

Developing a Digital Thread for AIV Engineering

A Model-Based Framework for Small Satellite Missions

AE5822: Thesis Space

Darpan Ramachandra Byahatti



Developing a Digital Thread for AIV Engineering

A Model-Based Framework for Small Satellite Missions

by

Darpan Ramachandra Byahatti

Thesis work was completed at the ISISPACE Group, in partial fulfilment of the requirements for the degree of Master of Science in Aerospace Engineering at the Delft University of Technology.

To be defended publicly on Friday August 21, 2026

Student number: 6167888
Project duration: January - August 2026
Thesis committee: Dr. Iklim Akay TU Delft, supervisor
Ir. J. Strengé ISISPACE, supervisor
Ir. L. M. Goh ISISPACE, mentor

An electronic version of this thesis is available at <http://repository.tudelft.nl/>.

Acknowledgement

Completing my Masters in Aerospace Engineering at TU Delft has been nothing short of a rollercoaster. Space engineering has been my core passion for more than a decade, so this feels like a deeply personal milestone. Having worked with many brilliant people on academic projects, several of whom have become my close friends, is something I will never forget. To everyone at Redwire, I would like to express my sincere gratitude for making my internship a fun, meaningful, and enriching experience. As for my thesis, I am extremely grateful to Dr. Iklim Akay, my supervisor at TU Delft, for being an inspiration, for always lending a listening ear, and for helping me develop the research mindset that carried me through this thesis. I am also very thankful to Dr. Stefano Speretta and Dr. Prem Sundaramoorthy for being part of my thesis committee.

As this thesis was completed in collaboration with ISISPACE, my supervisor Joachim has been the most understanding person since the moment I proposed working on MBSE implementation. From the very beginning, he pushed me out of my comfort zone, encouraged organisational adoption of the framework, and helped me explore new directions for Capella that ultimately led to its extension into AIV activities. I cannot thank Ming enough for being the best mentor I could have asked for. From day one, he has been curious about my direction and approach towards MBSE. The fortunate coincidence of him overhearing my concept for an AIV-based toolchain became one of the most interesting series of events of this journey. From the moment he sat down to teach me the basics of systems engineering using a water bottle as a demonstration prop, to watching me eventually model a full Earth Observation CubeSat, it has been an incredibly rewarding experience. Lastly, through all the discussions together, I have had an amazing time working closely with the systems engineering and AIT teams at ISISPACE.

On a more personal note, my parents have been my constant pillar of strength throughout my academic career. Through the highs and lows, my family has made the journey feel far less daunting than it often was. The friends I have made during my time in Delft have made this city feel like home, and I am grateful for their company not only in celebrating the wins but also through the long days and uncertain times. A special thanks to my friends back home for keeping me grounded and for being some of the best people I have had the privilege of knowing.

Darpan Ramachandra Byahatti

Delft, August 2026

Contents

Acknowledgement	i
List of Figures	v
List of Tables	viii
Nomenclature	ix
1 Introduction	1
1.1 Research Objective	2
1.2 Mission Context	3
1.3 Thesis Outline	4
2 Background	5
2.1 Small Satellite Missions	5
2.2 Mission Lifecycle	6
2.3 Traditional Systems Engineering	7
2.3.1 Document-Centred Engineering	7
2.3.2 The V-Model	8
2.4 Data-Driven Systems Engineering	8
2.4.1 Strengths	8
2.4.2 Limitations	9
2.5 Model-Based Systems Engineering	9
2.5.1 System Modelling Concepts	10
2.5.2 Strengths	10
2.5.3 Limitations	11
2.6 AIV/V Engineering in Small Satellite Missions	12
2.6.1 V&V Activities	12
2.6.2 AIT Challenges	14
2.7 Need for an Integrated Framework	15
2.7.1 Addressed Challenges	15

2.7.2	Evaluation Metrics for the Integrated Framework	16
3	Tool and Method Set-up	17
3.1	Capella	18
3.2	Valispace	19
3.3	Integration	20
3.3.1	Need for Capella-Valispace Toolchain	21
4	Methodology	23
4.1	Research Approach	23
4.1.1	Phase 1 - Problem Definition	24
4.1.2	Phase 2 - System Modelling	24
4.1.3	Phase 3 - Designing Framework	25
4.1.4	Phase 4 - Verification and Validation	25
4.2	Driving Requirements	25
5	System Modelling	27
5.1	Operational Analysis	27
5.2	System Analysis	29
5.3	Logical Architecture	32
5.4	Physical Architecture	36
5.5	Property Value Management Tool	41
6	Designing Framework	43
6.1	Capella-only Approach	43
6.1.1	Requirement Linkage	43
6.1.2	Verification Planning	44
6.1.3	Functional Test Modelling	45
6.1.4	Verification Tracking	49
6.2	Capella-Valispace Approach	51
6.2.1	Forward Thread	52
6.2.2	Reverse Thread	56
6.2.3	Reverse Thread	58
7	Framework Verification	61
7.1	Blocks Module	61
7.2	Requirements Module	64
7.3	V&V Module	65

7.4	PVMT	66
8	Results and Discussion	68
8.1	Evaluation Methodology	68
8.2	Results	69
8.2.1	Traceability	70
8.2.2	Consistency	72
8.2.3	Flexibility	74
8.2.4	Accessibility and Stakeholder Collaboration	76
8.3	Discussion	78
8.3.1	Inter-Rater Validation and Cohen's Kappa	79
8.3.2	Interpretation of the Kappa Score	80
9	Framework Validation	81
9.1	Research Findings	81
9.1.1	Research Question 1	81
9.1.2	Research Question 2	82
9.1.3	Research Question 3	84
9.2	Contributions	85
9.3	Limitations	86
9.3.1	System Modelling	87
9.3.2	Toolchain	87
9.3.3	Evaluation	88
9.4	Future Work	88
	References	90
A	Appendix A	95
B	Appendix B	102

List of Figures

1.1	Life-cycle cost as a function of time [3]	1
2.1	Spacecraft mission lifecycle [14]	6
2.2	The Vee Model [3]	8
2.3	The PMTE elements in context of models [32]	10
3.1	Capella modelling language, method and tool [50]	18
3.2	Layers of the ARCADIA methodology [51]	18
3.3	Engineering data pyramid [28]	20
4.1	Thesis methodology	24
5.1	[MSM] Spacecraft mission lifecycle	28
5.2	[OAB] Operational Architecture Blank	29
5.3	[MCB] Mission Capabilities Blank showing requirement flowdown and actor allocation	30
5.4	[SAB] System Architecture Blank showing the space segment and its external interfaces	31
5.5	[MSM] Satellite operative modes	32
5.6	[LCBD] Logical Component Breakdown Diagram of the space segment	33
5.7	[LFBD] Logical Function Breakdown Diagram for the hyperspectral imaging function	33
5.8	[MSM] ADCS subsystem mode transition diagram	34
5.9	[LAB] ADCS subsystem logical architecture	34
5.10	[LAB] System-level Logical Architecture Blank of the space segment	35
5.11	[LAB] Power and thermal interfaces of the space segment	36
5.12	[PCBD] Physical Component Breakdown Diagram of the space segment	37
5.13	[MSM] System Assembly, Integration and Testing Map	38
5.14	[PAB] Power interface diagram of the space segment	38
5.15	[PAB] Harness diagram for the flight model	40
5.16	Mass budget domain defined using PVMT	41
5.17	Power budget domain defined using PVMT	41
5.18	Requirements domain defined using PVMT	42
6.1	System requirements linked to the space segment at the Logical Architecture layer . .	44

6.2	Verification method and status enumerations defined using PVMT	45
6.3	User interface for assigning verification method and status in Capella	45
6.4	[PAB] I2C Signal Verification test architecture	46
6.5	[ES] I2C Signal Verification test procedure	47
6.6	Component status tracking using PVMT	48
6.7	Test activity status tracking using PVMT	48
6.8	Capturing expected test results using PVMT extensions	49
6.9	Verification phase tracking using PVMT	50
6.10	Verification phase extension using PVMT	51
6.11	User interface for assigning verification phase in Capella	51
6.12	Capella to Valispace data mapping via Excel intermediate layer	52
6.13	Forward digital thread toolchain architecture	53
6.14	Timestamped run log files generated during import into Valispace	56
6.15	Valispace to Capella data mapping via Excel intermediate layer	57
6.16	Reverse digital thread toolchain architecture	58
6.17	Valispace to Capella data mapping via Excel intermediate layer	58
6.18	Reverse digital thread toolchain architecture	59
7.1	Blocks module showing the product tree imported from Capella	62
7.2	Info view of the AOBC block with part number imported from Capella	62
7.3	Properties of the AOBC block in Valispace imported from Capella	63
7.4	The AOBC block and its connections imported from the Capella model	63
7.5	Requirements module showing requirement identifiers, text and their V&V items . . .	64
7.6	Requirements module showing an excerpt of the parent-child relationships	64
7.7	Valispace V&V module showing the folder and activity structure	65
7.8	Test procedure for the I2C signal verification imported from the Exchange Scenario . .	65
7.9	As_built_mass property on PVMT populated following platform inspection during AIV	66
7.10	Capella PVMT populated with requirement identifier, V&V method and status	67
A.1	[OEBD] Operational Entity Breakdown Diagram	95
A.2	[OCB] Operational Capabilities Blank	96
A.3	[MSM] Satellite configurations modelled as sub-modes of the nominal operational mode	96
A.4	[LDFB] Logical Dataflow Blank for the hyperspectral imaging function	97
A.5	[ES] Exchange Scenario for the imaging capability at the Logical Architecture layer . .	98
A.6	[LAB] Control, data and RF interfaces of the space segment	99
A.7	[MSM] ADCS equipment active during nominal mode configurations	99
A.8	[PAB] Data interface diagram of the space segment	100

A.9 [PAB] Control interface diagram of the space segment	100
A.10 [PAB] RF interface diagram of the space segment	101

List of Tables

1.1	Mission characteristics [11]	3
2.1	Correlation of SE Capability Elements with Project Success [34]	11
3.1	Valispace and Capella capabilities [8]	21
4.1	Driving Requirements	26
8.1	Evaluation scores for Traceability	70
8.2	Evaluation scores for Consistency	72
8.3	Evaluation scores for Flexibility	74
8.4	Evaluation scores for Accessibility and Stakeholder Collaboration	76
8.5	Overall score distribution across all 26 success factors	78
8.6	Linear weight matrix applied in the weighted Cohen's Kappa calculation	79
8.7	Agreement summary between author and independent expert across 78 rating units . .	79
8.8	Weighted Cohen's Kappa per metric	80
B.1	Independent expert evaluation scores for Traceability	103
B.2	Independent expert evaluation scores for Consistency	103
B.3	Independent expert evaluation scores for Flexibility	104
B.4	Independent expert evaluation scores for Accessibility and Stakeholder Collaboration .	105

Nomenclature

Abbreviations

Abbreviation	Definition
ADCS	Attitude Determination and Control System
AIV	Assembly, Integration and Verification
AIV/V	Assembly, Integration, Verification and Validation
AIT	Assembly, Integration and Testing
AOBC	Attitude On-Board Computer
API	Application Programming Interface
ARCADIA	Architecture Analysis and Design Integrated Approach
ASOT	Authoritative Source of Truth
CM	Configuration Management
ConOps	Concept of Operations
COTS	Commercial Off-The-Shelf
DDSE	Data-Driven Systems Engineering
EASE	Eclipse Advanced Scripting Environment
ECSS	European Cooperation for Space Standardization
EGSE	Electrical Ground Support Equipment
EO	Earth Observation
EPS	Electrical Power System
ES	Exchange Scenario
FR	Functional Requirement
ICD	Interface Control Document
INCOSE	International Council on Systems Engineering
IVV	Integration, Verification and Validation
LA	Logical Architecture
LAB	Logical Architecture Blank
LCBD	Logical Component Breakdown Diagram
LDFB	Logical Data Flow Blank

Abbreviation	Definition
LFBD	Logical Function Breakdown Diagram
LTDN	Local Time of Descending Node
MBSE	Model-Based Systems Engineering
MCB	Mission Capabilities Blank
MSM	Mode State Machine
NASA	National Aeronautics and Space Administration
NFR	Non-Functional Requirement
OA	Operational Analysis
OAB	Operational Architecture Blank
OBC	On-Board Computer
OCB	Operational Capabilities Blank
OEBD	Operational Entity Breakdown Diagram
PA	Physical Architecture
PAB	Physical Architecture Blank
PCBD	Physical Component Breakdown Diagram
PMTE	Process, Method, Tool and Environment
PVMT	Property Value Management Tool
REST	Representational State Transfer
RF	Radio Frequency
SA	System Analysis
SAB	System Architecture Blank
SAIT	System Assembly, Integration and Testing
SE	Systems Engineering
SEBoK	Systems Engineering Body of Knowledge
SoI	System of Interest
SRD	System Requirements Document
SSO	Sun-Synchronous Orbit
T/A/I/D	Test, Analysis, Inspection, Design
TSE	Traditional Systems Engineering
V&V	Verification and Validation
VCD	Verification Control Document
VCM	Verification Control Matrix

Introduction

The rapid evolution of the NewSpace ecosystem has fundamentally changed how spacecraft are designed, developed, and operated. Small satellite missions, particularly CubeSats, have enabled faster mission deployment, reduced development costs, and increased accessibility to space [1]. However, these advantages are achieved at the expense of reduced schedule margins, limited testing opportunities, and highly constrained engineering resources [2]. As a result, ensuring mission reliability while maintaining accelerated development timelines is of major significance. Systems engineering plays a critical role in managing complexity across the spacecraft lifecycle. As shown in Figure 1.1, the cost of correcting design errors increases significantly as development progresses.

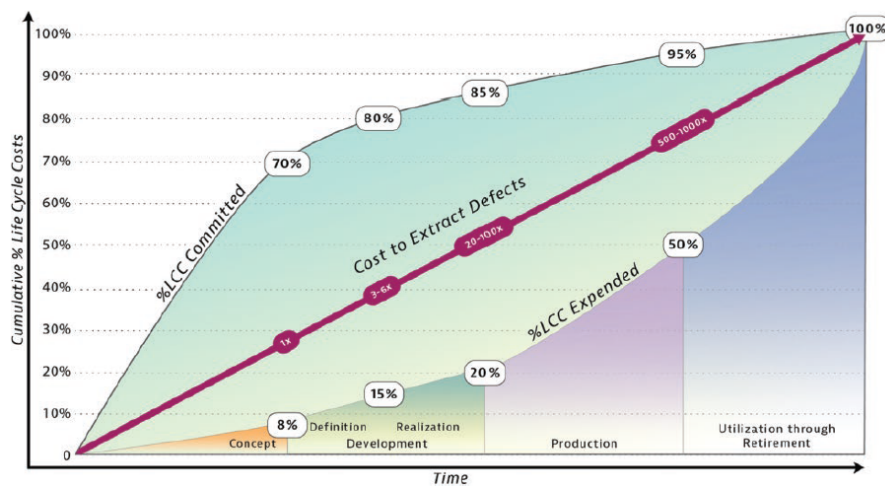


Figure 1.1: Life-cycle cost as a function of time [3]

Early-phase changes typically affect requirements or system models, whereas late-stage corrections frequently require hardware modification, re-integration, and repeated testing campaigns [4]. Consequently, engineering decisions made during early design phases strongly influence downstream Assembly, Integration, Verification and Validation (AIV/V) activities. During Phase D of the mission lifecycle, spacecraft hardware is assembled and subjected to functional and environmental qualification tests, subsystem and system integration, and verification in accordance with ECSS guidelines [5]. At this stage, discovery of requirement non-compliance may lead to:

- disassembly and rework of integrated hardware,
- requalification or regression testing,
- repetition of environmental test campaigns,
- delays impacting launch integration schedules.

Despite advances in digital engineering practices, AIV/V activities are still often supported by fragmented engineering data, manually maintained verification matrices, and weak traceability between system design decisions and verification evidence [6]. This disconnect between early systems engineering activities and later integration and testing phases frequently results in loss of information, duplicated effort, and reduced lifecycle continuity [7]. Recent developments in Model-Based Systems Engineering (MBSE) and Data-Driven Systems Engineering (DDSE) propose digital environments capable of maintaining a **single source of truth** throughout system development [8].

While these approaches have demonstrated benefits during early mission phases, their practical application within AIV/V engineering remains limited [9]. Verification planning, requirement compliance tracking, and test procedures are frequently maintained outside system models, resulting in fragmented information flows from early phases into Phase D [10]. This research is therefore motivated by the need to investigate whether an integrated MBSE-based framework can extend lifecycle continuity into AIV/V phases, enabling improved traceability and data consistency for small satellite missions.

1.1. Research Objective

The research objective for this thesis is:

To design, implement, and evaluate an integrated Model-Based and Data-Driven Systems Engineering framework supporting Assembly, Integration, Verification and Validation engineering for small satellite missions.

As the research objective is broad in scope, it is further broken down into three research questions:

- **RQ-1:** How can a DDSE tool be integrated with an MBSE-based solution to support AIV/V engineering processes in small satellite missions?
- **RQ-2:** To what extent does the integration of MBSE and DDSE environments influence lifecycle continuity between Phase C and Phase D?
- **RQ-3:** How effective is the MBSE-based integrated framework for a 6UXL CubeSat?

The hypotheses established for this thesis are:

- **H-1:** A “single source of truth” can be established by integrating MBSE and DDSE tools across the mission lifecycle of a small satellite.
- **H-2:** The integrated toolchain improves bidirectional traceability, data consistency, flexibility, and stakeholder collaboration.
- **H-3:** Extending MBSE utilisation into AIV/V phases positively influences project performance.

1.2. Mission Context

The research framework is demonstrated using a 6UXL CubeSat designed and developed by ISISPACE [11]. This mission carries a hyperspectral imaging payload integrated with a standardised CubeSat platform. The spacecraft orbit and attitude profiles are driven by mission and system requirements related to pointing performance, power generation, and data acquisition. The satellite interfaces with the ground segment for command and control operations and payload data downlink, with payload data transmitted via X-band communication and housekeeping telemetry exchanged through an S-band link. The key mission characteristics are summarised in Table 1.1.

Table 1.1: Mission characteristics [11]

Properties	Values
Reference orbit	SSO
Orbital altitude	500 km
Local Time of Descending Node (LTDN)	11:00
Platform size	6UXL
Spacecraft mass	~8 kg

1.3. Thesis Outline

This report is structured to introduce the research context, establish the state-of-the-art assessment, develop the integrated framework, and evaluate its application to AIV/V engineering through validation within a small satellite mission.

Chapter 2 reviews small satellite mission characteristics, lifecycle frameworks, and the limitations of traditional systems engineering, before introducing DDSE and MBSE as complementary paradigms. It establishes the gap between MBSE architectural definition and AIV/V execution as the central motivation for this research.

Chapter 3 describes Capella and Valispace¹ as the selected MBSE and DDSE platforms and briefly explains the integration mechanisms connecting them, including Python4Capella scripting and the Valispace REST API.

Chapter 4 presents the research approach, mission context, and framework scope definition. It establishes the driving requirements governing the framework design and outlines the qualitative evaluation approach used to assess the framework against reference workflows.

Chapter 5 documents the 6UXL Earth Observation CubeSat system model developed in Capella across all four ARCADIA layers, with particular emphasis on the Physical Architecture encompassing the product tree, interfaces, harnesses, functional test architectures and test procedures.

Chapter 6 presents the two proposed AIV workflow approaches. The Capella-only approach extends Capella into AIV activities through PVMT. The Capella-Valispace approach establishes the bidirectional digital thread connecting both tools through forward and reverse data flow architectures.

Chapter 7 evaluates the proposed framework against document-based and Capella-only reference approaches across 26 success factors and four qualitative metrics using a three-point Likert scale.

Chapter 8 answers the three research questions, assesses the hypotheses, and discusses the contribution relative to existing literature. Directions for future research are identified from partially and not-supported evaluation findings.

¹Valispace has been acquired by Altium and rebranded as the **Altium Requirements Portal**.

2

Background

This chapter establishes the engineering context required to understand the research problem addressed in this thesis. It introduces small satellite missions, spacecraft lifecycle characteristics, and systems engineering approaches currently applied within space projects. The reviewed literature focusses on Assembly, Integration, Verification and Validation (AIV/V) activities, where increasing system complexity and fragmented engineering environments create challenges for small satellite developments. The chapter concludes by motivating the need for integrated digital engineering frameworks.

2.1. Small Satellite Missions

Small satellites have transformed spacecraft development by enabling rapid mission deployment, reduced development costs, and increased accessibility to space. CubeSats, standardised around modular units of $10 \times 10 \times 10$ cm [12], represent the most widely adopted small satellite architecture and typically range from 1U to 16U configurations.

Unlike traditional large spacecraft programmes, CubeSat missions operate under constrained mass, power, volume, and schedule margins [13]. Hardware components are frequently integrated earlier in the development process, and subsystem maturity may evolve concurrently with testing activities. As a consequence, verification activities increasingly occur in parallel with design refinement rather than strictly following sequential development phases. Small satellite programmes must achieve comparable mission assurance with fewer resources, placing increased importance on efficient traceability, configuration management, and coordination between systems engineering and AIT teams.

2.2. Mission Lifecycle

Like most complex engineering products, satellites follow a structured mission lifecycle that organises development activities into defined phases to manage technical risk and ensure mission success. Systems engineering standards describe the lifecycle as a progression from concept definition to system realisation, operation, and disposal, enabling gradual maturation of requirements, architecture, and verification evidence [3]. The ECSS lifecycle shown in Figure 2.1 divides missions into Phases 0 through F, covering feasibility assessment (Phase A), preliminary definition (Phase B), detailed design (Phase C), qualification and acceptance (Phase D), operations (Phase E), and disposal (Phase F). AIV activities reach maximum intensity during Phase D, where hardware realisation and testing confirm compliance with mission requirements prior to launch [4].

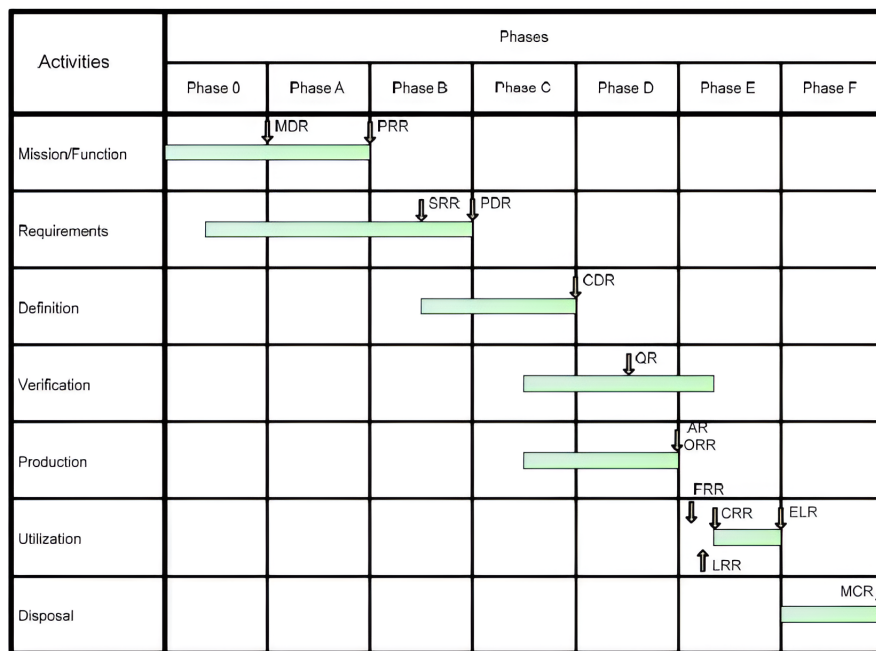


Figure 2.1: Spacecraft mission lifecycle [14]

Requirements defined during early phases are decomposed into subsystem and component specifications, which are subsequently verified through analysis, inspection, review of design, and testing activities [15]. As seen in Chapter 1, the cost of fixing defects increases exponentially in later stages of the lifecycle, making early verification planning critical. Modern engineering workflows, especially for small satellites, are characterised by evolving requirements and concurrent development activities that challenge these assumptions [10].

As small satellite missions operate under rapid schedules, smaller teams, and limited testing redundancy [16], Lifecycle activities often overlap. Verification planning may receive reduced focus during early design phases, and AIV specialists frequently inherit a finalised design rather than being involved from the start [6]. The impact of late-stage AIV/V issues becomes disproportionately severe; several

institutions attribute on-orbit failures to incomplete testing due to insufficient time and recommend dedicating half of a development schedule to testing from the outset [2]. Maintaining consistency between requirements, architectural models, and verification artefacts therefore becomes increasingly difficult without digitally supported traceability mechanisms and an authoritative source of truth [7]. This gap between lifecycle theory and operational practice highlights the need for engineering approaches capable of maintaining lifecycle continuity.

2.3. Traditional Systems Engineering

Traditional Systems Engineering (TSE) has been the foundation of spacecraft development for decades. It structures engineering work through a top-down breakdown of the system, managing its development lifecycle, and documentation-controlled baselines. Engineering knowledge in this approach is captured primarily through documents rather than executable system models [7].

2.3.1. Document-Centred Engineering

Document-Based Systems Engineering can be defined as “an approach to engineering systems that relies upon paper or digital textual documentation to record system specifications and other development project related information” [17]. Within traditional systems engineering, documents are the primary means of capturing and communicating system knowledge. Configuration management (CM) ensures that approved system baselines remain controlled throughout the lifecycle, providing “enduring truth,” trust, and traceability for the product’s configuration [3]. ISO systems engineering terminology further defines engineering documentation as any written or pictorial information describing activities, requirements, or results forming part of the lifecycle configuration baseline [18].

However, keeping large document sets consistent requires significant manual effort. Key limitations include:

- maintaining traceability between requirements, design artefacts, and verification evidence manually is time-consuming and prone to human error [19],
- when a requirement changes, every affected document must be updated manually, increasing the risk of contradictions across the data-pack,
- engineering knowledge is spread across disconnected artefacts, making it difficult to assess overall system consistency and completeness [20],
- teams spend up to 86% of their time on non-engineering tasks such as searching for information or manually entering data [21].

2.3.2. The V-Model

The V-model is the conceptual framework underlying traditional systems engineering, linking system definition activities directly to verification and validation processes [15].

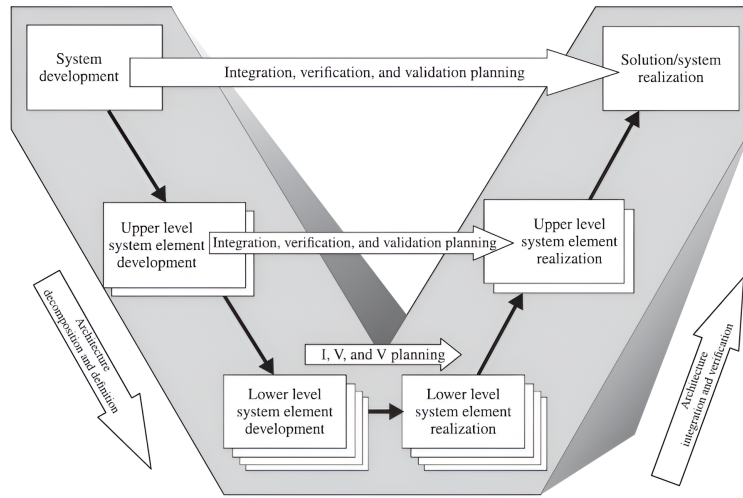


Figure 2.2: The Vee Model [3]

The left branch represents the progressive breakdown of the system from mission objectives down to the lowest level of detail, while the right branch covers integration and verification activities that confirm compliance at each level. Verification planning is therefore meant to run alongside requirement definition in the early project phases rather than beginning only after hardware is available [4]. The limitations of document-based engineering have driven the shift toward digitally managed environments that work with live data points rather than static reports.

2.4. Data-Driven Systems Engineering

Data-Driven Systems Engineering (DDSE) is a “methodology where engineering data and associated structure, links and connections constitute the foundation of the systems engineering process” [22]. Rather than exchanging information through disconnected documents, DDSE environments manage requirements, technical parameters, verification status, and engineering calculations within centralised databases, often termed a “Single Source of Truth” accessible to all stakeholders [21]. For spacecraft development, this shift points towards growing challenges in coordination between systems engineering, subsystem teams, and AIV organisations [23].

2.4.1. Strengths

DDSE platforms offer significant advantages in managing requirements, budgeting, and V&V processes [21]. In the scope of this thesis, the following capabilities are used extensively for the toolchain implementation.

Requirements Engineering

Modern platforms allow requirements to be stored with structured attributes, including verification method, ownership, compliance status, and traceability links [24]. This enables automated traceability, version control, and collaborative editing across distributed teams [25]. Continuous visibility of requirement verification status helps identify compliance risks early during testing campaigns [16].

Engineering Budget Management

Space missions operate under tight mass, power, thermal, and communication budgets, which are traditionally managed in spreadsheets and are prone to duplication and synchronisation errors [23]. DDSE platforms bring engineering calculations into a shared environment, so subsystem updates automatically flow through to system-level constraints. This is particularly useful in small satellite missions where margins are limited and parameters need to be monitored continuously [26].

Verification and Validation Management

Verification matrices, test evidence, and compliance status can be managed within a single engineering database rather than across separate static reports. This reduces the manual effort of maintaining and auditing verification documentation [27], and real-time dashboards help identify unverified requirements before they become schedule-critical issues.

2.4.2. Limitations

While DDSE significantly improves engineering data management, it does not inherently capture system architecture or operational behaviour [28]. Key limitations include:

- DDSE platforms typically lack the means to model hierarchical logical or physical architectures,
- system behaviour such as function sequencing, state-based logic, and operational modes is not well supported,
- system architecture is usually defined in a separate modelling tool, with Valispace serving only as a data management environment.

These limitations motivate the adoption of Model-Based Systems Engineering approaches that introduce formal, graphical system models as the primary engineering artefact to provide the necessary architectural context for engineering data [29].

2.5. Model-Based Systems Engineering

Model-Based Systems Engineering (MBSE) is defined by INCOSE 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” [3]. It represents a shift away from disconnected documents toward formal system models as the primary means of conveying engineering information [30]. MBSE has been widely adopted in aerospace, driven by increasing system complexity and the high cost of catching errors late in development [31].

2.5.1. System Modelling Concepts

MBSE methodologies combine processes, methods, and tools (PMTE) into a structured engineering workflow [32]. Processes define WHAT activities need to be performed, methods describe HOW they are carried out, and tools support their execution efficiently. Together, these elements form a methodology suited to a specific engineering domain, such as the ARCADIA method implemented in the Capella tool.

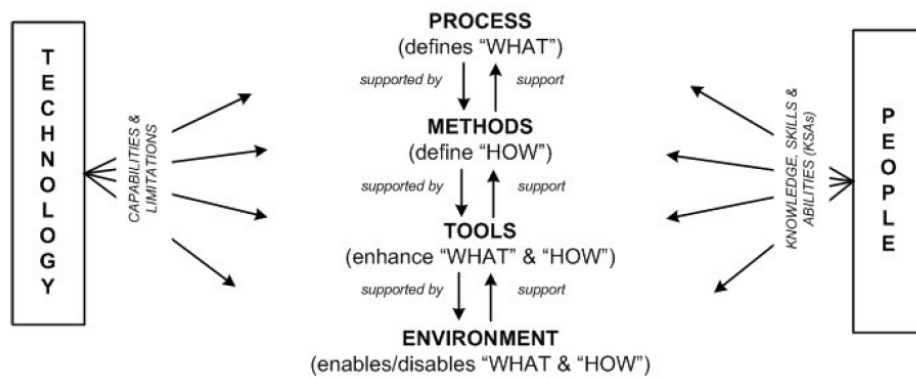


Figure 2.3: The PMTE elements in context of models [32]

2.5.2. Strengths

MBSE offers clear advantages over traditional systems engineering, including improved traceability, better collaboration, and earlier validation of design decisions [33]. Organisations using MBSE report improved communication among stakeholders, better management of system complexity, and earlier detection of defects, which reduces the cost of late-stage rework [3]. Projects implementing MBSE have also been shown to cost less and deliver on time more consistently compared to traditional approaches [17].

Table 2.1: Correlation of SE Capability Elements with Project Success [34]

Driver	Correlation
Total deployed SE	+0.49 → Very strong positive
Project planning	+0.46 → Very strong positive
Requirements development and management	+0.44 → Very strong positive
Verification	+0.43 → Very strong positive
Product architecture	+0.41 → Very strong positive
Configuration management	+0.38 → Strong positive
Trade studies	+0.38 → Strong positive
Project monitoring	+0.38 → Strong positive
Product integration	+0.33 → Strong positive
Validation	+0.33 → Strong positive
Risk management	+0.21 → Moderate positive
Integrated product team realization	+0.18 → Weak positive

As seen in Table 2.1, the SE capability elements most strongly correlated with project success are concentrated in the early lifecycle phases. Project planning (0.46), requirements development and management (0.44), and product architecture (0.41) all show very strong positive correlations. Elm and Goldenson further show that successful projects invest more in mission definition and early architectural work, while poor projects spend disproportionately on fixing integration problems later [34]. Of particular relevance to this thesis, verification (0.43) and product integration (0.33) both show strong positive correlations, confirming that structured investment in these activities leads to better project outcomes. The study also identifies MBSE as highly correlated with good project outcomes, and notes that successful early-stage SE is characterised by a focus on communications, traceability, flexibility, and integration. This reinforces the motivation to extend MBSE utilisation into AIV/V phases.

2.5.3. Limitations

Although MBSE improves architectural reasoning and early validation, its practical use tends to be concentrated in the early lifecycle phases. Surveys consistently report that MBSE is most effective during conceptualisation and requirements analysis, with utilisation dropping once detailed design and hardware integration begin [9]. Some key limitations include:

- limited native support for engineering budget management in descriptive modelling tools like Capella [35],
- MBSE tools can define test cases but rarely capture actual test results or verification status in real-time [23],

- architectural models and engineering databases remain separate, creating a gap between the design intent in the model and the as-built data in practice [21].

As a result, AIV/V teams often fall back on spreadsheets and document-based tools during integration and testing, breaking the continuity between the architectural design and the actual verification execution [23]. This gap motivates the need for an integrated toolchain that connects MBSE models with data-driven verification management environments [8].

Implications for AIV/V Engineering

Each paradigm addresses certain aspects of the engineering lifecycle, but none fully resolves the challenge of maintaining continuity between system design and verification execution [6]. MBSE falls short in supporting the AIV/V activities required in Phase D, while DDSE platforms are not suited for system architecture modelling. This creates a clear motivation for an integrated framework that combines the strengths of both, which is the central focus of this thesis.

2.6. AIV/V Engineering in Small Satellite Missions

Assembly, Integration, Verification and Validation (AIV/V) is the phase where system design assumptions are physically confirmed through testing and operational demonstration. ECSS verification standards define verification as demonstrating that system elements meet their specified requirements through analysis, inspection, review of design, or testing [15]. Validation confirms that the requirements for the intended use have been fulfilled, ensuring that the right product was built [18]. For space missions, AIV/V is one of the most resource-intensive stages of development, historically consuming approximately 35% of recurrent costs and 23% of the development schedule [6].

2.6.1. V&V Activities

The main Verification and Validation (V&V) processes such as requirement flowdown and verification planning begin in early stages of satellite development. Verification methods are defined and tracked through matrices to maintain traceability.

Requirement Flowdown

Verification planning establishes the strategy, responsibilities, and methods used to demonstrate requirement compliance throughout development [15], and should be performed concurrently with requirement definition rather than being postponed until hardware integration begins [3]. According to classical systems engineering practice, each requirement must be:

- clearly defined and unambiguous so it is interpreted consistently across the team,
- traceable vertically to parent and child requirements, and horizontally to architectural elements

and test cases,

- verifiable through a defined method (Test, Analysis, Inspection, or Review of Design),
- allocated to a responsible system element within the product breakdown structure.

In small satellite missions, consistent requirement flowdown is difficult to achieve due to the widespread use of Commercial Off-The-Shelf (COTS) components and budget constraints. The reliance on available hardware often leads to a flow-up approach, where subsystem capabilities limit higher-level system requirements rather than the other way around. Studies applying MBSE to CubeSat architectures show that insufficient requirement allocation frequently results in inconsistencies being discovered late in development [36].

Configuration Management

Configuration management (CM) ensures that all engineering artefacts remain consistent with the approved system baseline throughout the lifecycle [3]. ECSS guidelines require bidirectional traceability links and configuration-controlled documentation such as the Verification Control Document (VCD) [15]. In practice, document-driven workflows often result in fragmented traceability when requirements, design descriptions, and test results are maintained across separate tools. Without continuous digital traceability, the true verification status may only become clear during final integration, significantly increasing project risk [6].

Traceability

Traceability gaps are a recurring issue in small satellite missions, where approximately 40% of failures are attributed to unforeseen subsystem interactions and software behaviour rather than hardware defects [37]. MBSE studies on CubeSat missions show that missing links between requirements, architectural models, and verification artefacts frequently lead to duplicated testing or requirements that go unverified until launch [38]. Common problems include requirements without assigned verification methods, test procedures not linked to requirements, inconsistent baselines across teams, and manual verification tracking with no real-time visibility. These issues increase the risk of late-stage failures, which are particularly costly for CubeSat programmes with minimal schedule and budget margins.

Verification Methods

System- and subsystem-level requirements are verified through four ECSS-defined standard methods [15]:

- **Test** - using the realised end product to measure performance in representative environments,
- **Analysis** - using mathematical modelling and analytical techniques to predict compliance when a physical product is not yet available,

- **Inspection** - visually examining the realised product to verify physical design features, manufacturer identification, or workmanship,
- **Review of Design** - examining documentation such as drawings, specifications, and models to verify that the design meets the stated requirements without physical testing.

Selecting verification methods early, ideally during Phase A/B, allows integration campaigns to be planned efficiently and ensures requirements are verifiable before design complexity increases [9]. Model-based approaches have been shown to improve the alignment between requirements and verification strategies by shifting V&V activities earlier in the lifecycle [39].

Verification Matrices

According to ECSS standards, the VCD must be initialised during Phase B and updated incrementally until the completion of the verification campaign, the timing of which depends on the project scope and end-customer requirements. In practice, verification matrices in small satellite projects are often maintained in spreadsheets, which introduces risks of data duplication, outdated status tracking, and limited visibility across teams [23].

2.6.2. AIT Challenges

Assembly, Integration and Test (AIT) activities are where the system design is physically realised and tested. Small satellite missions face particular challenges in this phase due to compressed schedules, evolving designs, and fragmented engineering environments [40].

Interface Management

Interface management is a major driver of integration complexity, as spacecraft systems involve tightly coupled electrical, mechanical, thermal, and software boundaries that must all be compatible before integration begins. Keeping interfaces consistent becomes difficult when subsystems are developed in parallel or when teams independently update disconnected design artefacts [21]. A CubeSat's compact size makes de-integration and repair particularly difficult once assembly has begun. Literature [41] reports that Flatsat campaigns frequently expose inconsistencies between subsystem assumptions, leading to costly corrective actions late in development. Similar findings are reported in compact satellite AIV campaigns, where end-to-end testing revealed design flaws caused by complex system interactions [42].

Late Requirement Changes

CubeSat projects often have iterative development characteristics similar to agile environments. While this allows flexibility, it creates significant risks when requirement changes are not consistently propagated across design, verification, and configuration artefacts [21]. Late changes can lead to redesign of

already integrated subsystems, invalidation of completed tests, repeated verification campaigns, and schedule and cost overruns.

Tool Fragmentation

As discussed earlier, common challenge in spacecraft development is the use of multiple disconnected tools across different engineering activities. System architecture modelling, requirements management, budget tracking, verification management, and test documentation are often handled in separate tools with no formal link between them. This fragmentation is identified as a significant barrier to lifecycle continuity [43]. Even when MBSE tools are adopted for early design, AIT teams often revert to spreadsheets and static reports during implementation, breaking the connection between the architectural design and the actual verification execution [44]. Without integration, the benefits of digital engineering in automated change propagation and a single source of truth are yet to be fully realised [28].

2.7. Need for an Integrated Framework

The challenges described in the preceding sections are not purely technical, they are largely organisational and informational. Interface mismatches, late requirement changes, and fragmented tool environments all contribute to late-stage verification failures. These issues are especially critical in small satellite missions where testing opportunities are limited, teams are small, schedules are tight, and there is little room for rework. Although digital engineering practices are increasingly being adopted, lifecycle continuity between system design and verification execution remains limited.

2.7.1. Addressed Challenges

This framework specifically targets the following recurring challenges in spacecraft AIV/V engineering:

Redundant AIT Verification Flows

Verification planning is often rebuilt from scratch during integration rather than being derived directly from the system architecture model [6]. Requirements defined early in the project are frequently reinterpreted when generating verification matrices and test procedures, introducing duplicated effort and inconsistencies [2]. Verification activities often remain document-based even when model-based approaches are used in early design, which means the traceability established during system modelling is not carried through into the verification workflow.

Limited MBSE Utilisation Beyond Phase C

Although MBSE tools add significant value during conceptual and preliminary design, their use tends to decline once hardware realisation begins [10]. MBSE has consistently been used in the early project

phases, with utilisation dropping during verification planning and execution in Phase D. This reduces the long-term value of MBSE investments and limits their contribution to mission assurance [31].

Lack of Alignment Between Systems Engineering and AIT Teams

A gap between architecture definition and verification execution is a recurring source of late-stage integration risk. AIT specialists are typically brought in late, working in separate tool environments from the systems engineering team. As established earlier, inconsistencies between design intent and implemented hardware often only surface during integrated testing where they are most expensive to fix. This confirms that current engineering approaches do not yet provide a unified framework linking system architecture models with data-driven AIV/V execution environments [45].

2.7.2. Evaluation Metrics for the Integrated Framework

The first three evaluation criteria for the framework proposed in this thesis are derived from literature on digital engineering adoption and spacecraft verification practices [46]. Accessibility and Stakeholder Collaboration is additionally included as an evaluation metric, as the framework specifically targets improved alignment between systems engineering and AIV teams as a means of enhancing lifecycle continuity.

Traceability

Integrated environments should strengthen bidirectional traceability between system architecture, requirements, and verification data. This ensures every element is justified by a requirement and that compliance evidence is clearly connected back to the technical baseline [24].

Flexibility

The framework should support design and requirement changes without disrupting verification planning [46]. Automatic change propagation is particularly important in space missions where system parameters frequently change due to external constraints [21].

Data Consistency

Consistency between model property values and verification data is a key factor in maintaining a single source of truth. An integrated toolchain prevents data divergence by ensuring values remain consistent across tools, making it easier to detect discrepancies early [47].

Accessibility and Stakeholder Collaboration

Collaborative digital environments improve communication across multidisciplinary teams, reducing the gap that traditionally separate systems engineering and AIV activities [48]. Real-time visibility into verification status reduces the need for manual reporting and redundant communications.

3

Tool and Method Set-up

Model-Based Systems Engineering environments aim to replace document-centric engineering with structured system models, while Data-Driven Systems Engineering platforms focus on managing engineering information generated throughout project execution. The selection of Capella and Valispace as the MBSE and DDSE tools for this framework is based on both practical and technical considerations.

Capella was selected as the MBSE platform because it is open-source and freely available, making it accessible to small satellite programmes without the licence costs of commercial MBSE tools. Beyond accessibility, Capella has seen significant adoption in the aerospace and systems engineering industry across satellite development, defence, and transportation domains [8]. Its native implementation of the ARCADIA methodology provides a structured modelling process that guides engineers from operational needs through to physical implementation.

Valispace was chosen as the DDSE platform because an existing institutional licence was already in place at ISISPACE, where it had been used for preliminary requirements management. This meant the organisational infrastructure for data management was already established, reducing the overhead of introducing a new tool. The platform's user interface is accessible to engineers without a software background, which supports the role separation objective of the framework by allowing AIV engineers to work in Valispace without needing MBSE expertise.

3.1. Capella

Capella is an open-source MBSE tool that natively implements the ARCADIA methodology. It provides a structured modelling environment for early system definition, architecture analysis, and stakeholder communication [49]. Built on the Eclipse platform, it organises system knowledge as an interlinked set of architectural models rather than static documents, maintaining an Authoritative Source of Truth (ASOT) for the system design across the lifecycle [50].

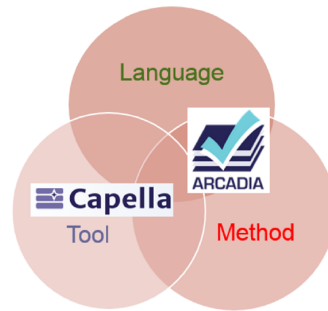


Figure 3.1: Capella modelling language, method and tool [50]

The ARCADIA (Architecture Analysis and Design Integrated Approach) methodology structures systems engineering activities into progressive architectural viewpoints as shown in Figure 3.2, guiding engineers from operational needs through to technical implementation while maintaining bidirectional traceability.

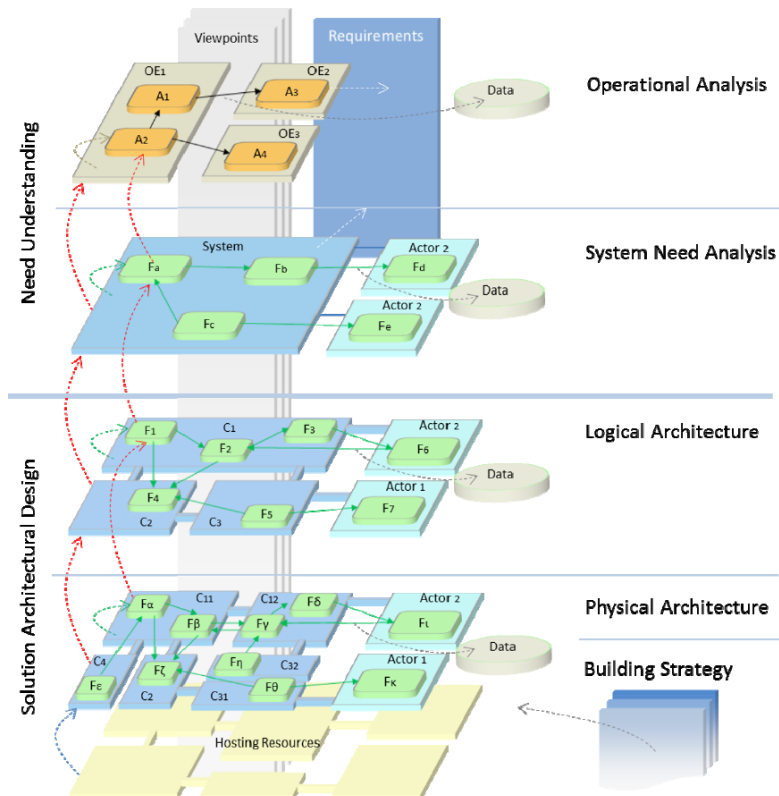


Figure 3.2: Layers of the ARCADIA methodology [51]

Capella organises system information through four architectural layers:

- *Operational Analysis* (OA) models the mission architecture by describing interactions between stakeholders, external actors, and mission services without yet defining the system itself.
- *System Analysis* (SA) introduces the System of Interest (SoI) and defines its interactions with external actors, translating operational needs into system functions.
- *Logical Architecture* (LA) identifies logical components, allocates functions to them, and enables interface definition and early performance reasoning.
- *Physical Architecture* (PA) maps logical elements to physical subsystem components, specifies interfaces, and supports design realisation.

By explicitly modelling interfaces and functional allocations, Capella helps identify inconsistencies earlier than traditional documentation workflows. The four layers provide a structured way to manage system complexity while maintaining continuous traceability from mission objectives to subsystem design. However, since Capella is primarily a design and architecture tool, verification data typically remains outside the MBSE environment.

3.2. Valispace

Valispace¹ is a centralised engineering platform that stores requirements, calculations, and verification data as live database entities rather than static text. Unlike document-based specifications, Valispace links engineering data through its internal calculation engine, ensuring requirements are continuously checked against current performance data [24]. The platform maintains bidirectional traceability by connecting requirements to analyses, design parameters (Valis), and verification evidence as shown in Figure 3.3.

¹The platform used in this toolchain is the **Altium Requirements Portal**, the successor to Valispace following its acquisition by Altium. Throughout this document, the platform is referred to as Valispace to maintain consistency with the API documentation, referenced literature, and toolchain scripts, which retain the Valispace designation.

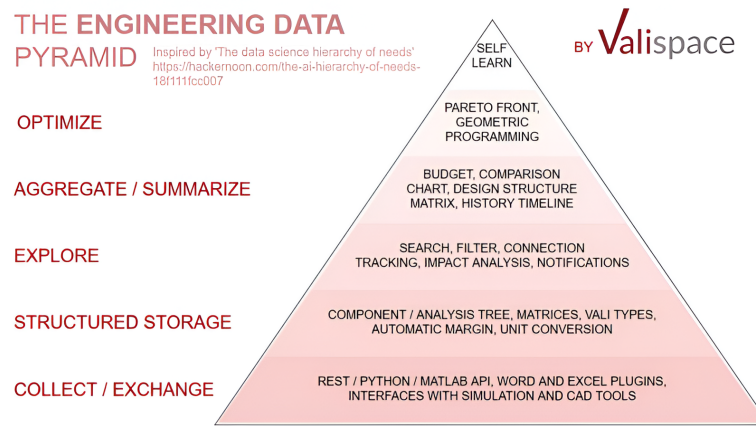


Figure 3.3: Engineering data pyramid [28]

A key strength of Valispace is its built-in Verification and Validation (V&V) capability, which consolidates requirements and testing within a single interface. This includes:

- *Verification methods* — specifying whether a requirement is verified through test, analysis, inspection, or review,
- *Verification status* — real-time dashboards showing how many requirements are pending, verified, or out of bounds,
- *V&V activities* — linking executed test procedure results directly back to system requirements.

This directly addresses the challenge in small satellite AIV/V campaigns where verification is often tracked in spreadsheets, leading to inconsistent information and late-stage problem discovery. Centralised verification management improves transparency and maintains an Authoritative Source of Truth (ASOT) for all compliance data during integration and testing.

3.3. Integration

While Capella excels at functional modelling and analysis, and Valispace provides strong capabilities for requirements management and verification tracking, lifecycle continuity is difficult to achieve when these tools operate independently. Architectural decisions are not directly connected to verification execution, requirement updates require manual synchronisation between environments, and lifecycle traceability becomes fragmented across disconnected tools [28].

Table 3.1 summarises the complementary capabilities of both tools, showing where each is strongest and how integration addresses individual limitations.

Table 3.1: Valispace and Capella capabilities [8]

	Capella	Valispace
Requirements management	Strong (flow-down, traceability with any model element, difficult to export)	Very Strong (digitization, flow-down, traceability, user-friendly, easy to export)
Concept of Operations	Very Strong (sequence diagrams)	Weak (limited to high level, no branching)
Modes and States	Strong (UML-inspired state machines, transitions modelled with detail)	Weak (components and their values can be allocated to modes and simulated)
Functional Analysis	Very Strong (detailed functionalities and functional interfaces modelling at different levels)	Weak (definition and trace but poor model)
Architecture management	Very Strong (detailed logical and physical architecture, allocation of functions to components)	Weak (only physical without modelling interfaces or trade-offs)
Budgets and simulation	Very Weak (class diagrams can be used to model static data structures)	Strong (components modelled with sizing values, end-to-end traceability and data consistency)
Interfaces management	Very Strong (detailed functional and physical interfaces)	Weak (limited to definition)
Teamwork	Strong (with add-on)	Very Strong (real-time, internal reviews, comments, notifications)
AIV/AIT management	Strong (architecture model can be re-used for AIV/AIT plan)	Strong (test procedures, requirements and verifications in the same tool)
Change tracking and history	Weak	Very Strong (full repercussion on the design)

3.3.1. Need for Capella-Valispace Toolchain

For small satellite missions where team sizes are limited, an integrated toolchain reduces duplicated effort and maintains an ASOT across lifecycle phases. Literature on integrated MBSE-DDSE environments reports the following advantages [45]:

- improved requirement traceability across lifecycle phases, with compliance evidence connected to the architectural baseline,
- reduced manual data duplication through a single source of truth where every data point is created, used, and stored consistently,
- earlier identification of integration risks before hardware realisation, enabling a left shift of V&V activities,
- greater consistency between design intent and verification execution, as architectural changes propagate through the shared database.

Despite these benefits, existing integrations remain limited in maturity:

1. Synchronisation between tools can introduce complexity if data ownership is not clearly defined.

2. Most reported studies focus on early design phases rather than the high-intensity AIV/V phases, so empirical evidence of Phase D benefits remains limited.
3. Successful integration requires alignment between systems engineers and AIT teams, which can conflict with existing project structures and working practices [52].

Python4Capella

Python4Capella provides scripting access to Capella models through Python-based automation, allowing engineers to extract, modify, and generate model elements programmatically [53]. It bridges Python with Java, enabling architectural data to be managed without manual model manipulation. Key information that can be exported includes system components and their hierarchies, functional allocations, interfaces and component exchanges, and requirement relationships. This reduces manual transcription errors and enables repeatable synchronisation workflows between the model and external tools. However, as Python4Capella operates through a Java-Python bridge, not all model element types and relationships are fully exposed through the scripting API. Some Capella constructs are simplified or unavailable in the Python layer, meaning certain model data cannot be extracted programmatically and must either be handled through workarounds or excluded from the digital thread scope.

REST API Integration

Valispace provides a REST API that allows external applications to read and write engineering data programmatically. Information extracted from Capella can be posted to Valispace, including requirements and verification status, mass and power budgets, test procedures linked to requirements, and subsystem parameters. Conversely, verification close-out data and as-built measurements recorded in Valispace can be written back into the Capella model. This two-way exchange forms the bidirectional digital thread that is the central contribution of this thesis. However, the REST API does not expose all platform functionality programmatically. Certain data assignments made through the Valispace user interface, such as per-requirement verification method and status values, are not accessible via the API in the current version. These gaps in API coverage introduce manual steps into an otherwise automated pipeline and represent a dependency on future API extensions by the tool vendor to achieve full end-to-end automation.

4

Methodology

The methodology follows a progressive research approach, moving from problem identification through framework design and implementation to structured evaluation. The research is conducted as a case study on a 6UXL Earth Observation CubeSat mission at ISISPACE, where the integrated framework is designed, implemented, and evaluated against baseline workflows.

4.1. Research Approach

The research process consists of four main phases as illustrated in Figure 4.1: Problem Definition, System Modelling, Designing Framework, and Verification and Validation. Each phase produces a defined output that feeds into the next, ensuring a structured progression from the identified research gap to the evaluated integrated framework.

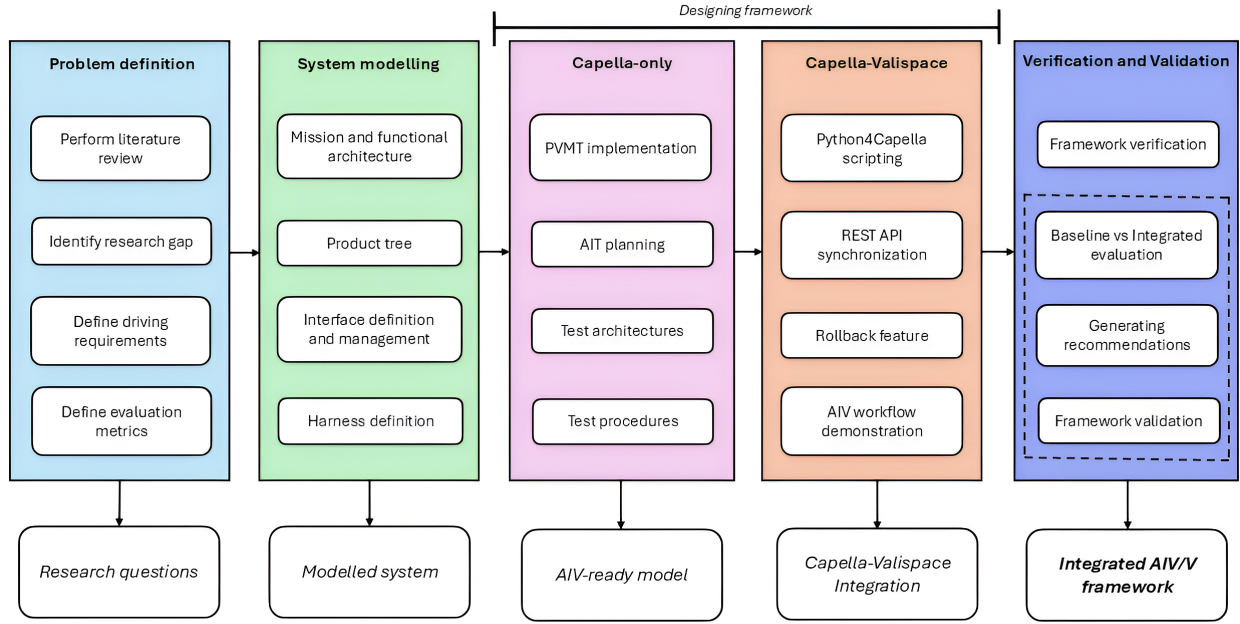


Figure 4.1: Thesis methodology

4.1.1. Phase 1 - Problem Definition

The first phase establishes the theoretical and engineering foundation of the research through a literature review of MBSE, DDSE, AIV/V engineering, and existing tool integration practices for small satellite missions. The review focuses on identifying discontinuities at the Phase C to Phase D hand-over, where engineering data defined during system design must be transferred to an AIV execution environment. This analysis motivates the integrated framework and informs the definition of driving requirements and evaluation metrics. The output of this phase is a set of research questions, hypotheses, and driving requirements that govern the framework scope and evaluation criteria.

4.1.2. Phase 2 - System Modelling

The second phase develops the 6U XL CubeSat system model in Capella across all four ARCADIA layers. The Operational Analysis layer establishes the mission architecture and requirements. The System Analysis layer introduces the System of Interest, defines system capabilities and external interfaces, and establishes the Concept of Operations. The Logical Architecture layer decomposes the system into logical components, allocates functions, and defines interface types and directions. The Physical Architecture layer finalises the hardware product tree, specifies physical interfaces and harness routing, and introduces the test architecture through PAB diagrams and Exchange Scenario test procedures. The output is a fully modelled system that serves as the ASOT and the primary data source for the forward digital thread.

4.1.3. Phase 3 - Designing Framework

The third phase designs and implements the two AIV workflow approaches that together constitute the integrated framework, developed as two parallel implementations.

The **Capella-only implementation** extends Capella's native capabilities into AIV engineering through PVMT and functional test modelling. PVMT property domains are configured to capture mass and power budget data, verification method and status, and requirement identifiers. Functional test architectures and procedures are modelled in the PA layer. This produces an AIV-ready Capella model that supports verification planning and tracking within the MBSE environment.

The **Capella-Valispace implementation** extends the Capella-only approach by implementing the bidirectional digital thread. Python4Capella export scripts extract model artefacts into structured Excel files. Python REST API import scripts post the extracted data into Valispace, populating the Blocks, Requirements, and V&V modules. A rollback mechanism based on timestamped run logs supports iterative development. The reverse digital thread returns as-built measurements and verification close-out data from Valispace to the Capella model as PVMT property values. This produces the operational Capella-Valispace integration that forms the primary research contribution.

4.1.4. Phase 4 - Verification and Validation

The fourth phase evaluates the integrated framework through a two-stage assessment. Framework verification confirms that the toolchain produces the expected engineering artefacts in each target environment, demonstrated through screenshots of the populated Valispace modules and the Capella PVMT following reverse flow execution. Framework validation conducts a structured qualitative evaluation comparing the integrated approach against document-based and Capella-only reference approaches across 26 success factors and four qualitative metrics. An independent expert evaluation is additionally conducted by a senior systems engineer at ISISpace, with inter-rater agreement assessed using weighted Cohen's Kappa. The output is the evaluated integrated AIV/V framework with documented findings, limitations, and directions for future development.

4.2. Driving Requirements

To address the research gap identified in Phase 1, this framework connects the capabilities of Capella and Valispace to support practical AIV/V workflows in CubeSat missions. The driving requirements governing the framework design and evaluation are presented in Table 4.1. Each requirement is motivated by a specific challenge identified in the literature and maps to one or more of the qualitative evaluation metrics used in Phase 4.

Table 4.1: Driving Requirements

ID	Driving Requirement	Description	Addressed Research Need	Evaluation Criterion
NFR-1	End-to-End Traceability	The framework shall maintain bidirectional traceability between mission objectives, system architecture, requirements, and verification evidence across the mission lifecycle.	Loss of traceability during AIV/V phases	Traceability completeness, automated link maintenance
NFR-2	Lifecycle Data Consistency	The framework shall maintain consistent engineering data, requirement status, and verification information across connected environments.	Data inconsistencies between SE and AIT teams	Data integrity, update propagation robustness
NFR-3	Stakeholder Collaboration	The framework shall enable shared access to engineering information for systems engineering and AIT stakeholders within a common digital environment.	Poor collaboration between teams	Accessibility, stakeholder interaction
FR-1	Lifecycle Continuity of MBSE Artifacts	The framework shall enable MBSE models created during early design phases to remain usable during AIV activities.	Limited MBSE utilisation beyond Phase C	Model reuse capability, lifecycle coverage
FR-2	Automated Engineering Data Exchange	The framework shall minimise manual transfer of engineering information through automated or semi-automated synchronization between tools.	Manual data handling and duplication	Automation level, synchronization reliability
FR-3	Support for AIV/V Engineering Workflows	The framework shall support verification planning activities such as requirement flowdown, verification matrices, configuration tracking, and verification status monitoring.	Inefficient AIV/V workflows	Verification management capability
FR-4	Practical Implementability for Small Satellite Missions	The framework shall be implementable using existing industrial tools, plugins, and APIs without requiring development of new proprietary software.	Lack of practical deployable solutions	Implementation effort, scalability

The non-functional requirements (NFR-1 to NFR-3) define the overarching quality characteristics the framework must demonstrate across all lifecycle phases. The functional requirements (FR-1 to FR-4) define the specific capabilities the toolchain must implement to support AIV engineering in practice. Together, these seven requirements form the basis of the 26 success factors and four evaluation metrics applied in Chapter 8.

5

System Modelling

The complementary strengths and limitations of both tools suggest that Valispace is well suited as a lifecycle data management and AIV/V coordination platform. When combined with Capella for system architecture and functional modelling, Valispace can extend model-based information into the verification execution and operational phases. In this relationship, the MBSE model serves as the master reference for component breakdown, functions, modes, and interfaces, while Valispace manages engineering data, budgets, and requirement compliance. This motivates the development of an integrated MBSE-DDSE toolchain capable of maintaining digital continuity across the small satellite lifecycle.

As part of this research, the 6UXL CubeSat system is modelled across all four ARCADIA layers in Capella to establish the engineering baseline required to support the AIV workflow approaches developed in Chapter 6.

5.1. Operational Analysis

The Operational Analysis layer establishes the mission context by identifying the key stakeholders and their roles. For this mission, ISISPACE is the primary stakeholder responsible for the design and development of the CubeSat. The first step is breaking down the major mission entities into operational actors representing the teams and individuals that contribute to the mission. This is captured in the Operational Entity Breakdown Diagram (OEBD), shown in Figure A.1 in Appendix A.

With the operational actors defined, mission requirements are structured through Operational Capabilities, which describe what the mission must achieve without prescribing how [54]. Six operational capabilities are defined in Capella and allocated to their respective entities, as shown in the Operational Capabilities Blank (OCB) in Figure A.2 in Appendix A.

The spacecraft mission lifecycle is modelled using a Mode State Machine (MSM) diagram as shown in Figure 5.1. States represent the phases the spacecraft undergoes from launch through to end-of-life, with transitions connecting them sequentially. This lifecycle definition provides the temporal context within which all AIV activities are planned and executed.

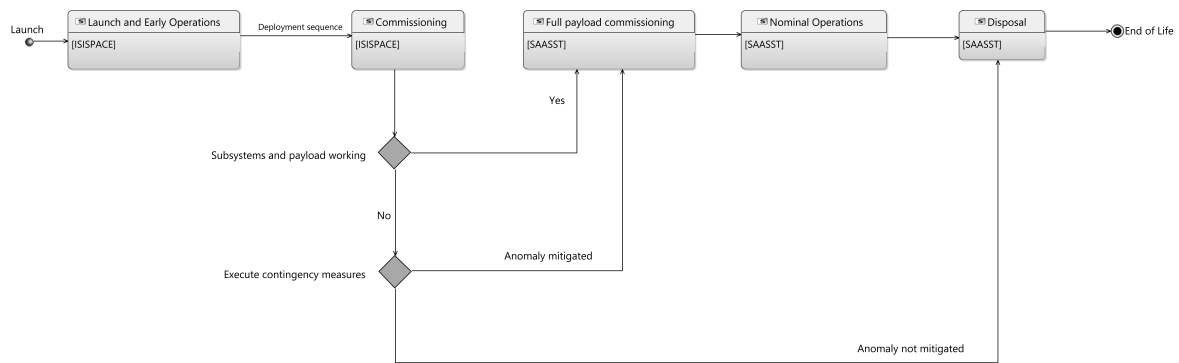


Figure 5.1: [MSM] Spacecraft mission lifecycle

The operational activities and interactions between entities [11] are captured in the Operational Architecture Blank (OAB), shown in Figure 5.2. Activities are allocated to their respective entities and interact sequentially to realise each operational capability. The coloured lines represent operational processes, each corresponding to the functional interaction chain that realises one capability across multiple entities. Communication between entities is represented through communication means, capturing the medium used to enable functional interaction.

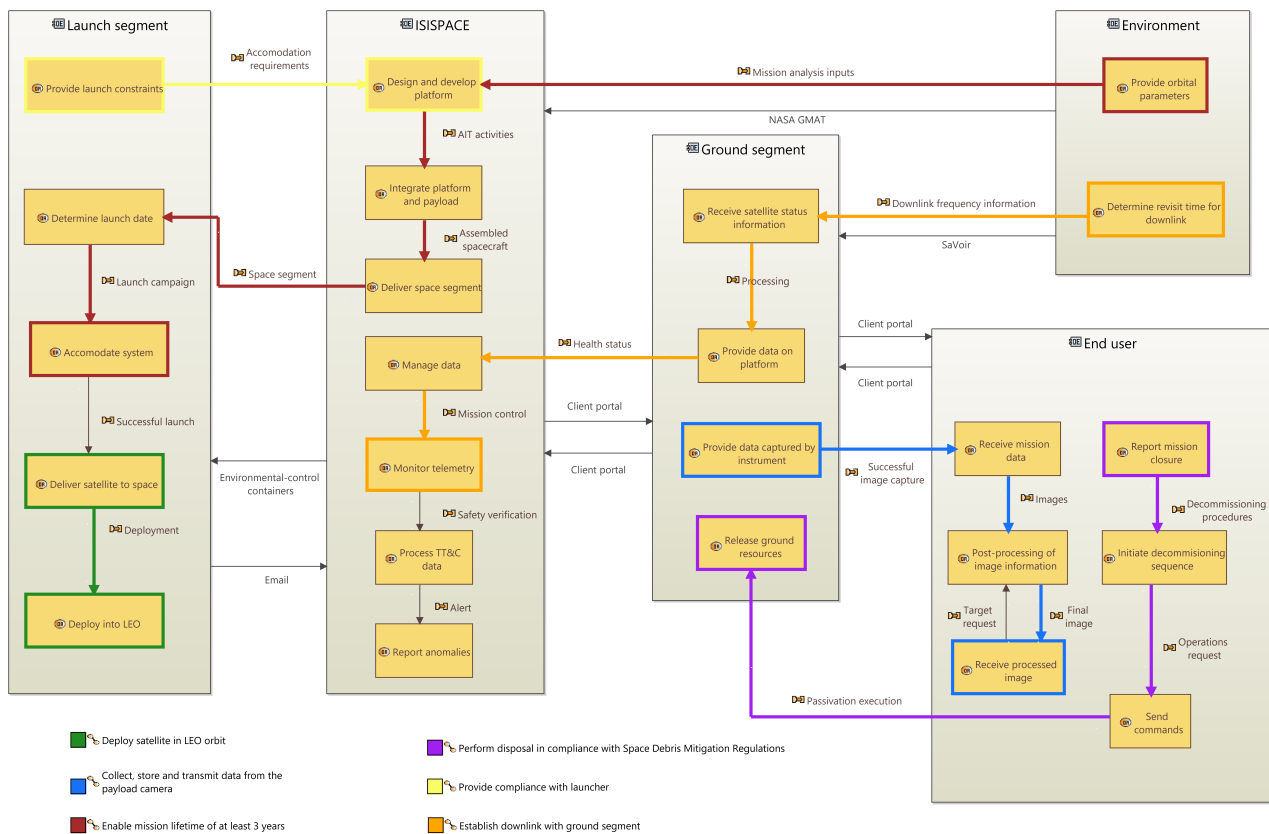


Figure 5.2: [OAB] Operational Architecture Blank

5.2. System Analysis

The System Analysis layer introduces the System of Interest, which in this mission is the space segment comprising the 6UXL platform and the payload instrument, and places it within the broader mission environment through the definition of system actors. These actors are derived from the operational entities established at the previous layer, maintaining continuity in the model. Mission requirements are flowed down by creating system missions from the operational capabilities defined earlier, and system capabilities are modelled to represent the specific requirements that the space segment and its actors must fulfil [54]. Each system capability is allocated to the responsible actor, and the capabilities within each system mission are connected to preserve traceability from mission intent to system responsibility. The resulting Mission Capabilities Blank (MCB) is shown in Figure 5.3. This diagram is of direct relevance to the integrated toolchain, as missions and capabilities at this layer form the source of the requirement hierarchy imported into Valispace during the forward digital thread.

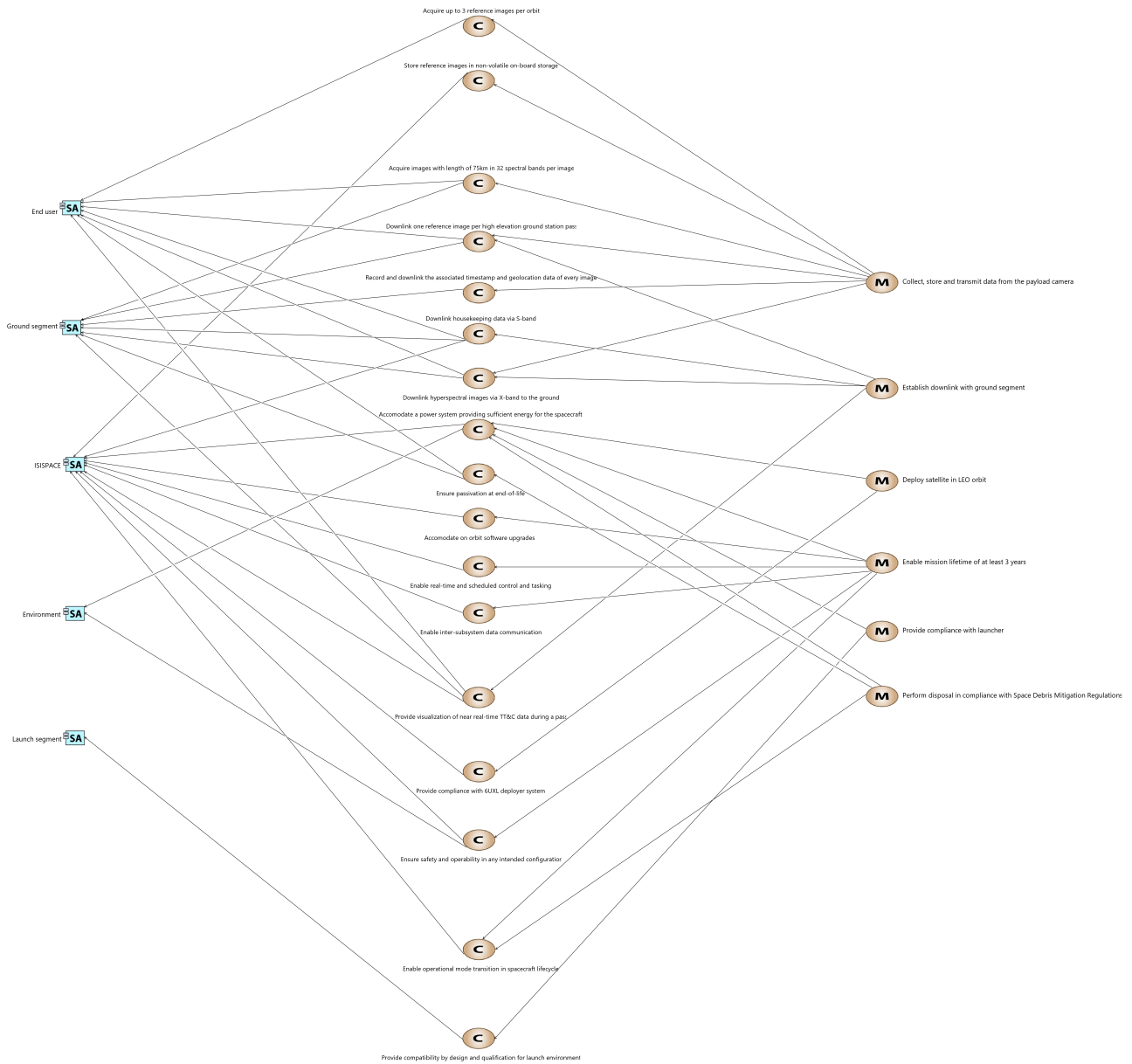


Figure 5.3: [MCB] Mission Capabilities Blank showing requirement flowdown and actor allocation

The system architecture, shown in Figure 5.4, defines the functions of the space segment and its interactions with external system actors. System functions are derived from the operational activities modelled at the previous layer, preserving the traceability established earlier. Functions and their exchanges are allocated to system actors, and parent functions are created for each subsystem to introduce an initial structural decomposition. Component exchanges define the interfaces between the space segment and external actors, with component ports associated to their corresponding functional ports. The System Architecture Blank provides the first model-based view of the space segment as a bounded system with well-defined external interfaces.

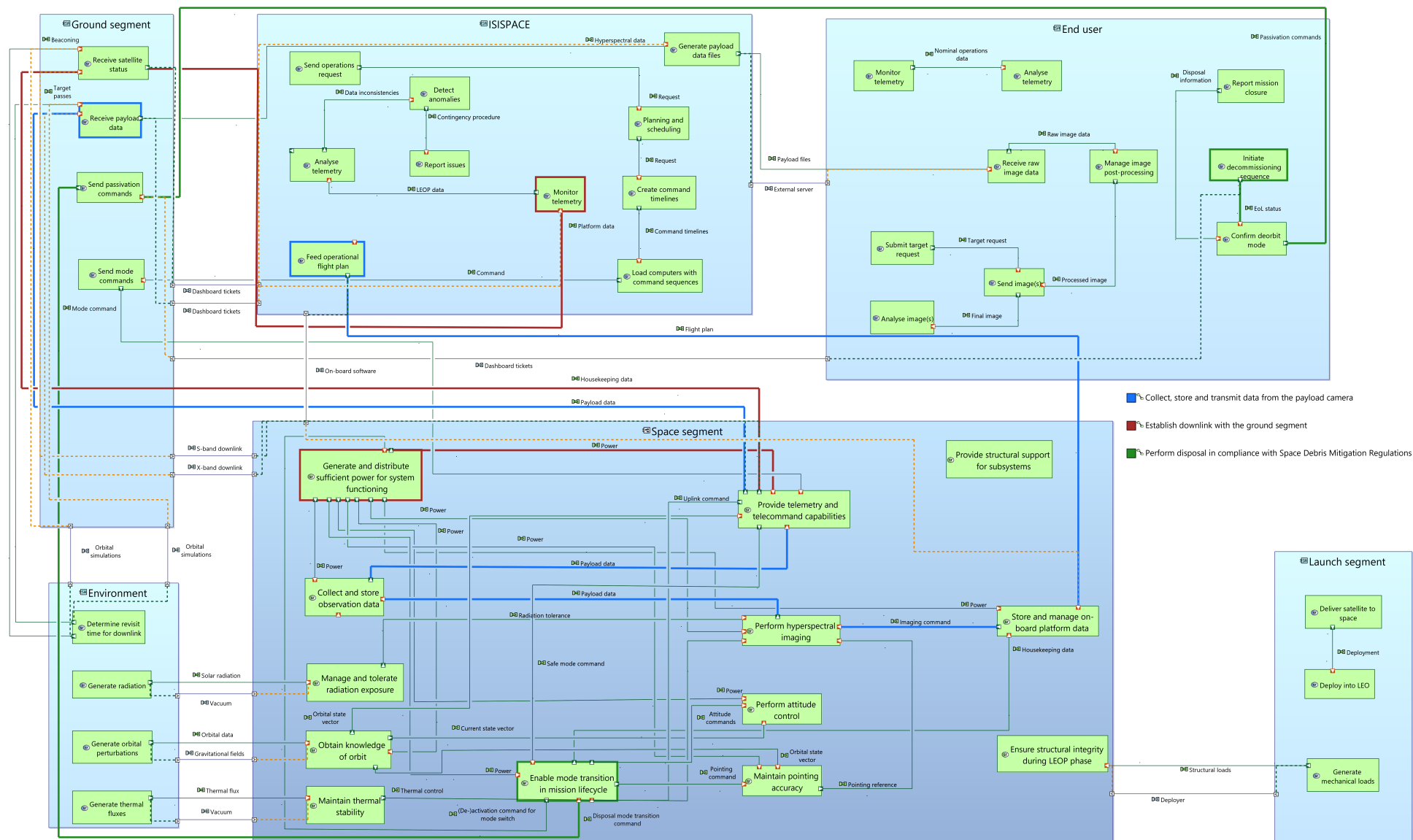


Figure 5.4: [SAB] System Architecture Blank showing the space segment and its external interfaces

The Concept of Operations is defined using a Mode State Machine (MSM) diagram, shown in Figure 5.5. Modes represent each operational state of the satellite [55], with transitions defined through algorithmic choices or direct state changes from deployment through to disposal. This diagram provides the operational context within which subsystem modes and satellite configurations are later defined at the Physical Architecture layer.

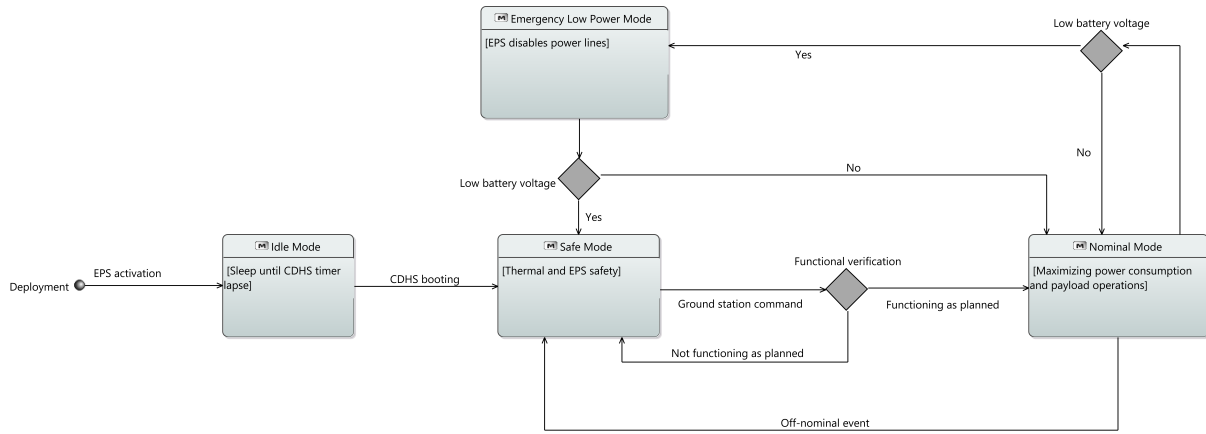


Figure 5.5: [MSM] Satellite operative modes

The satellite configurations within the nominal mode are also modelled at this layer using a separate MSM diagram, with each configuration represented as a sub-mode. This diagram is provided in Figure A.3 in Appendix A.

5.3. Logical Architecture

The transition from System Analysis to Logical Architecture marks a shift in focus from what the system must do to how it will be structured to do it. The artefacts from the System Analysis layer are carried over and progressively refined to define leaf components and their functions. The product tree of the space segment [11] is first established through the Logical Component Breakdown Diagram (LCBD), as shown in Figure 5.6, defining 7 subsystems and 29 equipment items that form the basis for functional allocation in subsequent diagrams. Note that the light blue boxes visible in the diagram represent logical actors rather than system components and are not part of the spacecraft product tree breakdown.

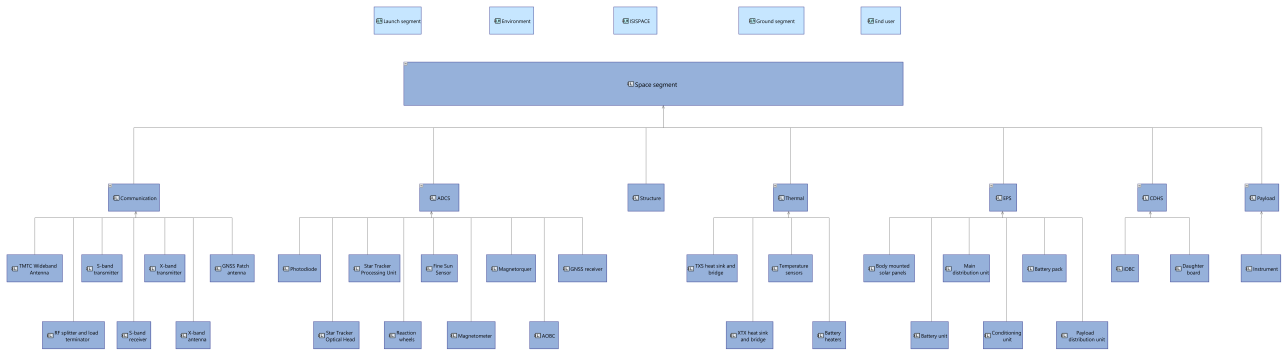


Figure 5.6: [LCBD] Logical Component Breakdown Diagram of the space segment

Parent functions from the System Analysis layer are then decomposed into leaf functions and allocated to their respective components. Functions are defined based on component allocation rather than being decomposed independently first, which keeps the model grounded in actual design intent and reduces the risk of defining functions with no physical realisation. The Logical Function Breakdown Diagram (LFBD) for the hyperspectral imaging function is shown in Figure 5.7 as a representative example. The LFBDs for all remaining parent functions are provided in Appendix A.

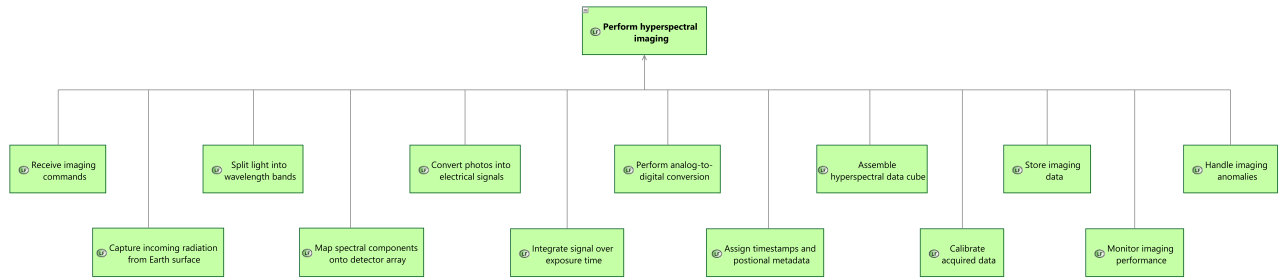


Figure 5.7: [LFBD] Logical Function Breakdown Diagram for the hyperspectral imaging function

Functional exchanges are then created between leaf functions to establish the data flow required to fulfil each parent function. This is captured in the Logical Dataflow Blank (LDFB) diagrams, with the hyperspectral imaging function shown as a representative example in Figure A.4 in Appendix A.

Subsystem mode transitions are subsequently defined using MSM diagrams for each subsystem, capturing the operational modes and transitions led by the onboard software. The ADCS subsystem MSM is shown in Figure 5.8 as a representative example, illustrating the mode transitions from detumbling after deployment to fine pointing during imaging passes [56].

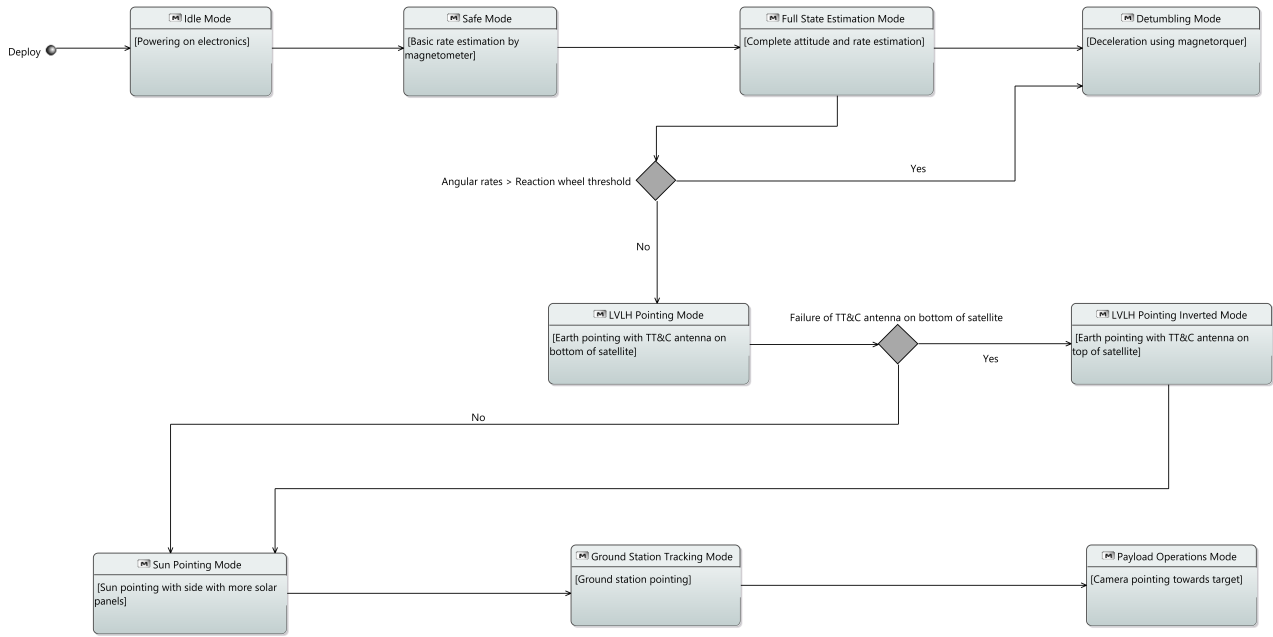


Figure 5.8: [MSM] ADCS subsystem mode transition diagram

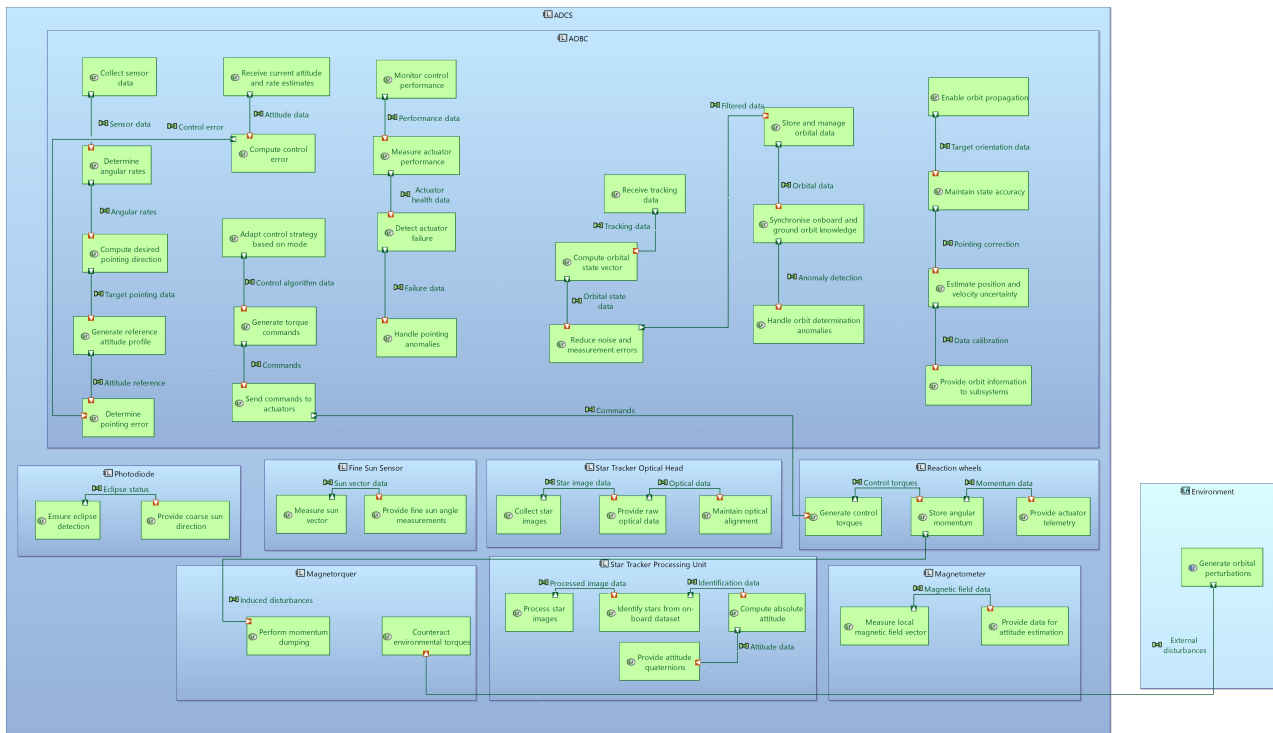


Figure 5.9: [LAB] ADCS subsystem logical architecture

The subsystem architectures are then combined into the system-level Logical Architecture Blank (LAB), shown in Figure 5.10. This step defines the inter-subsystem functional exchanges that cannot be captured within individual subsystem architectures, such as attitude data transfer from ADCS to OBC or power distribution from EPS to other subsystems.

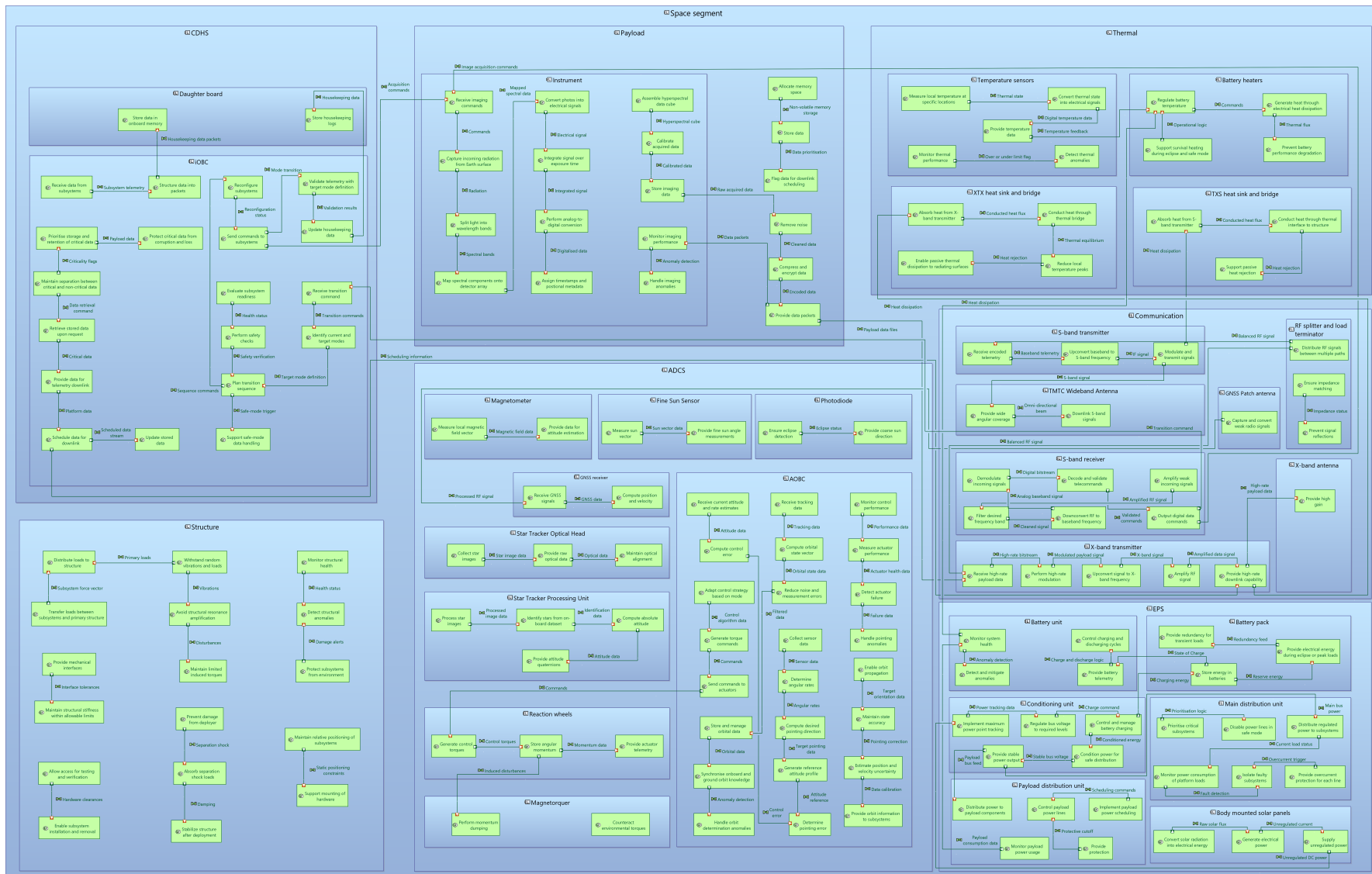


Figure 5.10: [LAB] System-level Logical Architecture Blank of the space segment

With subsystem modes and functions defined in sufficient detail, Exchange Scenarios are used to represent specific capabilities in the time domain, showing the ordered sequence of interactions needed to realise them. An Exchange Scenario for the capability “*Acquire images with length of 75km in 32 spectral bands per image*” is provided in Figure A.5 in Appendix A.

Interface definition is also addressed at this layer, where the type and direction of each interface is formally established using component exchanges. Interfaces are separated by type for clarity. The power and thermal interfaces are shown in Figure 5.11, and the control, data, and RF interfaces are provided in Figure A.6 in Appendix A. Establishing this separation at the logical level ensures it carries through consistently to the Physical Architecture layer, where each interface is assigned to a concrete hardware realisation.

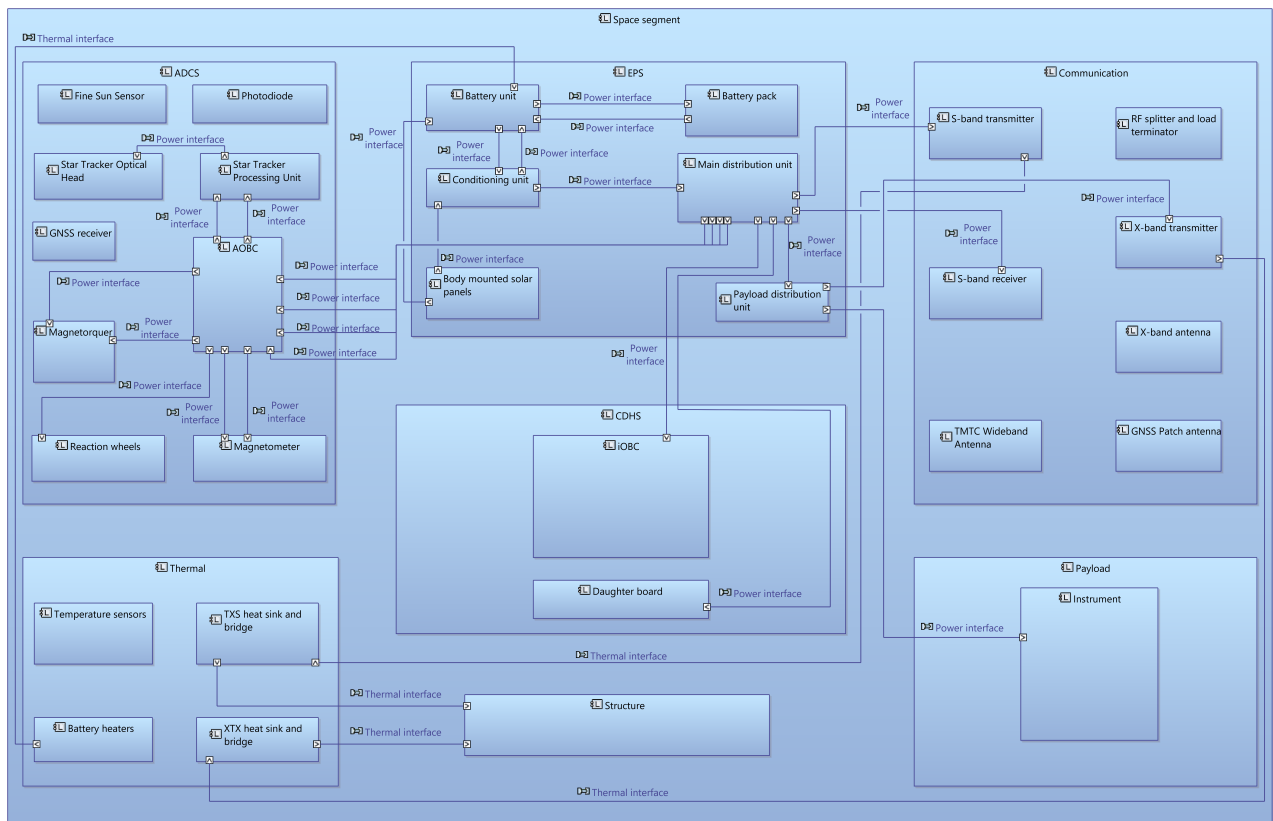


Figure 5.11: [LAB] Power and thermal interfaces of the space segment

5.4. Physical Architecture

The Physical Architecture layer is where the logical design converges with the hardware realisation of the satellite, and is the most consequential layer for the AIV engineering use case. Transitioning from the logical layer introduces Node and Behavioural Physical Components. Logical components transition as Behavioural Physical Components, preserving their functional responsibilities, while Node Physical Components are introduced to map them to the specific hardware items to be procured and assembled. Maintaining both representations preserves functional traceability while providing the

hardware specificity needed for procurement, configuration management, and integration planning.

The Physical Component Breakdown Diagram shown in Figure 5.12 illustrates this structure, with Node Physical Components placed adjacent to their behavioural counterparts and annotated with the hardware model designation [57]. TNote that the light blue boxes in the diagram represent physical actors such as the EGSE and ground support equipment, and are not part of the spacecraft hardware breakdown. This diagram directly feeds the integrated toolchain, as the Behavioural Physical Components and their deploying Node Physical Components form the source of the product tree and part number data imported into Valispace.

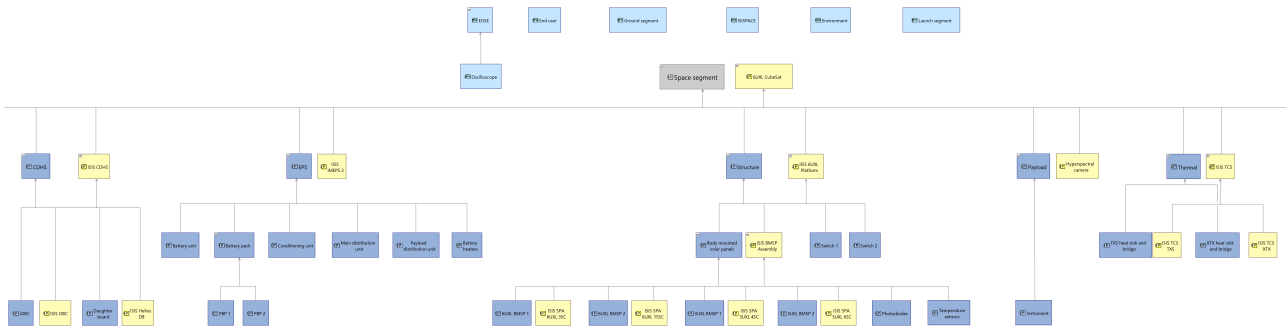


Figure 5.12: [PCBD] Physical Component Breakdown Diagram of the space segment

The Mode State Machine at this layer documents which components within each subsystem remain active during each satellite configuration [55]. Rather than defining new behavioural logic, it consolidates mode and configuration information from the system and logical layers into a hardware-centric view. The ADCS subsystem example is provided in Figure A.7 in Appendix A.

The SAIT map for the mission is modelled using the MSM diagram at this layer, where states represent verification phases and activities rather than operational modes. This deviates from the standard ARCADIA use of the MSM but is justified at the PA layer, where model elements do not propagate back to previous layers. The resulting SAIT plan captures the planned verification sequence from incoming inspection through environmental testing to final system integration, providing a shared model-based reference for the campaign [58].

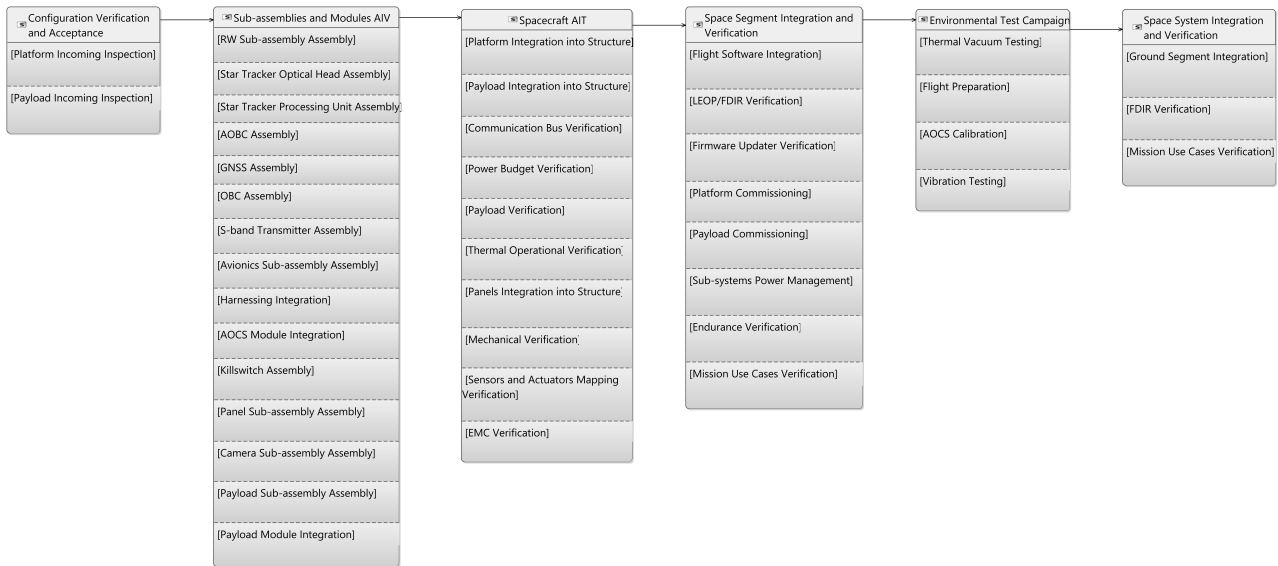


Figure 5.13: [MSM] System Assembly, Integration and Testing Map

The interfaces from the LA layer are transitioned and assigned to specific hardware connections, buses, and power rails [59]. The power interface diagram is shown in Figure 5.14, and the data, control, and RF interface diagrams are provided in Appendix A. This progression from logical to physical interface definition ensures every functional interface has a concrete physical realisation that can be inspected, tested, and verified. The component exchanges defined in these diagrams are extracted by the toolchain and imported into Valispace as interface connections between subsystem blocks.

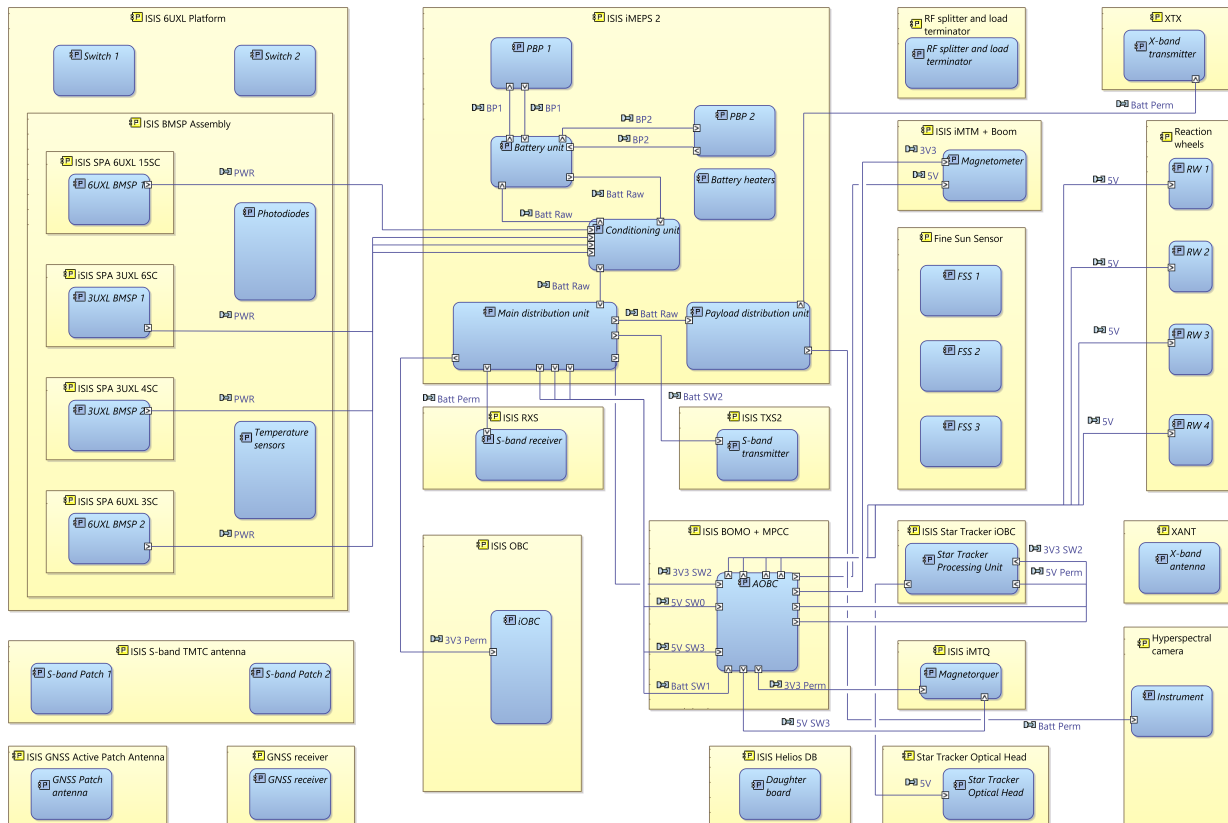


Figure 5.14: [PAB] Power interface diagram of the space segment

The physical interfaces are realised through harnesses, modelled in Capella using physical links and port features. The harness diagram for the flight model [60] is shown in Figure 5.15, with port labels corresponding to the harness identifiers used during integration activities. By encoding harness labels directly into the model, a direct link is established between the MBSE model and the physical integration documentation.

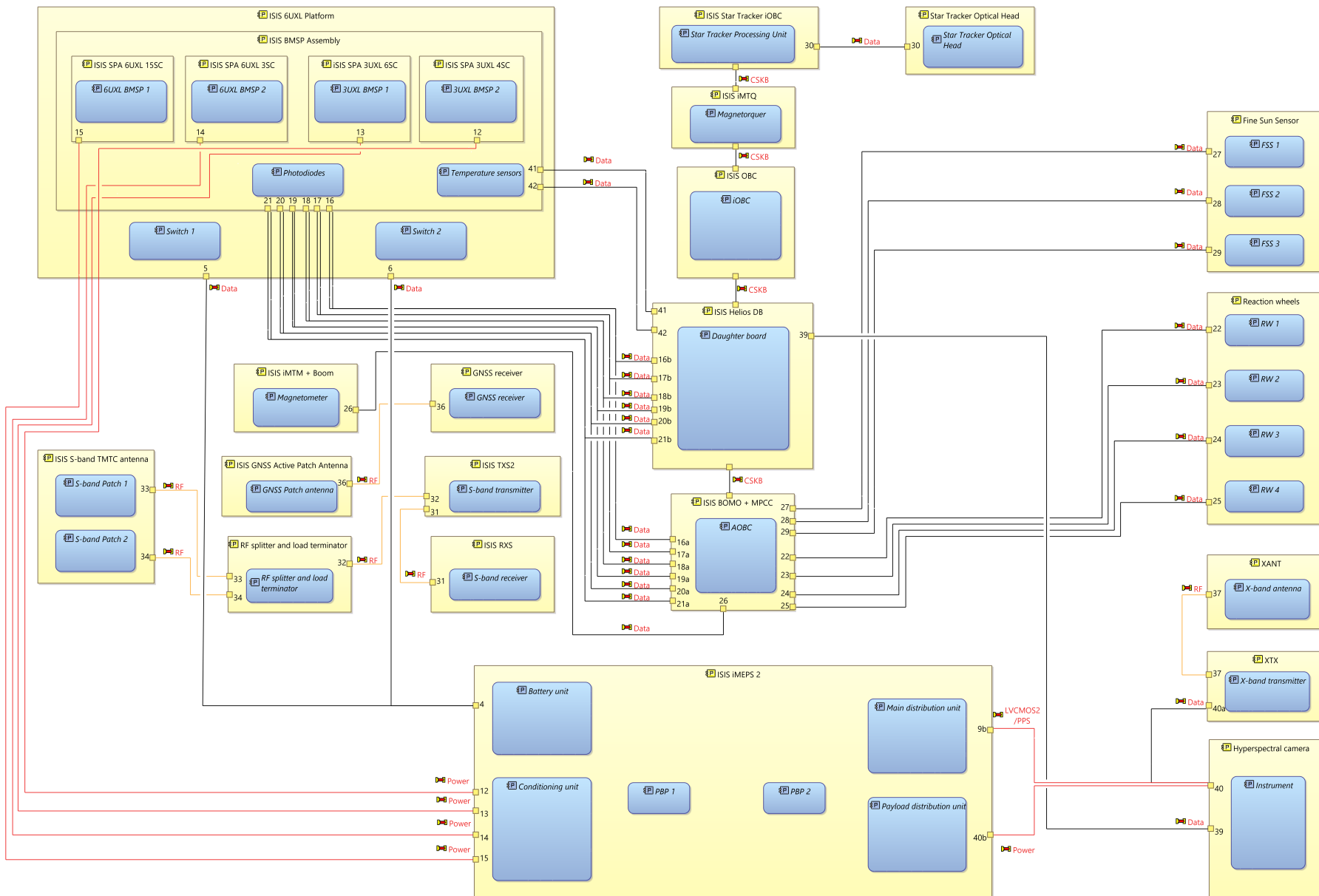


Figure 5.15: [PAB] Harness diagram for the flight model

5.5. Property Value Management Tool

The Property Value Management Tool (PVMT) is a Capella extension that allows typed property values to be defined and assigned to any model element. Within this framework, PVMT is the primary mechanism for embedding engineering data directly into the model, extending it beyond structural and functional representation into quantitative system assessment and verification management. Properties are organised into named domains, each grouping related properties scoped to the relevant model element types.

Two property domains are introduced as an example of modelling quantitative parameters for system budgeting. The mass budget domain, shown in Figure 5.16, stores component masses across different phases of design and integration, supporting consistency between the design record and the as-built state of the spacecraft [11]. The power budget domain, shown in Figure 5.17, captures the total maximum power consumed by each component. These two domains are the primary source of the budget properties extracted by the toolchain and imported into Valispace. While Capella does not natively aggregate component-level values into a system-level total, this is provided by the Valispace Modelists view once the properties are imported.

Name	Type	Default Value
>  Activity_status		
>  Component_status		
>  Functional_AIT		
▼  Mass_budget		
▼  Mass_Phases		
• Mass_SRR		
• Mass_PDR		
• Mass_CDR		
• Mass_FTRR		
• Mass_ETRR		
• Mass_QR		
• Mass_MRR		
▼  Designed_mass		
 Scope	[SYSTEM, LOGICAL]	
 Designed_mass	floatProperty	0.0 kg
 Mass_phase	Mass_Phases	Mass_SRR
▼  As_build_mass		
 Scope	[SYSTEM, LOGICAL]	
 As_build_mass	floatProperty	0.0 kg

Figure 5.16: Mass budget domain defined using PVMT

Name	Type	Default Value
>  Activity_status		
>  Component_status		
>  Functional_AIT		
>  Mass_budget		
▼  Power_budget		
▼  Power_consumed		
 Scope	[LOGICAL, PHYSICAL]	
 Total_max_power	floatProperty	0.0 W

Figure 5.17: Power budget domain defined using PVMT

A requirements domain is also defined to store the Valispace requirement identifiers assigned during the forward thread import. These fields remain empty until requirements are transferred from Capella to Valispace, at which point the generated identifiers are written back to the model through the reverse thread, establishing a persistent cross-tool reference. A verification control domain is additionally defined, containing enumeration properties for verification method and status on capability elements. These are populated by the reverse thread following AIV execution in Valispace, encoding the verification close-out record directly within the Capella model. The structure of the requirements domain as defined in PVMT is shown in Figure 5.18.

Name	Type	Default Value
>  Activity_status		
>  Component_status		
>  Functional_AIT		
>  Mass_budget		
>  Power_budget		
▼  Requirements		
▼  Requirement_values		
 Scope	[OPERATIONAL, SYSTEM]	
 Reference_images_per_orbit	floatProperty	00
 Image_length	floatProperty	00 km
 Spectral_bands_per_image	floatProperty	00
▼  Requirement_IDs		
 Scope	[SYSTEM, LOGICAL]	
 Mission_Requirements	stringProperty	
 System_Requirements	stringProperty	

Figure 5.18: Requirements domain defined using PVMT

With the system modelling complete, the following chapter describes how Capella, its add-ons, and Valispace are used to support AIT planning and verification engineering activities.

6

Designing Framework

This chapter presents the two AIV workflow approaches developed in this thesis. The first extends Capella's native capabilities into AIV engineering activities, establishing a model-based baseline for AIV planning, tracking, and verification management. The second extends this baseline by integrating Capella with Valispace through a bidirectional digital thread, forming the complete integrated MBSE-DDSE framework. Both approaches are applied to the 6U XL Earth Observation CubeSat mission.

6.1. Capella-only Approach

The Capella-only approach investigates the use of PVMT for verification planning, test procedure definition, and verification tracking within Capella. As this framework is built on the ARCADIA methodology, the scope is limited to functional AIT activities, which are most naturally expressed within the constructs Capella provides.

6.1.1. Requirement Linkage

The Requirements Viewpoint add-on is used to capture textual requirements directly within the Capella model, establishing formal links between each requirement and its corresponding mission or capability element at the System Analysis layer. System requirements are allocated to the space segment [54]. While the Requirements Viewpoint is limited to creating and linking requirements to model elements, the ARCADIA methodology supports further flowdown through the allocation of functions and components across the LA and PA layers, allowing all model elements to be assigned

the requirements they must satisfy. The allocation of system requirements to the space segment is shown in Figure 6.1.

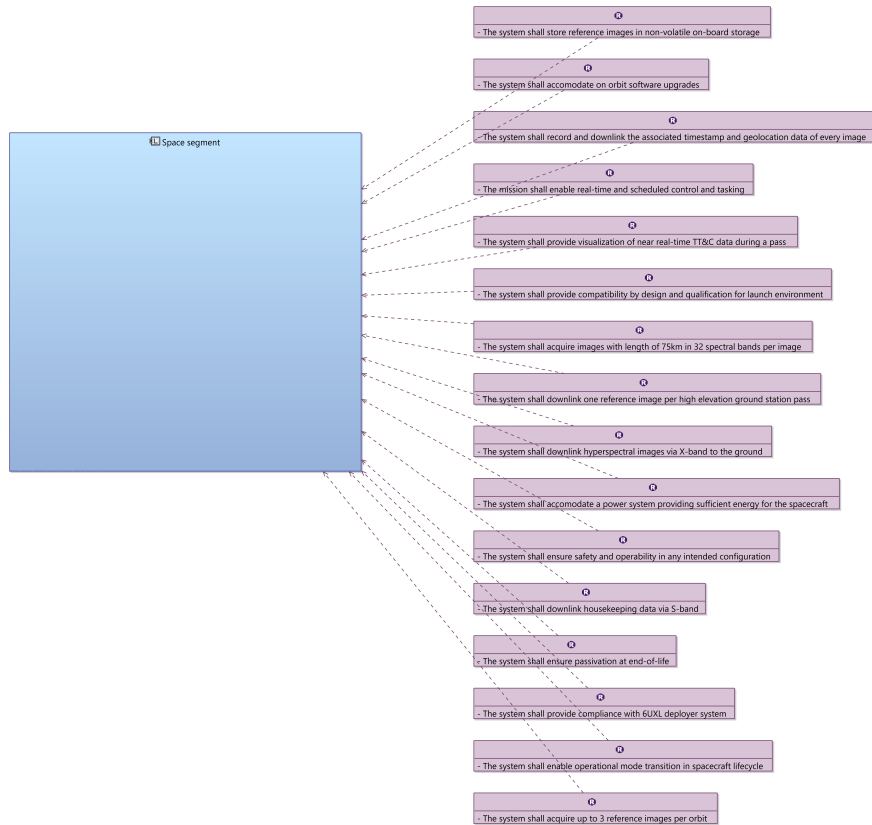


Figure 6.1: System requirements linked to the space segment at the Logical Architecture layer

6.1.2. Verification Planning

Verification planning establishes the methods and criteria used to demonstrate requirement compliance throughout development. A well-established principle is that this should begin alongside requirement definition, ideally from Phase A, rather than being deferred until hardware is available. Early planning ensures that verification methods and success criteria are defined while there is still opportunity to shape the verification campaign.

Within this framework, PVMT is used to implement verification control directly in the Capella model. Enumerations are defined for verification methods and status values, as shown in Figure 6.2, and made available for assignment to model elements. This embeds verification planning information alongside the design artefacts rather than in a separate document, keeping verification assignments traceable to the requirements they address.

Name	Type	Default Value
> Activity_status		
> Component_status		
> Functional_AIT		
> Mass_budget		
> Power_budget		
> Requirements		
> Verification_activities		
▼ Verification_control		
▼ Verification_status		
• Verified		
• Not_verified		
▼ Verification_method		
• Test		
• Inspection		
• Analysis		
• Review		
▼ Verification_control		
Scope	[SYSTEM]	
Verification_method	Verification_method	Test
Verification_status	Verification_status	Not_verified

Figure 6.2: Verification method and status enumerations defined using PVMT

When a model element is selected in Capella, the engineer is presented with the interface shown in Figure 6.3, which allows verification methods and status values to be assigned directly. This supports structured verification planning within the modelling environment and reduces the need for separate planning documents.

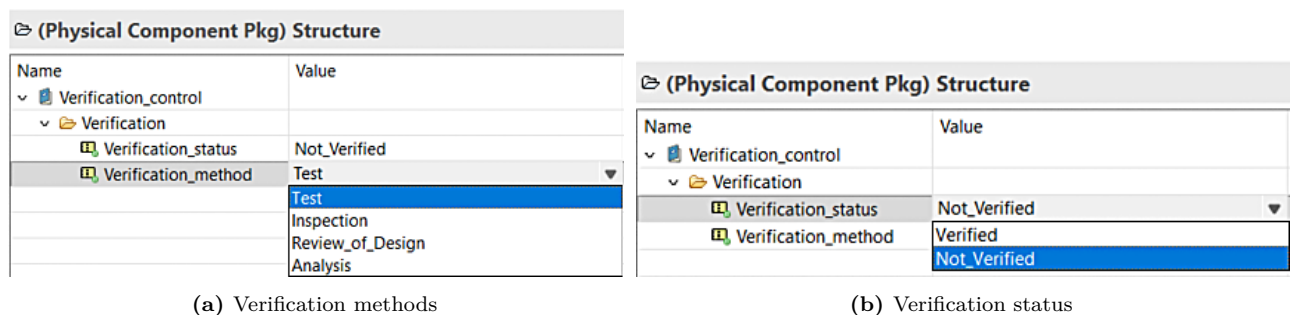


Figure 6.3: User interface for assigning verification method and status in Capella

6.1.3. Functional Test Modelling

A persistent challenge in spacecraft development is that requirements defined during conceptual design are often reinterpreted when generating verification matrices and test procedures, leading to duplicated effort and inconsistencies. This framework addresses that by defining test procedures directly within the Capella model, using physical functions as test activities and associating property values to capture expected results and performance parameters.

At the Physical Architecture layer, physical functions represent test activities and are allocated to the ISISPACE actor. Where applicable, the EGSE is introduced as a physical actor with its own

functions. The I2C signal verification test case is selected as the demonstration scenario.

An electrical integration test of the payload instrument was initially considered as a typical early AIV activity. However, as an equipment-level test, it does not map to any system requirement within the scope of this framework, which covers only mission and system requirements. Selecting such a test would leave the requirement-to-verification traceability chain incomplete. A platform-level functional test was therefore selected, as it satisfies a system-level capability requirement and demonstrates end-to-end traceability from mission objective through to verification close-out.

Test Architecture

The I2C Signal Verification test [61] is represented at the Physical Architecture layer through a PAB diagram capturing the test activities and associated actors, as shown in Figure 6.4. This diagram shows which components and functions are involved but does not impose a sequential order, as the same functions may appear in a different sequence in other procedures.

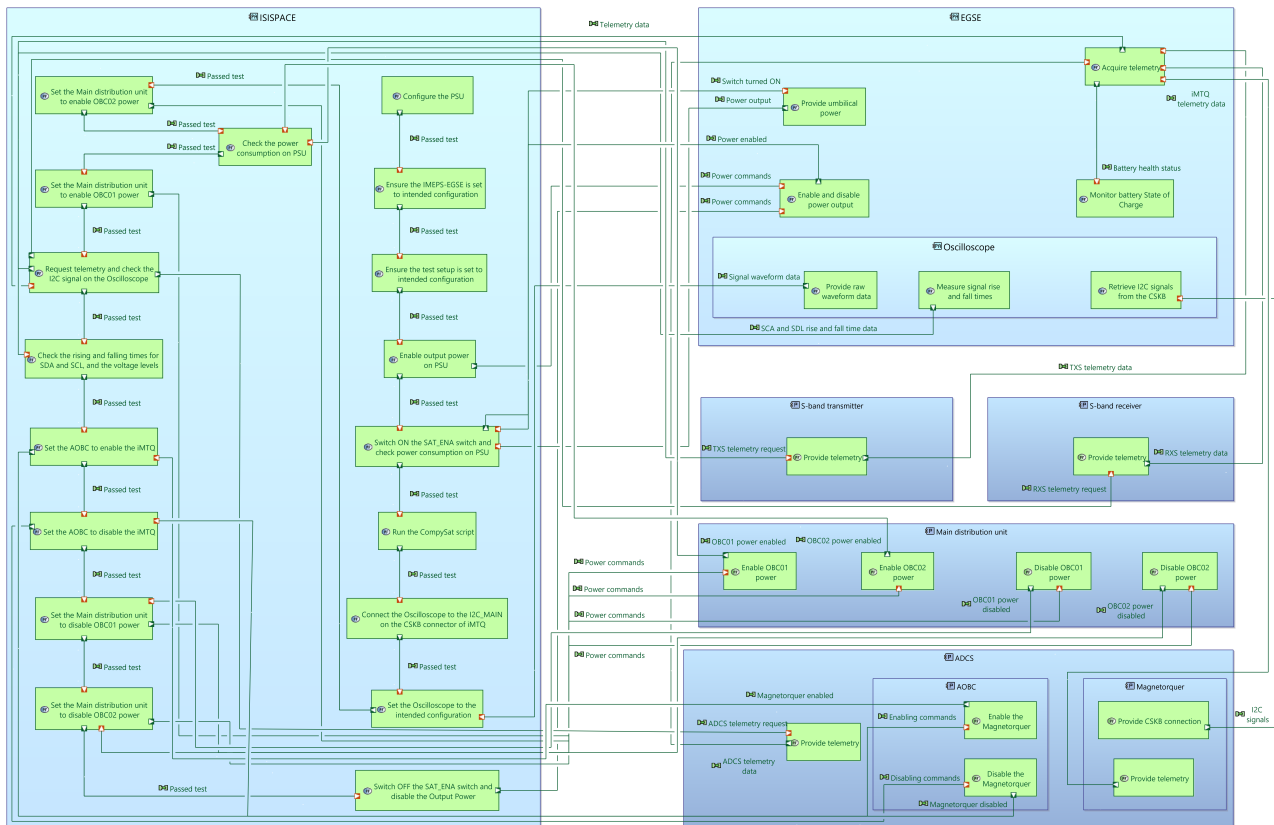


Figure 6.4: [PAB] I2C Signal Verification test architecture

Test Procedure

The sequential execution of test steps is captured using Capella's Exchange Scenario feature. Functions are arranged top to bottom to reflect the execution order, and each step is associated with the responsible component or actor. This produces a model-based test procedure directly traceable to the

capability it verifies. The Exchange Scenario for the I2C Signal Verification [62] is shown in Figure 6.5, with the ISISPACE actor on the left and its allocated physical functions representing the test steps.

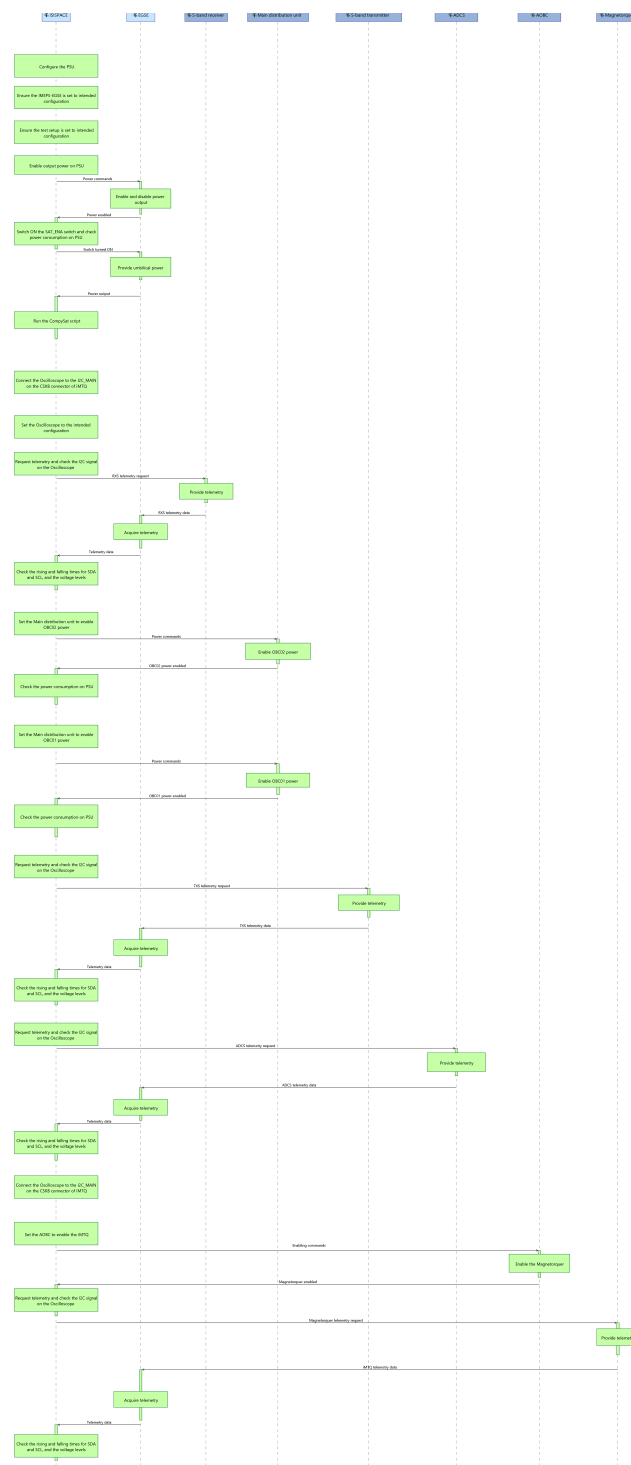


Figure 6.5: [ES] I2C Signal Verification test procedure

This approach captures procedures in their as-planned form only. As-run deviations and change records are not captured within the model, which is a recognised limitation addressed in the future work.

Test Results

Expected test results are captured as property values on physical functions. Enumerations represent the discrete modes and states a component is expected to exhibit at each step, as shown in Figure 6.6. The status of each test activity can also be tracked using enumeration properties, as shown in Figure 6.7.

Name	Type	Default Value
> Activity_status		
▼ Component_status		
▼ SAT_status		
• ON		
• OFF		
▼ OBC_Status		
• OFF		
• ON		
▼ BATT_Status		
• Connected		
• Disconnected		
▼ EGSE_status		
• Enabled		
• Disabled		
▼ COMPYSAT_Status		
• Running		
• Not running		
▼ XTX_MODE		
• IDLE		
• CONFIG		
• STANDBY		
• FLASH_ACCESS		
▼ PAYLOAD_RESET_REASON		
• Power_up		
• Instruction_cache_uncorrectable_error		
• Data_cache_uncorrectable_error		
• Bootloader_RAM_uncorrectable_error		
• Application_RAM_uncorrectable_error		
• Watchdog_time_out		
• Reset_command		
• Latch_up_power_off		
▼ PSU_Output		
• Enabled		
• Disabled		
▼ Status		
Scope	[PHYSICAL]	
SAT_Status	EGSE_status	Disabled
EGSE_Status	EGSE_status	Disabled
BATT_Status	BATT_Status	Disconnected
XTX_MODE	XTX_MODE	STANDBY
PAYLOAD_RESET_REASON	PAYLOAD_RESET_REASON	Power_up
COMPYSAT_Status	COMPYSAT_Status	Not running
PSU_Output	PSU_Output	Disabled
OBC_Status	OBC_Status	OFF

Figure 6.6: Component status tracking using PVMT

Name	Type	Default Value
▼ Activity_status		
▼ ACT_STATUS		
• PASS		
• FAIL		
• PARTIAL		
▼ Activity_status		
Scope	[PHYSICAL]	
ACT_Status	ACT_STATUS	FAIL

Figure 6.7: Test activity status tracking using PVMT

Extensions are used to define numerical expected values, with string and boolean types also supported, as shown in Figure 6.8. Together, these allow the expected outcome of each test step to be formally defined within the model.

Name	Type	Default Value
Functional_AIT		
AIT_results		
Scope	[PHYSICAL]	
MODE	integerProperty	0
current	floatProperty	0.0 mA
voltage	floatProperty	0.0
interface_type_identifier	integerProperty	0
board_identifier	integerProperty	0
system_type_identifier	integerProperty	0
uptime	floatProperty	0.0 sec
vip_input	floatProperty	0.0 V
vip_input_current	floatProperty	0.0 mA
vip_output	floatProperty	0.0 V
heater_active	integerProperty	0
under_voltage_cell	floatProperty	0.0 V
over_voltage_cell	floatProperty	0.0 V
balancing_cell	floatProperty	0.0 V
temperature	floatProperty	0.0 degC
volt_cell	floatProperty	0.0 mV
vipcc_output	floatProperty	0.0 V
harness_used	integerProperty	0
vip_vd	floatProperty	0.0 V
vip_obc	floatProperty	0.0 V
ovp_value	floatProperty	0.0 V
sn_id	integerProperty	0
xtx_id	integerProperty	0
frequency	floatProperty	0.0 MHz
symbol_rate	integerProperty	0 Msps
i2c_status	integerProperty	0
pa_vbat	floatProperty	0.0 mA
pa_vcc	floatProperty	0.0 mA
board_bat	floatProperty	0.0 mA
pa_temp	floatProperty	0.0 degC
pa_smtps_temp	floatProperty	0.0 degC
fpga_int_temp	floatProperty	0.0 degC
board_temp	floatProperty	0.0 degC
imager_product_id	integerProperty	0
imager_serial_number	floatProperty	0.0
firmware_version	floatProperty	0.0
start_up_state	integerProperty	0
subsystem_state	integerProperty	0
sda_voltage	floatProperty	0.0 V
sda_rising_time	floatProperty	0.0 micro_sec
sda_falling_time	floatProperty	0.0 micro_sec
scl_voltage	floatProperty	0.0 V
scl_rising_time	floatProperty	0.0 micro_sec

Figure 6.8: Capturing expected test results using PVMT extensions

6.1.4. Verification Tracking

Maintaining visibility of verification status throughout a test campaign is essential for identifying compliance risks before they become schedule-critical. Verification phases are defined as enumerations with individual activities as their literals, ensuring a consistent tracking structure aligned between systems and AIV engineers. The ISISPACE-defined verification phases [58] are modelled as enumeration literals as shown in Figure 6.9.

Name	Type	Default Value
▼  Verification_activities		
▼  Configuration_and_Acceptance		
• Platform_Incoming_Inspection		
• Completed		
• Payload_Incoming_Inspection		
• Not_initiated		
▼  Sub_Assembly_Module_AIV		
• RW_Sub_Assembly		
• Camera_Sub_Assembly_Assembly		
• Harnessing_Integration		
• Killswitch_Assembly_Into_Structure		
• Panel_Sub_Assemblies_Assembly		
• Platform_Module_Integration		
• AOCS_Module_Integration		
• Payload_Module_Integration		
• Payload_Sub_Assembly_Assembly		
• Avionics_Sub_Assembly_Assembly		
• STR_Sub_Assembly_Assembly		
• AOBC_Sub_Assemblies_Assembly		
• Not_initiated		
• Completed		
▼  Space_Segment_Integration_Verification		
• Flight_Software_Integration		
• FDIR_Verification		
• Firmware_Update_Verification		
• Platform_Commissioning		
• Payload-Commissioning		
• Subsystem_Power_Management		
• Endurance_Verification		
• Mission_Use_Cases_Verification		
• Not_initiated		
• Complete		
▼  Spacecraft_AIT		
• Platform_Integration_Into_Structure		
• Payload_Integration_Into_Structure		
• Communication_Bus_Verification		
• Power_Budget_Verification		
• Payload_Verification		
• Thermal_Operational_Verification		
• Panels_Integration_Into_Structure		
• Mechanical_Verification		
• Sensors_Actuators_Mapping_Verification		
• EMC_Verification		
• Not_initiated		
• Completed		

Figure 6.9: Verification phase tracking using PVMT

An extension under the same domain is modelled as enumeration properties, as shown in Figure 6.10, allowing the AIV engineer to assign and update the verification phase of each component or subsystem as the campaign progresses.

Name	Type	Default Value
> Activity_status		
> Component_status		
> Functional_AIT		
> Mass_budget		
> Power_budget		
> Requirements		
▼ Verification_activities		
> Configuration_and_Acceptance		
> Sub_Assembly_Module_AIV		
> Space_Segment_Integration_Verification		
> Spacecraft_AIT		
> Environment_Test_Campaign		
> System_Integration_Verification		
▼ Verification_activities		
Scope	[PHYSICAL]	
Configuration_and_Acceptance	Configuration_and_Acceptance	Not_initiated
Sub_Assembly_Module_AIT	Sub_Assembly_Module_AIV	Not_initiated
Spacecraft_AIT	Spacecraft_AIT	Not_initiated
Space_Segment_Integration_Verification	Space_Segment_Integration_Verification	Not_initiated
Environment_Test_Campaign	Environment_Test_Campaign	Not_initiated
System_Integration_Verification	System_Integration_Verification	Not_initiated

Figure 6.10: Verification phase extension using PVMT

The resulting user interface is shown in Figure 6.11. By assigning the current verification phase directly within Capella, the model becomes a live record of campaign progress, allowing the team to assess verification maturity at the component, subsystem, or system level at any point.

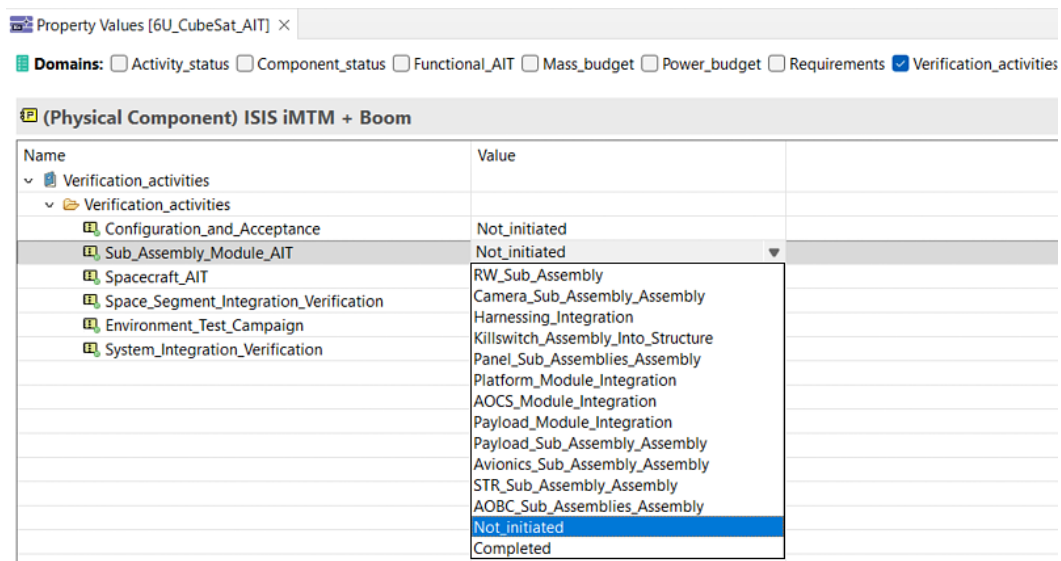


Figure 6.11: User interface for assigning verification phase in Capella

6.2. Capella-Valispace Approach

For small satellite missions, toolchain integration is particularly valuable. Limited team sizes place a premium on efficient information exchange between systems engineers and AIV teams, and maintaining consistency across disconnected tools becomes increasingly difficult as the design matures. An integrated environment reduces duplicated effort, supports collaboration, and preserves an Authoritative Source of Truth across all lifecycle phases.

6.2.1. Forward Thread

Figure 6.12 shows the data flow from Capella to Valispace through an Excel intermediate layer. Model artefacts are extracted using Python4Capella scripts and written to structured Excel files, which serve as a staging layer before import into Valispace. This intermediate step decouples extraction from import, allowing each script to operate independently and enabling data to be inspected and validated before being committed to Valispace. The data elements sourced from Capella and their corresponding targets in Valispace are as follows.

The physical product tree including BehaviorPCs and their deploying NodePCs is imported as a block hierarchy in Valispace. Designed mass and total maximum power PVMT properties are attached as numerical Vali entities on each block. Component exchanges are imported as connections between subsystem blocks representing physical interfaces. Missions and capabilities from the System Analysis layer are imported as mission and system requirements with their parent-child hierarchy preserved. Exchange Scenario steps from the Physical Architecture layer are imported as V&V activities and ordered step definitions. Physical links and port allocations are exported to Excel for reference only and are not imported into Valispace, as the platform does not natively support harness or port-level representations within the scope of this framework.

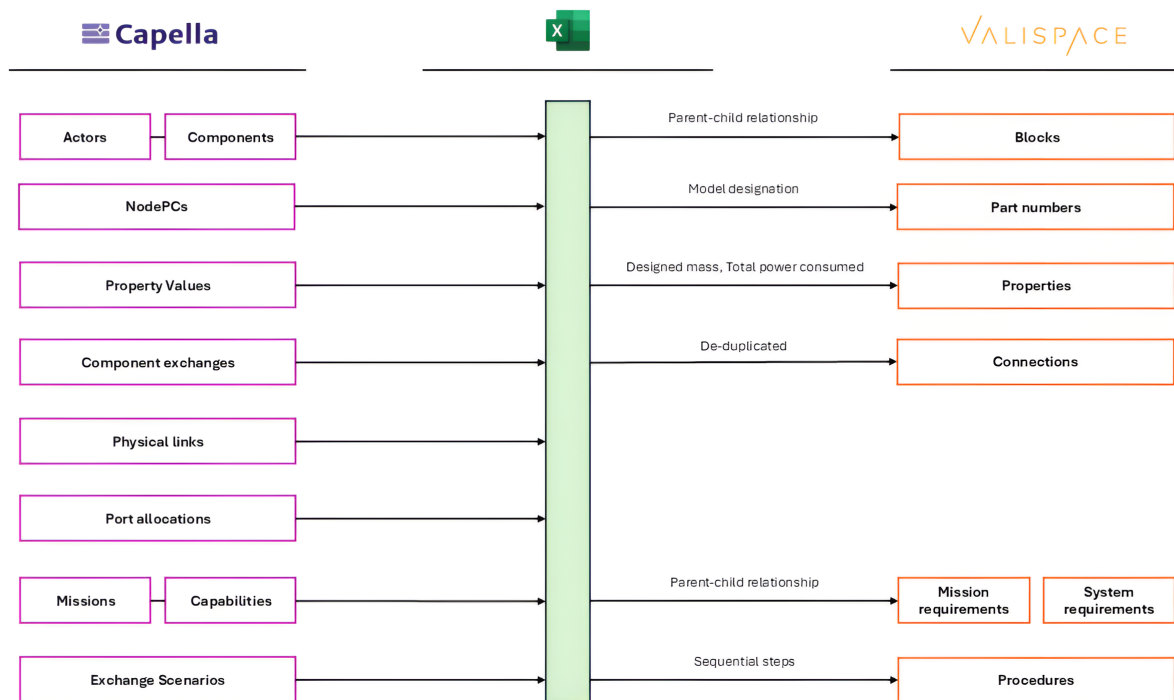


Figure 6.12: Capella to Valispace data mapping via Excel intermediate layer

The mapping applies specific transformations at each stage. Physical components (BehaviorPCs) and system actors are posted as blocks in Valispace, preserving the parent-child hierarchy from the Capella product tree. Designed mass and total maximum power properties are attached to each

block as numerical values. Component exchanges are de-duplicated during export and imported as connections between blocks representing physical interfaces. Missions and capabilities from the System Analysis layer are mapped to mission and system requirements in Valispace with their hierarchy preserved. Exchange Scenarios from the Physical Architecture layer are extracted as ordered steps and imported as V&V procedures, directly linking the Capella test architecture to the Valispace verification execution environment.

The forward thread is implemented as a two-stage pipeline. Python4Capella export scripts extract model artefacts into structured Excel files within Capella, and Python import scripts then read these files and post the data to Valispace via the REST API. Before execution, three configuration parameters must be set: the Valispace API URL, the project name, and a user access token from the Valispace settings. All import scripts run in dry-run mode by default, printing planned operations without committing data. Setting `DRY_RUN = False` enables the live import.

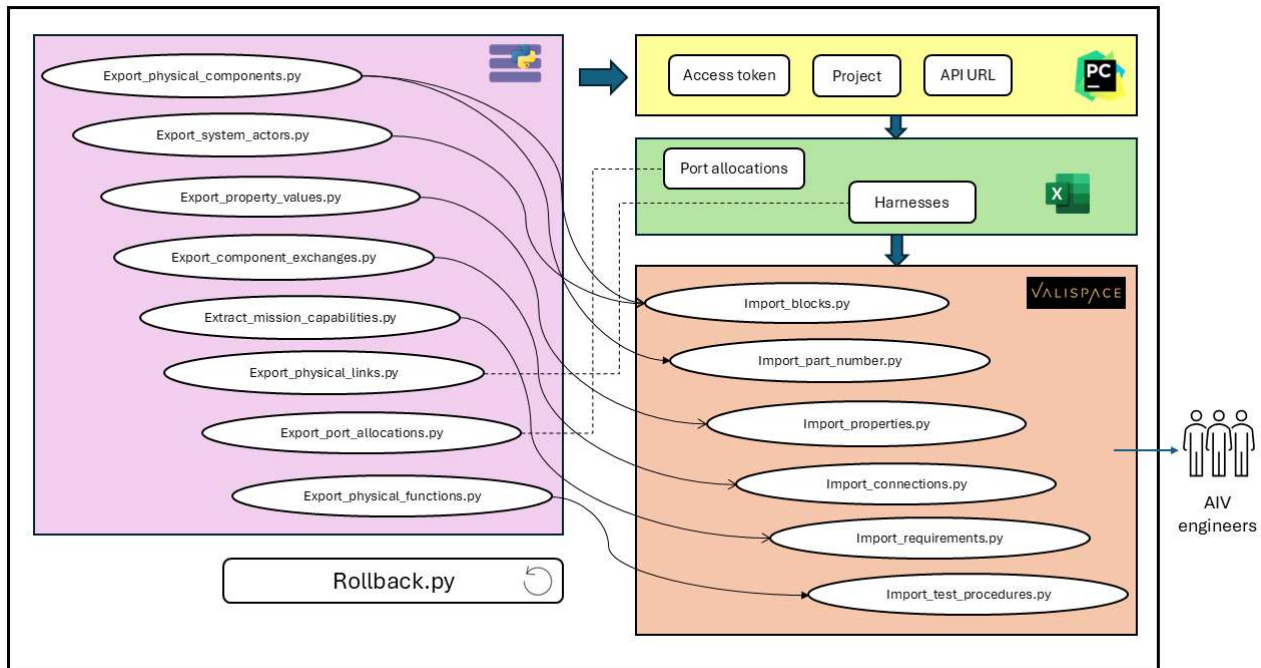


Figure 6.13: Forward digital thread toolchain architecture

Export Scripts

Eight Python4Capella scripts extract model artefacts from the Capella model and write them to Excel. These scripts are executed within Capella using the EASE scripting environment.

export_physical_components.py This script traverses the Physical Architecture component tree recursively from the root `PhysicalSystem`. A configurable depth limit controls how many levels of the hierarchy are extracted to prevent inclusion of unnecessarily detailed sub-components. For

each BehaviorPC, the script identifies the NodePC deployed on it, which serves as the hardware unit identifier for part number tracking in Valispace. It extracts the component name, element type, parent component name, and deploying NodePC designation, writing one row per component to the Excel file. Components with identical names at different levels of the hierarchy may cause matching ambiguity during import.

export_system_actors.py This script accesses the System Analysis layer and reads the actor elements defined there, capturing both operational and system actors to ensure all top-level block entities required for the Valispace product tree are extracted. Actors referenced informally within Exchange Scenarios but not formally defined at the SA level are not captured.

export_property_values.py This script uses the Python4Capella PVMT API to access the `Designed_mass` and `Total_max_power` properties from the `Mass_budget` domain on each physical component, writing the values alongside the component name and Capella element ID. Properties not set in Capella are exported as empty fields and skipped during import.

export_component_exchanges.py This script traverses the component exchange connections in the Physical Architecture layer, reading the name, source component, and target component for each interface. Interface names are sanitised during export to remove special characters that would cause API failures. A deduplication step ensures interfaces with identical names between the same component pair are not exported twice.

export_missions_capabilities.py This script traverses Mission elements in the Operational Analysis layer and Capability elements in the System Analysis layer, reading the name, description, parent element name, and any existing Valispace requirement identifier previously written as a PVMT string property during a prior import run. Only mission and capability elements are in scope; subsystem and equipment-level requirements must be entered manually in Valispace.

export_physical_functions.py This script traverses Exchange Scenario elements in the Physical Architecture layer. For each scenario, it reads the ordered message interactions and identifies the sending and receiving actors for each step. Steps allocated to actors other than the primary ISISPACE actor are flagged, with the actor name appended in square brackets to encode role responsibilities in the step label. Only Exchange Scenarios with formally defined ordered message sequences are extracted.

export_physical_links.py This script extracts harness routing data from physical link elements in the Physical Architecture layer and writes it to Excel for reference. This file is not imported into Valispace and serves as the harness design list reference document for the AIV team [63].

export_port_allocations.py This script extracts port allocation data from the PA layer and writes it to Excel for reference. Like the physical links file, it is not imported into Valispace.

Import Scripts

Six Python scripts read the exported Excel files and post the data to Valispace via the REST API, executed on the desktop using a standard Python environment.

import_blocks.py This script reads the PA product tree and SA actors Excel files and builds the full list of blocks to be created in Valispace. It fetches all existing blocks to build an idempotency map, then posts blocks from root to leaves using the POST `/components/` endpoint, ensuring parent blocks are created before children. Blocks with identical names within the same parent are treated as duplicates and skipped on re-runs. Renaming a component in Capella and re-running creates a new block rather than updating the existing one, requiring a prior rollback of the affected elements.

import_part_number.py This script reads the deploying NodePC information from the PA product tree Excel file and for each BehaviorPC updates the part number field in Valispace using a PATCH request to the `/components/id/` endpoint. This script requires **import_blocks.py** to have been executed beforehand.

import_properties.py This script reads the property values Excel file and for each component posts the `Designed_mass` and `Total_max_power` values as numerical Vali entities using the POST `/valis/` endpoint. Properties with empty or null values are skipped. The script requires **import_blocks.py** to have been executed first.

import_connections.py This script reads the component exchanges Excel file and resolves the source and target block IDs in Valispace by name. Where an exact match is not found, a parent chain fallback searches iteratively up the hierarchy. Once resolved, connections are created using the POST `/components/connections/` endpoint with a unique name combining the interface name and source and target component names. The fallback may resolve sub-components to an incorrect parent in highly nested hierarchies.

import_requirements.py This script posts mission requirements first to establish the top-level hierarchy, followed by system requirements as children using the POST `/requirements/` endpoint. After each successful post, the Valispace-assigned identifier is written back to the Excel file for use in the reverse flow. The Excel file must remain at the same path between the forward and reverse import runs.

import_test_procedures.py This script creates a folder structure in Valispace corresponding to the Exchange Scenario names, then creates a V&V activity and posts each step as a step definition using the POST /verification-and-validation/step-definition/ endpoint. Steps are posted in the order recorded in Excel to preserve the execution sequence. Steps allocated to non-primary actors are identified by the square bracket notation in the step name. Step definitions are derived from Capella and may require manual refinement in Valispace to be sufficiently descriptive for test execution.

Idempotency and Rollback

All import scripts implement idempotency checks before posting data to Valispace. Existing elements are fetched at the start of each run and compared against the data to be imported, with duplicates skipped. Each successful operation is recorded in a timestamped run log file, as shown in Figure 6.14.



Figure 6.14: Timestamped run log files generated during import into Valispace

If an import needs to be undone, **rollback.py** reads the run log and clears each created element via the Valispace API. The rollback covers all element types including blocks, valis, requirements, connections, V&V folders, activities, and step definitions. The part number field is cleared via a PATCH request rather than DELETE, as it is a property of the block and not an independent entity.

Beyond undoing imports, the run log provides a lightweight version control mechanism over the Valispace environment. Each execution generates a uniquely timestamped log, allowing the full history of toolchain imports to be traced. When the Capella model is revised, the engineer can roll back the previous import using the corresponding log and re-run the updated scripts, replacing the Valispace content without manual deletion. The combination of idempotency and rollback enables the Capella model to serve as the authoritative source of truth for the Valispace environment.

6.2.2. Reverse Thread

Figure 6.17 illustrates the reverse data flow from Valispace back into the Capella model through the Excel intermediate layer. Four categories of data are written back, each as PVMT property values on the corresponding model elements using Python4Capella scripts. The data elements sourced from Valispace and their corresponding targets in Capella are as follows.

As-built mass measurements logged by the AIV engineer are written to the `As_build_mass` property on physical components. Requirement identifiers generated by Valispace during the forward import are written to the `System_Requirements` and `Mission_Requirements` string properties on capability elements. Verification method and verification status values assigned following test execution are written to the corresponding enumeration properties on capability elements. This reverse flow closes the digital thread by ensuring that AIV execution data generated in Valispace is reflected in the authoritative system model in Capella, maintaining consistency between the as-executed state and the design record.

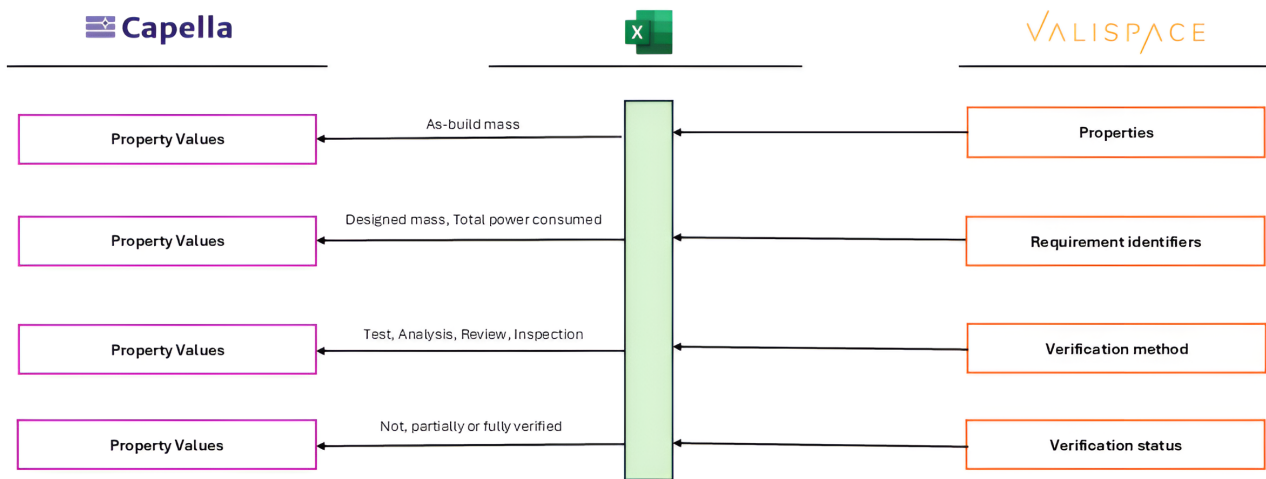


Figure 6.15: Valispace to Capella data mapping via Excel intermediate layer

As-built mass measurements logged by the AIV engineer in Valispace are exported to Excel and subsequently written to the corresponding PVMT property on each physical component in Capella, enabling a direct comparison between the designed mass and the physically measured as-built mass within the model. Requirement identifiers assigned by Valispace during the import of missions and capabilities are written back to the respective model elements as string property values, establishing a cross-tool reference that links each Capella element to its Valispace counterpart.

Verification method and verification status, assigned by the AIV engineer in Valispace following test execution, are written back to the corresponding mission and capability elements in Capella as enumeration property values. Together these four reverse flows complete the bidirectional digital thread, ensuring that the Capella model reflects the verified and as-built state of the spacecraft at the conclusion of Phase D.

The reverse digital thread uses the same two-stage pipeline as the forward flow, operating in the opposite direction. Python scripts executed on the desktop fetch engineering data from the Valispace REST API and write it to Excel files. Python4Capella scripts executed within Capella then read

these files and write the data as property values on the corresponding model elements. Figure 6.18 illustrates this architecture.

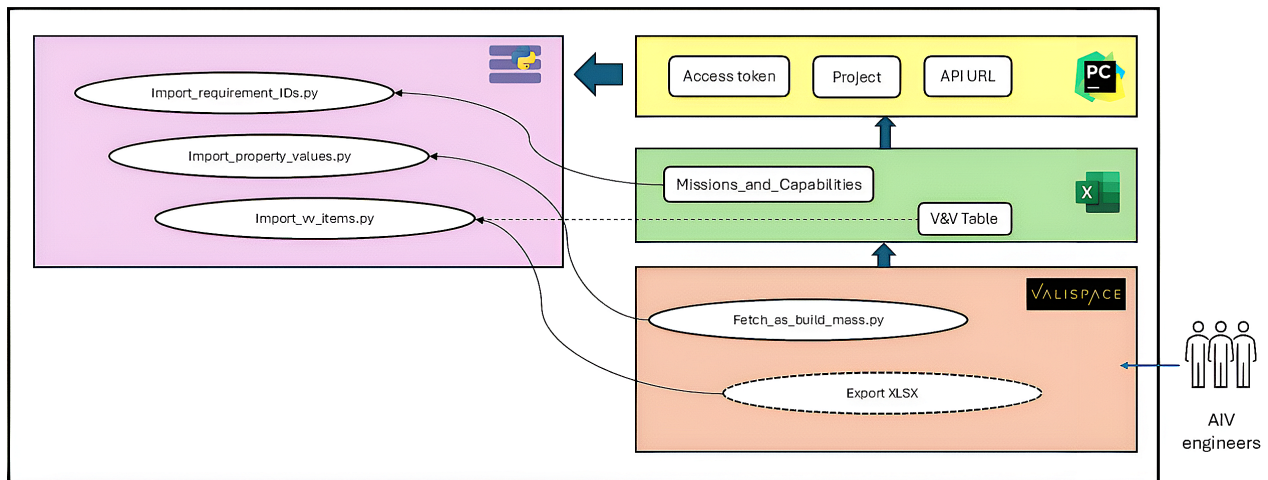


Figure 6.16: Reverse digital thread toolchain architecture

6.2.3. Reverse Thread

Figure 6.17 illustrates the reverse data flow from Valispace back into the Capella model through the Excel intermediate layer. Four categories of data are written back to Capella as PVMT property values, closing the digital thread by ensuring that AIV execution data from Valispace is reflected in the authoritative system model.

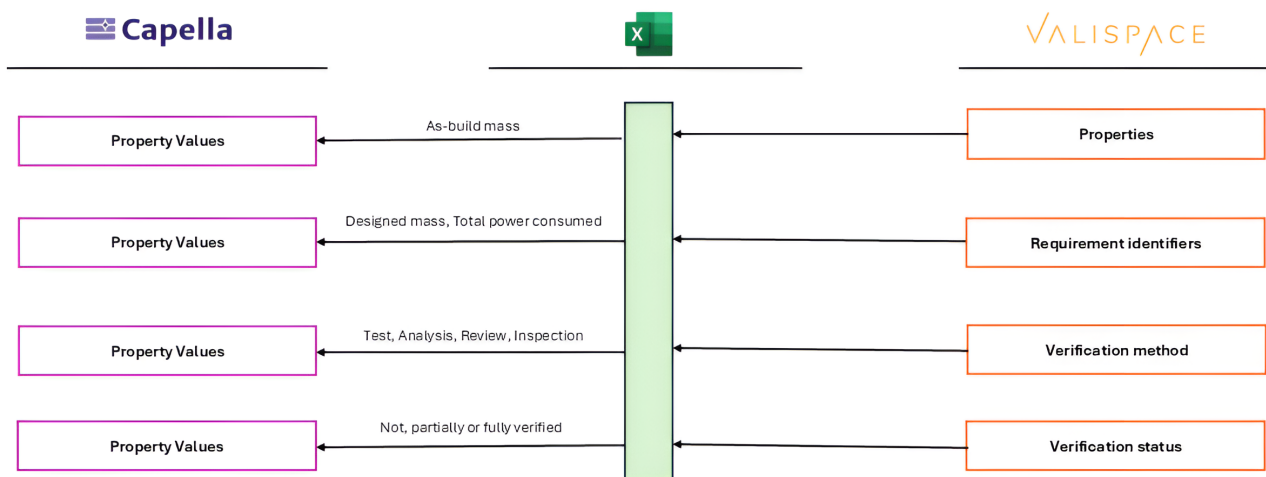


Figure 6.17: Valispace to Capella data mapping via Excel intermediate layer

As-built mass measurements logged in Valispace are written to the corresponding PVMT property on each physical component in Capella, enabling a direct comparison between the designed and measured as-built mass within the model. Requirement identifiers assigned by Valispace during import are written back as string property values, establishing a cross-tool reference between each Capella element

and its Valispace counterpart. Verification method and status values, assigned by the AIV engineer following test execution, are written back as enumeration property values on the corresponding mission and capability elements. Together, these four flows ensure that the Capella model reflects the as-built and as-verified state of the spacecraft at the end of Phase D.

The reverse thread uses the same two-stage pipeline as the forward flow but in the opposite direction. Python scripts on the desktop fetch data from the Valispace REST API and write it to Excel. Python4Capella scripts in Capella then read these files and write the data as property values on the corresponding model elements. Figure 6.18 illustrates this architecture.

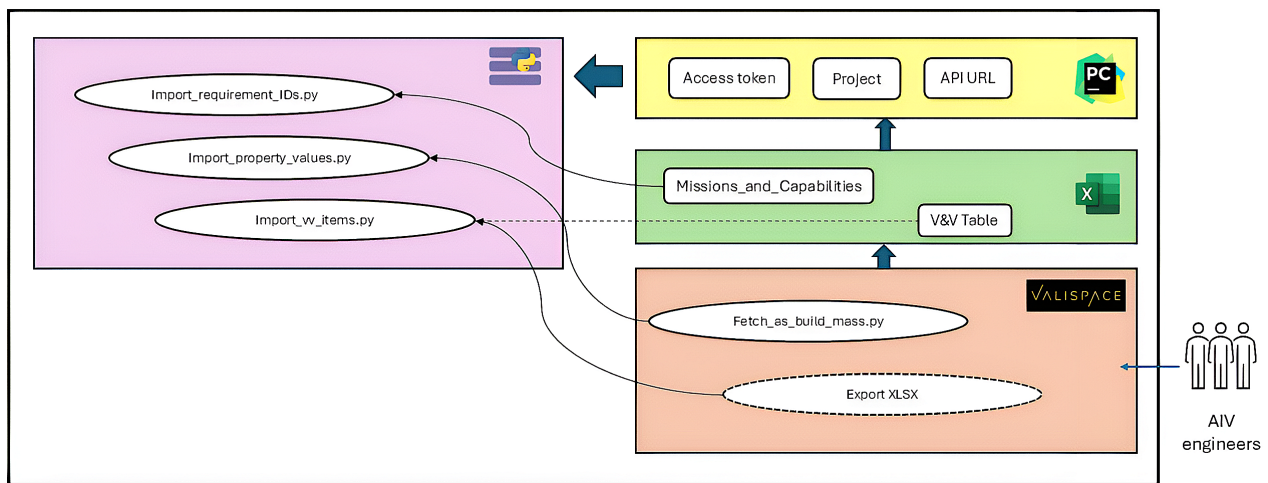


Figure 6.18: Reverse digital thread toolchain architecture

Fetch Scripts

Two Python scripts fetch data from Valispace and write it to Excel, executed on the desktop.

fetch_as_build_mass.py This script queries the Valispace REST API using a GET request to the `/valis/` endpoint, filtering by property name to retrieve `As_build_mass` values recorded by the AIV engineer after physically weighing each component. The returned values are matched to Capella component UUIDs using component names from the PA product tree Excel file as the lookup key. Components whose names differ between Capella and Valispace due to renaming or special character handling are not matched and are skipped.

V&V Table Export Retrieving per-requirement verification method and status assignments requires a manual export from the Valispace user interface. The API endpoints `/requirements/verification-methods/` and `/requirements/verification-statuses/` return global type definition lookup tables rather than per-requirement assignments, so programmatic retrieval is not possible with the current API version. The AIV engineer therefore manually exports the V&V table from the

Requirements module as an Excel file containing the requirement identifier, verification method, and status for each requirement. This limitation is discussed further in Section 6.2.3.

Import Scripts

Three Python4Capella scripts read the updated Excel files and write the fetched data as property values on the corresponding Capella model elements, executed within Capella using the EASE scripting environment.

import__property__values.py This script reads the PA product tree Excel file containing as-built mass values and Capella UUIDs. For each row, it retrieves the model element by UUID, accesses the Mass_budget PVMT group, and sets the As_build_mass property. For elements where the property has not previously been set, it is created before writing. Components not found by UUID are skipped and logged. No unit conversion is performed; values are written as received from Valispace.

import__requirement__ids.py This script reads the missions and capabilities Excel file, which contains the Valispace requirement identifiers written back during the forward import. For each row, it retrieves the corresponding Capella element by UUID and writes the identifier to the System_Requirements or Mission_Requirements PVMT string property. If the Excel file has been overwritten since the forward import, the identifiers may not be available.

import__vv__items.py This script reads the manually exported V&V Excel file from Valispace. For each row, it matches the requirement identifier to the corresponding Capella element by searching for elements whose PVMT identifier property matches, then sets the Verification_method and Verification_status enumeration properties to the appropriate literals. For elements where no enumeration property previously exists, the script creates and configures the property object before setting it.

API Limitation and Workaround

During implementation, a limitation was identified in the Valispace REST API regarding the retrieval of per-requirement verification assignments. The endpoints `/requirements/verification-methods/` and `/requirements/verification-statuses/` return global type definition lookup tables rather than the individual assignments made in the Valispace interface, making automated programmatic retrieval not possible with the current API version.

A practical workaround was implemented in which the AIV engineer manually exports the V&V table from Valispace as an Excel file. The `import_vv_items.py` script reads this file and writes the values to Capella PVMT. While this introduces a manual step into the otherwise automated reverse flow, it fully preserves the traceability of verification close-out data back into the Capella model. This limitation is documented in Chapter 9 and identified as an area for future development.

7

Framework Verification

This chapter demonstrates the integrated Capella-Valispace toolchain through the outputs produced following execution of both the forward and reverse digital thread scripts on the 6UXL CubeSat model. Framework verification here addresses whether the toolchain was implemented correctly and produces the expected engineering artefacts in each target environment, distinct from the qualitative evaluation conducted in Chapter 8.

The demonstration covers screenshots of all Valispace modules populated by the forward thread, including the Blocks module receiving the product tree, properties, and interface connections, the Requirements module receiving mission and system requirements with their hierarchy, and the V&V module receiving test procedures derived from the Capella Exchange Scenarios, as well as the Capella PVMT showing the reverse flow properties written back to the model.

7.1. Blocks Module

Figure 7.1 shows the Valispace Blocks module following execution of `import_blocks.py` and `import_part_number.py`. The physical component hierarchy from Capella is replicated as a parent-child block structure in Valispace, with the `PhysicalSystem` at the root and `BehaviorPC` components organised under their respective subsystem parents. This provides the AIV engineer with a structured view of the spacecraft hardware breakdown, forming the foundation for budget tracking, interface management, and verification planning in Valispace.

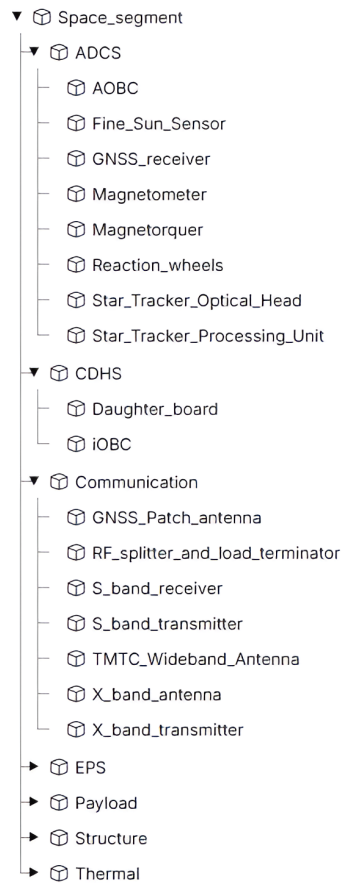


Figure 7.1: Blocks module showing the product tree imported from Capella

Figure 7.2 shows the info view of the AOBC block in Valispace, with the part number field populated by `import_part_number.py` with the deploying NodePC designation, establishing the hardware tracking relationship between the behaviour component and its physical host.

Other details

Part_number	ISIS BOMO + MPCC
State	
Tags	▼

Figure 7.2: Info view of the AOBC block with part number imported from Capella

The designed mass, total maximum power consumed, and CapellaID are imported by `import_properties.py` as shown in Figure 7.3.

Create Property					Info	Properties	Requirements	Modelists	Files	Connections
☐	Actions	Name ↓		Value	Display Unit ⓘ					
☐	...		<u>Total_max_power</u>	0.306	W					
☐	...		<u>Designed_mass</u>	0.107	kg					
☐	...		<u>CapellaID</u>	fcc19e63-2370-46af-a9dc-63ffe3043fe5						
☐	...		<u>As_build_mass</u>	0.107	kg					

Figure 7.3: Properties of the AOBC block in Valispace imported from Capella

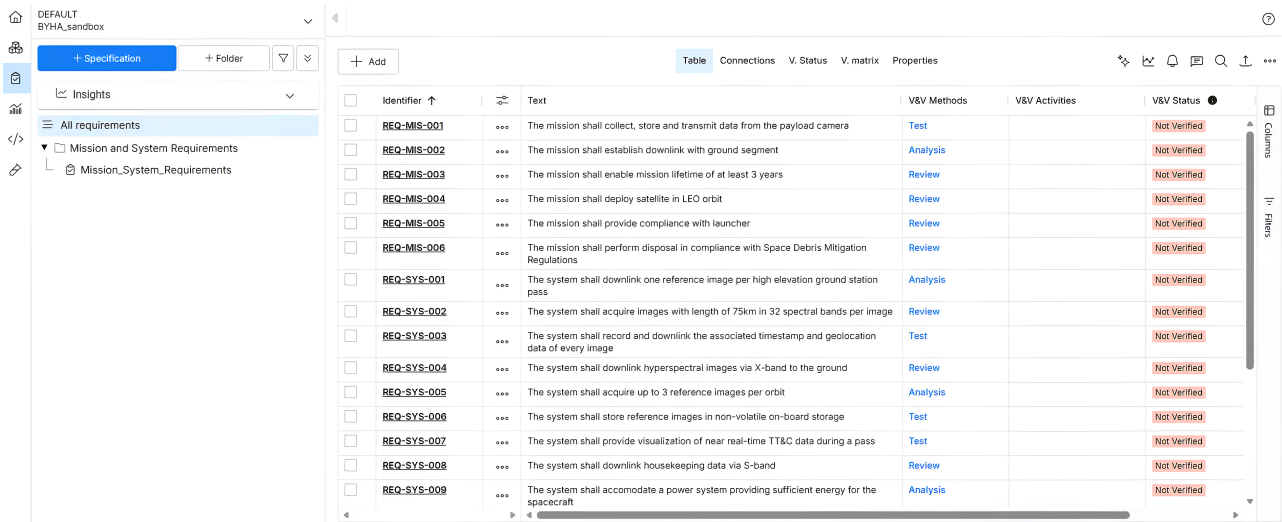
The interface connections of the AOBC to adjacent subsystem blocks, imported by `import_connections.py`, are shown in Figure 7.4. Together, these three data categories on a single block demonstrate the multi-dimensional engineering information consolidated in Valispace from a single Capella model element.

Name ↑	Endpoints
 3V3_AOBC_Magnetometer	AOBC Magnetometer
 3V3_Perm_AOBC_Magnetorquer	AOBC Magnetorquer
 3V3_SW2_AOBC_Star_Tracker_Processing_Unit	AOBC Star_Tracker_Processing_Unit
 3V3_SW2_Main_distribution_unit_AOBC	AOBC Main_distribution_unit
 5V_AOBC_Magnetometer	AOBC Magnetometer
 5V_AOBC_RW_1	AOBC Reaction_wheels
 5V_AOBC_RW_2	AOBC Reaction_wheels
 5V_AOBC_RW_3	AOBC Reaction_wheels
 5V_AOBC_RW_4	AOBC Reaction_wheels
 5V_Perm_AOBC_Star_Tracker_Processing_Unit	AOBC Star_Tracker_Processing_Unit
 5V_SW0_Main_distribution_unit_AOBC	AOBC Main_distribution_unit
 5V_SW3_AOBC_Magnetorquer	AOBC Magnetorquer
 5V_SW3_Main_distribution_unit_AOBC	AOBC Main_distribution_unit
 Analog_FSS_1_AOBC	AOBC Fine_Sun_Sensor
 Analog_FSS_2_AOBC	AOBC Fine_Sun_Sensor

Figure 7.4: The AOBC block and its connections imported from the Capella model

7.2. Requirements Module

The Requirements module is shown in Figure 7.5 following execution of `import_requirements.py`. Mission requirements (prefixed MISS) and system requirements (prefixed SYS) are organised within a folder and specification structure that maintains the mission-to-capability hierarchy from Capella. Each requirement carries a unique identifier assigned during import and written back to the Capella model as a PVMT property, establishing a persistent cross-tool reference. The V&V table within the module presents the requirement identifier, text, assigned verification method, linked V&V activity, and current verification status, providing real-time visibility of verification coverage without requiring a manually compiled report.



Identifier	Text	V&V Methods	V&V Activities	V&V Status
REQ-MIS-001	The mission shall collect, store and transmit data from the payload camera	Test		Not Verified
REQ-MIS-002	The mission shall establish downlink with ground segment	Analysis		Not Verified
REQ-MIS-003	The mission shall enable mission lifetime of at least 3 years	Review		Not Verified
REQ-MIS-004	The mission shall deploy satellite in LEO orbit	Review		Not Verified
REQ-MIS-005	The mission shall provide compliance with launcher	Review		Not Verified
REQ-MIS-006	The mission shall perform disposal in compliance with Space Debris Mitigation Regulations	Review		Not Verified
REQ-SYS-001	The system shall downlink one reference image per high elevation ground station pass	Analysis		Not Verified
REQ-SYS-002	The system shall acquire images with length of 75km in 32 spectral bands per image	Review		Not Verified
REQ-SYS-003	The system shall record and downlink the associated timestamp and geolocation data of every image	Test		Not Verified
REQ-SYS-004	The system shall downlink hyperspectral images via X-band to the ground	Review		Not Verified
REQ-SYS-005	The system shall acquire up to 3 reference images per orbit	Analysis		Not Verified
REQ-SYS-006	The system shall store reference images in non-volatile on-board storage	Test		Not Verified
REQ-SYS-007	The system shall provide visualization of near real-time TT&C data during a pass	Test		Not Verified
REQ-SYS-008	The system shall downlink housekeeping data via S-band	Review		Not Verified
REQ-SYS-009	The system shall accommodate a power system providing sufficient energy for the spacecraft	Analysis		Not Verified

Figure 7.5: Requirements module showing requirement identifiers, text and their V&V items

Figure 7.6 shows an excerpt of the parent-child relationship view in Valispace. System requirements are linked to their parent mission requirements, replicating the hierarchy from the Capella System Analysis layer. Capabilities exploited by multiple missions are correctly linked to all parent mission requirements, confirming that multi-parent relationships modelled in Capella are faithfully preserved in Valispace.

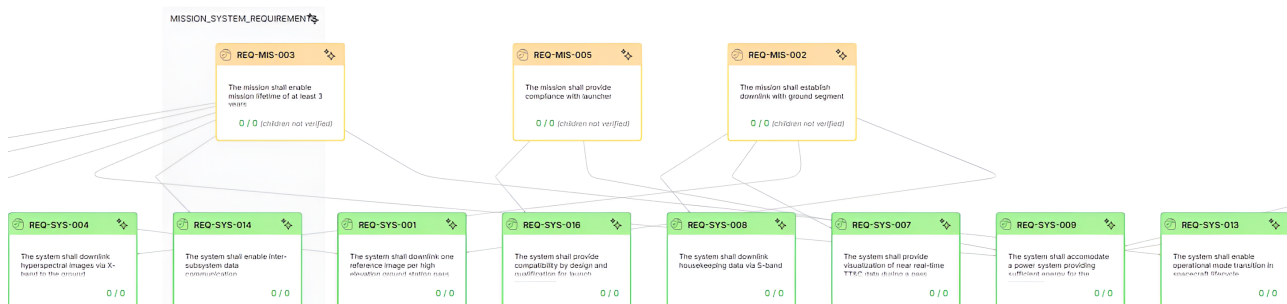


Figure 7.6: Requirements module showing an excerpt of the parent-child relationships

7.3. V&V Module

Figure 7.7 shows the Valispace V&V module following execution of `import_test_procedures.py`. Test activities are organised within folders corresponding to the functional test type identified from the Exchange Scenario name in Capella. This structure provides the AIV engineer with an overview of all planned verification campaigns by test type, without requiring knowledge of the underlying ARCADIA model.

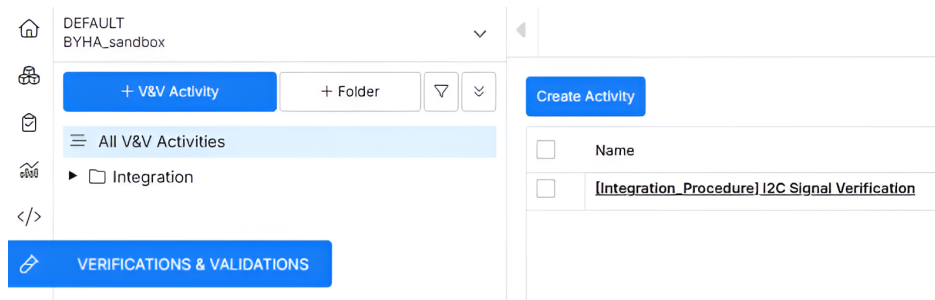


Figure 7.7: Valispace V&V module showing the folder and activity structure

Figure 7.8 shows the procedure table for the I2C signal verification activity in Valispace. Each step presents its step number, name, description, expected results, linked requirements, and attachment fields. Step names are derived from the physical functions allocated to actors in the Capella test architecture, with steps from non-primary actors labelled in square brackets to distinguish responsibilities during execution. The procedure is populated entirely from the Capella test architecture without manual authoring.

Create Step						
Activity Info Requirements Statuses Procedure Runs						
☐	Step Number	Name	Description	Expected Results	Requirements	Attachments
☐	1	Configure the PSU				
☐	2	Ensure the IMEPS-EGSE is set to intended configuration				
☐	3	Ensure the test setup is set to intended configuration				
☐	4	Enable output power on PSU				
☐	5	Switch ON the SAT_ENA switch and check power consumption on PSU				
☐	6	Run the CompySat script				
☐	7	Connect the Oscilloscope to the I2C_MAIN on the CSKB connector of IMTQ				
☐	8	Set the Oscilloscope to the intended configuration				
☐	9	Request telemetry and check the I2C signal on the Oscilloscope				
☐	10	Provide telemetry [S-band receiver]				
☐	11	Check the rising and falling times for SDA and SCL, and the voltage levels				
☐	12	Set the Main distribution unit to enable OBC02 power				
☐	13	Enable OBC02 power [Main distribution unit]				
☐	14	Check the power consumption on PSU				
☐	15	Set the Main distribution unit to enable OBC01 power				
☐	16	Enable OBC01 power [Main distribution unit]				
☐	17	Check the power consumption on PSU				
☐	18	Request telemetry and check the I2C signal on the Oscilloscope				
☐	19	Provide telemetry [S-band transmitter]				

Figure 7.8: Test procedure for the I2C signal verification imported from the Exchange Scenario

7.4. PVMT

The reverse digital thread is demonstrated through the PVMT properties updated on Capella model elements following execution of the reverse flow import scripts. Four categories of data are written back: as-built mass, requirement identifiers, verification method, and verification status. Together these close the digital thread by reflecting the as-executed and as-verified state of the spacecraft within the authoritative system model.

Figure 7.9 shows the as-built mass property written to a component in Capella following execution of `import_as_build_mass.py`. The `As_build_mass` property under the `Mass_budget` domain is populated with the value recorded by the AIV engineer in Valispace after weighing the component, enabling a direct comparison with the `Designed_mass` property without consulting external spreadsheets.

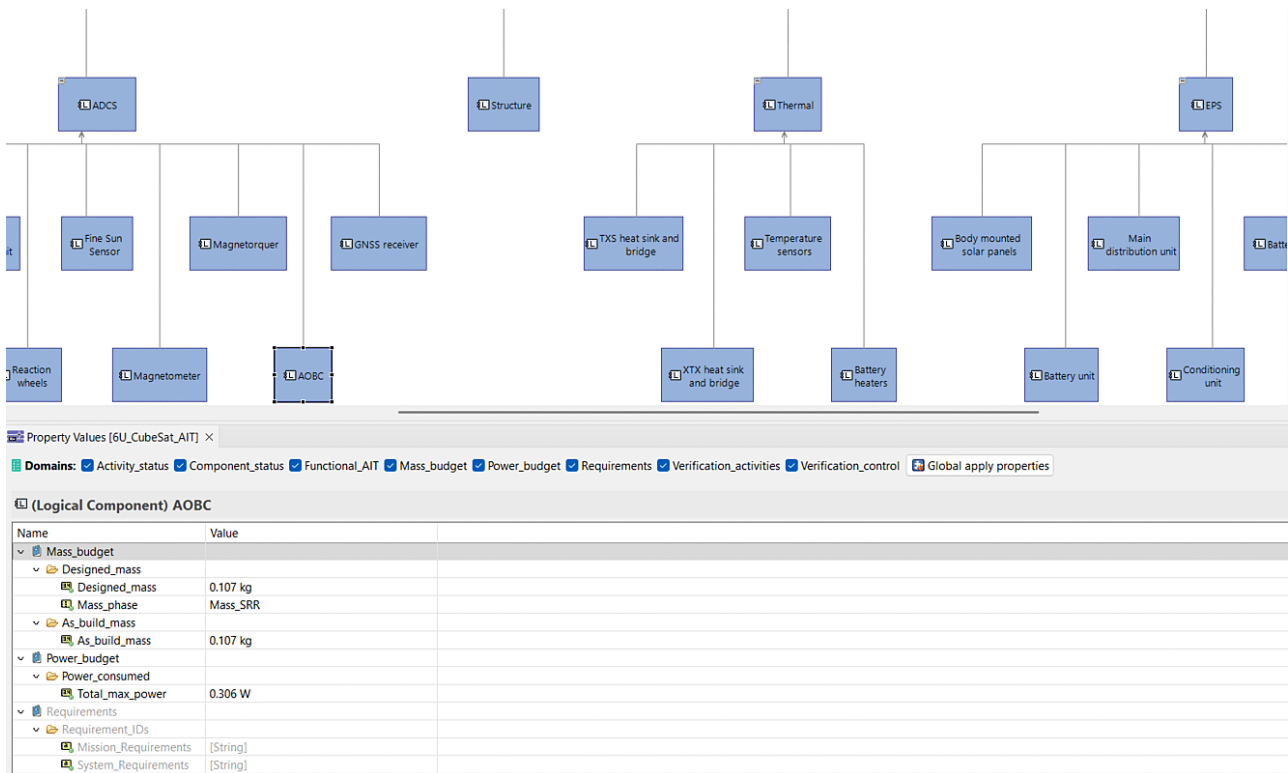


Figure 7.9: `As_build_mass` property on PVMT populated following platform inspection during AIV

Figure 7.10 shows the verification-related properties written to a capability element in Capella following execution of `import_requirement_ids.py` and `import_vv_items.py`. The `System_Requirements` property holds the Valispace requirement identifier assigned during the forward import, establishing a persistent cross-tool reference. The `Verification_method` property records how the requirement was verified, and the `Verification_status` property records the close-out decision from the Valispace V&V module. Together, these three properties encode the complete verification close-out record for each system requirement directly within the Capella model.

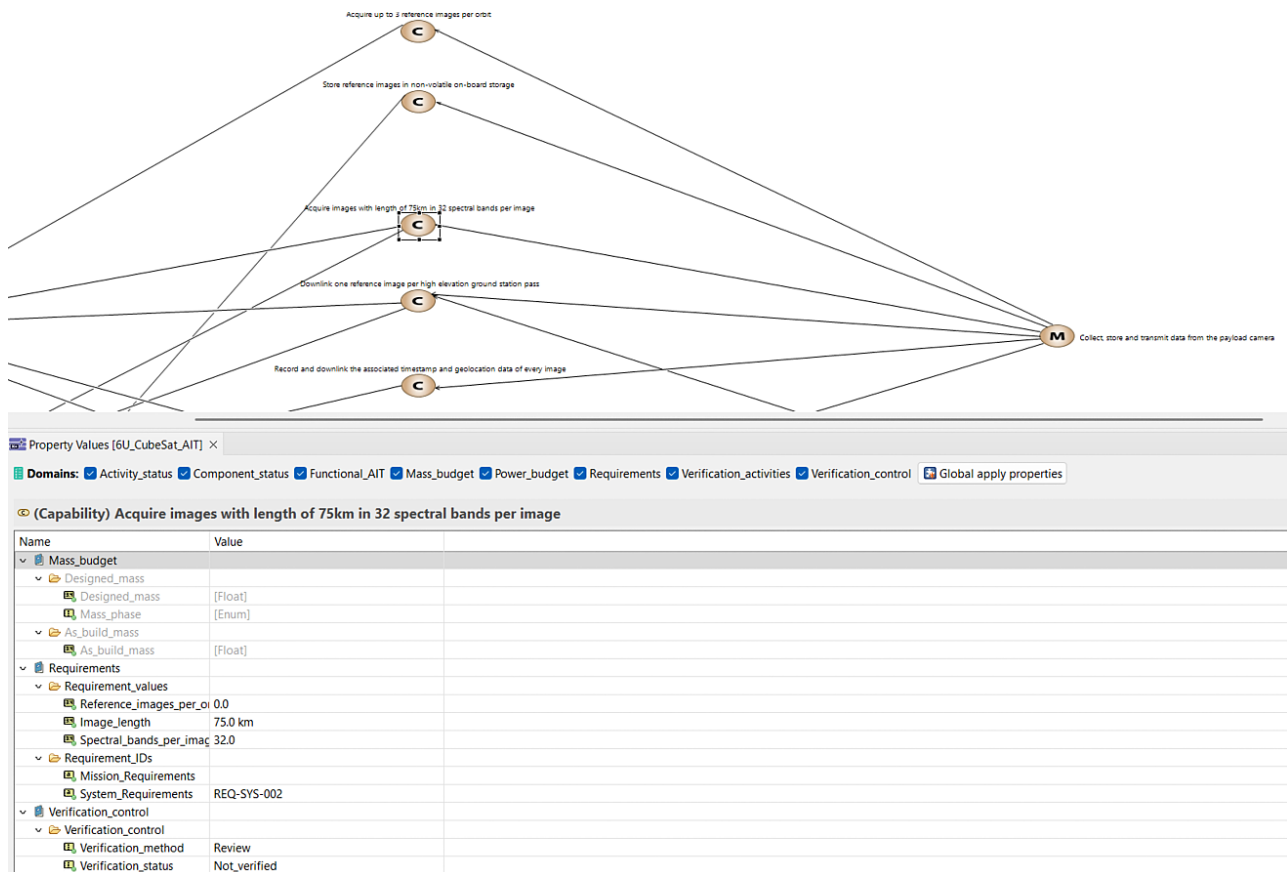
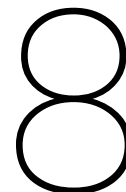


Figure 7.10: Capella PVMT populated with requirement identifier, V&V method and status



Results and Discussion

This chapter presents the qualitative evaluation of the integrated Capella-Valispace framework against the document-based and Capella-only reference approaches. The evaluation methodology is first described, covering the three approaches compared, the 26 success factors across four qualitative metrics, and the three-point scoring scale. The results for each metric are then presented with colour-coded scoring tables and per-factor justifications. The chapter concludes with a discussion of the overall findings, the inter-rater validation, and the interpretation of the weighted Cohen's Kappa score.

8.1. Evaluation Methodology

Three approaches to CubeSat AIV engineering are compared in this evaluation. The document-based approach represents current practice using Word documents, Excel spreadsheets, and shared folders, consistent with ECSS verification obligations fulfilled through document artefacts. The Capella-only approach uses Capella and the ARCADIA methodology with the Requirements Viewpoint, PVMT, and functional test modelling capabilities described in Chapter 5. The integrated Capella-Valispace approach extends the Capella-only approach with the bidirectional digital thread established in Chapter 6. All three approaches are capable of satisfying ECSS verification requirements through different mechanisms.

The evaluation employs a structured qualitative assessment of built-in framework capabilities against defined success factors. Each factor is rated on a three-point Likert scale based on direct observation

of framework features during implementation and demonstration on the 6UXL CubeSat mission:

- **1 - Not supported:** the framework has no built-in mechanism for this factor and relies entirely on manual effort outside the framework.
- **2 - Partially supported:** the framework provides some built-in mechanism but it is incomplete, requires significant manual intervention, or depends on workarounds.
- **3 - Fully supported:** the framework has a dedicated built-in mechanism that addresses this success factor systematically.

The primary scoring criterion is the presence and completeness of built-in features supporting each factor. The degree of automation versus manual effort is considered a secondary factor, reflected in the distinction between scores of 2 and 3 where a partially automated mechanism scores lower than a fully systematic one. A factor is not penalised for requiring some manual input if the framework provides a clear and structured mechanism for it; it is penalised only where that mechanism is absent or dependent on workarounds outside the framework boundary.

Scores reflect the specific implementations developed in this thesis, covering the modelling layers, add-ons, scripts, and data flows described in the earlier chapters. They do not represent a general assessment of Capella or Valispace as tools, nor of MBSE or DDSE as methodologies. Capabilities that fall outside the scope of the implemented framework are not considered. The evaluation covers 26 success factors across four metrics: Traceability, Consistency, Flexibility, and Accessibility and Stakeholder Collaboration. Success factors where the document-based approach retains advantages are explicitly identified to ensure an unbiased assessment.

The evaluation criteria are derived from literature on digital engineering adoption and spacecraft verification practices [46], with individual factors further grounded in ECSS standards and CubeSat AIV practice literature.

8.2. Results

The results of the qualitative evaluation are presented across four metrics. Each evaluation table uses the following notation: **DB** refers to the Document-Based approach, **C** refers to the Capella-only approach, and **C-V** refers to the integrated Capella-Valispace approach. Scores of 1, 2, and 3 denote Not supported, Partially supported, and Fully supported respectively.

8.2.1. Traceability

Traceability is defined as the degree to which engineering artefacts from mission requirements to verification close-out can be linked, navigated, and maintained throughout the AIV campaign. Six success factors are assessed, covering requirement-to-verification linkage, evidence navigation, component allocation, test procedure linkage, lifecycle continuity, and cross-discipline accessibility.

Table 8.1: Evaluation scores for Traceability

ID	Factor	AIV Engineering Need	DB	C	C-V
T1	Requirement-to-verification coverage	Can every requirement be linked to at least one verification activity?	1	2	3
T2	Verification evidence linkage	Can verification results be navigated back to their originating requirement?	1	2	3
T3	Component-to-requirement allocation	Can each hardware component be linked to the requirements it must satisfy?	1	2	3
T4	Test procedure-to-requirement linkage	Is each test procedure explicitly linked to the requirement it verifies?	1	3	3
T5	Lifecycle traceability continuity	Is traceability maintained from Phase C handover through verification close-out without manual re-establishment?	1	2	2
T6	Cross-discipline traceability	Is traceability accessible to all engineering roles without specialist tool expertise?	3	1	1

T1 - Requirement-to-verification coverage. This factor assesses whether a formal mechanism exists to link every requirement to at least one verification activity. In the document-based approach, requirements in the SRD and verification activities in the VCD are kept in separate documents with no enforced linkage, meaning uncovered requirements can only be caught through manual review (score 1). The Capella-only approach allows capabilities to be manually linked to verification activities modelled in PVMT, providing a structural link but requiring manual effort not enforced by the tool (score 2). The integrated approach imports requirements into Valispace and generates V&V activities from ES diagrams, with the V&V table providing a built-in coverage matrix (score 3).

T2 - Verification evidence linkage. This factor assesses the ability to navigate from a verification result back to its originating requirement. In the document-based approach, test reports reference requirement identifiers textually but navigation requires manual searching (score 1). In the Capella-only approach, Exchange Scenarios are linked to capabilities within the model but tracing results back requires ARCADIA familiarity (score 2). In the integrated approach, V&V activities in Valispace are associated with requirements through the platform’s traceability view, enabling direct navigation through the user interface (score 3).

T3 - Component-to-requirement allocation. This factor assesses whether hardware components are formally linked to the requirements they must satisfy. The document-based approach maintains this allocation informally across ICDs and system documents with no enforced structural link (score 1). The Capella-only approach supports allocation through the Requirements Viewpoint but requires additional modelling effort (score 2). The integrated approach imports physical components as blocks in Valispace where they can be linked to requirements in the V&V module, with the reverse thread additionally connecting as-built mass back to the physical components in Capella (score 3).

T4 - Test procedure-to-requirement linkage. This factor assesses whether each test procedure is explicitly linked to the requirement it verifies. The document-based approach references requirements informally within procedure text (score 1). The Capella-only approach formally links Exchange Scenarios to the capabilities they verify, providing structural traceability within the ARCADIA model (score 3). The integrated approach preserves this linkage in Valispace where imported V&V activities are associated with their corresponding requirements (score 3).

T5 - Lifecycle traceability continuity. This factor assesses whether traceability is maintained from Phase C through verification close-out without manual re-establishment. No approach scores 3 because some deliberate human action is required at the Phase C to Phase D transition in every case. The document-based approach requires complete manual re-establishment of traceability at handover (score 1). The Capella-only approach maintains traceability within the model but requires deliberate extraction for AIV use (score 2). The integrated approach improves lifecycle continuity but still requires engineers to trigger script execution, manage verification planning, and account for design changes through to verification close-out (score 2). For a concrete example of what this means in practice, consider that even with the integrated toolchain, the systems engineer must deliberately run the import scripts on the approved model version at handover rather than traceability being maintained automatically. This is an intentional design decision preserving a human validation checkpoint.

T6 - Cross-discipline traceability. This factor assesses whether traceability is accessible to all engineering roles without specialist tool knowledge, and is the only document-based advantage in this metric. Word and Excel artefacts are universally readable without tool installation or methodology training (score 3). The Capella-only approach requires ARCADIA familiarity and Capella installation (score 1). The integrated approach requires a Valispace licence for the AIV side and Capella expertise for the systems engineering side (score 1).

The integrated approach achieves 67% fully supported in Traceability (T1, T2, T3, T4), compared to 17% for the Capella-only approach (T4 only) and 17% for the document-based approach (T6 only). The document-based approach scores not supported on 83% of traceability factors, confirming the near-complete absence of formal traceability in document-centric AIV workflows.

8.2.2. Consistency

Consistency is defined as the degree to which engineering data remains accurate and non-contradictory across the AIV campaign, from Phase C handover through to verification close-out. Six success factors are assessed covering data accuracy, budget integrity, requirement integrity, as-built accuracy, configuration control, and formal document revision control.

Table 8.2: Evaluation scores for Consistency

ID	Factor	AIV Engineering Need	DB	C	C-V
C1	Data accuracy at handover	Is engineering data received by the AIV team accurate and complete at the start of Phase D?	2	2	2
C2	Budget data integrity	Are mass and power values consistent between the design record and AIV tracking?	1	3	3
C3	Requirement integrity	Is requirement text, hierarchy, and identifier consistent throughout the AIV campaign?	2	2	3
C4	As-built data accuracy	Are as-built measurements accurately captured and reflected in the system record?	1	2	3
C5	Configuration control	Is there a clear mechanism to prevent unauthorised or untracked changes to AIV data?	2	1	2
C6	Formal document revision control	Is versioning and revision history of AIV artefacts formally maintained and customer-visible?	3	1	2

C1 - Data accuracy at handover. This factor assesses whether engineering data received by the AIV team at the start of Phase D is accurate and complete. All three approaches score 2 because handover data accuracy depends on engineer discipline and source data quality, not the tool framework. The document-based approach depends on the completeness of the handover package (score 2). The Capella-only approach depends on model completeness at extraction (score 2). The integrated approach automates transfer but cannot validate source data accuracy before import (score 2). This is a universal limitation across all approaches.

C2 - Budget data integrity. This factor assesses whether mass and power values are consistent between the design record and AIV tracking. The document-based approach maintains budgets in separate spreadsheets with no link to the design record, making divergence undetectable without manual checking (score 1). The Capella-only approach enforces typed mass and power properties through PVMT, maintaining consistency within the model (score 3). The integrated approach transfers the same values to Valispace through the forward thread and returns as-built mass through the reverse thread, maintaining consistency across both tools (score 3).

C3 - Requirement integrity. This factor assesses whether requirement text, hierarchy, and identifiers stay consistent throughout the AIV campaign. The document-based approach requires manual propagation of updates across the SRD, VCD, and test procedures, introducing divergence risk at each copy (score 2). The Capella-only approach formally defines requirements in the model but does not connect them to the AIV execution environment (score 2). The integrated approach imports requirements with generated identifiers written back to the Capella model, enforcing consistent text and hierarchy across both tools (score 3).

C4 - As-built data accuracy. This factor assesses whether as-built measurements are accurately captured and reflected in the system record. The document-based approach records as-built data in spreadsheets with no formal link to the design model (score 1). The Capella-only approach supports as-built mass as a PVMT property but requires direct manual entry into Capella during AIV (score 2). In the integrated approach, the AIV engineer logs as-built mass in Valispace and the reverse thread automatically transfers the value to the Capella PVMT, providing a fully built-in mechanism for reflecting as-built measurements in the system record (score 3).

C5 - Configuration control. This factor assesses whether a built-in mechanism prevents unauthorised or untracked changes to AIV data. The document-based approach relies on file naming conventions and folder access controls, providing partial protection without formal approval workflows (score 2). The Capella-only approach has no built-in change approval workflow; changes are uncontrolled once the model file is shared (score 1). The integrated approach benefits from Valispace's built-in change tracking and history, though rollback in the reverse thread is not formally enforced (score 2).

C6 - Formal document revision control. This factor assesses whether versioning and revision history of AIV artefacts are formally maintained and customer-visible, and is the only document-based advantage in this metric. ECSS-compliant Word and Excel templates include revision history headers and signature blocks directly submittable to customer review boards (score 3). The Capella-only approach has no native document generation mechanism for formal sign-off (score 1). The integrated approach has Valispace's internal data history but lacks native ECSS-compliant signature blocks and approval chains (score 2).

Consistency results reveal a nuanced picture. The integrated approach is the only one to eliminate not-supported scores, achieving 50% fully supported (C2, C3, C4) and 50% partially supported. The document-based and Capella-only approaches each achieve 17% fully supported but for opposite reasons, the document-based approach scores 3 on C6 while the Capella-only approach scores 3 on C2. C1 scores 2 universally, reflecting a process-dependent limitation no tool framework can fully address.

8.2.3. Flexibility

Flexibility is defined as the degree to which the framework accommodates changes to requirements, architecture, or test procedures during the AIV campaign without disproportionate rework. Seven success factors are assessed, including two where the document-based approach has advantages over both model-based approaches.

Table 8.3: Evaluation scores for Flexibility

ID	Factor	AIV Engineering Need	DB	C	C-V
F1	Requirement change accommodation	When a requirement changes, how much rework is required to update the AIV record?	1	1	2
F2	Scope change accommodation	Can component additions or removals be accommodated without rebuilding the AIV structure?	1	2	2
F3	Test procedure modification	Can test procedures be updated or extended without rebuilding the documentation structure?	2	2	2
F4	Re-use across missions	Is the framework applicable to a different CubeSat mission without significant restructuring?	1	2	2
F5	Iterative verification support	Is re-verification after a design change supported without loss of previous evidence?	2	2	3
F6	Informal change management	Can ad-hoc deviations and non-conformances during test execution be accommodated without tool intervention?	3	1	1
F7	Infrastructure independence	Is the framework operational without server access, API connectivity, or software licences?	3	2	1

F1 - Requirement change accommodation. This factor assesses the rework burden when a requirement changes. All three approaches require significant manual effort when a requirement changes. The document-based approach requires manual propagation across the SRD, VCD, test procedures, and AIV plan (score 1). The Capella-only approach requires manual identification and update of all linked Exchange Scenarios (score 1). The integrated approach reduces rework through script re-execution, but rollback of the previous import and deliberate re-import are still required (score 2).

F2 - Scope change accommodation. This factor assesses whether component additions or removals can be reflected in the AIV structure without rebuilding from scratch. The document-based approach requires manual updates across parts lists, ICDs, and verification matrices (score 1). The Capella-

only approach requires architectural updates in the model with downstream AIV documents updated separately (score 2). The integrated approach enables propagation of structural changes through the forward thread, but similar limitations apply due to architectural modifications and propagation into test procedures (score 2).

F3 - Test procedure modification. All three approaches score 2. Document-based procedures in Word are easily modified but consistency with requirements must be manually verified. Exchange Scenarios in Capella cannot be freely modified without rebuilding the test architecture, meaning modified scenarios must be re-exported and re-imported. Step definitions in Valispace can be directly edited by the AIV engineer without model changes, providing a supplementary path but at the cost of consistency between both tools.

F4 - Re-use across missions. The document-based approach requires manual population of AIV artefacts for each new mission (score 1). The Capella-only approach benefits from mission-independent ARCADIA structures and PVMT definitions (score 2). The integrated approach's Python scripts operate on any Capella model with the same PVMT structure, requiring only configuration parameter updates for a different mission (score 2). Re-use is assessed at the framework level; the Capella model itself must be rebuilt for a new mission.

F5 - Iterative verification support. The document-based approach retains previous reports but re-verification requires manual VCD updates (score 2). The Capella-only approach retains PVMT values but re-verification requires updating the model or modifying ES diagrams (score 2). The integrated approach supports iterative verification through Valispace's as-run procedure history, recording each execution separately without overwriting previous evidence, with updated close-out status written back to Capella (score 3). This is the only fully supported factor for the integrated approach in this metric.

F6 - Informal change management. This is the primary document-based advantage in this metric. An AIV engineer can annotate or amend a test procedure document immediately during test execution without any tool dependency (score 3). Changes to the Capella-only approach require modifying Exchange Scenarios in the model, which is not accessible on the test bench (score 1). The integrated approach requires Valispace access, which is not suitable for fast-paced test bench operations (score 1).

F7 – Infrastructure independence. This is the second document-based advantage. Word and Excel operate fully offline (score 3). Capella requires a Java/Eclipse environment but no server dependency (score 2). The integrated approach requires Valispace server access, API connectivity, and a valid access token, which is particularly limiting in test facility environments where network

connectivity cannot be guaranteed (score 1).

Flexibility results present the most balanced distribution across the three approaches. The document-based approach achieves 29% fully supported (F6, F7) through certain operational advantages. The Capella-only approach achieves 0% fully supported but 71% partially supported. The integrated approach achieves 14% fully supported (F5 only) and 57% partially supported, with F6 and F7 representing an inherent trade-off between structured digital engineering and operational flexibility during test execution.

8.2.4. Accessibility and Stakeholder Collaboration

Accessibility and Stakeholder Collaboration is defined as the degree to which the framework is accessible to different engineering roles without specialist expertise, and supports concurrent working between systems and AIV engineers. Seven success factors are assessed, including three where the document-based approach shows significant promise.

Table 8.4: Evaluation scores for Accessibility and Stakeholder Collaboration

ID	Factor	AIV Engineering Need	DB	C	C-V
A1	Role-appropriate access	Can AIV engineers access and contribute without requiring systems engineering tool expertise?	3	1	1
A2	Verification status visibility	Can stakeholders view current verification status without a manually compiled report?	1	1	3
A3	Concurrent working	Can SE and AIV engineers work simultaneously without blocking each other?	1	1	3
A4	Handover clarity	Is the Phase C to Phase D handover formally defined, complete, and unambiguous?	2	1	2
A5	New engineer onboarding	Can a new AIV engineer understand the campaign structure and begin contributing quickly?	2	1	2
A6	Formal sign-off and audit readiness	Do AIV artefacts support formal review board approval, signature blocks, and customer submission?	3	1	2
A7	Stakeholder access without tool licence	Can customers, authorities, or auditors access AIV records without installing specialist tools?	3	1	1

A1 - Role-appropriate access. This factor assesses whether AIV engineers can access and contribute without systems engineering tool expertise. Word and Excel require no specialist training (score 3). The Capella-only approach requires ARCADIA familiarity and tool installation for any AIV access (score 1). The integrated approach provides AIV engineers with a Valispace environment for test execution, result logging, and status tracking without Capella expertise. However, managing imported data and maintaining the toolchain configuration still requires systems engineering involvement, distinguishing between contributing to the campaign and understanding the full toolchain

context (score 1).

A2 - Verification status visibility. The document-based approach requires a manually updated VCD snapshot, typically produced periodically rather than in real time (score 1). The Capella-only approach stores status as PVMT properties on individual elements without a consolidated dashboard (score 1). The integrated approach provides real-time Valispace dashboards accessible to any stakeholder with an account (score 3).

A3 - Concurrent working. In document-based workflows, systems and AIV engineers share files on a common folder. When the systems engineer updates the SRD or VCD, the AIV engineer risks version conflicts when working on related artefacts simultaneously (score 1). Capella without the Teams for Capella add-on is a single-user application, preventing simultaneous access (score 1). The integrated approach separates roles naturally: the systems engineer works in Capella while the AIV engineer works concurrently in Valispace, with minimal conflicts between the two environments (score 3).

A4 - Handover clarity. The document-based approach provides a readable handover package, but version ambiguity and implicit knowledge dependencies can cause issues during the AIV campaign (score 2). The Capella-only approach has no formal handover mechanism; the model is shared informally (score 1). The integrated approach defines what data flows at handover through the import scripts, but does not formally enforce that scripts were run on the approved model version or that the AIV team accepted the Valispace content before beginning the campaign (score 2).

A5 - New engineer onboarding. A new engineer can read a Word-based AIV plan without tool training, though understanding the document structure still requires a familiarisation period (score 2). The Capella-only approach does not capture the campaign structure effectively and requires ARCADIA familiarity (score 1). The integrated approach distributes campaign structure across Capella and Valispace, requiring a new AIV engineer to navigate Valispace and understand the relationship between tools without a dedicated onboarding guide (score 2). This highlights an onboarding overhead that grows with toolchain complexity.

A6 – Formal sign-off and audit readiness. ECSS-compliant Word documents with revision history and signature blocks satisfy formal submission requirements directly (score 3). The Capella-only approach has no native document generation for formal sign-off (score 1). The integrated approach allows export of V&V tables and requirement matrices from Valispace as Excel or PDF, but native ECSS-compliant signature blocks and formal approval chains are absent (score 2).

A7 – Stakeholder access without tool licence. PDF and Word documents are universally accessible without tool installation (score 3). Accessing Capella data requires tool installation; accessing

Valispace data requires a licence and account (score 1 for both model-based approaches).

Accessibility results show the most dramatic divergence across approaches. The document-based approach achieves 43% fully supported (A1, A6, A7) through familiar tools requiring no specialist training. The Capella-only approach achieves 0% fully supported and 100% not supported, as all data resides within a specialist tool inaccessible without Capella installation and ARCADIA training. The integrated approach achieves 29% fully supported (A2, A3) through real-time dashboards and role separation. The A5 result warrants particular attention: all three approaches score either 1 or 2, but for different reasons. A new AIV engineer can navigate a Word-based AIV plan with less guidance than a Valispace environment whose data originates from an ARCADIA model. This regression in onboarding clarity is a real trade-off of the integrated approach.

8.3. Discussion

The qualitative evaluation assessed the three AIV workflow approaches across 26 success factors organised into four metrics. The overall score distribution for each approach is presented in Table 8.5.

Table 8.5: Overall score distribution across all 26 success factors

Score	DB	C	C-V
3 – Fully supported	7 (27%)	2 (8%)	10 (38%)
2 – Partially supported	7 (27%)	12 (46%)	11 (42%)
1 – Not supported	12 (46%)	12 (46%)	5 (19%)

The integrated approach achieves the highest proportion of fully supported factors at 38%, compared to 27% for the document-based approach and 8% for the Capella-only approach. The reduction in not-supported factors to 19% for the integrated approach confirms that it provides built-in mechanisms for a greater number of AIV engineering needs. The Capella-only approach achieves the lowest proportion of fully supported factors at 8%, limited to budget data integrity (C2) and test procedure-to-requirement linkage (T4), consistent with the literature showing that MBSE tools offer limited built-in AIV support beyond the modelling boundary without external integration.

The document-based approach achieves 27% fully supported across seven factors where document accessibility and infrastructure independence are practical advantages over both model-based approaches: cross-discipline traceability (T6), formal revision control (C6), informal change management (F6), infrastructure independence (F7), role-appropriate access (A1), formal sign-off readiness (A6), and stakeholder access without tool licences (A7). These results confirm that the document-based approach retains real operational advantages in areas where tool dependency and MBSE expertise

become barriers.

The integrated approach shows the most pronounced improvements in Traceability and Accessibility. In Traceability, the absence of a cross-tool digital thread in both reference approaches limits formal requirement-to-verification linkage, whereas the integrated approach connects model artefacts directly to the AIV execution environment. In Accessibility, the Capella-only approach scores 0% fully supported and 100% not supported, as all data resides within a specialist tool inaccessible to AIV engineers without MBSE expertise. The integrated approach addresses this through a dedicated Valispace environment.

Flexibility presents the most balanced distribution. The F6 and F7 advantages of the document-based approach highlight an inherent trade-off of any integrated digital engineering environment: the automation that improves traceability and consistency introduces tool and connectivity dependencies that the document-based approach does not carry. This is acknowledged as a fundamental characteristic of integrated toolchains rather than a limitation specific to this implementation.

8.3.1. Inter-Rater Validation and Cohen's Kappa

To partially address the limitation of self-assessed scoring, an independent structured expert evaluation was conducted by a systems engineer at ISISPACE with direct experience of document-based and model-based AIV workflows. All 26 success factors were rated independently using the same three-point scale and factor definitions. Weighted Cohen's Kappa was selected over the standard formulation because the three-point scale is ordinal and a one-point difference is less significant than a two-point difference. A linear weight matrix was applied as shown in Table 8.6.

Table 8.6: Linear weight matrix applied in the weighted Cohen's Kappa calculation

	Score 1	Score 2	Score 3
Score 1	1.0	0.5	0.0
Score 2	0.5	1.0	0.5
Score 3	0.0	0.5	1.0

The agreement across all 78 rating units (26 factors across 3 approaches) is summarised in Table 8.7.

Table 8.7: Agreement summary between author and independent expert across 78 rating units

Agreement level	Count	Percentage
Exact agreement	59	75.6%
Near agreement	19	24.4%
Large disagreement	0	0.0%
Total	78	100%

The weighted observed agreement ($p_{o,w}$) was computed by applying the weight matrix to the confusion matrix of scores, yielding $p_{o,w} = 0.878$. The weighted expected agreement by chance ($p_{e,w}$) was computed using the marginal score distributions of the two raters, yielding $p_{e,w} = 0.577$. The weighted Kappa was calculated as:

$$\kappa_w = \frac{p_{o,w} - p_{e,w}}{1 - p_{e,w}} = \frac{0.878 - 0.577}{1 - 0.577} = \frac{0.301}{0.423} = 0.712 \quad (8.1)$$

The per-metric weighted Kappa values are presented in Table 8.8.

Table 8.8: Weighted Cohen's Kappa per metric

Metric	Weighted Kappa (k_w)	Interpretation	Exact agreement
Traceability	0.700	Substantial	13/18 (72%)
Consistency	0.727	Substantial	14/18 (78%)
Flexibility	0.669	Substantial	16/21 (76%)
Accessibility and Stakeholder Collaboration	0.714	Substantial	16/21 (76%)
Overall	0.712	Substantial	59/78 (76%)

8.3.2. Interpretation of the Kappa Score

A weighted Kappa of $k_w = 0.712$ falls within the Substantial agreement range of 0.61 to 0.80 as defined by Landis and Koch [64], indicating that the two raters independently arrived at broadly consistent assessments across all 26 success factors and three approaches. Exact agreement was achieved on 75.6% of rating units, with the remaining 24.4% differing by one point. No rating unit differed by two points.

The Kappa values are consistent across all four metrics, ranging from 0.669 to 0.727, indicating that the evaluation framework performs reliably across different types of AIV engineering criteria. All four metrics independently demonstrate substantial inter-rater agreement, confirming that no single metric is driving the overall result. The remaining one-point disagreements reflect an expected interpretive difference between the two raters. The author evaluates built-in capability based on the toolchain design, while the domain expert evaluates based on practical operational experience. This type of variation is normal in expert-based qualitative assessments. These disagreements are acknowledged in the limitations section and discussed in the relevant factor justifications above.

A k_w of 0.712 confirms that the evaluation scores are reproducible. The success factor definitions and scoring anchors were clear enough for an independent expert to apply them consistently, and the relative performance of the three approaches is unlikely to change under an independent assessment. This partially compensates for the limitation of the evaluation being conducted by a single assessor within a single institution.

Framework Validation

This chapter presents the outcomes of the research by addressing the three research questions established in Chapter 1. It identifies the contributions of this work in MBSE-based AIV engineering for small satellite missions, discusses its limitations, and suggests directions for future research.

9.1. Research Findings

This section addresses the three research questions using evidence from the system modelling, toolchain implementation, and qualitative evaluation conducted in the preceding chapters. Each question is answered with reference to specific implementation outcomes and evaluation results.

9.1.1. Research Question 1

How can a DDSE tool be integrated with an MBSE-based solution to support AIV/V engineering processes in small satellite missions?

The integration is achieved through a three-layer pipeline that separates data extraction from the MBSE model, staging and validation of that data, and import into the DDSE environment. Python4Capella scripts form the extraction layer, reading architectural elements, property values, requirement definitions, and Exchange Scenario sequences from the Capella model. Structured Excel files form the intermediate staging layer, enabling inspection and validation before data is committed to Valispace. Python REST API scripts form the import layer, posting the staged data as structured engineering artefacts in Valispace.

The prerequisite for this integration is a sufficiently complete and structured MBSE model. The Capella model was developed across all four ARCADIA layers, with the Requirements Viewpoint formally linking mission and system requirements to their corresponding model elements, and PVMT attaching typed budget and verification properties to physical components and capability elements. The implementation shows that integration is most effective when operating on formally structured and typed data, such as PVMT property values, Exchange Scenario sequences, and requirement hierarchies, as these provide consistent and predictable extraction targets across model revisions.

The integration supports five systems and AIV engineering processes previously managed outside the MBSE model. Requirement flowdown is supported by importing missions and capabilities as mission and system requirements in Valispace with persistent cross-tool identifiers. Product configuration is supported by importing the physical component breakdown as a block hierarchy in Valispace, with part numbers linking each BehaviorPC to its deploying NodePC. Interface management is supported by importing component exchanges as connections between subsystem blocks. Verification planning is supported by importing Exchange Scenarios as ordered V&V activities and step definitions, enabling the AIV engineer to execute procedures derived directly from the MBSE model. Verification close-out is partially supported through the reverse thread, where as-built mass measurements and verification close-out data logged in Valispace are written back to the Capella model as PVMT property values.

One technical limitation was identified: the Valispace REST API does not expose per-requirement verification method and status assignments programmatically, requiring a manual export step before the reverse import script can execute. This introduces a single manual step into an otherwise automated pipeline.

In summary, the integration is technically feasible using existing scripting interfaces and commercial APIs without proprietary software development, and is reusable across different CubeSat missions through configuration-only parameter changes.

9.1.2. Research Question 2

To what extent does the integration of MBSE and DDSE environments influence lifecycle continuity between Phase C and Phase D?

Lifecycle continuity is understood as the degree to which engineering data defined during Phase C is preserved, accessible, and usable during Phase D AIV execution without manual re-establishment of traceability or consistency. The integration addresses this continuity gap across four dimensions: traceability, data consistency, flexibility, and collaborative access.

In terms of **traceability**, the integrated approach achieves 9 out of 18 rating units as fully supported

in the Traceability metric, compared to 3 for the document-based approach and 2 for the Capella-only approach. The most significant improvements are in T1, T3, and T4, all of which score 1 in the document-based approach and 3 in the integrated approach. Requirements defined in Capella are imported into Valispace with persistent identifiers, test procedures are derived from the MBSE test architecture and linked to the requirements they verify, and physical components are represented as blocks in Valispace. The traceability chain from mission objective through system requirement to verification close-out is formally maintained across the Phase C to Phase D boundary without manual re-establishment.

In terms of **data consistency**, the integration eliminates manual transcription of engineering data at the handover boundary. Budget data integrity (C2) and requirement integrity (C3) show the most pronounced improvements in the Consistency metric. The reverse thread further maintains consistency by reflecting as-built measurements and verification close-out data back into the Capella model at the end of Phase D.

In terms of **collaborative access**, the role separation enabled by the toolchain addresses the single-user constraint of the Capella-only approach and the version conflict risks of the document-based approach. The systems engineer works in Capella during Phase C while the AIV engineer works simultaneously in Valispace during Phase D. Concurrent working (A3) improves from not supported in both reference approaches to fully supported in the integrated approach.

The **Flexibility** metric also supports lifecycle continuity indirectly. When design changes occur during the Phase C to Phase D overlap, the toolchain can accommodate updates to the product tree, budgets, and verification status without rebuilding the AIV campaign structure. The script-based update mechanism allows the two roles to maintain a consistent engineering baseline in their respective tools without blocking each other, reducing coordination overhead at the handover boundary.

Two lifecycle continuity gaps remain. T5 scores 2 across all three approaches because deliberate script execution is required at handover rather than automatic synchronisation, which is an intentional design decision that preserves a human validation checkpoint before data is committed to the AIV execution environment. Satellite operational modes are also not transferred to Valispace, requiring the AIV team to consult the Capella model separately when planning mode-dependent verification activities.

In summary, the integration significantly improves lifecycle continuity across traceability, data consistency, and concurrent working. The most substantial contribution is the elimination of manual re-transcription at the Phase C to Phase D boundary, reducing information discontinuity from a near-complete break in the document-based approach to a structured and largely automated handover.

More broadly, the toolchain enforces a way of working that promotes lifecycle continuity by design. Engineering data has a defined origin, a structured transfer mechanism, and a traceable destination in each direction of the digital thread. While a disciplined human process is still required around the toolchain to handle exceptions and design changes, the framework establishes the structural foundation needed to support lifecycle continuity in small satellite AIV engineering.

9.1.3. Research Question 3

How effective is the MBSE-based integrated framework for a 6U XL CubeSat?

The effectiveness of the framework is assessed through both the functional coverage demonstrated during implementation and the qualitative evaluation results.

Functionally, the framework successfully populated all major Valispace modules from the Capella model without manual data entry. The Blocks module received the full physical product tree with part numbers, budget properties, and interface connections. The Requirements module received the complete mission and system requirement hierarchy with cross-tool identifiers and parent-child linkage. The V&V module received functional test procedures from the Exchange Scenarios as ordered step definitions ready for execution. The PVMT demonstration confirmed that as-built mass measurements and verification close-out data were correctly written back to the Capella model, closing the bidirectional digital thread.

Beyond the direct outputs of the forward thread, the framework establishes a foundation in Valispace from which the requirements hierarchy can be extended further. Once mission and system requirements are imported, the systems engineer can continue the flowdown in Valispace by adding subsystem and equipment-level requirements as children of the imported system requirements. This extended hierarchy can inform further modelling in Capella, where subsystem requirements can be translated into functional chains or extended capabilities. The corresponding test architectures and procedures can then be modelled in Capella and imported into Valispace through the same scripts, extending the digital thread to lower levels of the requirement hierarchy. This demonstrates that the framework is not a one-time transfer but a repeatable process that evolves alongside the mission.

From the evaluation, the integrated approach achieves 38% fully supported and 19% not supported, compared to 27% and 46% for the document-based approach, and 8% and 46% for the Capella-only approach. The improvements are most pronounced in the areas that most challenge CubeSat AIV practice: fragmented tool environments, manual traceability maintenance, limited concurrent working, and lack of real-time verification status visibility.

The framework's effectiveness is bounded by several factors. The scope covers mission and system

requirements only; lower-level requirements must be entered manually in Valispace unless the iterative workflow described above is followed. The document-based approach retains advantages in informal change management (F6) and infrastructure independence (F7), reflecting genuine trade-offs of any integrated digital engineering environment. The new engineer onboarding factor (A5) shows a regression compared to both reference approaches, as distributing the campaign structure across two tools reduces intuitiveness for engineers new to the framework. These trade-offs are genuine characteristics of integrated digital engineering environments rather than remediable deficiencies.

The inter-rater validation yielded a weighted Cohen's Kappa $k_w = 0.712$, indicating substantial agreement between the author's scores and those of an independent senior systems engineer at ISISPACE, supporting the reproducibility of the evaluation findings.

In summary, the MBSE-based integrated framework is effective for a 6UXL CubeSat AIV campaign at the mission and system requirement level, demonstrating measurable improvements in traceability, consistency, and stakeholder collaboration over both reference approaches. It is practically implementable using existing tools and APIs, reusable across missions, and grounded in a real industrial mission context.

9.2. Contributions

This research makes three primary contributions to the MBSE and AIV engineering literature for small satellite missions.

The first and most significant contribution is the demonstration of a working bidirectional digital thread between an MBSE tool and a DDSE tool on an industrial CubeSat mission. Existing work [65] establishes MBSE AIV methodologies within a single tool environment but does not address integration with an external AIV execution environment. This framework shows that such integration is technically feasible using existing scripting interfaces and commercial APIs without proprietary tool development, and is reusable across missions through configuration-only parameter changes.

The second contribution is the automated derivation of executable test procedures from Physical Architecture Exchange Scenarios. This directly addresses the redundant AIV verification flow identified in Chapter 2, where test procedures are typically authored manually by AIV engineers re-interpreting design information. By deriving test steps from physical functions allocated to actors in the Capella test architecture, the framework removes a significant source of manual effort and traceability discontinuity between the design record and the AIV execution environment.

The third contribution is the evaluation framework itself: a three-approach, 26-factor qualitative assessment that includes success factors where the document-based approach holds genuine advantages

over both model-based approaches. This ensures the evaluation provides an honest comparative analysis and identifies the trade-offs that practitioners should consider before adopting an integrated MBSE-DDSE toolchain.

The three hypotheses are assessed as follows.

- **H-1** is confirmed: the framework establishes a single source of truth across the AIV lifecycle by ensuring all engineering artefacts originate from the Capella model and are consistently represented in Valispace through the automated digital thread. The Excel intermediate layer reinforces this by providing an inspectable staging layer that maintains data integrity between tools rather than introducing a parallel source of information. The manual export step required for the V&V status reverse flow does not contradict this principle, as the data originates from and is ultimately written back to the authoritative Capella model; the manual step affects only the mechanism of transfer, not the ownership of the data.
- **H-2** is confirmed: the toolchain improves traceability through formal requirement-to-verification linkage across tools, consistency through automated data transfer that eliminates manual transcription, and collaboration through role separation that enables systems and AIV engineers to work concurrently. Flexibility is supported through scope change accommodation, iterative verification support, and cross-mission reusability. Valispace further supports flexibility through its as-run procedure capability, which preserves previous verification evidence when re-verification is needed, and through the ability for AIV engineers to modify test procedures directly in Valispace without changing the Capella model. The document-based approach retains advantages in informal change management (F6) and infrastructure independence (F7), which reflect trade-offs inherent to any integrated digital engineering environment.
- **H-3** is confirmed: extending MBSE utilisation into AIV phases positively influences project performance across both planning and execution. The toolchain improves traceability, consistency, and stakeholder collaboration during AIV preparation, while structured V&V activities, ordered step definitions, and real-time dashboards in Valispace directly support the AIV engineer during test execution. The document-based approach retains specific operational advantages in informal change management and infrastructure independence, which are inherent trade-offs of integrated digital engineering rather than a limitation of the framework's effectiveness.

9.3. Limitations

The following limitations are acknowledged to provide an honest assessment of the scope and boundaries of the framework and its evaluation.

9.3.1. System Modelling

The system model covers mission and system-level requirements only and does not extend to subsystem or equipment-level requirements. This restricts the scope of the digital thread to the upper levels of the requirement hierarchy, with lower-level requirements requiring manual entry in Valispace.

Satellite operational modes and configurations are modelled in Capella but are not transferred to Valispace as part of the forward thread. The AIV engineer must therefore consult the Capella model separately when planning mode-dependent verification activities.

The Physical Architecture layer in this framework is used primarily for AIV engineering activities rather than for its intended design purpose within the ARCADIA methodology. Physical functions, which conventionally represent the physical realisation of logical functions, are repurposed here to represent test steps and activities. As a result, the standard PA design scope covering detailed hardware specifications, physical behaviour modelling, and design trade-offs is not captured in the current model. This is an intentional scope decision focused on AIV activities, but practitioners seeking to extend the framework into a full lifecycle design model should account for this deviation.

The functional test architecture and procedures modelled in the PA layer are limited to functional integration and testing. Environmental qualification tests, software validation campaigns, and non-functional verification activities are outside the scope of the current modelling approach.

9.3.2. Toolchain

The Valispace REST API does not expose per-requirement verification method and status assignments programmatically. The reverse flow for verification close-out data therefore requires a manual export step from the Valispace interface before the import script can execute. This is a known API constraint and is identified as a target for future automation.

Harnesses are exported to Excel for reference but are not imported into Valispace, as the platform does not natively support harness-level interface representation. The exported data serves as a reference document for the AIV team but does not form part of the digital thread.

The toolchain depends on Valispace server access, API connectivity, and a valid access token. In test facility environments where network connectivity cannot always be guaranteed, this may limit availability during active test sessions.

No formal non-conformance or deviation management workflow is integrated within the toolchain. Deviations identified during test execution must be handled outside the framework and cannot be formally linked to the originating V&V activity or requirement in Valispace.

Re-running import scripts after model changes requires a prior rollback of affected elements in Valispace. The rollback mechanism relies on timestamped run log files being retained and correctly identified by the user, introducing a manual dependency in the model update workflow.

The toolchain does not implement a formal version control or configuration management strategy at the system level. The Capella model and Python scripts are managed as individual files without integration into a version control system such as Git, meaning there is no automated mechanism to track changes to the scripts, associate specific script versions with specific model revisions, or roll back the toolchain configuration to a known state. In a team environment where multiple engineers contribute to the model or scripts over time, the absence of formal configuration management at this level introduces a risk of inconsistency between the toolchain and the model baseline it operates on.

9.3.3. Evaluation

The evaluation is conducted on a single mission and assessed by two raters both affiliated with ISISPACE. The findings may not generalise across organisations with different tool configurations, team structures, or ECSS compliance requirements.

All scoring was based on direct observation of framework features during implementation rather than measured operational outcomes such as time saved or error rate reduction. No quantitative performance metrics were collected, which limits the conclusions that can be drawn about practical performance benefits in an operational AIV campaign.

The document-based reference approach was assessed against the ECSS-tailored workflow in practice at ISISPACE, providing a directly relevant baseline rather than a generic reference. However, document-based AIV workflows vary across organisations, and the scores may not be representative of document-based practice elsewhere.

9.4. Future Work

The following directions for future development correspond to limitations or partially supported areas identified in the evaluation.

Valispace API extension for verification close-out. The most immediately actionable improvement is the programmatic retrieval of per-requirement verification method and status assignments from the Valispace REST API. Once the API is extended to support per-requirement data retrieval, the manual export step in the reverse thread can be replaced with a fully automated fetch, completing the reverse digital thread without user intervention.

Satellite modes and configurations. The forward thread does not currently transfer operational

mode definitions from Capella to Valispace. Extending the toolchain to include Mode State Machine elements as structured metadata in Valispace would give the AIV engineer mode context during test planning and execution without needing to access the Capella model separately.

Non-conformance and deviation management. A structured non-conformance workflow integrated within Valispace, formally linked to the relevant V&V activity and requirement, would allow the AIV engineer to record deviations directly in the execution environment while maintaining traceability. This would address the informal change management limitation and bring the toolchain closer to ECSS non-conformance management requirements.

Configuration management and version control. The current toolchain does not implement formal version control at the system level. Integrating the Capella model and Python scripts into a version control system such as Git would enable change tracking across model revisions, associate script versions with specific model baselines, and complement the existing run log mechanism with a more robust and auditable form of configuration control.

New engineer onboarding. A dedicated onboarding guide documenting the toolchain workflow from both the systems engineer and AIV engineer perspectives, combined with a cross-tool campaign status dashboard showing the relationship between Capella and Valispace, would reduce the onboarding overhead identified through the A5 regression in the evaluation.

Subsystem and equipment-level requirement integration. Formalising the iterative workflow where subsystem and equipment-level requirements added in Valispace inform further modelling in Capella, and the resulting test procedures are flowed back, would complete the traceability chain from mission objective to component-level specification and reduce manual effort at the lower levels of the requirement hierarchy.

Formal sign-off and audit readiness. Developing a post-processing step that generates ECSS-compliant V&V close-out documents from Valispace data, including verification evidence references, approval signatures, and revision history headers, would address the formal sign-off limitation. This could be implemented as a Python script generating a formatted Word or PDF document for customer review board submission.

Extension to Epsilon3 for operational test execution. Integrating a dedicated procedure execution platform such as Epsilon3 as a third tool in the digital thread would extend the framework into real-time test execution with telemetry integration, role-based sign-off, and as-run procedure history. Capella would serve as the system design environment, Valispace as the requirements and verification planning environment, and Epsilon3 as the operational execution environment, addressing the informal change management and formal sign-off limitations simultaneously.

References

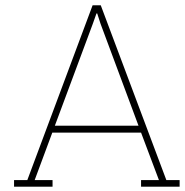
- [1] Thyrso Villela et al. “Towards the thousandth CubeSat: A statistical overview”. In: *International Journal of Aerospace Engineering* 2019.1 (2019).
- [2] Catherine Venturini et al. “Improving mission success of CubeSats”. In: (2018).
- [3] International Council on Systems Engineering. *INCOSE Systems Engineering Handbook - A Guide for System Life Cycle Processes and Activities (5th Edition)*. John Wiley & Sons, 2023. ISBN: 978-1-119-81429-0. URL: <https://app.knovel.com/hotlink/pdf/id:kt0149IGQ1/systems-engineering-handbook/system-life-cycle-concepts>.
- [4] National Aeronautics and Space Administration. *NASA Systems Engineering Handbook*. Special Publication NASA/SP-2016-6105. Revision 2. Washington, D.C.: National Aeronautics and Space Administration, 2016. URL: <https://ntrs.nasa.gov/citations/20170001761>.
- [5] ECSS Secretariat. *Space Engineering: Testing guidelines*. ECSS-E-HB-10-03A. European Cooperation for Space Standardization (ECSS). Noordwijk, The Netherlands, May 2022. URL: <https://ecss.nl/home/ecss-e-hb-10-03a-testing-guidelines-31-may-2022/>.
- [6] Eduardo Escobar Bürger. “A conceptual MBSE framework for satellite AIT planning”. In: *CEP* 12 (2019).
- [7] Azad M Madni and Michael Sievers. “Model-based systems engineering: Motivation, current status, and research opportunities”. In: *Systems Engineering* 21.3 (2018), pp. 172–190.
- [8] Paolo Minacapilli et al. “Small satellites mission design enhancement through MBSE and DDSE toolchain”. In: *Proceedings of the Model Based Space Systems and Software Engineering (MBSE2022), Toulouse, France* (2022), pp. 22–24.
- [9] Eileen A Bjorkman, Shahram Sarkani, and Thomas A Mazzuchi. “Using model-based systems engineering as a framework for improving test and evaluation activities”. In: *Systems Engineering* 16.3 (2013), pp. 346–362.
- [10] Simon Vanden Bussche, Marco Witzmann, and Louise Lindblad. “IAC-17, D3, 4, 7, x40657 THE HARDWARE DEVELOPMENT TOOL STACK FOR FUTURE SPACE EXPLORATION”. In: (2017).
- [11] ISISPACE. *6U EO CubeSat Mission Design Description Document*. Internal Technical Document. Confidential. Delft, The Netherlands: ISIS - Innovative Solutions In Space.

- [12] California Polytechnic State University. *CubeSat Design Specification (CDS) Rev. 14.1*. Document No. CP-CDS-R14.1. California Polytechnic State University (Cal Poly), San Luis Obispo. San Luis Obispo, CA, USA, Feb. 2022. URL: <https://www.cubesat.org/cubesatinfo>.
- [13] Chantal Cappelletti, Simone Battistini, and Benjamin K Malphrus. *Cubesat handbook: From mission design to operations*. Academic Press, 2020.
- [14] European Cooperation for Space Standardization. *Space project management: Project planning and implementation*. Tech. rep. ECSS-M-ST-10C Rev.1. Noordwijk, The Netherlands: ECSS Secretariat, ESA-ESTEC Requirements & Standards Division, Mar. 2009. URL: <https://ecss.nl/standard/ecss-m-st-10c-rev-1-project-planning-and-implementation/>.
- [15] ECSS Secretariat. *Space Engineering: Verification guidelines*. ECSS-E-HB-10-02A. European Cooperation for Space Standardization (ECSS). Noordwijk, The Netherlands, Dec. 2010. URL: <https://ecss.nl/hbstms/ecss-e-10-02a-verification-guidelines/>.
- [16] L Iossa et al. “From design to delivery in three months: the fast development of a 3U CubeSat”. In: *74th International Astronautical Congress (IAC)*. IAF. 2023.
- [17] Edward Ralph Carroll and Robert Joseph Malins. “Systematic literature review: How is model-based systems engineering justified?” In: (2016).
- [18] ISO/IEC/IEEE. *Systems and software engineering - Vocabulary*. Tech. rep. ISO/IEC/IEEE 24765:2017(E). International Organization for Standardization, 2017. URL: <https://www.iso.org/standard/71952.html>.
- [19] Iqtiaar Siddique. “MBSE implementation in small satellite systems: Rationale for adoption over traditional document-based systems engineering”. In: *Available at SSRN 5149862* (2024).
- [20] Serena Campioli et al. “Towards a unified modeling and simulation framework for space systems: Integrating model-based systems engineering with open source multi-domain simulation environments”. In: *Aerospace* 12.8 (2025), p. 745.
- [21] Marco Ackley. *Efficient Engineering 101 (II): Document-Driven vs. Data-Driven Systems Engineering (DDSE)*. Accessed [12-Feb-2026]. Valispace. Oct. 2019. URL: <https://www.valispace.com/efficient-engineering-101-ii-document-driven-vs-data-driven-systems-engineering-ddse/>.
- [22] Louise Lindblad, Marco Witzmann, and Simon Vanden Bussche. “Data-driven systems engineering. turning mbse into industrial reality”. In: *ESA International Systems & Concurrent Engineering for Space Applications Workshop*. 2018.
- [23] Supreet Kaur et al. “A Dialogue on the Digitization of Requirements, Verification, and Test Management with Data-Driven Systems Engineering (DDSE)”. In: *Proceedings of the 71st International Astronautical Congress (IAC)*. Paper ID: IAC-20,D5,1,10,x60415. International Astronautical Federation (IAF). Oct. 2020. URL: <https://www.valispace.com/wp-content/uploads/2020/11/IAC-20D5110x60415-Valispace-IAC-2020.pdf>.

- [24] Valispace GmbH. *Valispace - Where the world builds hardware*. Accessed [21-Jan-2026]. Valispace. 2021. URL: <https://www.valispace.com>.
- [25] Mert Ozkaya, Geylani Kardas, and Mehmet Alp Kose. “The Analysis of Requirements Engineering Tools for the Needs of Practitioners”. In: (2023).
- [26] F Weijand. “Thesis report-AE5810”. In: ().
- [27] Jan Kreisinger and Sanjay Chatterjee. “Verification Plan in Requirements Management Tool: Simple Traceability and Automated Interface to Regression Manager”. In: ().
- [28] Carlotta Deiana. “Integrating data-driven and model-based systems engineering for end-to-end small-satellite design”. MA thesis. Politecnico di Torino, 2025.
- [29] Juan José Mejía González. “Implementation of agile methodologies and model-based systems engineering for the management, design and development of a low earth orbit cubesat mission”. In: (2024).
- [30] Sanford Friedenthal, Alan Moore, and Rick Steiner. *A practical guide to SysML: the systems modeling language*. Morgan Kaufmann, 2014.
- [31] Azad M Madni and Shatad Purohit. “Economic analysis of model-based systems engineering”. In: *Systems* 7.1 (2019), p. 12.
- [32] Jeff A Estefan et al. “Survey of model-based systems engineering (MBSE) methodologies”. In: *IncoSE MBSE Focus Group* 25.8 (2007), pp. 1–12.
- [33] Kaitlin Henderson and Alejandro Salado. “Value and benefits of model-based systems engineering (MBSE): Evidence from the literature”. In: 24.1 (2021), pp. 51–66.
- [34] INVESTMENT IN QUALITY EARLY-PHASE. “THE ENDURING PATH TO SYSTEM SUCCESS”. In: (2019).
- [35] Miriam Amato. “Pre-Phase A design through MBSE of CUBE mission”. In: (2021).
- [36] David Kaslow and Azad M Madni. “Validation and verification of MBSE-compliant CubeSat reference model”. In: *Disciplinary Convergence in Systems Engineering Research*. Springer, 2017, pp. 381–393.
- [37] Gargi Sunil Pantoji. “MBSE-driven early Risk Management: A Framework for the NEBULA-Xplorer mission”. MSc Thesis. Delft, The Netherlands: Delft University of Technology, Nov. 2025. URL: <https://repository.tudelft.nl>.
- [38] Charlotte Strobbe, Sophia Salas Cordero, and Rob Vingerhoeds. “Open-source CubeSat MBSE approach to address traceability: Power subsystem”. In: *2023 IEEE international systems conference (SysCon)*. IEEE. 2023, pp. 1–8.
- [39] Johan Cederbladh and Daniel Krems. “Successfully Integrating Early Validation and Verification in Industrial MBSE”. In: *INCOSE International Symposium*. Vol. 35. 1. Wiley Online Library. 2025, pp. 1886–1900.

- [40] Zachary Scott Decker. “A systems-engineering assessment of multiple CubeSat build approaches”. PhD thesis. Massachusetts Institute of Technology, 2016.
- [41] João P Monteiro et al. “Integration and verification approach of ISTSat-1 CubeSat”. In: *Aerospace* 6.12 (2019), p. 131.
- [42] Sebastian Kottmeier et al. “The Eu: CROPIS assembly, integration and verification campaigns: building the first DLR compact satellite”. In: *69th International Astronautical Congress 2018*. 2018.
- [43] INCOSE. *Systems Engineering Vision 2035: Engineering Solutions for a Better World*. Accessed 10 Feb. 2026. International Council on Systems Engineering (INCOSE). San Diego, CA, Dec. 2021. URL: <https://sevisionweb.incose.org>.
- [44] Jean-Charles Chaudemar and Pierre de Saqui-Sannes. “MBSE and MDAO for early validation of design decisions: a bibliography survey”. In: *2021 IEEE international systems conference (SysCon)*. 2021, p. 0.
- [45] Ms Serena Campioli, Ms Carlotta Deiana, and Sabrina Corpino. “Integrating Concurrent Engineering with Data-Driven and Model-Based Systems Engineering for End-to-End Small-Satellite Design”. In: (2025).
- [46] Vasileios Sideris. “Advancing Model-Based Systems Engineering (MBSE) in the Development of Systems Architecture: Exploring the Value of MBSE during Early-Stage Naval Vessel Design”. MSc Thesis. Delft, The Netherlands: Delft University of Technology, July 2023. URL: <https://repository.tudelft.nl/record/uuid:42c74b48-a4e7-4542-87ae-5320d11a027b>.
- [47] Louise Lindblad et al. “Systems Engineering from a web browser: turning MBSE into industrial reality”. In: *Systems Engineering and Concurrent Engineering for Space Applications (SECESA)* (2016).
- [48] 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.
- [49] 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.
- [50] Pascal Roques. *Systems architecture modeling with the Arcadia method: a practical guide to Capella*. Elsevier, 2017.
- [51] Jean-Luc Voirin. *Model-based system and architecture engineering with the arcadia method*. Elsevier, 2017.
- [52] Tom McDermott et al. “Framework for and progress of adoption of digital and model-based systems engineering into engineering enterprises”. In: *Conference on Systems Engineering Research*. Springer. 2023, pp. 69–82.
- [53] Eclipse Foundation. *Capella: Open Source Solution for Model-Based Systems Engineering*. Accessed [13-Jan-2026]. Eclipse Foundation. 2022. URL: <https://mbse-capella.org/>.

- [54] ISISPACE. *System Requirements and VCD*. Internal Technical Document. Confidential - not publicly available. Delft, The Netherlands: ISIS - Innovative Solutions In Space.
- [55] ISISPACE. *System Concept of Operations*. Internal Technical Document. Confidential - not publicly available. Delft, The Netherlands: ISIS - Innovative Solutions In Space.
- [56] ISISPACE. *Operations User Manual*. Internal Technical Document. Confidential - not publicly available. Delft, The Netherlands: ISIS - Innovative Solutions In Space.
- [57] ISISPACE. *FlatSat Hardware Tracking List*. Internal Technical Document. Confidential - not publicly available. Delft, The Netherlands: ISIS - Innovative Solutions In Space.
- [58] ISISPACE. *System Assembly, Integration and Test Plan*. Internal Technical Document. Confidential - not publicly available. Delft, The Netherlands: ISIS - Innovative Solutions In Space.
- [59] ISISPACE. *Flight Model Interface Diagrams*. Internal Technical Document. Confidential - not publicly available. Delft, The Netherlands: ISIS - Innovative Solutions In Space.
- [60] ISISPACE. *Flight Model Harness Diagrams*. Internal Technical Document. Confidential - not publicly available. Delft, The Netherlands: ISIS - Innovative Solutions In Space.
- [61] ISISPACE. *System Assembly, Integration and Test Report*. Internal Technical Document. Confidential - not publicly available. Delft, The Netherlands: ISIS - Innovative Solutions In Space.
- [62] ISISPACE. *Engineering Model Platform Electrical Integration Test Procedure*. Internal Technical Document. Confidential - not publicly available. Delft, The Netherlands: ISIS - Innovative Solutions In Space.
- [63] ISISPACE. *System Harnesses Design List*. Internal Technical Document. Confidential - not publicly available. Delft, The Netherlands: ISIS - Innovative Solutions In Space.
- [64] J. Richard Landis and Gary G. Koch. “The Measurement of Observer Agreement for Categorical Data”. In: *Biometrics* 33.1 (1977), pp. 159–174. DOI: 10.2307/2529310.
- [65] Nicolas Oidtmann. “Implementation and Application of MBSE in Post-CDR Satellite Projects”. MSc Thesis. Delft University of Technology, May 2026.



Appendix A

This appendix contains the Capella system modelling diagrams referenced throughout Chapter 5 that are not included in the main body of the report. They provide additional architectural detail across the four ARCADIA layers and are included here to maintain the readability of the main text while ensuring completeness of the system model documentation for the 6UXL Earth Observation CubeSat mission.

Operational Analysis

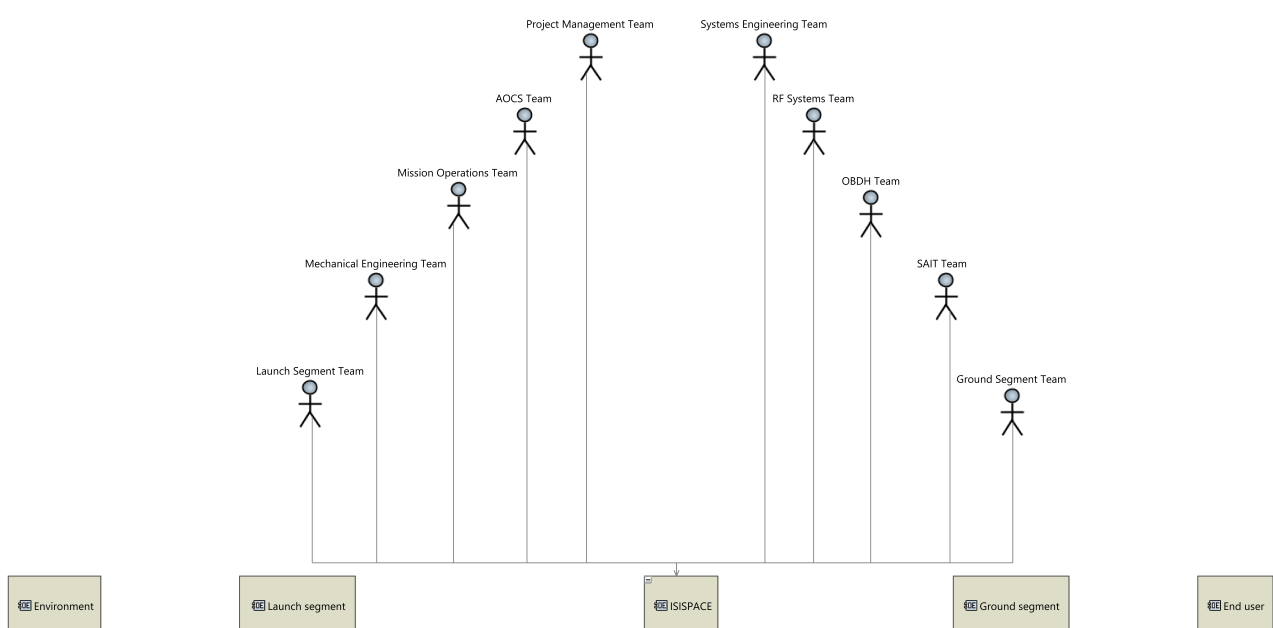


Figure A.1: [OEBD] Operational Entity Breakdown Diagram

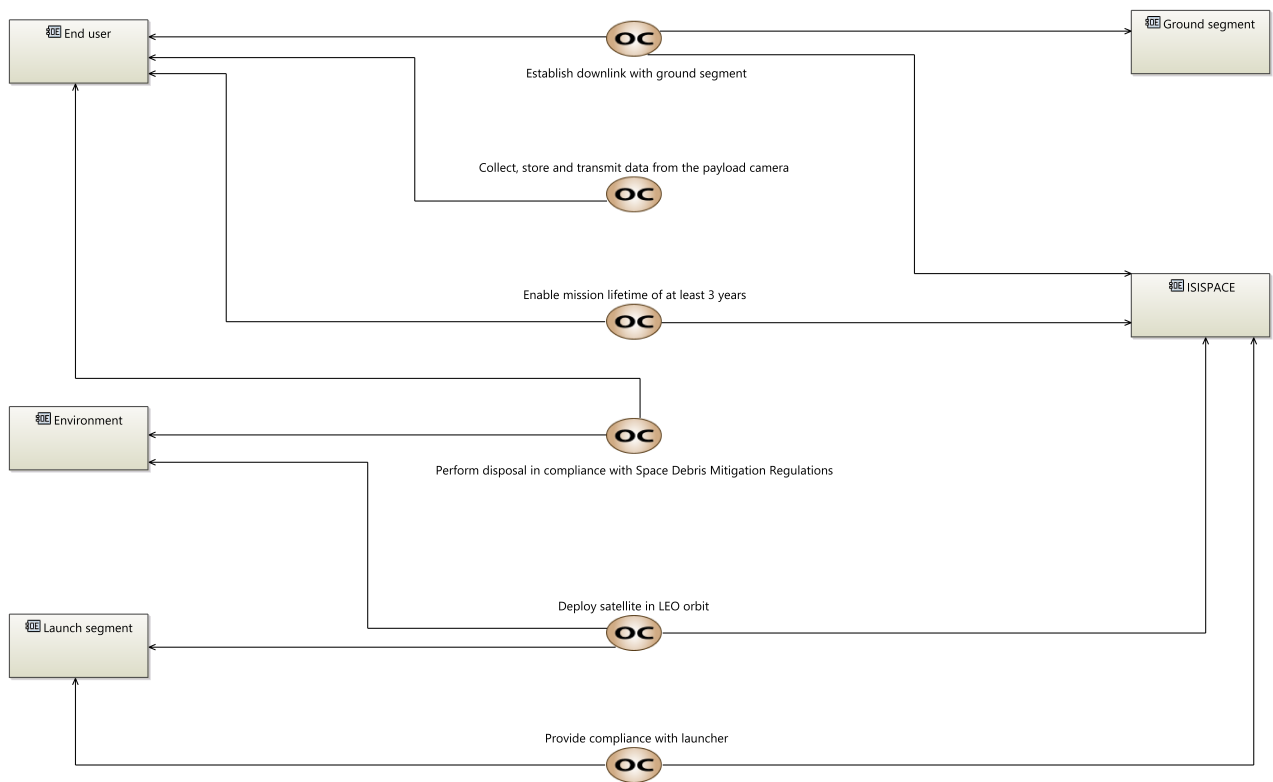


Figure A.2: [OCB] Operational Capabilities Blank

System Analysis

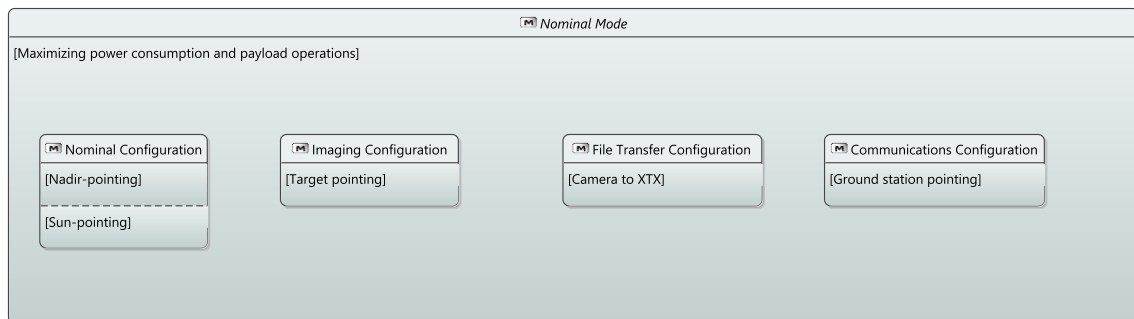


Figure A.3: [MSM] Satellite configurations modelled as sub-modes of the nominal operational mode

Logical Architecture

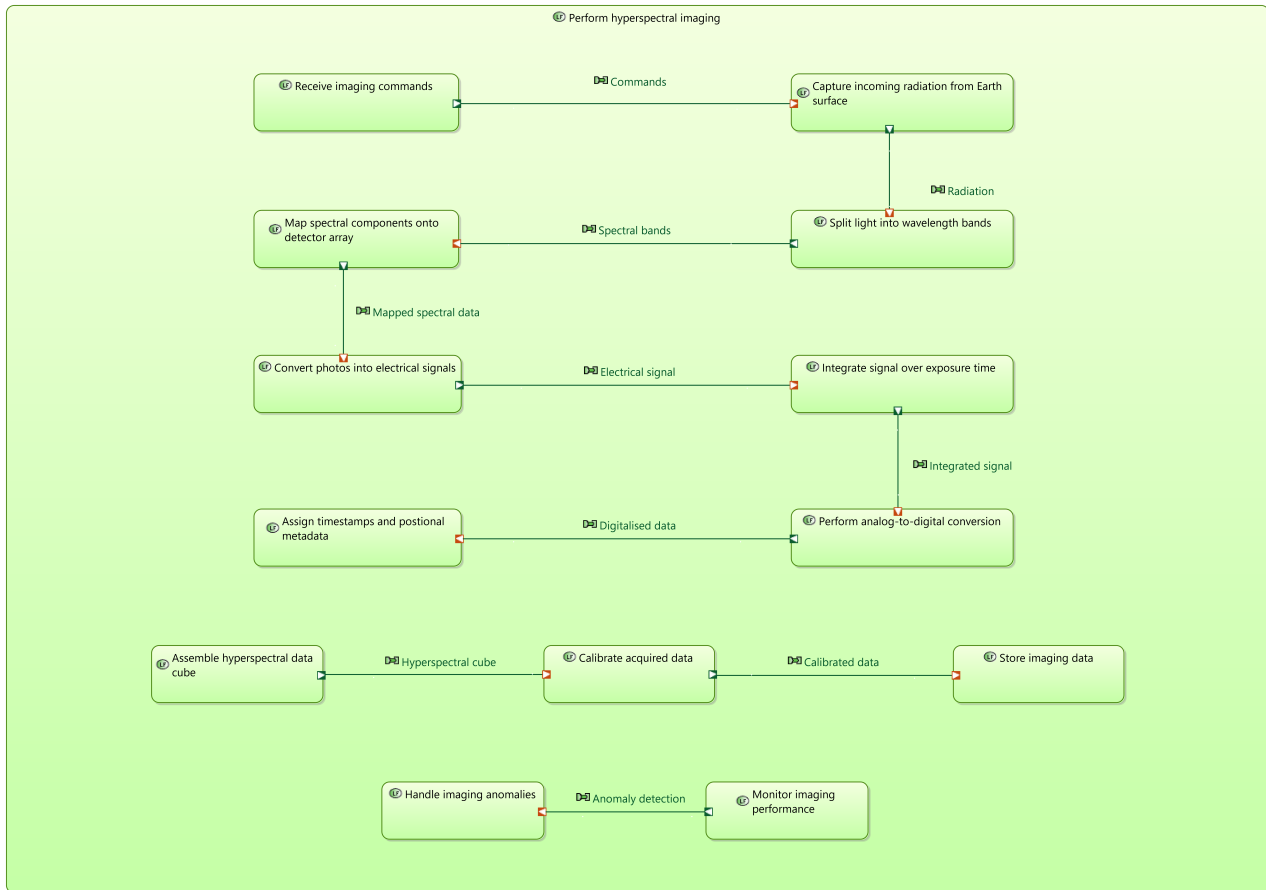


Figure A.4: [LDFB] Logical Dataflow Blank for the hyperspectral imaging function

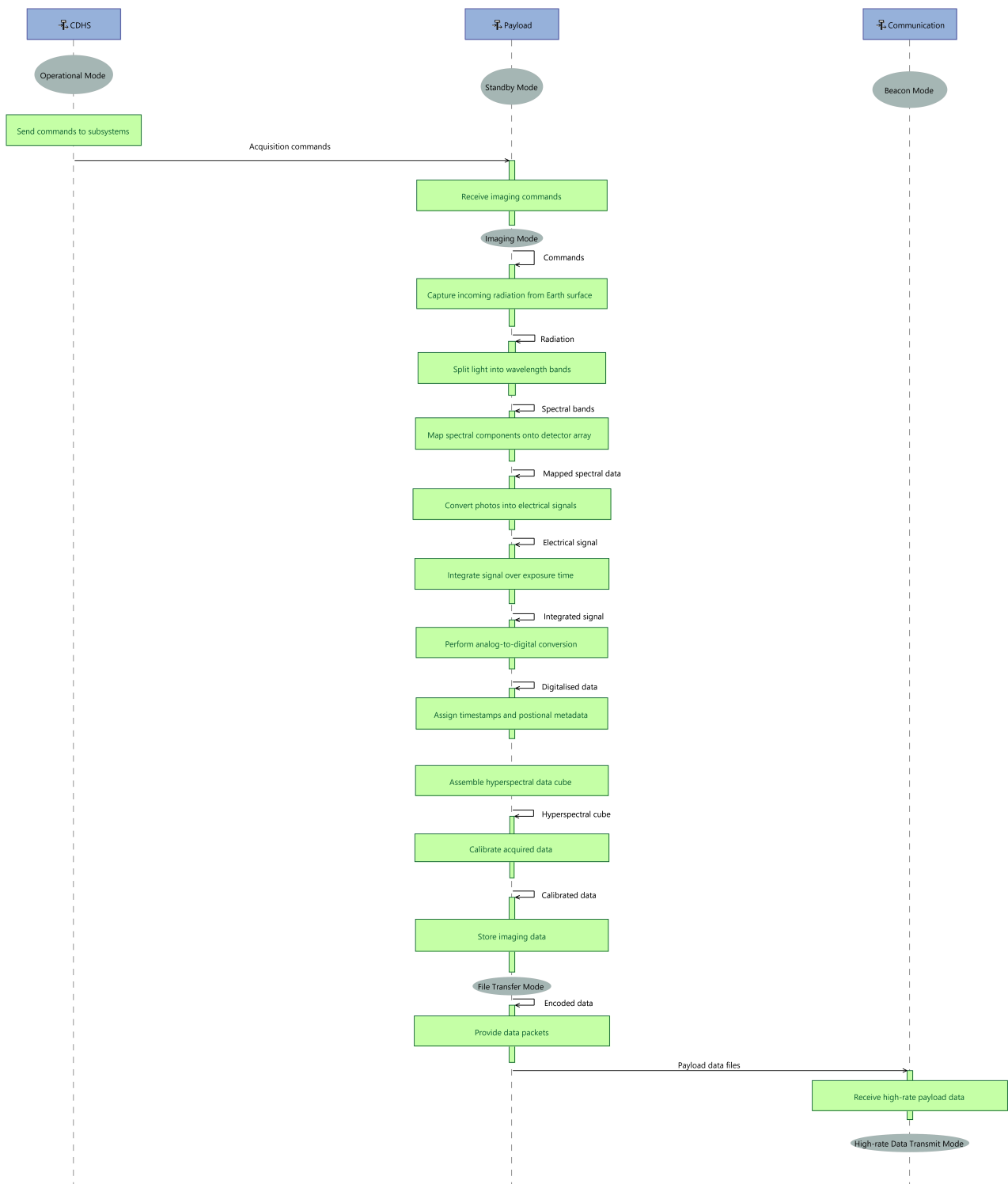


Figure A.5: [ES] Exchange Scenario for the imaging capability at the Logical Architecture layer

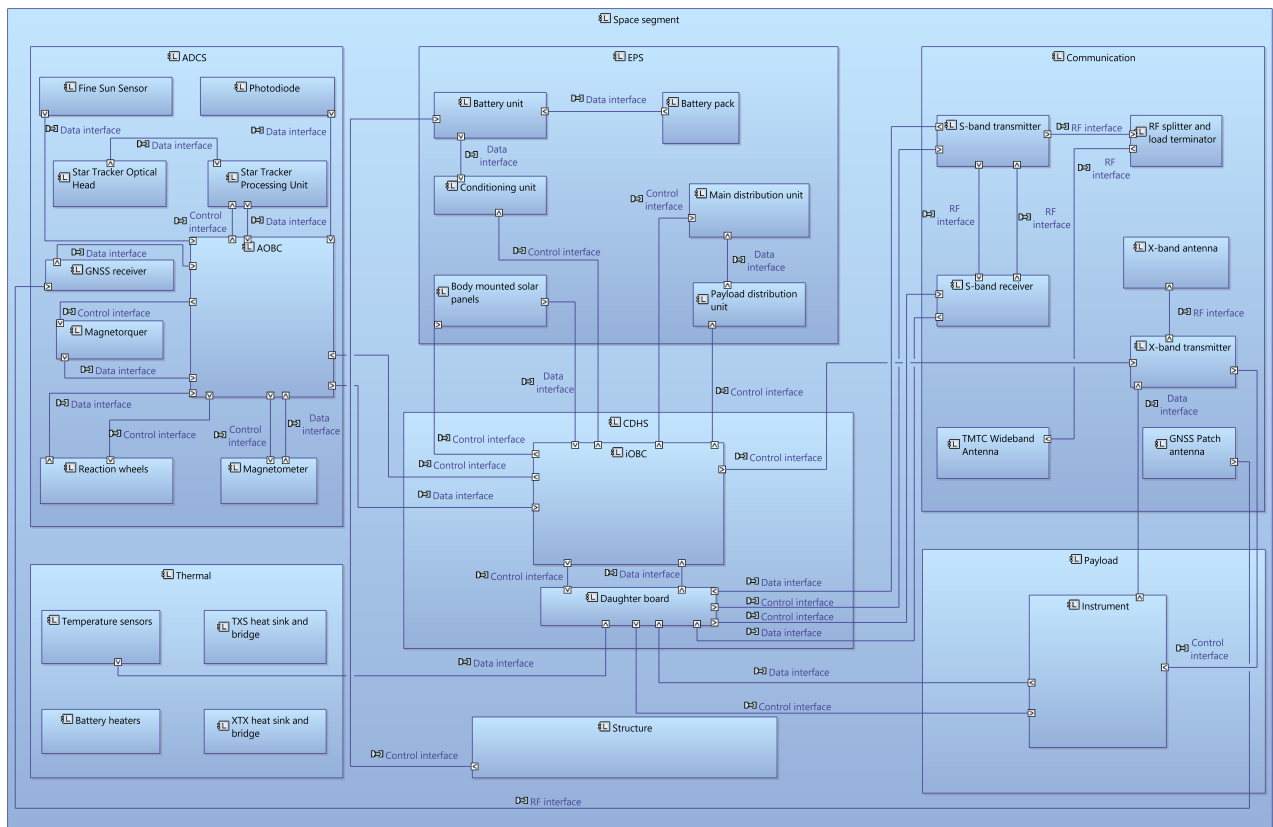


Figure A.6: [LAB] Control, data and RF interfaces of the space segment

Physical Architecture

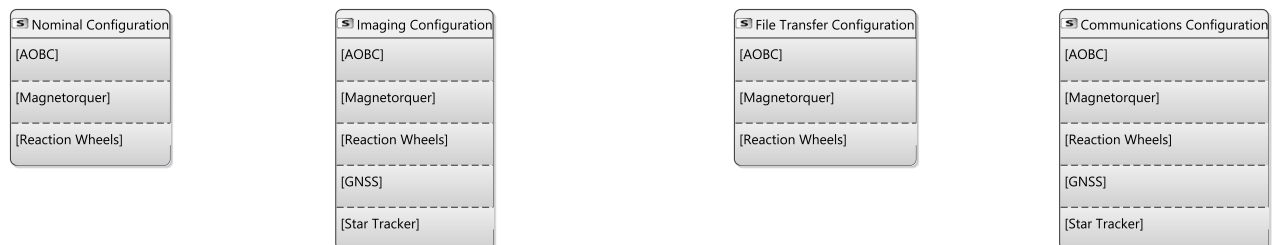


Figure A.7: [MSM] ADCS equipment active during nominal mode configurations

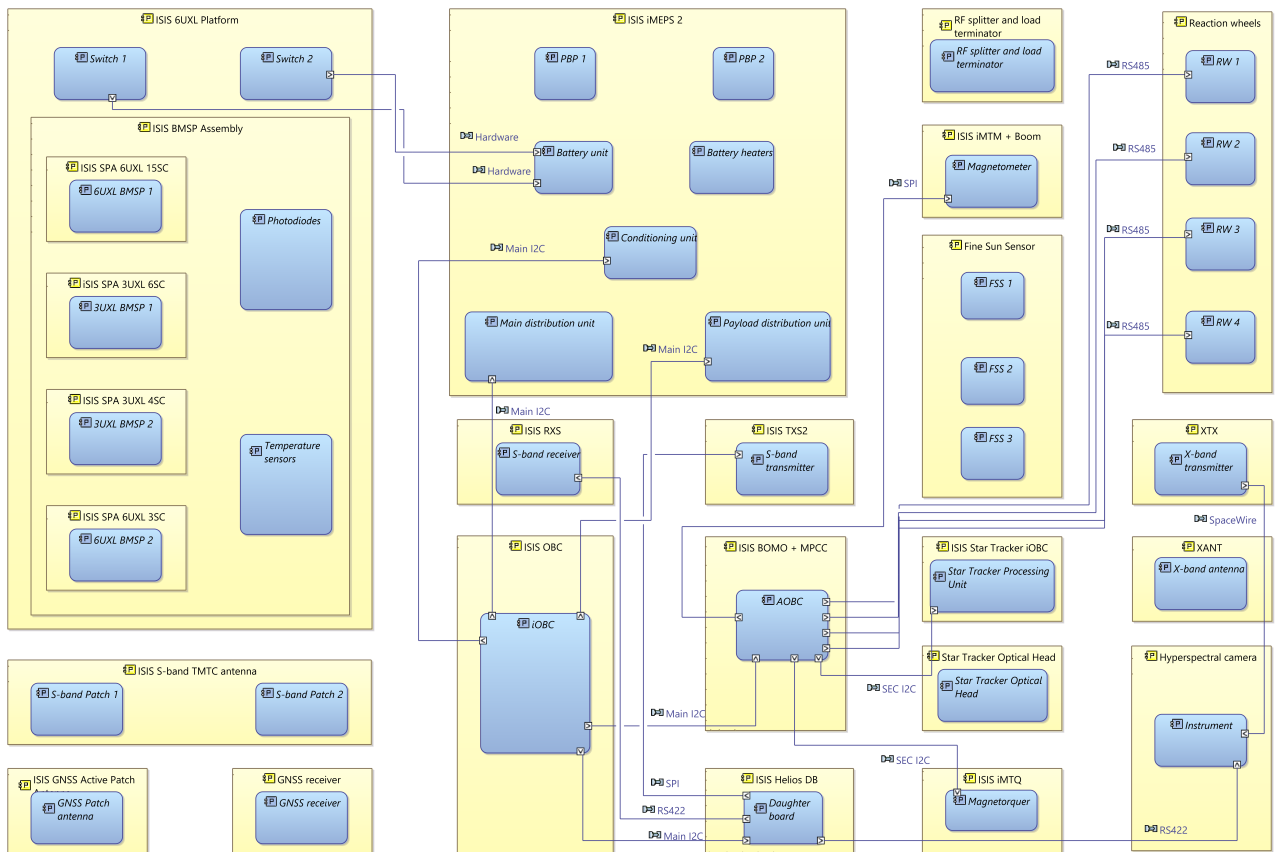


Figure A.8: [PAB] Data interface diagram of the space segment

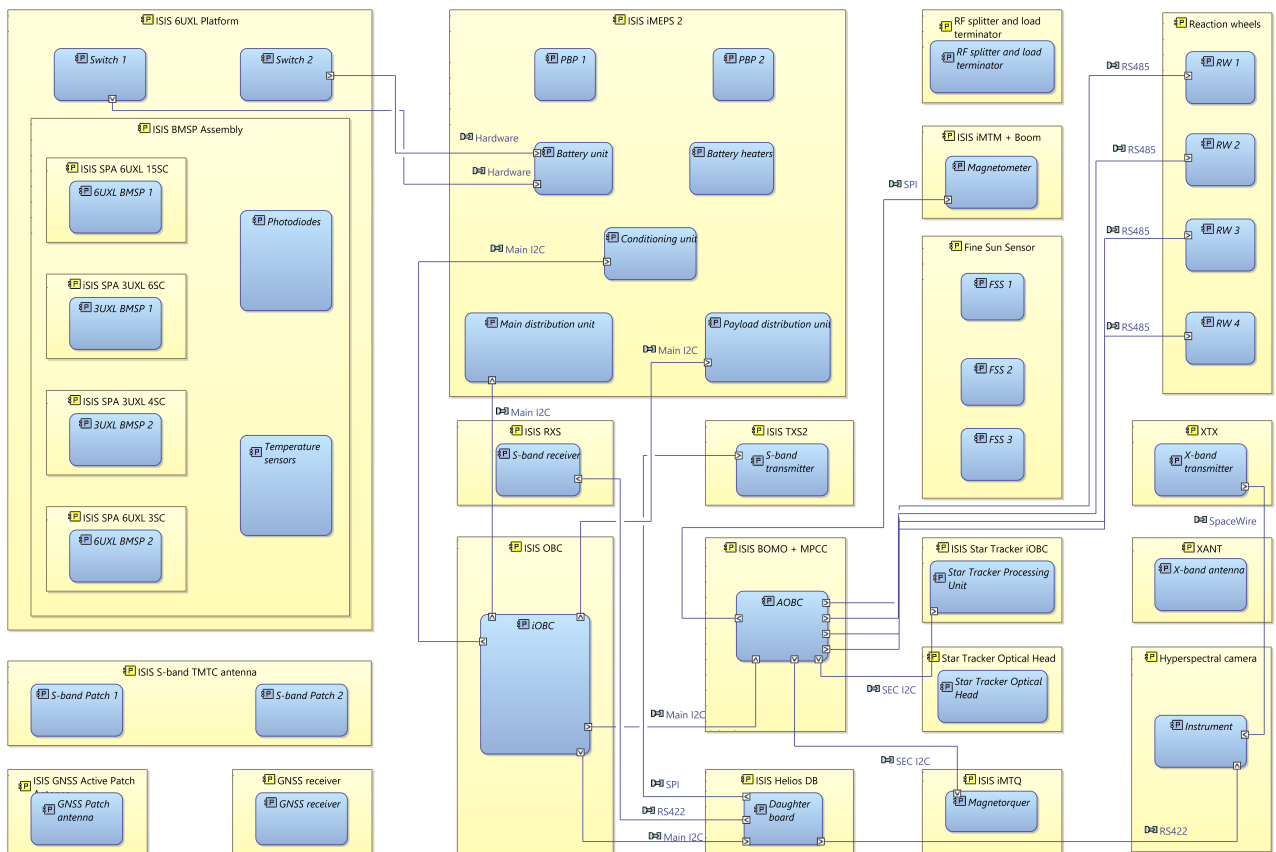


Figure A.9: [PAB] Control interface diagram of the space segment

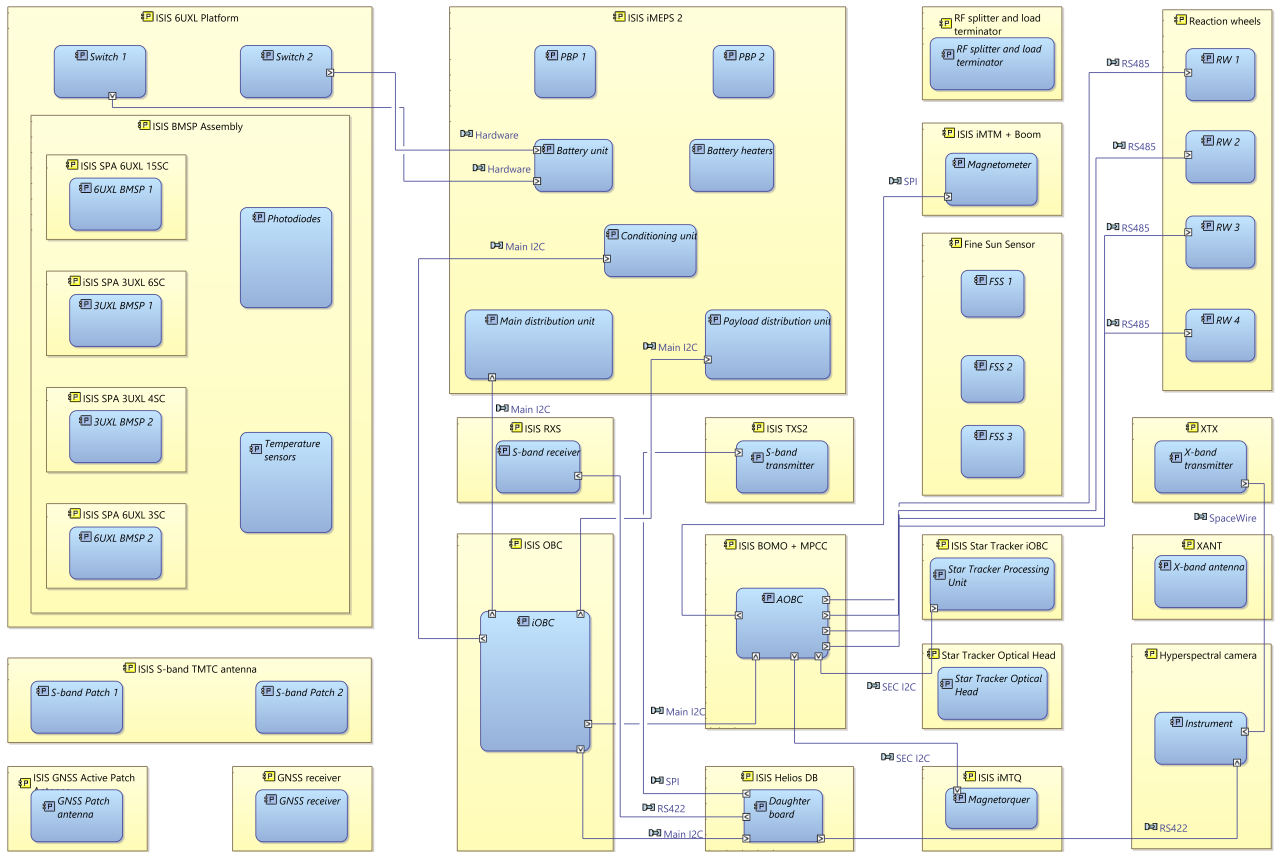


Figure A.10: [PAB] RF interface diagram of the space segment

B

Appendix B

This appendix presents the independent structured expert evaluation of the three AIV workflow approaches conducted by a senior systems engineer at ISISpace. The four tables below present the expert's scores for each success factor across the Traceability, Consistency, Flexibility, and Accessibility and Stakeholder Collaboration metrics, using the same colour convention as the author's evaluation tables in Chapter 8: green indicates fully supported (3), yellow indicates partially supported (2), and red indicates not supported (1). A weighted Cohen's Kappa of $k_w = 0.712$ was calculated between the two independent assessments, indicating substantial inter-rater agreement and supporting the reproducibility of the evaluation scores reported in Chapter 8.

Traceability

Table B.1: Independent expert evaluation scores for Traceability

ID	Factor	AIV Engineering Need	DB	C	C-V
T1	Requirement-to-verification coverage	Can every requirement be linked to at least one verification activity?	1	1	3
T2	Verification evidence linkage	Can verification results be navigated back to their originating requirement?	1	3	3
T3	Component-to-requirement allocation	Can each hardware component be linked to the requirements it must satisfy?	1	2	2
T4	Test procedure-to-requirement linkage	Is each test procedure explicitly linked to the requirement it verifies?	1	2	3
T5	Lifecycle traceability continuity	Is traceability maintained from Phase C handover through verification close-out without manual re-establishment?	1	1	2
T6	Cross-discipline traceability	Is traceability accessible to all engineering roles without specialist tool expertise?	3	1	1

Consistency

Table B.2: Independent expert evaluation scores for Consistency

ID	Factor	AIV Engineering Need	DB	C	C-V
C1	Data accuracy at handover	Is engineering data received by the AIV team accurate and complete at the start of Phase D?	1	2	2
C2	Budget data integrity	Are mass and power values consistent between the design record and AIV tracking?	1	2	3
C3	Requirement integrity	Is requirement text, hierarchy, and identifier consistent throughout the AIV campaign?	1	2	3
C4	As-built data accuracy	Are as-built measurements accurately captured and reflected in the system record?	1	2	3
C5	Configuration control	Is there a clear mechanism to prevent unauthorised or untracked changes to AIV data?	2	1	2
C6	Formal document revision control	Is versioning and revision history of AIV artefacts formally maintained and customer-visible?	3	1	1

Flexibility

Table B.3: Independent expert evaluation scores for Flexibility

ID	Factor	AIV Engineering Need	DB	C	C-V
F1	Requirement change accommodation	When a requirement changes, how much rework is required to update the AIV record?	1	1	1
F2	Scope change accommodation	Can component additions or removals be accommodated without rebuilding the AIV structure?	1	1	1
F3	Test procedure modification	Can test procedures be updated or extended without rebuilding the documentation structure?	1	2	2
F4	Re-use across missions	Is the framework applicable to a different CubeSat mission without significant restructuring?	1	2	2
F5	Iterative verification support	Is re-verification after a design change supported without loss of previous evidence?	2	2	2
F6	Informal change management	Can ad-hoc deviations and non-conformances during test execution be accommodated without tool intervention?	3	1	1
F7	Infrastructure independence	Is the framework operational without server access, API connectivity, or software licences?	3	2	1

Accessibility and Stakeholder Collaboration

Table B.4: Independent expert evaluation scores for Accessibility and Stakeholder Collaboration

ID	Factor	AIV Engineering Need	DB	C	C-V
A1	Role-appropriate access	Can AIV engineers access and contribute without requiring systems engineering tool expertise?	3	1	1
A2	Verification status visibility	Can stakeholders view current verification status without a manually compiled report?	1	2	3
A3	Concurrent working	Can SE and AIV engineers work simultaneously without blocking each other?	1	1	2
A4	Handover clarity	Is the Phase C to Phase D handover formally defined, complete, and unambiguous?	1	1	2
A5	New engineer onboarding	Can a new AIV engineer understand the campaign structure and begin contributing quickly?	2	1	1
A6	Formal sign-off and audit readiness	Do AIV artefacts support formal review board approval, signature blocks, and customer submission?	3	1	1
A7	Stakeholder access without tool licence	Can customers, authorities, or auditors access AIV records without installing specialist tools?	3	1	1