marker (scale)} \\ 3 & \text{the } j\text{-th functionality is well characterized by the } i\text{-th marker (scale)} \\ 4 & \text{the } j\text{-th functionality is greatly characterized by the } i\text{-th marker (scale)} \end{cases} \quad (4.9)$$

Concerning the case of non boolean values, if the functionality marker has value 4 and the same alternative marker has value 4, it means that the alternative perfectly satisfies the functionality according to that marker. In case of non-boolean markers, the **AFM** is compiled computing the difference between the i -th functionality marker value and the i -th alternative marker value (as long as they are different from 0 or 1). If the computed number is equal to 0, it means that what is required by the functionality (e.g. a value of 4 for a marker) is perfectly satisfied, according to that marker, by the k_w -th alternative, and so the highest value of 1 is assigned to the x_{ik_w} element of the **AFM**. If the difference is 1, a lower value is assigned, meaning that the alternative still well satisfies the functionality according to that marker, but not perfectly. The maximum difference can be 2, on that case a minimum value will be assigned. Actually a 10% increment is assigned when the difference between the functionality and the alternative is lower than 0, meaning that the alternative satisfies the functionality according to that marker more than needed (while in the other case it is less satisfied). The value 0 is assigned only if the f_{ij} or the m_{ik_w} are 0. It is recalled that a marker can have value 0 even if it is not classified as boolean. The discussed rules are summarized in Equation 4.10:

$$x_{ik_w} = \begin{cases} 0 & \text{if } f_{ij} = 0 \vee m_{ik_w} = 0 \\ 1 & \text{if } f_{ij} = 1 \wedge m_{ik_w} = 1 \\ 1 & \text{if } f_{ij} - m_{ik_w} = 0 \\ 1 - \frac{|f_{ij} - m_{ik_w}|}{3} & \text{if } f_{ij} - m_{ik_w} > 0 \\ \left(1 - \frac{|f_{ij} - m_{ik_w}|}{3}\right) \cdot 1.1 & \text{if } f_{ij} - m_{ik_w} < 0 \end{cases} \quad (4.10)$$

Summing up, each decision w -th having p_w alternatives will be characterized by a 3D tensor containing m **AFM** matrices of dimension $[n \times p_w]$, one for each functionality.

Another output comes from this step, called **coverage**. Each decision will be characterized by m matrices of this kind, having n rows and number of columns equal to the number alternatives of the decision, therefore forming a 3D tensor. It is similar to the **AFM**, however it is compiled differently as the Algorithm 2 in Appendix B.2 shows. It will be used to exclude those architectures which do not cover all the markers asked by

the functionalities.

$$\mathbf{coverage}_j^w = \begin{bmatrix} c_{11w} & \cdots & c_{1k_w} & \cdots & c_{1p_w} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ c_{i1w} & \cdots & c_{ik_w} & \cdots & c_{ip_w} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ c_{n1w} & \cdots & c_{nk_w} & \cdots & c_{np_w} \end{bmatrix} \quad (4.11)$$

Step 2: the *Output Functionality Matrix* and the *Coverage Alternatives*

Now, the j -th **AFM** has to be converted into a vector whose elements represent the degree of satisfaction of the k_w -th alternative with respect to the j -th functionality. To do that, a simple average on the columns is done (and so on the “illuminated” markers of the alternatives), obtaining the desired vector called *Output Functionality Vector* for each functionality:

$$\mathbf{OFV}_j^w = \left[y_{1wj} = \frac{\sum_{i=1}^n x_{i1w}}{n}, \dots, y_{k_wj} = \frac{\sum_{i=1}^n x_{ik_w}}{n}, \dots, y_{p_wj} = \frac{\sum_{i=1}^n x_{ip_w}}{n} \right] \quad (4.12)$$

The computed vectors are reported as columns into a matrix called *Output Functionality Matrix*, one for each decision w having p_w alternatives, that has the following expression:

$$\mathbf{OFM}^w = \begin{bmatrix} y_{1w1} & \cdots & y_{1wj} & \cdots & y_{1wm} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ y_{k_w1} & \cdots & y_{k_wj} & \cdots & y_{k_wm} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ y_{p_w1} & \cdots & y_{p_wj} & \cdots & y_{p_wm} \end{bmatrix} \quad (4.13)$$

Its rows are single alternatives and its columns are functionalities. Higher the y_{k_wj} value, better the k_w -th alternative is in accomplishing the j -th functionality. This matrix is also a decision matrix: the ranking of the alternatives is obtained from the resolution of a decision-making problem that solves the **OFM**, as the next subsection shows.

Similarly, using the **coverage** matrices, for each alternative the so-called **Coverage Alternatives** is computed (Equation 4.14) summing all the values in the *coverage* rows. Each alternative will then have a $[1 \times m]$ vector indicating how much that alternative satisfies each functionality in terms of markers coverage, and each decision will be characterized by a matrix built as the **OFM**. The elements of the **Coverage Alternatives** matrices are different from those obtained in the **OFM** because the degree of satisfaction is not considered here. Indeed, it may happen that in the **OFM** an alternative has a higher value with respect to a different one because of the higher values coming from Equation 4.10 and at the same time covering a lower number of markers, therefore having a lower value in the **Coverage Alternatives** matrix.

$$\mathbf{Coverage Alternatives}_j^w = \left[ca_{1wj} = \sum_{i=1}^n c_{i1w}, \dots, ca_{p_wj} = \sum_{i=1}^n c_{ip_w} \right] \quad (4.14)$$

Before moving to the following step, an important condition involving the contemporary functionalities is here applied. If an alternative is totally wrong for a functionality (value 0 in the **OFM** or in the **CoverageAlternatives**, a threshold value can also be selected instead of 0), it means that the behaviour of such functionality is compromised. If another functionality has to be done at the same time of the former, a condition is activated to assign value 0 also to the cell of both the **OFM** and the **CoverageAlternatives** corresponding to the second functionality. This way, that alternative is excluded from the solution. Without such condition, an alternative may be selected by a functionality and at the same time compromising the behaviour of a contemporary functionality per which that alternative is totally unsuitable. The **F** matrix presented in 4.1 is here exploited to tell the code whether functionalities are contemporary (Appendix B.2, Algorithm 3).

Step 3: the Proposed Architecture by Functionalities

This step consists in the extraction of the best alternative for each decision, according to each single functionality, therefore leading to the preferred architecture from the single functionalities. This is not properly a fundamental step of the tool, however it can be useful later to compare the overall architecture with the functionalities “preferences”. To do that, values in the **OFM** and in the **CoverageAlternatives** are simply sorted and for each decision the best alternative is selected, leading to two architectures for each functionality. The outputs are two vectors with the identification number of the best alternative for each decision:

$$\mathbf{Archifun}_{OFM_j}^w = \left[alt_{OFM_1}, \dots, alt_{OFM_l} \right] \quad (4.15)$$

$$\mathbf{Archifun}_{coverage_j}^w = \left[alt_{cov_1}, \dots, alt_{cov_l} \right] \quad (4.16)$$

Step 4: Performance Scores of the Alternatives for each Decision

The pairwise matrix automatically computed, reported in Section 4.1.1, is furnished as input to the AHP function which implements the Analytic Hierarchy Process to derive the weights assigned to each functionality (see Section 4.2.1) and used in the here described decision-making problem.

Once the user selects the MCDM method among those available, the previous **OFM** and **CoverageAlternatives** matrices are solved as they are here treated as decision matrices, where in the rows there are the alternatives values belonging to the w -th decision and the columns are functionalities, which represent multiple decision criteria.

The output of this step are two vectors for each decision, containing the Performance Scores of the alternatives computed applying the selected MCDM method respectively to the **OFM** and the **CoverageAlternatives**:

$$\mathbf{PS}_{OFM}^w = \left[PS_{OFM}^w(1_w), \dots, PS_{OFM}^w(k_w), \dots, PS_{OFM}^w(p_w) \right] \quad (4.17)$$

$$PS_{coverage}^w = \left[PS_{cov}^w(1_w), \dots, PS_{cov}^w(k_w), \dots, PS_{cov}^w(p_w) \right] \quad (4.18)$$

To summarize, a 3D tensor for each decision ($AFM^w(i, k_w, j)$) was converted into a matrix for each decision ($OFM^w(k_w, j)$ and $CoverageAlternatives^w(k_w, j)$) and then into two vectors ($PS_{OFM}^w(k_w)$ and $PS_{coverage}^w(k_w)$).

Step 5: Architectures Ranking

At the end of the Step 4, each alternative of each decision is characterized by two Performance Scores which tells how much that alternative is suitable for the whole set of functionalities. Taking one alternative for each decision means building an architecture. The aim of the Step 5 is to evaluate all the possible architectures and rank them.

To accomplish the presented task, an overall Performance Score is computed for each architecture as the product between the Performance Scores of the alternatives that compose it. Actually two Performances Scores are computed, one considers the **OFM** and one the **CoverageAlternatives**, as in Equations 4.19 and 4.20:

$$PS_{archi_{OFM}}(f) = \prod_{w=1}^l PS_{OFM}^w(k_w) \quad (4.19)$$

$$PS_{archi_{coverage}}(f) = \prod_{w=1}^l PS_{coverage}^w(k_w) \quad (4.20)$$

In the previous equations f is the identification number of the evaluated architecture, which are in total equal to the product between the number of alternatives per each decision; l is the total number of decisions; k_w is the alternative under cycle of the w -th decision. This way, all the combinations are evaluated.

Now an overall parameter which merges the previous two is introduced, so that each architecture from now on is quantified by one single number. It is computed as in the Equation 4.21, where w_{OFM} and $w_{coverage}$ are weights which summed must be equal to one and can be set by the user (i.e. 0.5 each):

$$J(q) = PS_{archi_{OFM}}(q) \cdot w_{OFM} + PS_{archi_{coverage}}(q) \cdot w_{coverage} \quad (4.21)$$

Once J is computed for each architecture, the values are sorted decreasingly, preserving the indexes of the alternatives which constitute each q -th architecture.

Step 6: the *Final Proposed Architectures*

At this point, all the architectures are distinguished by an identification number and a ranking value J . However it is not ensured that each architecture actually covers all the markers required by the functionalities, therefore a skimming takes place here in order

to exclude those architectures which do not satisfy all the functionalities, and so all the markers.

A new matrix is introduced for each architecture, called **CoveredMarkers**, with dimensions equal to the IFM ($[n \times m]$). It is filled assigning the value 1 to the (i,j) cell whenever at least one alternative of the architecture has value different from 0 in **coverage** for the i -th marker and the j -th functionality, meaning that such marker is satisfied for that functionality, otherwise the value 0 is assigned (Equation 4.22).

$$CoveredMarkers_q(i, j) = \begin{cases} 1 & \text{if (i,j) marker satisfied by at least one alternative} \\ 0 & \text{if (i,j) marker is not satisfied by at least one alternative} \end{cases} \quad (4.22)$$

Now, if one architecture has at least one row in **CoveredMarkers** with only all null values, it means that the i -th marker is not satisfied by any alternative. Therefore a value 0 is assigned to J of the q -th architecture. This way, only the architectures which cover all the required markers by the functionalities are preserved.

The last passage of this step consists in verifying that the skimmed architectures effectively satisfy all the functionalities with a further check involving the outcomes from the Step 2 about the **OFM**. The previous skimming, indeed, does not ensure that all functionalities are actually satisfied, because it was conducted using the **coverage** matrices which do not consider the zeroing of some alternatives coming from the condition about contemporary functionalities, performed in the Step 2. It is recalled that each decision is characterized by one **OFM**, having pw lines (one for each alternative of the decision) and m columns. If for each alternative of each decision of the architecture under cycle the value in the **OFM** corresponding to the j -th column (or functionality) is 0, such architecture is excluded assigning value 0 to J .

The *Final Proposed Architectures* are those with a J value different from 0; higher the value, better the architecture for the desired functionalities. The **FPA** is a matrix with number of rows equal to the total number of evaluated architectures and number of columns equal to the number of decisions; each cell contains a number which identifies the alternative of the decision it belongs (i.e. column 1 is the decision 1) for the q -th architecture. The algorithm related to the Step 6 is presented in Appendix B.2, Algorithm 4.

4.2.3 Level 2 Architectures Selection

Step 7: Satisfaction Degree Computation of Second Level Alternatives

In Section 4.1.2 the second level decisions were presented: each first level alternative contains a number of second level decisions, each one with its own set of second level alternatives. The scope of this second part of the algorithm is to select, for each archi-

ecture coming from the Level 1, a number of second level alternatives which ensure that all the first level alternatives in the L1 architecture are accomplished, and therefore functionalities too. In order to ease the readability of this section, from here on the following nomenclature is adopted:

- D1 = first level decision;
- A1 = first level alternative (contained in a D1);
- D2 = second level decision (contained in a A1);
- A2 = second level alternative (contained in D2).

The first step of the algorithm consists in computing, for each A2, the degree of markers coverage asked by the A1. This is done for all the A1s belonging to the first level architecture selected by the user, expressed as in Equation 4.23:

$$\mathbf{Archi}_q = [A1_q(1), \dots, A1_q(w), \dots, A1_q(l)] \quad (4.23)$$

A similar approach to the one applied for the first level clustering is here presented, introducing the **satisfaction** matrices $[n \times q_h]$, where q_h is the total number of A2 contained in the h -th D2. Each D2 will be characterized by a number of **satisfaction** matrices equal to the number of functionalities, therefore obtaining a 3D tensor. Actually each decision h and alternative g_h should have the pedix k_w , not reported in this section to ease the readability. Equation 4.24 shows a generic matrix of the h -th decision and j -th functionality, while the rules for the matrix filling are directly reported in Appendix B.2, Algorithm 5.

$$\mathbf{satisfaction}_j = \begin{bmatrix} s_{11} & \cdots & s_{1g_h} & \cdots & s_{1q_h} \\ \vdots & \cdots & \vdots & \vdots & \vdots \\ s_{i1} & \cdots & s_{ig_h} & \cdots & s_{iq_h} \\ \vdots & \cdots & \vdots & \vdots & \vdots \\ s_{n1} & \cdots & s_{ng_h} & \cdots & s_{nq_h} \end{bmatrix} \quad (4.24)$$

The sum on the markers (index i) and on the functionalities (index j) is done for each g_h -th A2 leading to a scalar called *SatisfactionTotal*, engaged in the next steps, that tells the goodness of that alternative in satisfying the A1 markers it belongs:

$$SatisfactionTotal_{g_h} = \sum_{i=1}^n \sum_{j=1}^m satisfaction(i, g_h, j) \quad (4.25)$$

Step 8: Feasible Second Level Architectures Evaluation

Now, for each A1 selected in the first level architecture, the purpose is to find the second level architecture which guarantees the highest markers coverage. It is recalled that as each A1 contains a number of D2, a second level architecture is here intended as a set

of A2 selected by the A1. Therefore there will be a number of second level architectures equal to the number of D1 (or A1, as in the first level just one A1 is selected by a D1). The overall architecture instead merges all of them.

The driving parameters are once again the markers, as the aim is to guarantee the coverage of those “illuminated” by each selected A1. For each A1, all combinations of A2 are evaluated. As baseline, the code selects just one A2 for each D2; however it is not ensured that all the markers required by the A1 they belongs will be satisfied. Therefore, if none of the second level architectures related to an A1 covers all the markers selecting just one A2 for each D2, new A2 are added until all the markers are covered.

The Step 8 passes through the definition of a newly matrix called **CoverageTot**, defined for each D2 and with size $[n \times q_h]$. It is filled with values 1 whenever all the A1 markers are covered, took from the **AFM** in order to ensure that functionalities too are satisfied. Their coverage is verified looking at the sum of **satisfaction** values for the A2s contributing to the second level architecture. This way, since the first level architecture ensure the satisfaction of functionalities markers, if all markers of such first level architecture are satisfied by the “assembly” of the second level ones, it means that the overall architecture for sure will be suitable for the asked functionalities.

Step 9: Final Proposed Overall Architectures

As done for the first level, all the overall architectures that passed the previous skimming algorithm are ranked. This time a different parameter is used to evaluate how much an architecture is suitable for the input functionalities. It is called *ValueArchi* and it is computed as the sum of the *SatisfactionTotal* values associated to each A2 of the considered architecture:

$$ValueArchi_r = \sum_{t=1}^{n_{A2}} SatisfactionTotal(t) \quad (4.26)$$

where n_{A2} is the number of A2s belonging to the r -th architecture.

Step 10: Back to the Capella Environment

The last step consists in exploiting a library of modeled components, which represent all the A2s (leaves of the decision tree), in an MBSE tool such as Capella. Once the user selects the overall architecture, he/she can directly move to Capella and work with the already modeled components in terms of basic functionalities as well as a first grid of requirements, as in Figure 4.3. The user then can adds Functional Exchanges, Functions, Physical Links, requirements and new components if needed. It is clarified that the initially defined functionalities should be modeled within the System Analysis in Arcadia, that is the level at which the user’s project should be in order to properly use the tool (he/she should have a clear idea of what his/her system has to do); components are

instead modeled at Physical Architecture. This way the user is forced to bridge SA and PA passing through the Logical Architecture, in which further considerations about how the system has to work will surely arise and eventually new needs and functionalities. The components obtained by the tool, at that point, may not satisfy yet all the functionalities, therefore it is advised to re-run the tool adding the new ones. In this sense, the tool can also be used to evaluate if changes in the required behaviour of the system influence the components selection and how, suggesting the best architecture which suits best to the needs providing a good support to an MBSE solution.

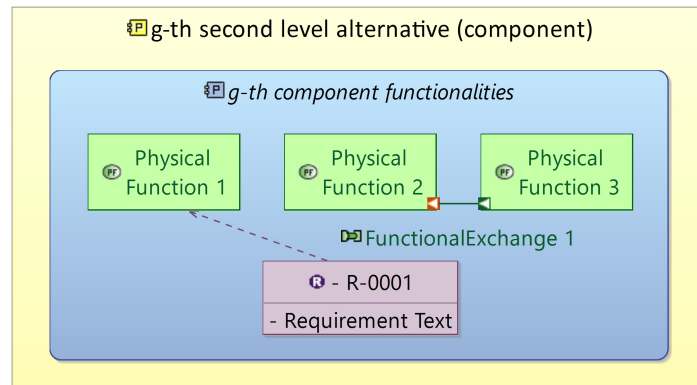


Figure 4.3: Example of Components Modeling in Capella

4.3 Simulations and Validation

This section reports some simulations with the purpose of validating the tool, interpret the results and evaluating the limits of the algorithm. A review of the technologies that constitute the decision tree is firstly presented to provide the rationale behind the markers assigned.

4.3.1 Subsystems Decision Tree

For the following simulations the ADCS, the EPS, the TT&C and the Propulsion subsystems have been considered, each one characterized by a number of decisions and alternatives, here briefly presented one by one. The main reference used to review small satellites technologies is the *NASA State-of-the-Art Small Spacecraft Technology 2020 edition* [80]. At this preliminary design level it is important to choose the set of decisions and alternatives that are limited in number but at the same time effective in describing the system and providing an architecture.

ADCS

For the ADCS subsystem, three first level decisions are defined, related to the most common stabilization techniques adopted for small satellites: 3-axis, spin and passive. Two second level decisions follows, as reported in Figure 4.4: sensors for absolute and relative attitude determination and actuators to control the satellite. A review of their common technologies is provided in Tables 4.4 and 4.5, which represent the second level alternatives of the tree. Other classes can be added in future, such as the control algorithm selection, the position determination techniques and/or the inertial measurements technologies.

Propulsion

Not all small satellites have a propulsion subsystem, here intended just for orbital maneuvers and not for attitude scopes. However, an assumption is done for the following simulations, that is the need of such subsystem for orbit changes. This way it is possible to better understand how the tool works, and how it behaves with the selection of the propulsion type too. The first level decision for this subsystem contains two AIs: chemical and electric propulsion. They become in turn second level decisions, with the technologies reported in Tables 4.6 and 4.7 which indicate the second level alternatives. The tree is reported in Figure 4.5.

EPS

For the Electric Power Subsystem one first level decision is a sort of architecture type, containing two first level alternatives: satellites with just solar panels and satellite with both solar panels and secondary batteries. Primary non-rechargeable batteries are ex-

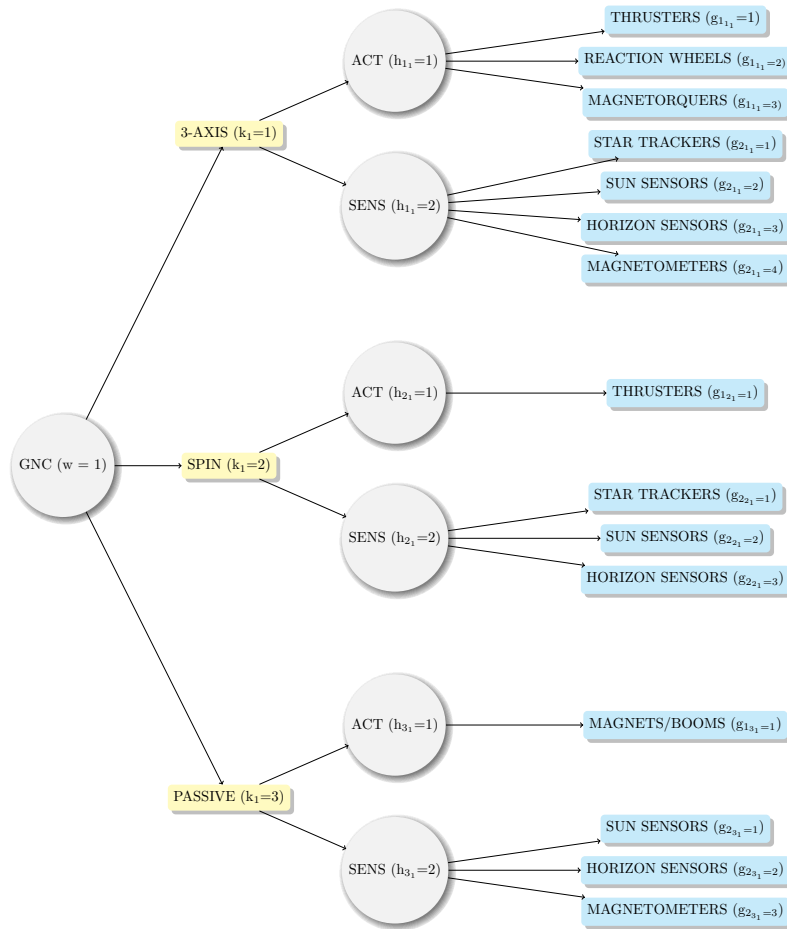


Figure 4.4: Decision Tree of ADCS

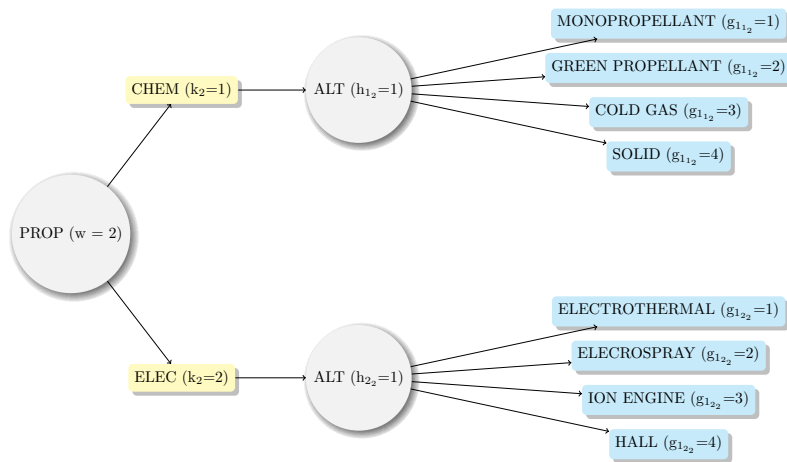


Figure 4.5: Decision Tree of Propulsion Subsystem

Table 4.4: Common actuators for small satellites [80]

Thrusters	They are used for attitude control and desaturation operations on small satellites generating a thrust displaced from the center of mass of the platform. Their lifetime is limited by the amount of propellant on-board.
Reaction Wheels	Provide spacecrafts with a three-axis precision pointing capability generating a torque around the spin axis. A minimum of 3 wheels are required for full 3-axis control. Reaction wheels need to be periodically desaturated once they reach their maximum speed rate using an actuator that provides an external torque. The most accurate among the actuators, can be used for fast slew maneuvers.
Magnetorquers	They work only in presence of a local external magnetic field (i.e. Earth's one) generating a torque perpendicular to it. One magnetic torquer cannot provide full 3-axis stabilization alone. Less accurate with respect to other actuators.
Magnets/Booms	Can be used for Passive Magnetic and/or Gravity Gradient Stabilization techniques. They do not provide controllability.

Table 4.5: Common sensors for small satellites [80]

Star Trackers	A star tracker can provide an accurate estimate of three-axis attitude by capturing digital images and comparing them with multiple stars in a catalog.
Sun Sensors	Sun sensors are used to estimate the direction of the Sun in the spacecraft body frame for attitude estimation. To obtain a three-axis attitude estimate at least one additional independent source of attitude information is required.
Horizon Sensors	Horizon sensors can be simple infrared horizon crossing indicators (HCI), or more advanced thermopile sensors that can be used to detect temperature differences between the poles and equator. They detect Earth edges to calculate the roll and the pitch angle of the satellite.
Magnetometers	They provide a measurement of the local magnetic field to provide both attitude and orbital position.

cluded as they are used only for one-time short use [42], so very brief missions of up to one-week typically; approximately 85% of all nano-satellites are equipped with solar panels and rechargeable batteries [80]. However, it is clarified that the decision tree, reported in Figure 4.6, is not frozen and new decisions and alternatives can be freely added.

Table 4.6: Common chemical propulsion technologies for small satellites [80]

Monopropellant (i.e. Hydrazine-based)	They use catalyst structures to decompose hydrazine or a derivative such as monomethyl hydrazine (MMH) to produce hot gases. Hydrazine specific impulses are achievable in the 200 – 235 second range for 1-N class or larger thrusters.
Green Propellant-based	The so-called “green propellants” (i.e. HAN, ADN) have reduced toxicity due in large part to the lower danger of component chemicals. Green propellants also provide higher specific impulse performance than the current state-of-the-art hydrazine monopropellant and have lower minimum storage temperatures which may be beneficial in power-limited spacecraft. However, the technology is less mature.
Cold Gas	Cold gas systems are relatively simple systems that provide limited spacecraft propulsion and are one of the most mature technologies for small spacecraft. Thrust is produced by the expulsion of a propellant which can be stored as a pressurized gas or a saturated liquid.
Solid Propellant	Solid rocket technology is typically used for impulsive maneuvers such as orbit insertion or quick de-orbiting. Due to the solid propellant, they achieve moderate specific impulses and high thrust magnitudes that are compact and suitable for small buses. Not restartable.

The first A1, only solar panels, becomes a D2 with three alternatives: only body mounted panels, body mounted plus fixed wings and body mounted plus gimbaled wings. Actually the last alternative may be constraining, as the selection of orientable solar panels should be done after some iterations within the subsystems design, as it introduces complexity. However it is kept in order to assess in which cases the tool opts for it.

For the second D1, besides the D2 about the solar panels architecture, another D2 is introduced: the secondary batteries type. The survey in [12] indicates the following secondary batteries as the most adopted for small satellites: 66% Lithium-ion (Li-ion), 16% Nickel-Cadmium (Ni-Cd), 12% Lithium-polymer (Li-pol) and 4% Lithium-Chloride (Li-Cl). The decision tree in Figure 4.6 reports the three most used, presented in Table 4.8.

TT&C

This subsystem is introduced just for what concern the downlink of telemetry and scientific data, enough to evaluate the ability of the tool in the selection process. Two first level decisions are defined: high gain antenna, typically engaged for high data download in a

Table 4.7: Common electric propulsion technologies for small satellites [80]

Electrothermal (i.e. Resistojet, Arcjet)	Electrothermal technologies use electrical energy to increase the enthalpy of a propellant (chemical rely on exothermal chemical reactions). Once heated, the propellant is accelerated and expelled through a conventional converging-diverging nozzle to convert the acquired energy into kinetic energy.
Electrosprays (i.e. FEEP)	Electrospray propulsion systems generate thrust by electrostatically extracting and accelerating ions or droplets from a low-vaporpressure, electrically-conductive, liquid propellant.
Ion Engines	Gridded-ion propulsion systems ionize gaseous propellant via a plasma discharge, and the resultant ions are subsequently accelerated via electrostatic grids. An external neutralizer cathode is needed to maintain plume charge neutrality. High specific impulses can be achieved, but the thrust density is fundamentally limited by space-charge effects.
Hall	The Hall-effect thruster (HET) is arguably the most successful in-space EP technology by quantity of units flown. They generate thrust by creating and accelerating ionized gas via magnetic and electrostatic fields.

Table 4.8: Common secondary batteries for small satellites [80]

Nickel-Cadmium	Conventional Ni-Cd batteries were widely used during the first 30 years in aerospace industry. They have high cycle life but a low energy density. The cell voltage is approximately constant until it is nearly fully discharged. The temperature is a critical parameter that affects the battery life and must be maintained in a narrow range. Repeated cycling to a deep depth of discharge can cause cracking in the cell plate structure. [49]
Lithium-Ion	Li-Ion is a high energy density technology, can accept deep discharges, therefore more of the available energy can be used and for a long number of cycles [49]. They are the most adopted for space applications.
Lithium-Polimer	Li-pol cells are traditionally having a high energy density and pouch format. This provides them the benefit of flexible size, slim profile, and generally reduced weight. However, due to the mechanical attributes, they might be prone to damage in the space environment (vacuum) if not carefully constructed [42]

restricted time interval, and low gain one, non directional and able to transmit a lower amount of data in the same time interval with respect to the HGA. They become second

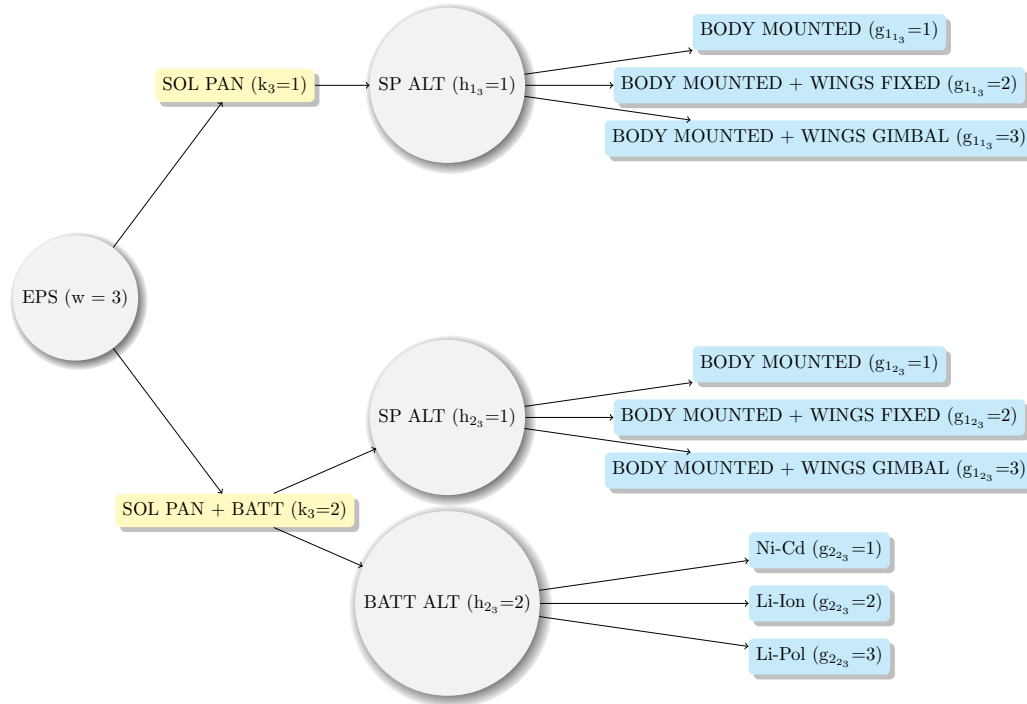


Figure 4.6: Decision Tree of EPS

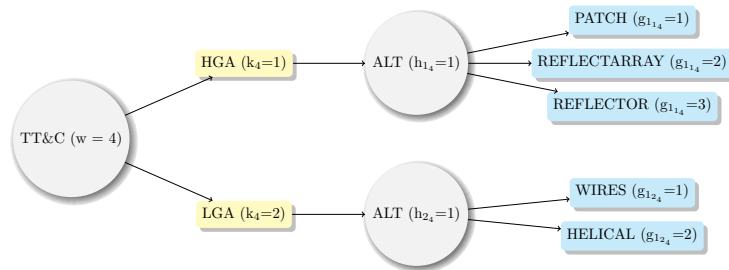
level decisions in the tree; the associated second level alternatives are reported in Tables 4.9 and 4.10, and in the tree of Figure 4.7.

Table 4.9: Common high gain antennas for small satellites

Patch	They have gained special attention for CubeSats, owing to their low profile and relative ease of fabrication. A variety of patch antenna designs have been investigated at the VHF, UHF, and S bands [53].
Reflectarray	They can provide high gain while easily integrating with the CubeSat structure. Since their structure consists of flat panels, it is possible for them to be folded and stowed on the CubeSat [53]. They were mounted on MarCO mission CubeSats [31].
Reflector	Reflectors offer the possibility of high gain and fine resolution, but they come with increased mechanical complexity. One of the first CubeSats to integrate a deployable reflector system was the Aeneas mission [2], featuring an S-band umbrella reflector with a 0.5-m diameter [53]. This is an emergent technology for small satellites.

Table 4.10: Common low gain antennas for small satellites

Wires (monopoles, dipoles)	These antennas typically are placed on the external face of the CubeSat chassis, allowing space for other electronics. During flight, the wire antennas are often stowed within the satellite volume and deployed once in orbit. Wire antennas are especially common for high frequency (HF), very HF (VHF), and ultra HF (UHF) applications, where the wavelength is long and achieving good radiation efficiency within a small volume is challenging [53].
Helical	They consist of one or more conducting wires wound in the form of a helix. They typically occupy more volume with consequent constraints on the configuration.

**Figure 4.7:** Decision Tree of TT&C

4.3.2 Results and Discussion

This section aims to apply the implemented tool for some small satellites mission scenarios, comparing the results with the architecture of similar missions. Two simulations are presented, one to assess the goodness of the selected architectures ranking, and another one in which the algorithm is stressed in order to check whether it effectively solves conflicting situations and how.

It is recalled that each alternative of the previously presented decision trees, be it an A1 or an A2, is characterized by n markers. The values assigned to them are reported in Appendix B.1, Tables B.3, B.4, B.5 and B.6. The following results are used to validate the implemented algorithm checking the coherence of the output architectures and to highlight its limits, being a preliminary prototype whose purpose is essentially to demonstrate the feasibility of the approach. In particular, the limits are mostly related to the substantial solutions changing with the tool-embedded tree parameters, therefore requiring a refinement in terms of more precise meanings associated to the markers, as well as training with a data set from previous missions in order to better compile them. It is also stressed that, according to the way missions are defined and given as input to the tool, the obtained output embrace a casuistry rather than a specific mission. However, as the

number of input functionalities is increased, the tool can converge to precise needs of a particular scenario, getting a “tailored” output for it.

Simulation 1

Let consider three very simple functionalities:

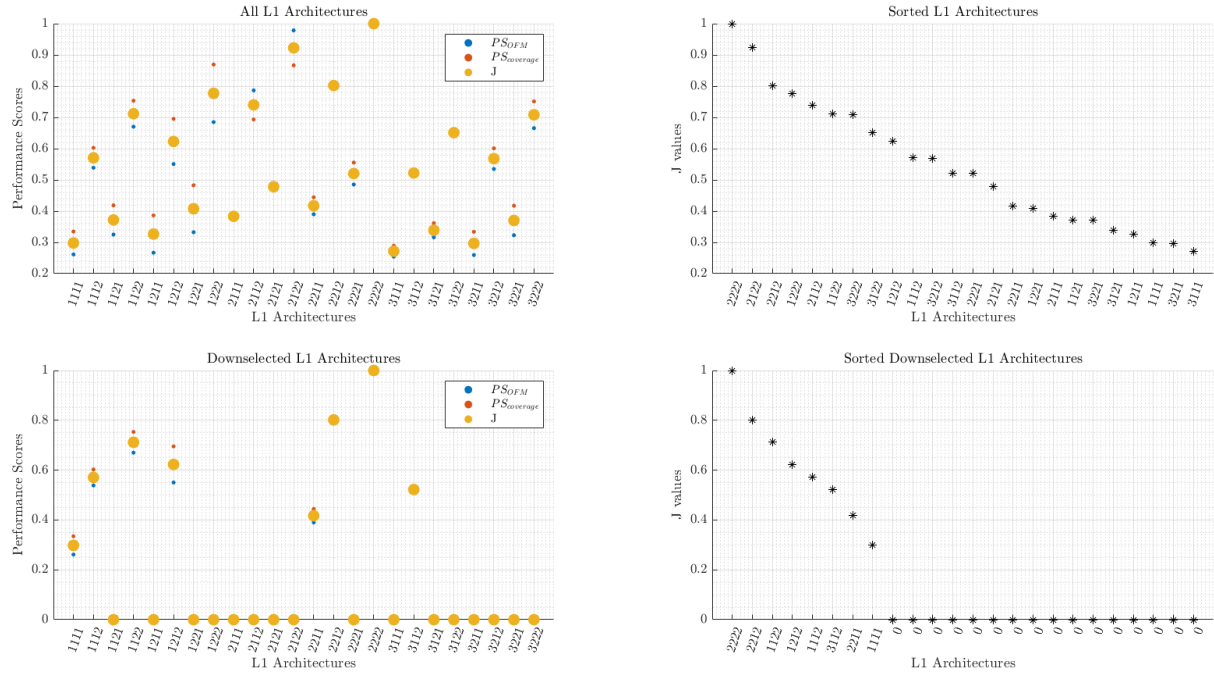
- F1 = Execute transfer to operative orbit;
- F2 = Transmit telemetry;
- F3 = Point inertial target.

The markers describing them are reported in Appendix B.1, Table B.1. Whenever the acronym NB (Non Beneficial) appears, it means that higher the marker value better it is (the marker *mass* is NB, therefore if value 4 is assigned it means that a low mass is desired). The above listed are non contemporary functionalities, therefore the \mathbf{F} matrix will have only zero values. Equal importance is assigned to them, consequently \mathbf{v}_{imp} is a $[1 \times 3]$ vector with all zero elements. The Sparsity Factor is set to 1 and the MCDM method used is the Weighted Sum.

Results related to the L1 architecture are shown in Figure 4.8. The horizontal axis reports a four digit number indicating the architecture; the first digit is the alternative of the first decision, the second digit is the alternative of the second decision and so on up to the fourth first level decision. The top-left diagram reports the ranked J values of each architecture, computed as the average between the two PS values, as explained in Section 4.2.2. Changing the weights assigned to the PS_{OFM} and $PS_{coverage}$ values, that for this simulation are both set equal to 0.5, the J value moves in between the interval. It is noted that most of the L1 architectures have PS_{OFM} values lower than $PS_{coverage}$, while for the architectures 2-1-1-2 and 2-1-2-2 the opposite happens, highlighting the conceptual distinction between **OFM** and **CoverageAlternatives**. The diagram in the top-right reports the same ranking values sorted from the highest to the lowest, suggesting as best L1 architecture the one with indexes 2-2-2-2 composed by spin stabilization, electric propulsion, solar panels + batteries, low gain antenna (recall the decision trees for the indexes).

Diagrams in the bottom of the Figure 4.8 represent the tool output after the Step 6. Eight *Final Proposed Architectures* are downselected, while the remaining ones have zero values because of their inability to satisfy all the functionalities markers. It is noted that, as the functionality F2 requires small amount of data to be downlinked, most of the selected alternatives suggest the adoption of the low gain antenna (number 2 in the fourth digit), however some solutions suggest the HGA one.

Once the L1 architectures are computed, the user can move to the selection of one or more of them in order to derive the L2 architectures too. Selecting the L1 architecture with


Figure 4.8: Simulation 1 - L1 Architectures

the highest ranking (2-2-2-2), the output coming from the second part of the algorithm is showed in Figure 4.9. For each D1, some architectures are computed, which are the combination of all the A2 belonging to that D1. For example the EPS, that is the first level decision #3, contains three A2 for the D2 #1 and three A2 for the D2 #2 (recall the tree in Figure 4.6). Nine L2 architectures related to it will be then evaluated. It is recalled that whenever the number of architectures exceed the total number of combinations (i.e. in this case 9), it means that the tool is selecting more than one A2 for a D2, as discussed in Section 4.2.3. Each red point in Figure 4.9 represents then a combination. Selecting the best L2 architecture for each decision, the overall architecture in Table 4.11 is obtained.

Table 4.11: Simulation 1 - Overall Architecture related to the 2-2-2-2 L1 Architecture

	L1	L2 D₂₁	L2 D₂₂
D1₁	2 = SPIN	1 = THR	2 = SS
D1₂	2 = ELEC	2 = ELTH	-
D1₃	2 = SP + BATT	2 = BM + WF	1 = Ni-Cd
D1₄	2 = LGA	1 = WIRES	-

The proposed best architecture is coherent with the requests, however it is noted that this preliminary prototype of the tool does not include a method to evaluate the influence of an alternative on the others. This is actually a decision, since the risk of introducing such relations is to stiffen the solver imposing strong constraints. As example, in this case, one may have inserted a condition telling that a spin stabilized satellite prohibits wings solar panels, throwing all the solutions that select them together. Such statement is

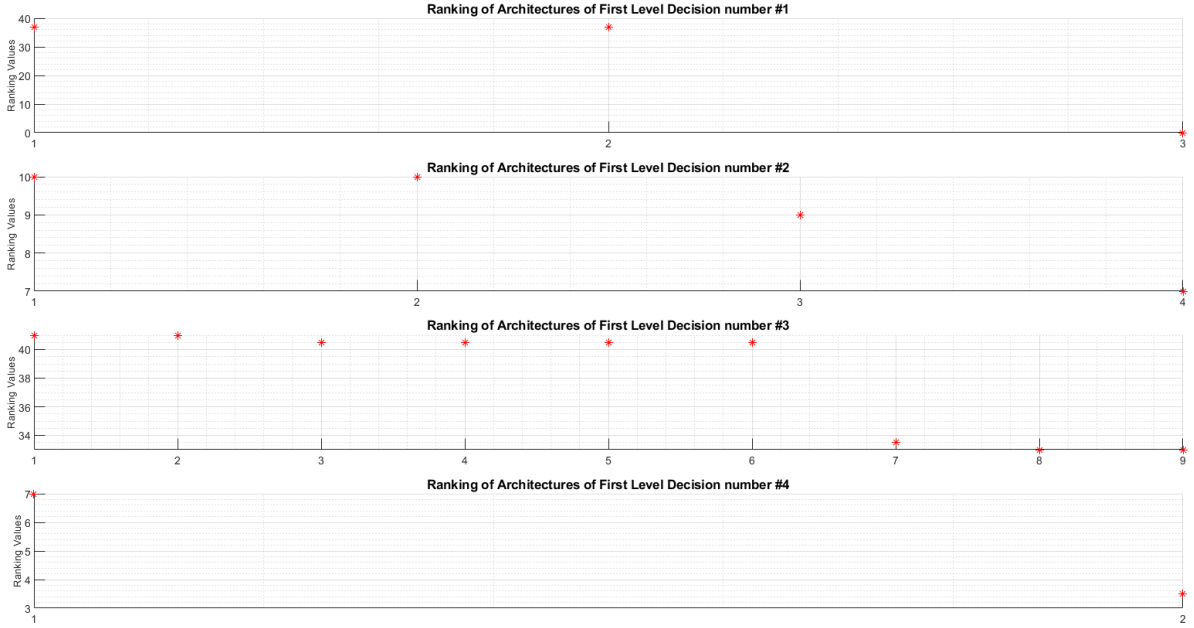


Figure 4.9: Simulation 1 - L2 Architectures related to the 2-2-2-2 L1 Architecture

actually not true as low velocity spinners can still mount deployable solar panels, therefore no conditions like it have been introduced in order to overconstrain the tool.

Simulation 2

For this simulation four functionalities are defined:

- F1 = Perform continuous imaging of the debris;
- F2 = Execute relative maneuvers;
- F3 = Transmit large data files to ground;
- F4 = Execute transfer to operative orbit.

They are more constraining with respect to those of the previous simulation, in particular the functionalities related to target pointing and relative maneuvers, as highlighted also in markers filled in Appendix B.1, Table B.2. F1 and F2 are also contemporary, therefore the \mathbf{F} matrix is filled as in Equation 4.3.2; more importance is also assigned to them with respect to F3 and F4, as the \mathbf{v}_{imp} in Equation 4.3.2 suggests. The Sparsity Factor is set to 1 and the MCDM method used is the Weighted Sum.

$$\mathbf{F}_{sim1} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad (4.27)$$

$$\mathbf{v}_{imp_{sim1}} = \begin{bmatrix} 0 & 1 & 0 & 0 \end{bmatrix} \quad (4.28)$$

The simulation is ran for the L1 architectures downselection and the results reported in Figure 4.10 are obtained. The algorithm clearly prioritizes those solutions that admit a 3-axis stabilization, as expected due to the multiple accurate pointing required.

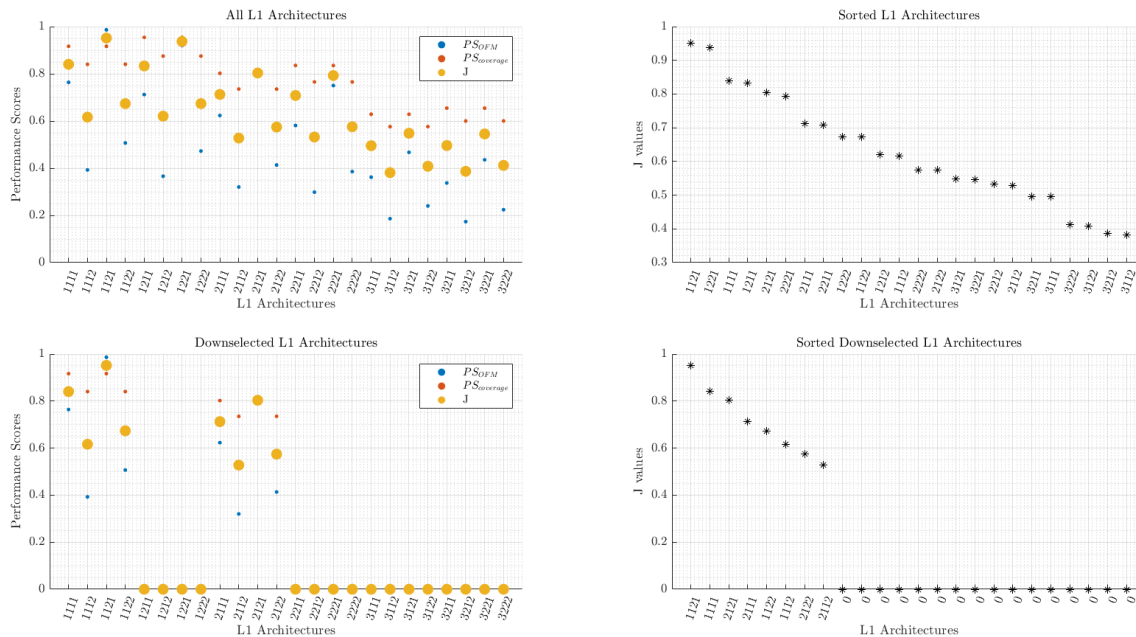


Figure 4.10: Simulation 2 - L1 Architectures

The best L1 architecture is the 1-1-2-1, that is 3-axis stabilization, chemical propulsion, solar panels + batteries and high gain antenna; the latter is preferred as the markers were filled asking for a high amount of data to be downloaded. This L1 architecture is selected to move to the L2 architectures selection, which outputs are furnished in Figure 4.11.

In this case, something happened for the L1 decision #1. The total number of A2 for the ADCS is 7 (3 coming from the actuators and 4 from the sensors), as illustrated in the tree in Figure 4.4, therefore one would expect a total of 12 combinations if the alternatives were selected in pairs. However 70 combinations are computed (p.n. some repeated solutions may be reported in the diagram), meaning that more than 2 alternatives are selected by the tool for what regard the ADCS. In fact, this is what happens for this simulation as the multiple requests for attitude precision knowledge require more than one kind of sensor, as the Table 4.12 reporting the best L2 architecture obtained indicates. The RW is selected due to the consistent slewing maneuvers requirements expressed in the form of marker as input; another actuator is of course needed to desaturate it, however the tool still does not implement a marker or a step that includes such kind of

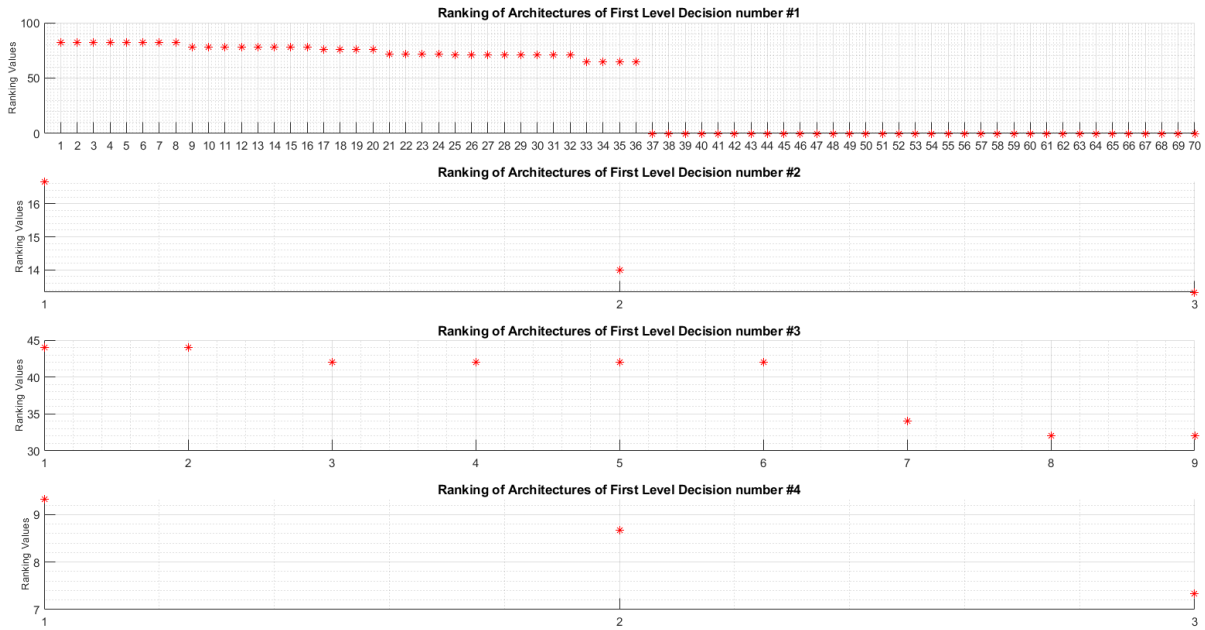


Figure 4.11: Simulation 2 - L2 Architectures related to the 1-1-2-1 L1 Architecture

finer considerations, which can be a step further for a future enhancement of the algorithm.

As any alternative in the tree has been modeled in Capella, the user can move to that library of components and check the grid of requirements proposed, also furnished as output. An example of such kind of modeled elements is reported in Figure 4.12, where the RW component with its functions alerts the presence of a requirement. The tool proposed again the Ni-Cd batteries as best alternative even though Li-Ion ones are typically engaged for small satellites; a further investigation on the markers assigned in the trees is then needed and eventually a data mining approach may be implemented using as data set previous small satellites mission, in order to optimally tune the overall embedded tree building, that up to now is manually compiled.

Table 4.12: Simulation 2 - Overall Architecture related to the 1-1-2-1 L1 Architecture

	L1	L2 D2₁	L2 D2₂ #1	L2 D2₂ #2
D1₁	1 = 3-AXIS	2 = RW	1 = ST	2 = SS
D1₂	1 = CHEM	1 = MONO	1 = Ni-Cd	-
D1₃	2 = SP + BATT	2 = BM + WF	1 = Ni-Cd	-
D1₄	1 = HGA	1 = PATCH	-	-

To conclude, the tool provides reliable results, even though the user should approach the solutions cautiously for the reasons expressed before. The best way to exploit such preliminary version of the tool is to associate it to some quantitative analysis and architecture design to assess the feasibility of the proposed architectures. The MBSE environment

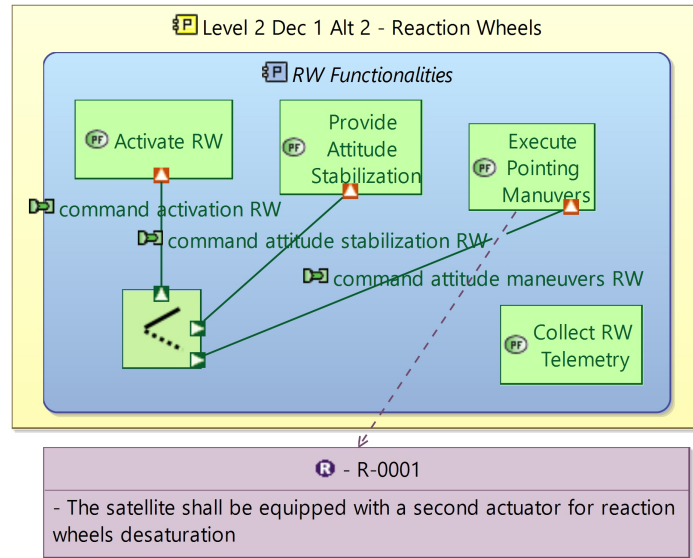


Figure 4.12: Reaction Wheel Node Component for Decision-Making Tool

should also be targeted to exploit all the outputs, as in the case seen before about the RW. Moreover, an MBSE solution such as Capella improves the system thinking providing a natural terrain to experiment with functional analysis having a set of components, as described in the Step 10 of Section 4.2.3, therefore an iterative approach which exploits Capella and the developed decision-making tool can be thought.

Chapter 5

Conclusions and Future Work

This thesis investigated the adoption of an MBSE methodology applied to small satellites mission, merged with a prototype version of a decision-making tool for preliminary architectures automatic generation and downselection, to enhance the entire system life-cycle. This final chapter provides a summary of the achieved results and proposes some recommendations to extend the study for future developments.

5.1 Summary of the Results

The research conducted within this work set out to improve the small satellites design lifecycle using an MBSE solution in order to assess whether it is worth applying such approach to this kind of systems. Through a complete case study of the ESA e.Inspector mission Phase A, it was demonstrated that implementing MBSE from the beginning of a project and using a defined methodology to enable the capture of all system aspects, results to be effective and suitable for a complex space system. The precise syntax and semantics of the ARCADIA language, merged with the powerful Capella tool, allow to express complex concepts and articulated architectures in a concise and intuitive way, coherently with the main systems engineering drivers which are requirements. The methodology accompanies systems engineers in their definition from the very high level mission objectives up to the components definition, guaranteeing consistency among levels and providing a clear vision of the entire system to any involved team member and/or stakeholder. Additionally, the model is not a standalone product used just once in the system lifecycle, but continuously evolves as the design proceeds. Concerning the small satellites field of applicability, it also provides a strong basis for the on-board software development thanks to its object-oriented nature. This study represents a step forward for the MBSE community, not only space-related, as it reports a complete analysis of an MBSE solution applied to a multidisciplinary complex system assessing its feasibility and highlighting some lacks and open points, and is in line with the INCOSE MBSE roadmap reported in Figure 5.1.

This thesis also extends the capabilities of usual systems engineering approaches intro-

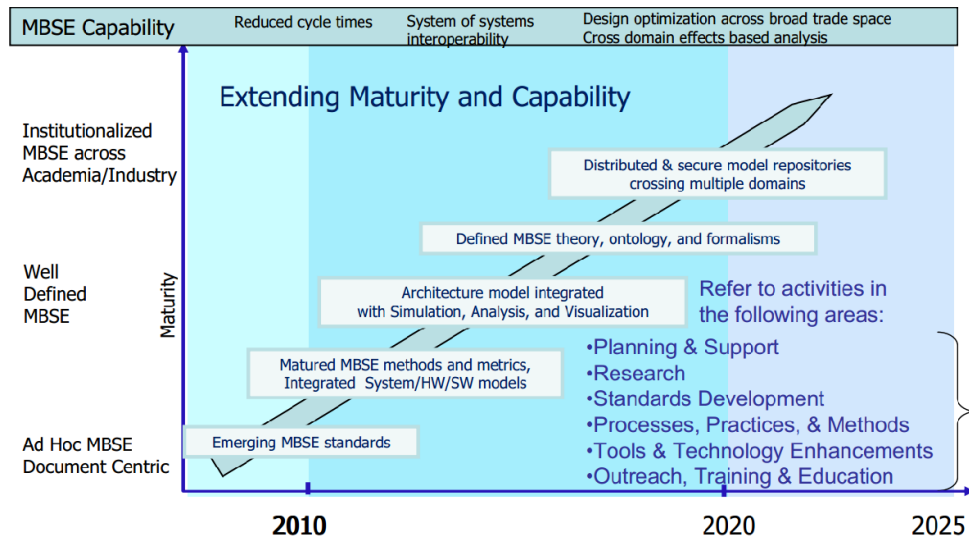


Figure 2: INCOSE MBSE Roadmap [3]

Figure 5.1: INCOSE MBSE Roadmap [71]

ducing a decision-making algorithm for the selection of one or more preliminary architectures, intended to be used in the feasibility study of small satellites when it is difficult to reduce the number of design alternatives due to the highly qualitative domain. The tool has been validated and results are promising, highlighting its ability to skim the architectures basing on the inputs provided in the form of functionalities addressed to the system to be developed. A way to merge the tool with the MBSE environment enhancing the overall mission design has been presented too.

5.2 Limitations of the Study and Future Developments

The work related to the MBSE approach presents some limitations that actually can also be interpreted as future works. Firstly, the study excluded the parameterization of the whole architecture model and a consequent interface with an analytical and numerical tool to run simulations, as ARCADIA/Capella do not provide this kind of interface. The nearest ARCADIA concept to such parametrization is the adoption of Class diagrams to precisely model quantities exchanged between functions and so components, and to have a data repository within the model. Such diagrams were not investigated within this work mainly due to early stage of the project, however a future development of the model for the mission should include them too.

One of the aspects emerged from the MBSE case study is the absence of dedicated requirements diagrams for the trees generation; actually for this work a solution to solve this issue was found, that is the adoption of OAB diagrams. However, the model would benefit from having a dedicated set of diagrams to enhance the requirements modeling,

therefore representing an open point for the future. The same considerations can be applied to the AIV/AIT plan development, with dedicated functions and model elements. A possibility is to develop both of them in the form of Capella add-ons (or viewpoints).

Concerning the proposed decision-making tool, the presented prototype version opens the road to many future developments. Firstly, a more formal interface with an MBSE tool, such as Capella, can be developed. The embedded decision tree can be improved increasing its details and revising the assigned markers using machine learning techniques and data mining that exploit a statistical set of data built up from the literature information on past concluded space missions, addressing a more precise matrices filling with respect to the current manually compiled ones. Also, new blocks can be introduced to the current algorithm such as a cross-relation block to evaluate how the selection of a particular component influences the others, being careful to not stiffen the tool introducing too much constraining conditions. Other interesting developments concern the introduction of sizing blocks which implement mission analysis and basic computations of subsystems parameters, in order to get as output a preliminary quantitative sizing too. Such blocks could be used to add some more decision-making conditions expanding the components selection to an available catalog of COTS, also determining their number for each alternative, leading to a more complete preliminary architecture with relative sizing of the system.

5.3 Final Thoughts

Although MBSE still has many social hurdles to overcome, the author expects a gradual awareness from the space community about the benefits a system design lifecycle can gain from it, as demonstrated in this work. Interfacing MBSE solutions with intelligent tools such as the prototype one developed for this thesis represents a way to overcome the stringent requirements asked by the new space systems and to face up the less relaxed development times required by the incoming space economy.

Appendix A

Appendix - Capella Diagrams

A.1 Requirements Trees

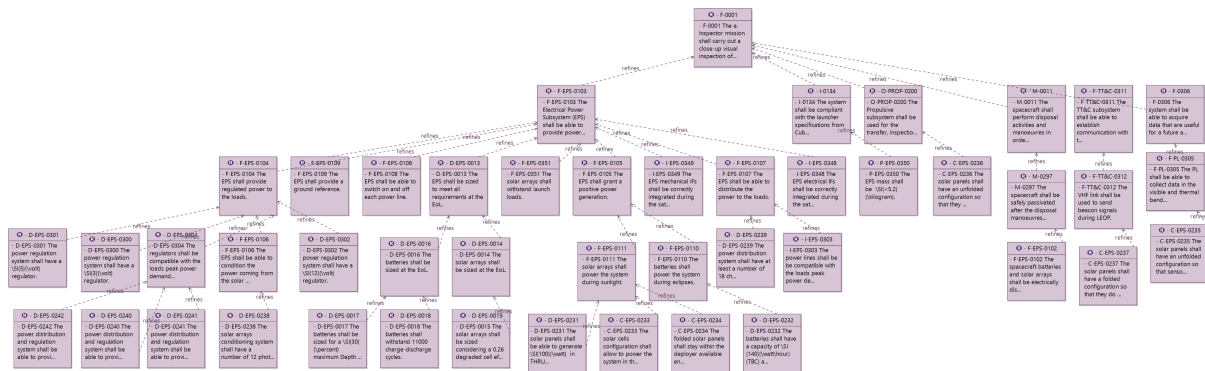


Figure A.1: [OAB] Requirements - EPS

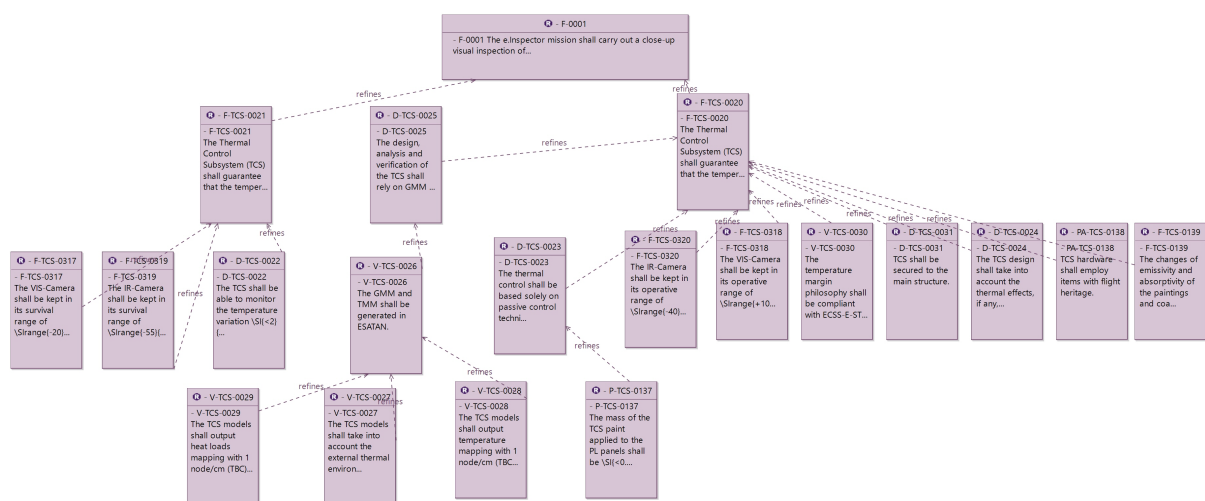


Figure A.2: [OAB] Requirements - TCS

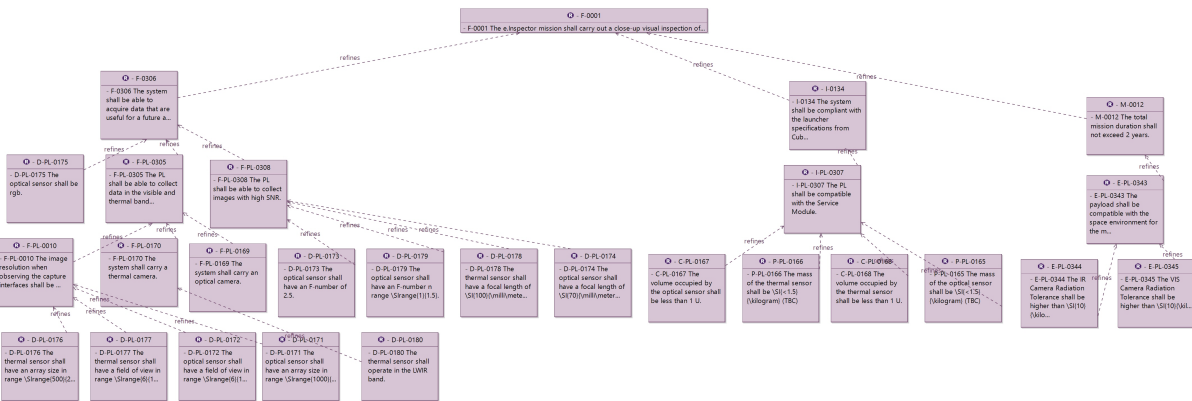


Figure A.6: [OAB] Requirements - PL

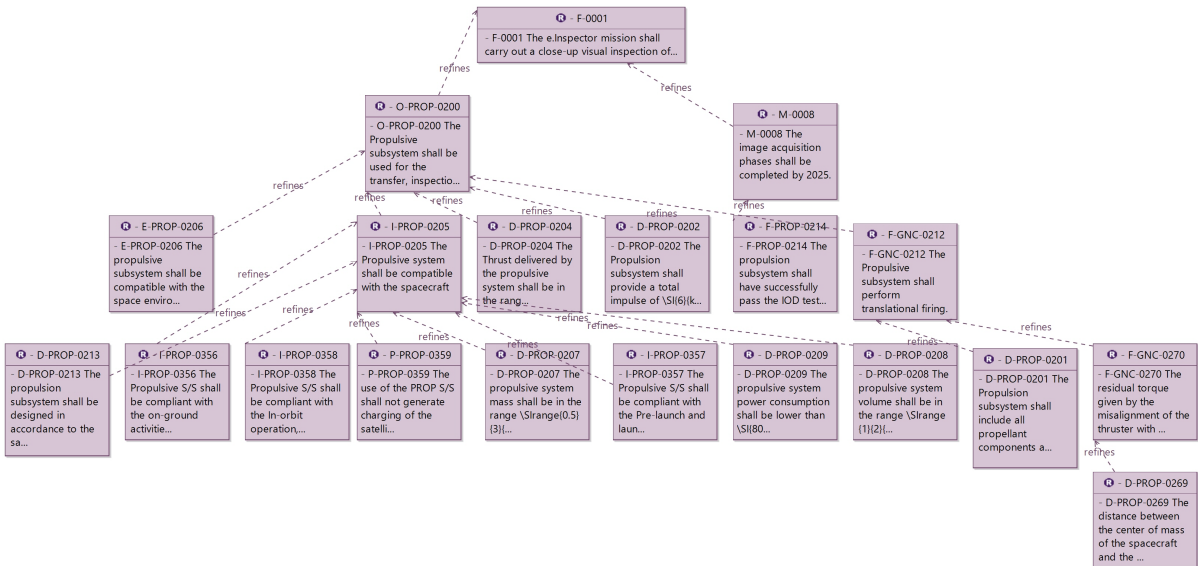


Figure A.7: [OAB] Requirements - PROP

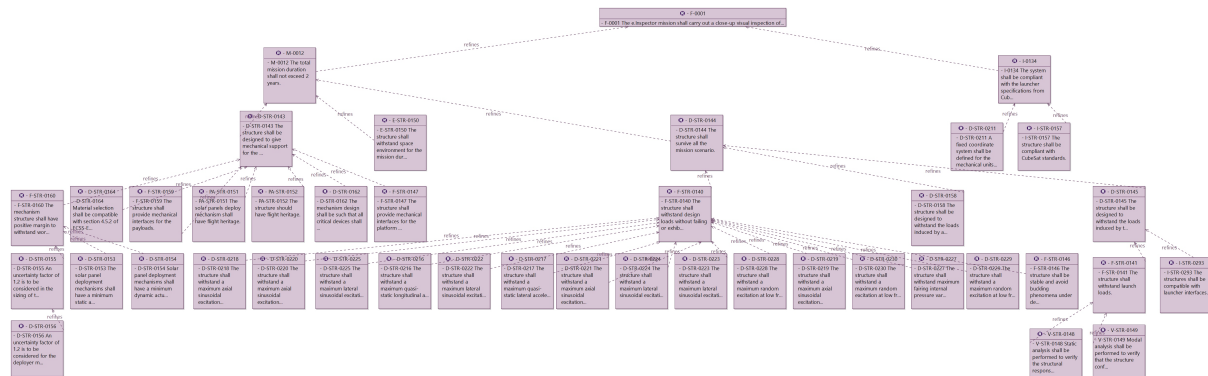


Figure A.8: [OAB] Requirements - STR&MECH

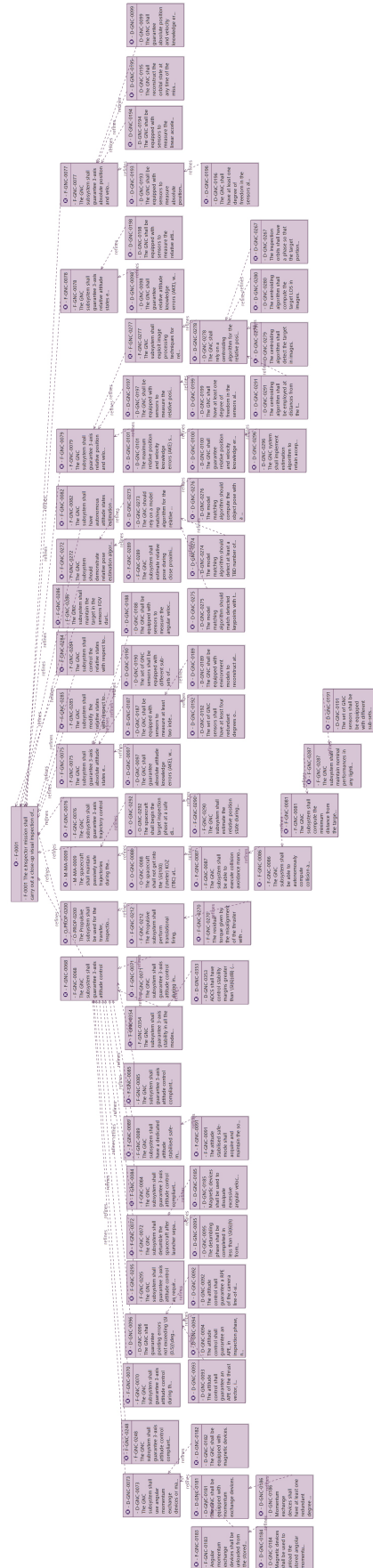


Figure A.9: [OAB] Requirements - GNC

A.2 System Analysis Diagrams

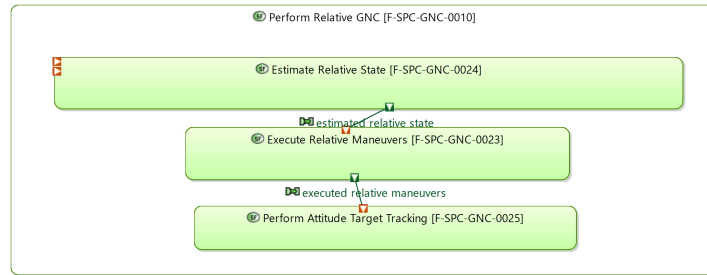


Figure A.10: [SDFB] Approach Target Debris

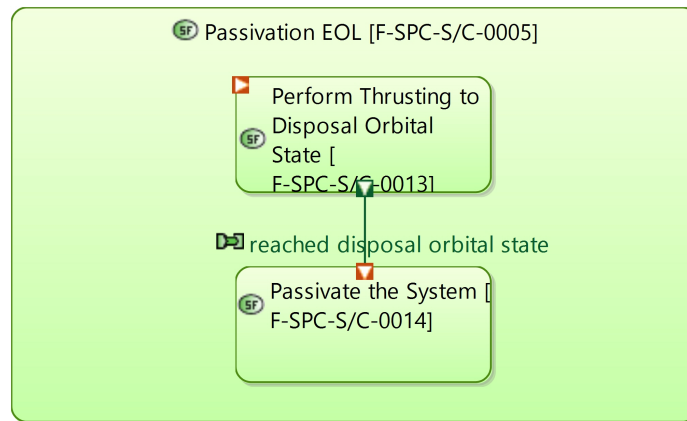


Figure A.11: [SDFB] Deorbit at EOL

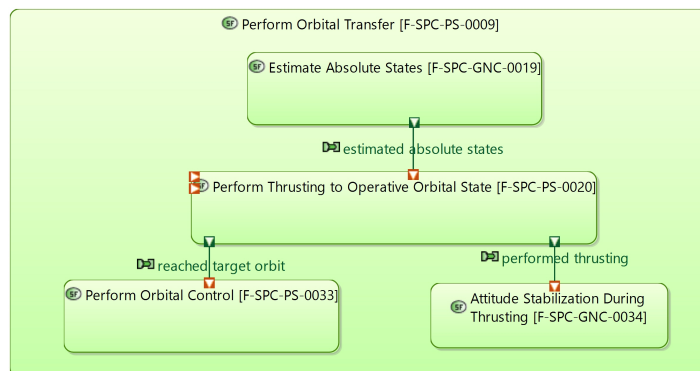


Figure A.12: [SDFB] Provide Propulsion

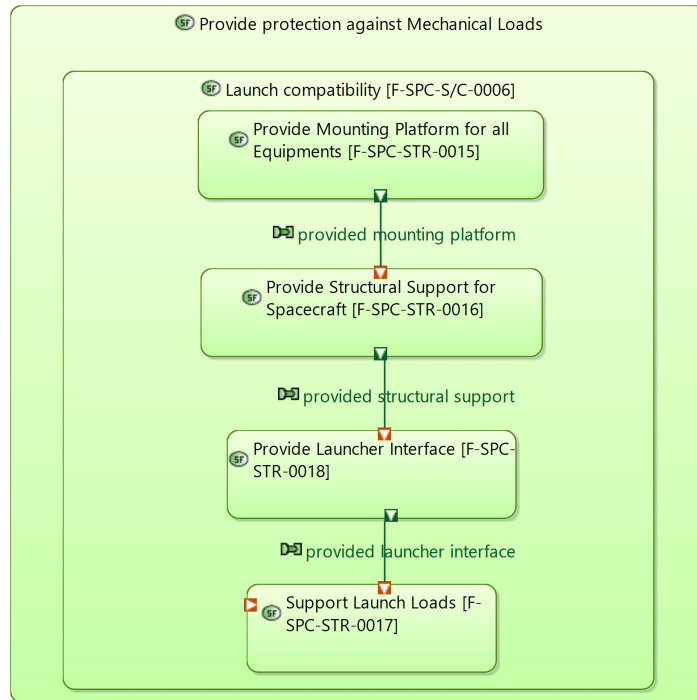


Figure A.13: [SDFB] Provide Protection against Mechanical Loads

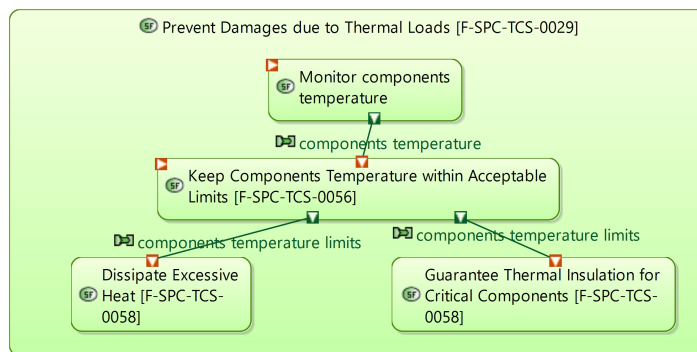


Figure A.14: [SDFB] Provide Protection against Temperature

A.3 Logical Architecture Diagrams

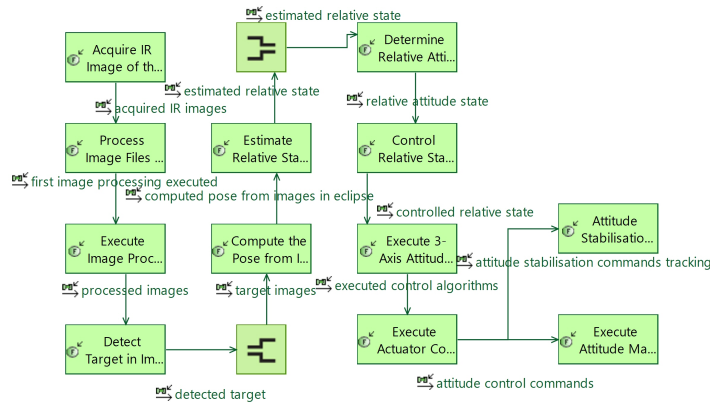


Figure A.15: [LFCD] GNC SS - Close Proximity Relative Navigation during Eclipse

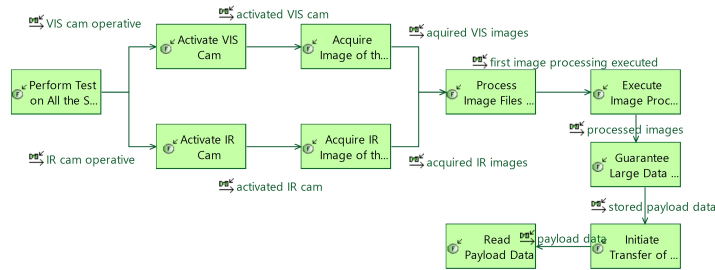


Figure A.16: [LFCD] OBDH SS - Payload Data Acquisition, Storage and Transmission

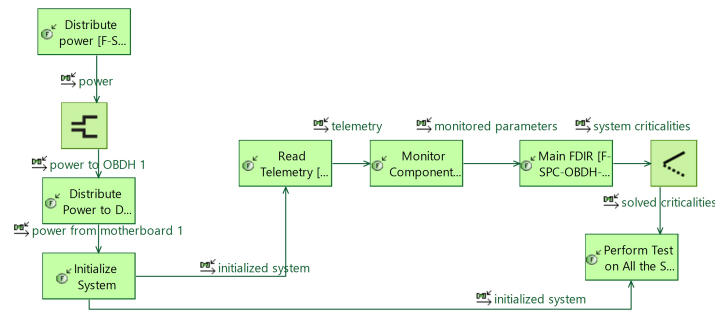


Figure A.17: [LFCD] OBDH SS - System Initialization

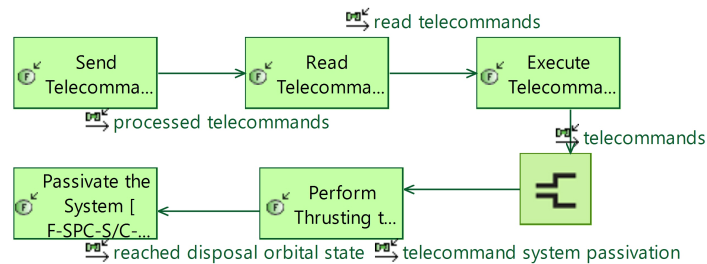


Figure A.18: [LFCD] OBDH SS - System Passivation

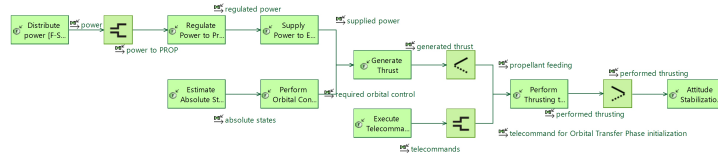


Figure A.19: [LFCD] PROP SS - Thrusting to Drifting Orbit Initialization

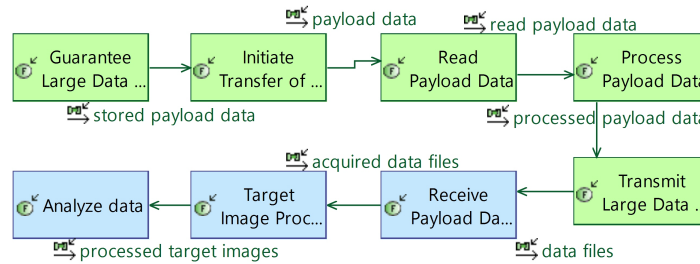


Figure A.20: [LFCD] TT&C SS - Acquired Target Images Downlink Line

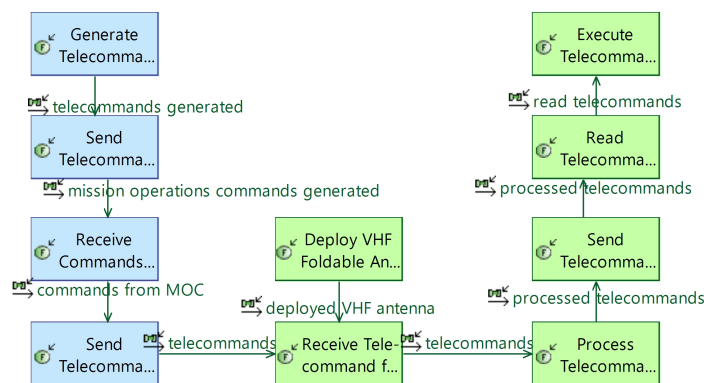


Figure A.21: [LFCD] TT&C SS - Telecommands Uplink Line

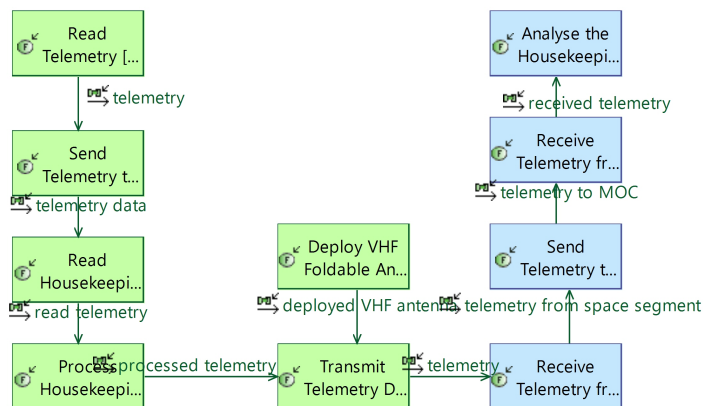


Figure A.22: [LFCD] TT&C SS - Telemetry Downlink Line

A.4 AIV/AIT Diagrams

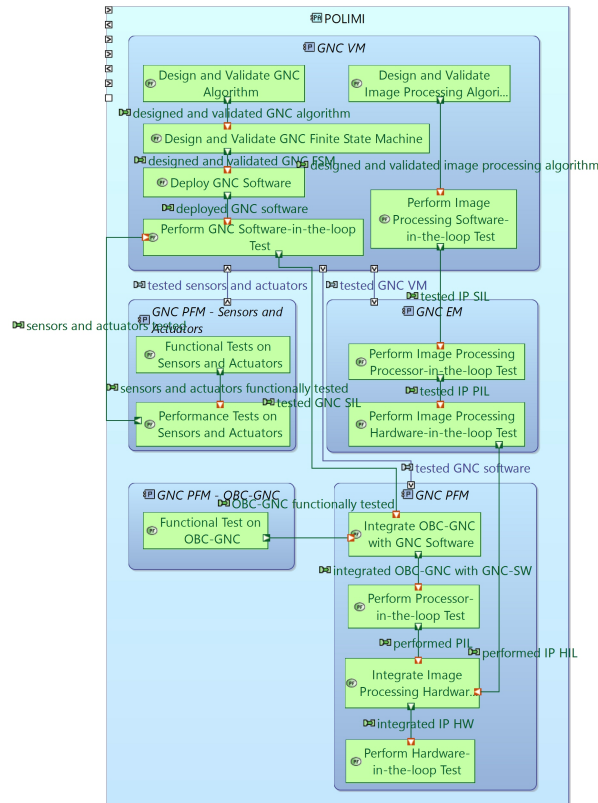


Figure A.23: [PAB] AIV/AIT GNC - Overall Plan

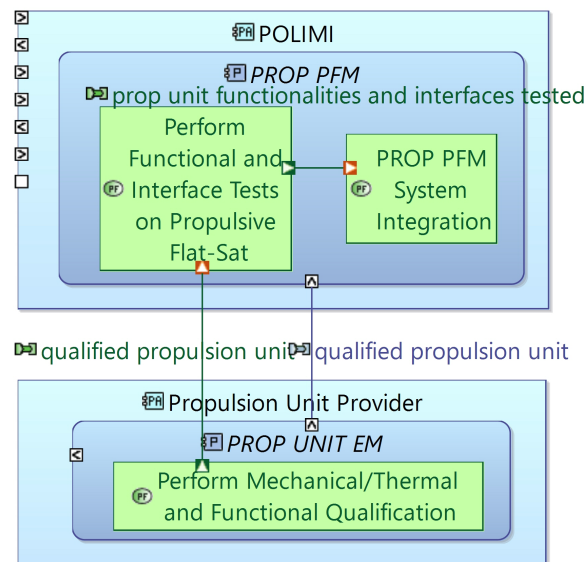


Figure A.24: [PAB] AIV/AIT PROP - Overall Plan

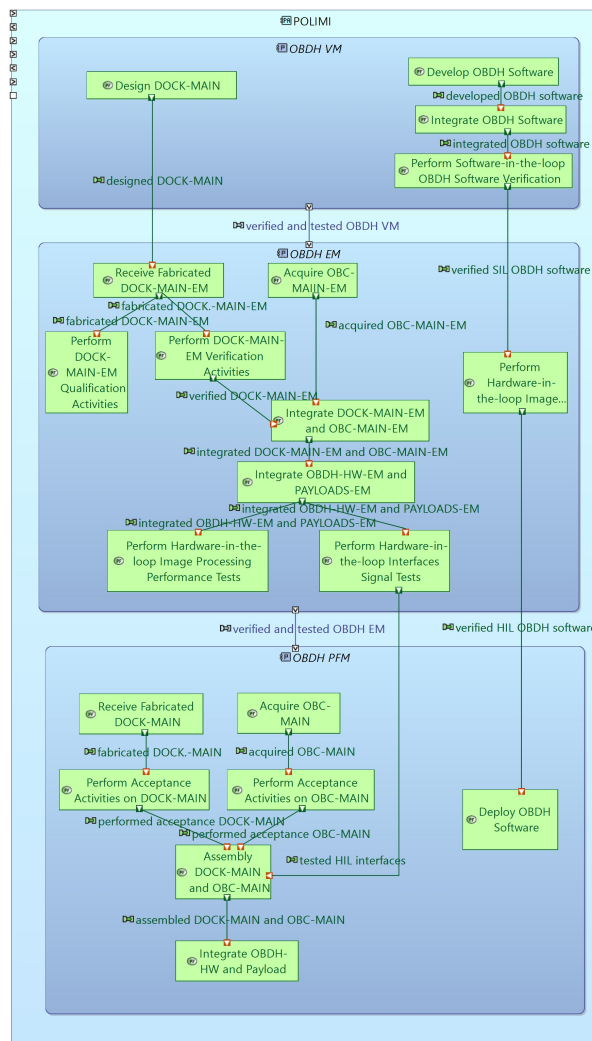


Figure A.25: [PAB] AIV/AIT OBDH - Overall Plan

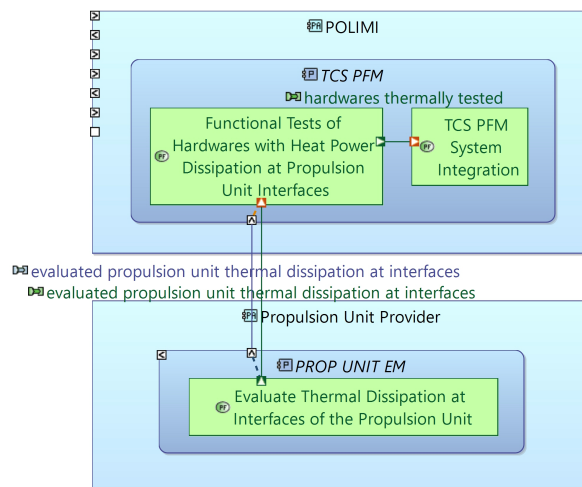


Figure A.26: [PAB] AIV/AIT TCS - Overall Plan

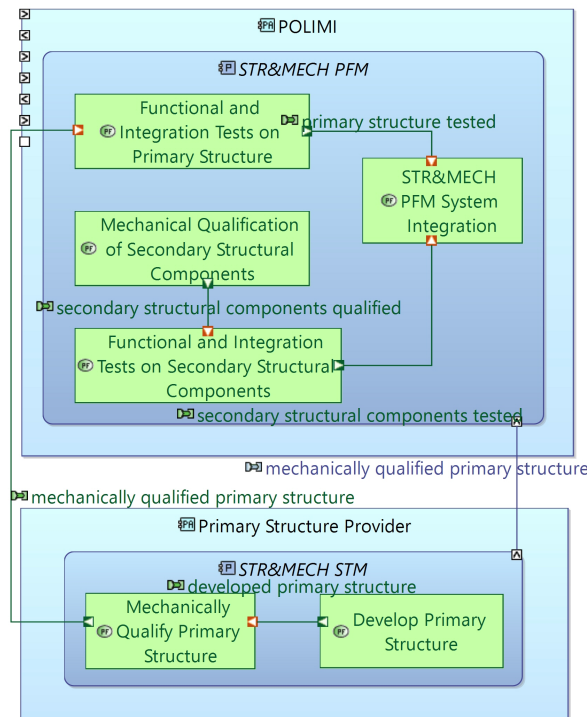


Figure A.27: [PAB] AIV/AIT STR&MECH - Overall Plan

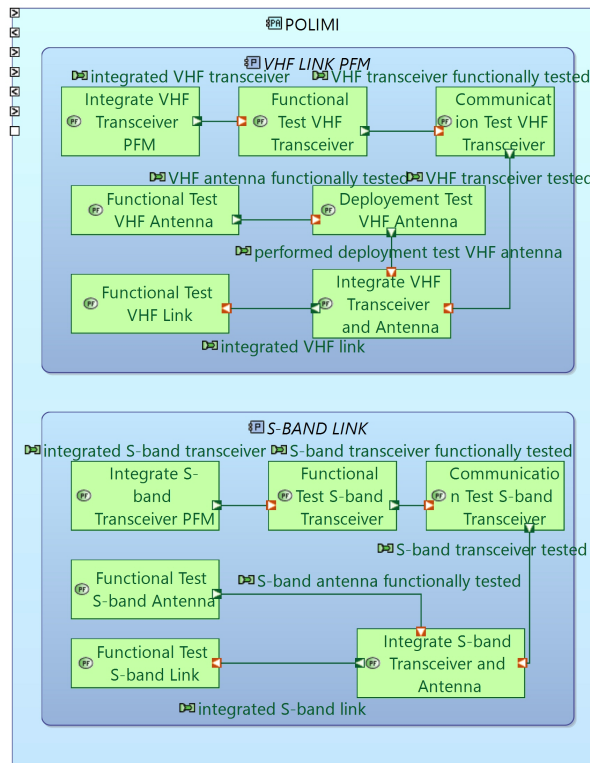


Figure A.28: [PAB] AIV/AIT TT&C - Overall Plan

Appendix B

Appendix - Decision-Making Tool

B.1 Functionalities and Alternatives Markers

Table B.1: Functionalities Markers of Simulation 1

	F 1	F 2	F 3
Nadir Pointing (e.g. Earth)	0	1	0
Inertial Target (e.g. Star, Sun)	0	0	1
Relative Target Pointing (e.g. debris)	0	0	0
Multiple pointing required	0	0	0
Knowledge Accuracy > 5°	0	0	1
Knowledge Accuracy < 5°	0	0	0
Knowledge Accuracy < 1°	0	0	0
Pointing Accuracy (actuators)	2	2	3
Maneuverability (attitude slewing)	2	2	2
Controllable over 6000 km orbit (Earth)	0	0	0
Controllable over 1000 km orbit (Earth)	0	0	1
Mass (NB)	3	3	3
Power required (NB)	4	4	4
Complexity (NB)	4	4	4
Lifetime	2	2	2
Time to perform maneuvers (NB)	0	0	0
Thrust-to-weight ratio	3	0	0
Multiple starts	1	0	0
Eclipses present	1	1	1
Temperature sensitivity (NB)	3	3	3
Sensitivity to Sun Angle (NB)	3	3	3
Power produced	2	2	2
Volume (NB)	4	4	4
Amount of data	0	2	0

Table B.2: Functionalities Markers of Simulation 2

	F 1	F 2	F 3	F 4
Nadir Pointing (e.g. Earth)	0	0	1	0
Inertial Target (e.g. Star, Sun)	1	1	0	0
Relative Target Pointing (e.g. debris)	1	0	0	0
Multiple pointing required	0	1	1	1
Knowledge Accuracy > 5°	0	0	1	0
Knowledge Accuracy < 5°	0	1	1	1
Knowledge Accuracy < 1°	1	0	0	0
Pointing Accuracy (actuators)	4	4	3	3
Maneuverability (attitude slewing)	4	3	2	2
Controllable over 6000 km orbit (Earth)	0	0	0	0
Controllable over 1000 km orbit (Earth)	0	0	0	0
Mass (NB)	3	3	3	3
Power required (NB)	3	3	3	3
Complexity (NB)	2	2	2	2
Lifetime	2	2	2	2
Time to perform maneuvers (NB)	0	4	0	3
Thrust-to-weight ratio	0	4	0	3
Multiple starts	0	1	0	1
Eclipses present	0	0	0	0
Temperature sensitivity (NB)	3	3	3	3
Sensitivity to Sun Angle (NB)	2	2	2	2
Power produced	4	4	4	4
Volume (NB)	3	3	3	3
Amount of data	4	0	4	0

Table B.3: ADCS L1 and L2 Alternatives Markers

	LEV 1 - ADCS				LEV 2 - ADCS - ACTUATORS				LEV 2 - ADCS - SENSORS			
	3-AXIS	SPIN	PASSIVE	THR	RW	MGTOR	MAG/BO	ST	SS	HS	MGTM	
Nadir Pointing (e.g. Earth)	1	1	1	1	1	1	1	1	1	1	1	
Inertial Target (e.g. Star, Sun)	1	1	1	1	1	1	1	1	1	1	1	
Relative Target Pointing (e.g. debris)	1	0	0	1	1	1	0	1	1	1	1	
Multiple pointing required	1	0	0	1	1	1	0	1	1	1	1	
Knowledge Accuracy $>5^\circ$	1	1	1	0	0	0	0	0	1	1	1	
Knowledge Accuracy $<5^\circ$	1	1	0	0	0	0	0	0	1	1	1	
Knowledge Accuracy $<1^\circ$	1	0	0	0	0	0	0	1	0	0	0	
Pointing Accuracy (actuators)	4	3	2	3	4	2	2	0	0	0	0	
Maneuverability (attitude slewing)	4	2	0	3	4	2	0	0	0	0	0	
Controllable over 6000 km orbit (Earth)	1	1	0	1	1	1	0	1	1	1	0	
Controllable over 1000 km orbit (Earth)	1	1	0	1	1	1	1	1	1	1	0	
Mass (NB)	2	3	4	3	2	4	4	2	3	3	4	
Power required (NB)	2	3	4	3	2	4	4	2	3	3	4	
Complexity (NB)	2	4	4	3	2	4	4	2	3	3	4	
Lifetime	3	3	4	2	3	4	4	3	3	3	3	
Time to perform maneuvers (NB)	0	0	0	0	0	0	0	0	0	0	0	
Thrust-to-weight ratio	0	0	0	0	0	0	0	0	0	0	0	
Multiple starts	0	0	0	0	0	0	0	0	0	0	0	
Eclipses present	0	0	0	0	0	0	0	0	0	0	0	
Temperature sensitivity (NB)	0	0	0	0	0	0	0	0	0	0	0	
Sensitivity to Sun Angle (NB)	0	0	0	0	0	0	0	0	0	0	0	
Power produced	0	0	0	0	0	0	0	0	0	0	0	
Volume (NB)	0	0	0	0	0	0	0	0	0	0	0	
Amount of data	0	0	0	0	0	0	0	0	0	0	0	

Table B.4: Propulsion Subsystem L1 and L2 Alternatives Markers

	LEV 1 - PROP			LEV 2 - PROP - GREEN			LEV 2 - PROP - CHEMICAL			LEV 2 - PROP - ELECTRIC		
	CHEM	ELEC	MONO	GREEN	CG	SOL	ELTHE	ELSPR	ION	HALL		
Nadir Pointing (e.g. Earth)	0	0	0	0	0	0	0	0	0	0		
Inertial Target (e.g. Star, Sun)	0	0	0	0	0	0	0	0	0	0		
Relative Target Pointing (e.g. debris)	0	0	0	0	0	0	0	0	0	0		
Multiple pointing required	0	0	0	0	0	0	0	0	0	0		
Knowledge Accuracy > 5°	0	0	0	0	0	0	0	0	0	0		
Knowledge Accuracy < 5°	0	0	0	0	0	0	0	0	0	0		
Knowledge Accuracy < 1°	0	0	0	0	0	0	0	0	0	0		
Pointing Accuracy (actuators)	0	0	0	0	0	0	0	0	0	0		
Maneuverability (attitude slewing)	0	0	0	0	0	0	0	0	0	0		
Controllable over 6000 km orbit (Earth)	0	0	0	0	0	0	0	0	0	0		
Controllable over 1000 km orbit (Earth)	0	0	0	0	0	0	0	0	0	0		
Mass (NB)	0	0	0	0	0	0	0	0	0	0		
Power required (NB)	2	4	3	2	4	3	3	3	3	2		
Complexity (NB)	3	2	3	2	4	3	4	3	2	2		
Lifetime	2	3	2	4	3	2	2	3	3	4		
Time to perform maneuvers (NB)	4	2	3	3	2	4	4	2	2	3		
Thrust-to-weight ratio	4	2	3	4	2	4	4	2	3	3		
Multiple starts	1	1	1	1	1	0	1	1	1	1		
Eclipses present	0	0	0	0	0	0	0	0	0	0		
Temperature sensitivity (NB)	0	0	0	0	0	0	0	0	0	0		
Sensitivity to Sun Angle (NB)	0	0	0	0	0	0	0	0	0	0		
Power produced	0	0	0	0	0	0	0	0	0	0		
Volume (NB)	0	0	0	0	0	0	0	0	0	0		
Amount of data	0	0	0	0	0	0	0	0	0	0		

Table B.5: EPS L1 and L2 Alternatives Markers

	LEV 1 - EPS			LEV 2 - SOLAR PANELS			LEV 2 - BATT		
	SP	SP+BATT	BM	BM+WF	BM+WG	Ni-Cd	Li-Ion	Li-Pol	
Nadir Pointing (e.g. Earth)	0	0	0	0	0	0	0	0	
Inertial Target (e.g. Star, Sun)	0	0	0	0	0	0	0	0	
Relative Target Pointing (e.g. debris)	0	0	0	0	0	0	0	0	
Multiple pointing required	0	0	0	0	0	0	0	0	
Knowledge Accuracy > 5°	0	0	0	0	0	0	0	0	
Knowledge Accuracy < 5°	0	0	0	0	0	0	0	0	
Knowledge Accuracy < 1°	0	0	0	0	0	0	0	0	
Pointing Accuracy (actuators)	0	0	0	0	0	0	0	0	
Maneuverability (attitude slewing)	0	0	0	0	0	0	0	0	
Controllable over 6000 km orbit (Earth)	0	0	0	0	0	0	0	0	
Controllable over 1000 km orbit (Earth)	0	0	0	0	0	0	0	0	
Mass (NB)	4	2	4	2	2	2	4	3	
Power required (NB)	0	0	0	0	0	0	0	0	
Complexity (NB)	4	2	4	3	2	3	3	4	
Lifetime	0	3	0	0	0	3	4	4	
Time to perform maneuvers (NB)	0	0	0	0	0	0	0	0	
Thrust-to-weight ratio	0	0	0	0	0	0	0	0	
Multiple starts	0	0	0	0	0	0	0	0	
Eclipses present	0	1	1	1	1	1	1	1	
Temperature sensitivity (NB)	3	3	2	3	4	4	3	3	
Sensitivity to Sun Angle (NB)	3	3	2	3	4	0	0	0	
Power produced	3	3	2	4	4	3	4	3	
Volume (NB)	4	2	4	3	3	0	0	0	
Amount of data	0	0	0	0	0	0	0	0	

Table B.6: TT&C L1 and L2 Alternatives Markers

	LEV 1 - TT&C		LEV 2 - HGA				LEV 2 - LGA	
	HGA	LGA	PATCH	RARR	REFL	WIRES	HEL	
Nadir Pointing (e.g. Earth)	0	0	0	0	0	0	0	
Inertial Target (e.g. Star, Sun)	0	0	0	0	0	0	0	
Relative Target Pointing (e.g. debris)	0	0	0	0	0	0	0	
Multiple pointing required	0	0	0	0	0	0	0	
Knowledge Accuracy > 5°	0	0	0	0	0	0	0	
Knowledge Accuracy < 5°	0	0	0	0	0	0	0	
Knowledge Accuracy < 1°	0	0	0	0	0	0	0	
Pointing Accuracy (actuators)	0	0	0	0	0	0	0	
Maneuverability (attitude slewing)	0	0	0	0	0	0	0	
Controllable over 6000 km orbit (Earth)	0	0	0	0	0	0	0	
Controllable over 1000 km orbit (Earth)	0	0	0	0	0	0	0	
Mass (NB)	0	0	0	0	0	0	0	
Power required (NB)	0	0	0	0	0	0	0	
Complexity (NB)	2	4	4	3	2	4	3	
Lifetime	0	0	0	0	0	0	0	
Time to perform maneuvers (NB)	0	0	0	0	0	0	0	
Thrust-to-weight ratio	0	0	0	0	0	0	0	
Multiple starts	0	0	0	0	0	0	0	
Eclipses present	0	0	0	0	0	0	0	
Temperature sensitivity (NB)	0	0	0	0	0	0	0	
Sensitivity to Sun Angle (NB)	0	0	0	0	0	0	0	
Power produced	0	0	0	0	0	0	0	
Volume (NB)	3	4	4	3	2	4	2	
Amount of data	4	2	3	3	4	3	4	

B.2 Algorithms

Algorithm 2: STEP 1 - Markers clustering and AFM and *coverage* computation.

Input: IFM = Input Functionalities Matrix, $D1$ = Level 1 Decisions and Alternatives Markers

Output: AFM, *coverage*

// Begin

n = Number of Markers

m = Number of Functionalities

l = Number of D1

for $w = 1 \rightarrow l$ **do**

p_w = Number of Alternatives in the w -th D1

for $k = 1 \rightarrow p_w$ **do**

for $j = 1 \rightarrow m$ **do**

for $i = 1 \rightarrow n$ **do**

if $IFM(i, j) = 0$ **then**

$AFM\{w\}(i, k, j) = 0$

$coverage\{w\}(i, k, j) = 0$

else if $IFM(i, j) \neq 0 \wedge D1\{w\}(i, k) = 0$ **then**

$AFM\{w\}(i, k, j) = 0$

$coverage\{w\}(i, k, j) = 0$

else if $IFM(i, j) = 1 \wedge D1\{w\}(i, k) = 1$ **then**

$AFM\{w\}(i, k, j) = 1$

$coverage\{w\}(i, k, j) = 1$

else

if $IFM(i, j) - D1\{w\}(i, k) = 0$ **then**

$AFM\{w\}(i, k, j) = 1$

$coverage\{w\}(i, k, j) = 1$

else if $IFM(i, j) - D1\{w\}(i, k) > 0$ **then**

$AFM\{w\}(i, k, j) = 1 - \frac{|IFM(i, j) - D1\{w\}(i, k)|}{3}$

$coverage\{w\}(i, k, j) = 0.5$

else if $IFM(i, j) - D1\{w\}(i, k) < 0$ **then**

$AFM\{w\}(i, k, j) = (1 - \frac{|IFM(i, j) - D1\{w\}(i, k)|}{3}) \cdot 1.1$

$coverage\{w\}(i, k, j) = 1$

end

end

end

end

end

// End

Algorithm 3: STEP 2 - Computation of OFM and *CoverageAlternatives*, check on contemporary functionalities.

Input: AFM, *coverage*, F

Output: OFM, *CoverageAlternatives*

// Begin

n = Number of Markers

m = Number of Functionalities

l = Number of D1

for $w = 1 \rightarrow l$ **do**

p_w = Number of Alternatives in the w -th D1

for $k = 1 \rightarrow p_w$ **do**

for $j = 1 \rightarrow m$ **do**

$$OFM\{w\}(k, j) = \frac{\sum_{i=1}^n AFM\{w\}(i, k, j)}{n}$$

$$coverage_{alternatives}\{w\}(k, j) = \sum_{i=1}^n coverage\{w\}(i, k, j)$$

end

end

end

// Check on contemporary functionalities

for $j1 = 1 \rightarrow m$ **do**

for $j2 = 1 \rightarrow m$ **do**

if $F(j1, j2) = 1$ **then**

for $w = 1 \rightarrow l$ **do**

p_w = Number of Alternatives in the w -th D1

for $k = 1 \rightarrow p_w$ **do**

if $OFM\{w\}(k, j1) = 0 \wedge coverage_{alternatives}\{w\}(k, j1) = 0$ **then**

$OFM\{w\}(k, j1) = 0$

$coverage_{alternatives}\{w\}(k, j1) = 0$

$OFM\{w\}(k, j2) = 0$

$coverage_{alternatives}\{w\}(k, j2) = 0$

if $OFM\{w\}(k, j2) = 0 \wedge coverage_{alternatives}\{w\}(k, j2) = 0$ **then**

$OFM\{w\}(k, j2) = 0$

$coverage_{alternatives}\{w\}(k, j2) = 0$

$OFM\{w\}(k, j1) = 0$

$coverage_{alternatives}\{w\}(k, j1) = 0$

end

end

end

end

// End

Algorithm 4: STEP 6 - L1 Architectures Skimming and *Final Proposed Architectures*

Input: *coverage*, OFM, IFM, *SortedIndexes_{ARCHI}* = Sorted Indexes of Alternatives of Each Architecture (i.e. [1 2 1 2])

Output: FPA

// Begin

n = Number of Markers

m = Number of Functionalities

l = Number of D1

n_{archi} = Number of L1 Architectures

// Compute *covered_{markers}*

for *f* = 1 → *n_{archi}* **do**

for *w* = 1 → *l* **do**

for *j* = 1 → *m* **do**

for *i* = 1 → *n* **do**

if *coverage*{*w*}(*i*, *SortedIndexes_{ARCHI}*(*f*, *w*), *j*) ≠ 0 **then**

 | *covered_{markers}*{*f*}(*i*, *j*) = 1

else

 | *covered_{markers}*{*f*}(*i*, *j*) = 0

end

end

end

end

// Exclude non compliant architectures

for *f* = 1 → *n_{archi}* **do**

for *j* = 1 → *m* **do**

for *i* = 1 → *n* **do**

if *covered_{markers}*{*f*}(*i*, :) = 0 ∨ *IFM*(*i*, *j*) ≠ 0 **then**

 | *J*(*f*) = 0

 | *SortedIndexes_{ARCHI}*(*f*, :) = 0

end

end

end

Delete all architectures having at least one marker of an alternative not satisfied

by at least one alternative of the L1 architecture;

Rank the architectures according to the *J* values and provide the associated

indexes of the alternatives;

// The obtained architectures represent the FPA

// End

Algorithm 5: STEP 7 - Satisfaction degree computation of second level alternatives.

Input: AFM, D1, D2 = Level 1 Decisions and Alternatives Markers, architecture

Output: *satisfaction*, *SatisfactionTotal*

// Begin

n = Number of Markers

m = Number of Functionalities

l = Number of D1

for $w = 1 \rightarrow l$ **do**

$alt_1 = architecture(w)$

$alt_2 = D2\{w\}\{alt_1\}$

d_{alt_1} = Number of D2 belonging to the alt_1 under cycle

for $h = 1 \rightarrow d_{alt_1}$ **do**

q_h = Number of A2 belonging to the D2 under cycle

for $g = 1 \rightarrow q_h$ **do**

for $j = 1 \rightarrow m$ **do**

for $i = 1 \rightarrow n$ **do**

if $AFM\{w\}(i, alt_1, j) = 0$ **then**

 | $satisfaction\{w\}\{alt_1\}(i, g, j) = 0$

else if $AFM\{w\}(i, alt_1, j) \neq 0 \vee D2\{w\}\{alt_1\}\{h\}(i, g) = 0$ **then**

 | $satisfaction\{w\}\{alt_1\}(i, g, j) = 0$

else if $AFM\{w\}(i, alt_1, j) \neq 0 \vee D2\{w\}\{alt_1\}\{h\}(i, g) \neq 0$ **then**

if $D1\{w\}(i, alt_1) = 1 \vee D2\{w\}\{alt_1\}\{h\}(i, g) = 1$ **then**

 | $satisfaction\{w\}\{alt_1\}(i, g, j) = 1$

else if $D1\{w\}(i, alt_1) \neq 1 \vee D2\{w\}\{alt_1\}\{h\}(i, g) \neq 1$ **then**

if $D1\{w\}(i, alt_1) - D2\{w\}\{alt_1\}\{h\}(i, g) = 0$ **then**

 | $satisfaction\{w\}\{alt_1\}(i, g, j) = 1$

else if $D1\{w\}(i, alt_1) - D2\{w\}\{alt_1\}\{h\}(i, g) > 0$ **then**

 | $satisfaction\{w\}\{alt_1\}(i, g, j) =$

$1 - \frac{|D1\{w\}(i, alt_1) - D2\{w\}\{alt_1\}\{h\}(i, g)|}{3}$

else if $D1\{w\}(i, alt_1) - D2\{w\}\{alt_1\}\{h\}(i, g) < 0$ **then**

 | $satisfaction\{w\}\{alt_1\}(i, g, j) =$

$(1 - \frac{|D1\{w\}(i, alt_1) - D2\{w\}\{alt_1\}\{h\}(i, g)|}{3}) \cdot 1.5$

end

end

$SatisfactionTotal\{w\}\{alt_1\}\{h\}(g) =$

$\sum_{i=1}^n \sum_{j=1}^m satisfaction\{w\}\{alt_1\}(i, g, j)$

end

end

end

// End

References

- [1] Object Management Group (OMG). *OMG Systems Modeling Language*. URL: <http://www.omg.sysml.org/>. (accessed: 26.06.2021).
- [2] Michael Aherne et al. “Aeneas–Colony I meets three-axis pointing”. In: (2011).
- [3] Shashank P Alai. “Evaluating arcadia/capella vs. oosem/sysml for system architecture development”. PhD thesis. Purdue University Graduate School, 2019.
- [4] Jose Lorenzo Alvarez et al. “Model-based system engineering approach for the Euclid mission to manage scientific and technical complexity”. In: *Modeling, Systems Engineering, and Project Management for Astronomy VII*. Vol. 9911. International Society for Optics and Photonics. 2016, p. 99110C.
- [5] Louise Anderson et al. “Enterprise modeling for CubeSats”. In: *2014 IEEE Aerospace Conference*. IEEE. 2014, pp. 1–16.
- [6] Modelica Association. *Modelica Language*. URL: <https://modelica.org/modelicalanguage.html>. (accessed: 25.06.2021).
- [7] M Bandecchi, B Melton, and F Ongaro. “Concurrent engineering applied to space mission assessment and design”. In: *ESA bulletin 99*. Journal Article (1999).
- [8] Todd Bayer. “Is MBSE helping? Measuring value on Europa Clipper”. In: *2018 IEEE Aerospace Conference*. IEEE. 2018, pp. 1–13.
- [9] Barry W Boehm. “A spiral model of software development and enhancement”. In: *Computer* 21.5 (1988), pp. 61–72.
- [10] Stephane Bonnet, Jean-Luc Voirin, and Juan Navas. “Augmenting requirements with models to improve the articulation between system engineering levels and optimize V&V practices”. In: 29.1 (2019), pp. 1018–1033.
- [11] Stéphane Bonnet et al. “Modeling system modes, states, configurations with Arcadia and Capella: method and tool perspectives”. In: 27.1 (2017), pp. 548–562.
- [12] Jasper Bouwmeester and Jian Guo. “Survey of worldwide pico-and nanosatellite missions, distributions and subsystem technology”. In: *Acta Astronautica* 67.7-8 (2010), pp. 854–862.
- [13] S. Battistini C. Cappelletti and B. K. Malphrus. *Cubesat Handbook: From Mission Design to Operations*. Elsevier, 2020.

- [14] Miriam Calo. “Model Based System Engineering applied to Small Satellite Systems”. PhD thesis. Politecnico di Torino, 2020.
- [15] S Chhaniyara et al. “Model based system engineering for space robotic systems”. In: *Proceedings of 11th Symposium on Advanced Space Technologies in Robotics and Automation*. 2011.
- [16] Aerospace Corporation. *First Aerocubes defined using MBSE now in orbit*. URL: <https://aerospace.org/story/first-aerocubes-defined-using-mbse-now-orbit>. (accessed: 27.06.2021).
- [17] Aerospace Corporation. *Going into action with Aerocube-10*. URL: <https://aerospace.org/article/going-action-aerocube-10>. (accessed: 27.06.2021).
- [18] Dov Dori. “Why significant UML change is unlikely”. In: *Communications of the ACM* 45.11 (2002), pp. 82–85.
- [19] Dov Dori et al. “OPCAT—An Object-Process CASE Tool for OPM-Based Conceptual Modelling”. In: *1st International Conference on Modelling and Management of Engineering Processes*. University of Cambridge Cambridge, UK. 2010, pp. 1–30.
- [20] *e.Inspector AIT/AIV Plan*. 2021.
- [21] *e.INSPECTOR Mission Description Document*. 2020.
- [22] Harald Eisenmann. “MBSE has a good start; requires more work for sufficient support of systems engineering activities through models”. In: *Insight* 18.2 (2015), pp. 14–18.
- [23] ESA. *Applying MBSE to a space mission*. URL: <https://blogs.esa.int/cleanespace/2017/08/28/applying-mbse-to-a-space-mission/>. (accessed: 30.06.2021).
- [24] Stéphane Estable et al. “Systems Modelling and Simulation of the ESA e. Deorbit Space Debris Removal Mission”. In: *NAFEMs*. 2017.
- [25] Jeff A Estefan et al. “Survey of model-based systems engineering (MBSE) methodologies”. In: *IncoSE MBSE Focus Group* 25.8 (2007), pp. 1–12.
- [26] Kevin Forsberg and Harold Mooz. “The relationship of system engineering to the project cycle”. In: *INCOSE International Symposium*. Vol. 1. 1. Wiley Online Library. 1991, pp. 57–65.
- [27] Sanford Friedenthal, Alan Moore, and Rick Steiner. *A practical guide to SysML: the systems modeling language*. Morgan Kaufmann, 2014.
- [28] J Guo, EKA Gill, and S Figari. “Model Based Systems Engineering to support the development of nano satellites”. In: *IAC. IAF*. 2014, pp. 1–10.
- [29] Arthur David Hall. *A methodology for systems engineering*. Vol. 24. van Nostrand, 1962.
- [30] Cecilia Haskins et al. “Systems engineering handbook”. In: *INCOSE*. Vol. 9. 2006, pp. 13–16.

-
- [31] Richard E Hodges et al. “A Deployable High-Gain Antenna Bound for Mars: Developing a new folded-panel reflectarray for the first CubeSat mission to Mars.” In: *IEEE Antennas and Propagation Magazine* 59.2 (2017), pp. 39–49.
- [32] IBM. *IBM Engineering Systems Design Rhapsody*. URL: <https://www.ibm.com/products/systems-design-rhapsody>. (accessed: 25.06.2021).
- [33] INCOSE. *History of Systems Engineering*. URL: <https://www.incose.org/about-systems-engineering/history-of-systems-engineering>. (accessed: 20.06.2021).
- [34] INCOSE. *Space Systems Working Group: Mission & Objectives*. URL: <https://www.incose.org/incose-member-resources/working-groups/Application/space-systems>. (accessed: 26.06.2021).
- [35] Project Management Institute. *A guide to the Project Management Body of Knowledge (PMBOK guide)*. 6th ed. 2017. ISBN: 9781628251845.
- [36] Stephen J Kapurch. *NASA systems engineering handbook*. Diane Publishing, 2010.
- [37] David Kaslow. “CubeSat Model Based System Engineering (MBSE) Reference Model-Application in the Concept Lifecycle Phase”. In: *AIAA SPACE 2015 Conference and Exposition*. 2015, p. 4474.
- [38] David Kaslow et al. “Developing a cubesat model-based system engineering (mbse) reference model-interim status”. In: *2015 IEEE Aerospace Conference*. IEEE. 2015, pp. 1–16.
- [39] David Kaslow et al. “Integrated model-based systems engineering (MBSE) applied to the Simulation of a CubeSat mission”. In: *2014 IEEE Aerospace Conference*. IEEE. 2014, pp. 1–14.
- [40] Mehdi Keshavarz Ghorabae et al. “Multi-criteria inventory classification using a new method of evaluation based on distance from average solution (EDAS)”. In: *Informatica* 26.3 (2015), pp. 435–451.
- [41] Javeed Kittur et al. “Comparison of different MCDM techniques used to evaluate optimal generation”. In: *2015 international conference on applied and theoretical computing and communication technology (iCATccT)*. IEEE. 2015, pp. 172–177.
- [42] Vaclav Knap, Lars Kjeldgaard Vestergaard, and Daniel-Ioan Stroe. “A review of battery technology in cubesats and small satellite solutions”. In: *Energies* 13.16 (2020), p. 4097.
- [43] E. Kulu. *Nanosats Database*. URL: <https://www.nanosats.eu/>. (accessed: 26.06.2021).
- [44] Jose Lorenzo Alvarez et al. “Best Practices for Model Based Systems Engineering in ESA Projects”. In: *2018 AIAA SPACE and Astronautics Forum and Exposition*. 2018, p. 5327.
- [45] Azad M Madni and Shatad Purohit. “Economic analysis of model-based systems engineering”. In: *Systems* 7.1 (2019), p. 12.

- [46] Saikat Ranjan Maity and Shankar Chakraborty. “Tool steel material selection using PROMETHEE II method”. In: *The International Journal of Advanced Manufacturing Technology* 78.9-12 (2015), pp. 1537–1547.
- [47] James N Martin. *Systems engineering guidebook: A process for developing systems and products*. Vol. 10. CRC press, 1996.
- [48] *Model Based For System Engineering*. URL: <https://essr.esa.int/project/mb4se-model-based-for-system-engineering>. (accessed: 27.06.2021).
- [49] RA Nelson. “Spacecraft Battery Technology”. In: *VIA SATELLITE-POTOMAC-14* (1999), pp. 104–149.
- [50] V Normand and D Exertier. “Model-driven systems engineering: SysML & the MDSysE approach at Thales”. In: *Ecole d’été CEA-ENSIETA, Brest, France* (2004).
- [51] Anthony M Olver and Michael J Ryan. “On a useful taxonomy of Phases, Modes, and States in Systems Engineering”. In: (2014).
- [52] Armen Poghosyan and Alessandro Golkar. “CubeSat evolution: Analyzing CubeSat capabilities for conducting science missions”. In: *Progress in Aerospace Sciences* 88 (2017), pp. 59–83.
- [53] Yahya Rahmat-Samii, Vignesh Manohar, and Joshua Michael Kovitz. “For Satellites, Think Small, Dream Big: A review of recent antenna developments for CubeSats.” In: *IEEE Antennas and Propagation Magazine* 59.2 (2017), pp. 22–30.
- [54] Pascal Roques. “MBSE with the ARCADIA Method and the Capella Tool”. In: *8th European Congress on Embedded Real Time Software and Systems (ERTS 2016)*. 2016.
- [55] Pascal Roques. *Systems Architecture Modeling with the Arcadia Method: A Practical Guide to Capella*. Elsevier, 2017.
- [56] Winston W Royce. “Managing the development of large software systems: concepts and techniques”. In: *Proceedings of the 9th international conference on Software Engineering*. 1987, pp. 328–338.
- [57] Wolahan A. Biesbroek. R. Innocenti L. Morales Serrano S. and de Koning H-P. “Model Based Systems Engineering Applied to ESA’s e.Deorbit Mission”. In: 2017.
- [58] Thomas L Saaty. “What is the analytic hierarchy process?” In: *Mathematical models for decision support*. Springer, 1988, pp. 109–121.
- [59] TL Saaty. “The analytic hierarchy process. mcgrawhill international”. In: *New York* (1980).
- [60] Albert Sanders and John Klein. “Systems engineering framework for integrated product and industrial design including trade study optimization”. In: *Procedia Computer Science* 8 (2012), pp. 413–419.

-
- [61] Josh Schoolcraft, Andrew Klesh, and Thomas Werne. “MarCO: interplanetary mission development on a CubeSat scale”. In: *Space Operations: Contributions from the Global Community*. Springer, 2017, pp. 221–231.
- [62] Cal Poly SLO. “CubeSat Design Specification Rev. 14”. In: (2020).
- [63] European Cooperation for Space Standardization. “ECSS-E-ST-10-02C Rev. 1 Space engineering - Verification”. In: (2018).
- [64] European Cooperation for Space Standardization. “ECSS-E-ST-10-06C Space engineering - Technical requirements specification”. In: (2009).
- [65] European Cooperation for Space Standardization. “ECSS-E-ST-10C Rev.1 – System engineering general requirements”. In: (15 February 2017).
- [66] European Cooperation for Space Standardization. “ECSS-M-ST-10C Rev. 1 – Space project management: project planning and implementation”. In: (6 March 2009).
- [67] Sara C Spangelo et al. “Applying model based systems engineering (MBSE) to a standard CubeSat”. In: *2012 IEEE aerospace conference*. IEEE. 2012, pp. 1–20.
- [68] Sara C Spangelo et al. “Model based systems engineering (MBSE) applied to Radio Aurora Explorer (RAX) CubeSat mission operational scenarios”. In: *2013 IEEE Aerospace Conference*. IEEE. 2013, pp. 1–18.
- [69] Dassault Systems. *Cameo Systems Modeler*. URL: <https://www.3ds.com/products-services/catia/products/no-magic/cameo-systems-modeler/>. (accessed: 25.06.2021).
- [70] Sparks Systems. *Enterprise Architect*. URL: <https://sparxsystems.com/products/mdg/tech/sysml/index.html>. (accessed: 25.06.2021).
- [71] International Council on Systems Engineering. *Systems Engineering Vision 2020*. INCOSE Technical Operations, Seattle, WA, 2007.
- [72] Thales. *Let yourself be guided with ARCADIA*. URL: <https://www.eclipse.org/capella/arcadia.html>. (accessed: 25.06.2021).
- [73] Evangelos Triantaphyllou et al. “Multi-criteria decision making: an operations research approach”. In: *Encyclopedia of electrical and electronics engineering* 15.1998 (1998), pp. 175–186.
- [74] Carnegie Mellon University. *Architecture Analysis and Design Language*. URL: https://www.sei.cmu.edu/our-work/projects/display.cfm?custome1_datapageid_4050=191439&custome1_datapageid_4050=191439. (accessed: 25.06.2021).
- [75] Jean-Luc Voirin. *Conception architecturale des systèmes basée sur les modèles avec la méthode Arcadia*. Vol. 3. ISTE Group, 2018.
- [76] Jean-Luc Voirin et al. “Simplifying (and enriching) SysML to perform functional analysis and model instances”. In: *INCOSE International Symposium*. Vol. 26. 1. Wiley Online Library. 2016, pp. 253–268.

- [77] Roger Walker et al. “Deep-space CubeSats: thinking inside the box”. In: *Astronomy & Geophysics* 59.5 (2018), pp. 5–24.
- [78] Charles S Wasson. “System Phases, Modes, and States Solutions to Controversial Issues”. In: *Wasson Strategics, LLC*. <http://www.wassonstrategics.com> (2010).
- [79] James R Wertz and Wiley J Larson. “Space mission analysis and design, Micro-cosm”. In: *Inc.*, (1999), p. 497.
- [80] Sasha Weston et al. “State of the art: Small spacecraft technology”. In: (2020).
- [81] Sirous Yasseri. “Application of systems engineering to subsea development”. In: *Underwater Technology* 32.2 (2014), pp. 93–109.