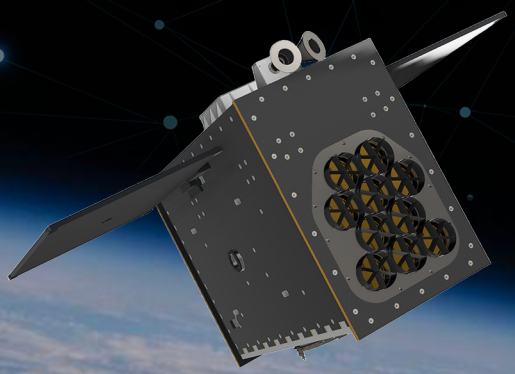
MBSE-driven early Risk Management

A Framework for NEBULA-Xplorer mission

AE5822: Thesis Space

Gargi Sunil Pantoji

Delft University of Technology



TU Delft

SRON

MBSE-driven early Risk Management

A Framework for NEBULA-Xplorer mission

by

Gargi Sunil Pantoji

Thesis work was completed at the Space Research Organisation Netherlands(SRON), 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 18th November 2025

Student number:	6032702	
Project duration:	February 2025 – November 2025	
Thesis committee:	Dr. I. Akay,	TU Delft, Supervisor
	Ir. M. Grim,	SRON, Supervisor
	Dr. J. Guo,	TU Delft
	Dr. ir. G. la Rocca,	TU Delft

This thesis is confidential and cannot be made public until December 31, 2025.

TU Delft

SRON

Contents

List of Figures

List of Tables

Nomenclature

Abbreviations

Abbreviation	Definition
CDR	Critical Design Review
ConOps	Concept of Operations
DBSE	Document Based Systems Engineering
ECSS	European Cooperation for Space Standardization
EM	Engineering Model
FDIR	Fault Detection, Isolation and Recovery
FEMMP	Framework for the Evaluation of MBSE Methodologies and Processes
FMEA	Failure Modes and Effects Analysis
FMECA	Failure Modes, Effects and Criticality Analysis
FRP	Function Risk Profile
FRS	Failure Resilience Score
HK	Housekeeping (telemetry data)
LAB	Logical Architecture Diagram
LCBD	Logical Component Breakdown Diagram
LDFB	Logical Data Flow Blank
LFBD	Logical Functional Breakdown Diagram
MBSE	Model-Based Systems Engineering
MRD	Mission Requirements Document
OAB	Operational Architecture Diagram
OEBD	Operational Entities Breakdown Diagram
PA	Product Assurance
PDR	Preliminary Design Review
PI	Principal Investigator
PVMT	Property Values Management Tool
RAMS	Reliability, Availability, Maintainability and Safety
SAB	System Architecture Blank
SDFB	System Data Flow Blank
SE	Systems Engineering
SEBoK	Systems Engineering Body of Knowledge
SPOF	Single Point of Failure
SRD	System Requirements Document
SRR	System Requirements Review

1

Introduction

This chapter introduces the thesis and sets the stage for the research. First, it outlines the context and motivation that drive this work. Finally, it presents the objectives and provides an overview of the structure of the report.

1.1. Motivation

Ensuring the reliability of space systems is one of the most critical challenges in mission design. Mission success primarily hinges on how early potential faults are surfaced and addressed in design. When issues are discovered late, during qualification testing or on-orbit, design freedom is limited, infeasible or expensive. Even small satellite missions, such as those led by universities, face this challenge: while they operate with lower budgets, the scientific and educational costs of mission failure remain high.

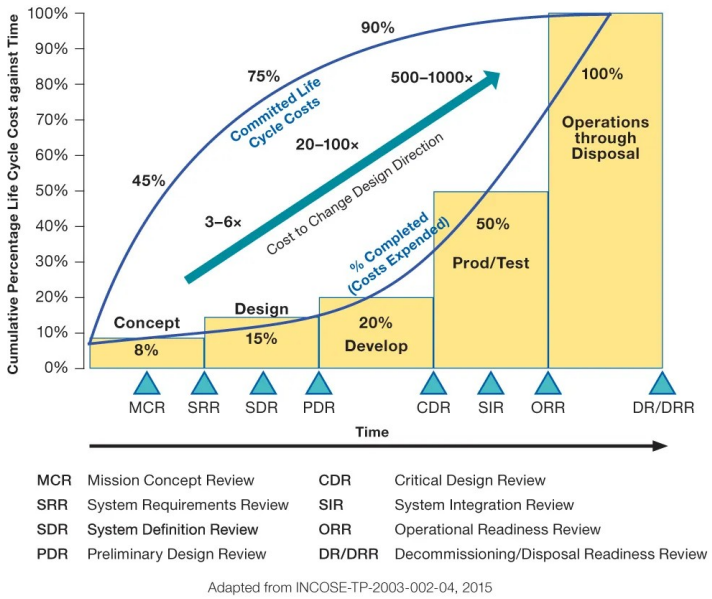


Figure 1.1: Life-cycle cost impact curve [1]

The well-established systems-engineering insight is that the cost to change a design rises steeply with lifecycle maturity. Decisions made in the concept and preliminary design phases lock in the majority (about 75%) of a system's life-cycle cost (Figure 1.1).

Despite this, fault detection and risk assessment in practice are still dominated by document-centric, late-phase methods and component-focused tests, with limited emphasis on system-level behaviour early in design. The result is a persistent gap between mission complexity and validation methods. According to NASA reports, nearly 40% of small satellite missions over the past two decades have ended in partial or total failure. Many of these failures are not due to major hardware defects, but arise from software behaviour, subsystem interactions, communication protocols, or lack of integration testing [2]. NASA's Small Satellite Reliability Initiative has similarly identified gaps in standardised processes and system-level validation as major contributors to mission losses [3].

Model-Based Systems Engineering (MBSE) is a promising framework to address these challenges. It provides a platform to model and analyse system architectures from the earliest stage [4]. However, while MBSE is widely recognised for its potential in enhancing traceability, documentation, and design integration, proactive early-stage fault detection is not popular in current practices [5]. If leveraged beyond documentation, MBSE can move fault detection and risk reasoning into the conceptual and preliminary design phases, when it's cheapest and most impactful to implement. This thesis is motivated by that opportunity: to develop and demonstrate an **MBSE-based fault detection and risk-assessment framework** that implements early, system-level checks rather than relying solely on late-phase, document-driven analysis.

1.2. Problem statement

Current fault detection and risk assessment activities are concentrated in the later stages of system development. Methods such as FMEA, FMECA, FTA, and operational FDIR, while valuable for insights, are typically applied during verification or operations, when design freedom is limited and corrections are costly.

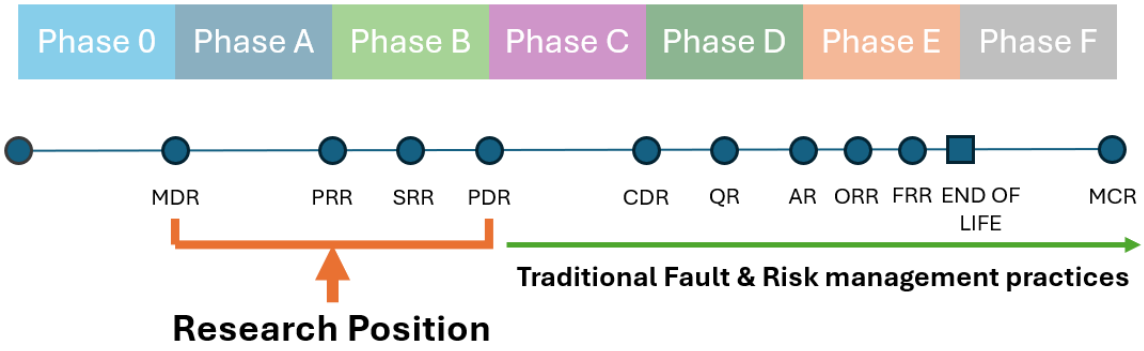


Figure 1.2: Research Position of this thesis

As shown in Figure 1.2, most existing practices are positioned at later phases of the systems engineering lifecycle. In contrast, Phases A–B remain underutilised for fault detection and risk assessment, even though decisions made here determine the majority of life-cycle cost.
This thesis directly addresses this lack of balance by focusing on the early phases of the development lifecycle. By shifting fault detection and risk reasoning upstream, it seeks to provide actionable feedback when it is most effective, laying the groundwork for the research objectives and questions that follow.

1.3. Case Study Context: NEBULA-Xplorer

To ground the research in a realistic setting, this thesis adopts **NEBULA-Xplorer** as its case study. NEBULA-Xplorer is an innovative X-ray astronomy mission at Space Research Organisation Netherlands (SRON), featuring a novel instrument concept designed to investigate extreme gravity and relativistic behaviour of binary systems [6]. The mission combines a novel scientific payload with standard satellite subsystems, and therefore faces the typical challenges of student satellite projects: limited resources, multidisciplinary integration, and stringent reliability requirements.

These characteristics make NEBULA-Xplorer an ideal context to demonstrate an MBSE-based framework for early fault detection and risk assessment. The mission is sufficiently complex to reveal the strengths and limitations of the approach, while still at a stage in its lifecycle where early design interventions can influence key architectural choices. A more detailed description of the mission, including its objectives, structure, and requirements, is provided in Chapter 4. The mission will serve as a practical and relevant case study to evaluate how early-stage fault detection could potentially affect design decisions.

1.4. Objectives and research questions

The Research Objective for this thesis is as follows:

To develop a fault detection framework integrated with MBSE to improve early-phase design fault identification and risk assessment for the NEBULA-Xplorer mission.

The main research question is broad in its scope. Therefore, it can be broken down into the following sub-questions to aid in the completion of the project:

- **RQ1:** How can MBSE models be developed to represent fault-critical information necessary for early-stage fault identification?
- **RQ2:** What fault patterns can be identified from early-phase MBSE models?
- **RQ3:** How can an external analysis tool be integrated with MBSE tools to enable the automated detection of early-stage faults?
- **RQ4:** How effective is the MBSE-based fault detection and risk assessment framework in identifying early-stage faults within the NEBULA Xplorer mission context?

1.5. Thesis outline

The content in this thesis is structured around four research questions.

Chapter 1 introduces the problem context, motivation, and defines the research questions and objective.

Chapter 2 and 3 address **RQ1**, reviewing existing methods for early fault detection and risk assessment in the context of space systems. The state of the art is analysed to identify gaps and to derive directions for framework development.

Chapter 3 answers **RQ2** by designing the proposed MBSE-based framework. The methodology for integrating fault detection, failure analysis, and risk assessment layers is described, supported by the definition of attributes, rules, and analysis steps.

Chapters 4, 5 and 6 collectively respond to **RQ3**. They introduce the NEBULA-Xplorer case study, describe the tool environment and implementation, and demonstrate the application of the framework through worked examples across all three analysis layers.

Chapter 7 addresses **RQ4**, validating the framework against its requirements and evaluating it through the FEMMP criteria. The chapter highlights strengths, limitations, and alignment with standards.

Chapter 8 concludes the thesis by summarising the answers to all research questions and reflecting on the contributions.

Chapter 9 provides recommendations for future work, outlining extensions and improvements that could increase the scope of the framework.

2

Background

This chapter provides the background needed to position the thesis. It defines terminology, reviews current practices in fault detection and risk assessment, highlights approaches to early-phase failure analysis, and introduces Model-Based Systems Engineering (MBSE) in the context of space missions. The chapter concludes by identifying the gaps that motivate the proposed framework.

2.1. Terminology

2.1.1. Fault and Failure

In dependability engineering, ***fault*** and ***failure*** form different points on a causal chain. According to ECSS, fault is defined as the inability or deviation of an item from its expected performance. Failure is the event resulting in an item to no longer performing its intended function. The presence of a fault, if left undetected, can manifest as failure [7].

Space systems are designed to be fault-tolerant, meaning that they can continue to achieve the intended mission objectives even in the presence of one or more faults. This is typically achieved through design measures such as redundancy or recovery mechanisms. A good fault-tolerant design can only be achieved through early fault detection as hidden or cascading faults can lead to failure if not identified and mitigated early.

2.1.2. Risk

In systems engineering, ***risk*** is considered a measure of combination of **likelihood** of an undesirable event with the **severity** of its consequences. The INCOSE Systems Engineering Handbook defines risk as "the potential consequence of a future event, which may be desirable or undesirable, and its probability of occurrence" [8]. NASA similarly frames risk as "the potential for performance shortfalls, which may be realised in the future, with respect to achieving explicitly established and stated performance requirements"[9].

Within the context of space missions, risks can be technical, programmatic, cost, or schedule-driven. This thesis focuses on **technical risks**: those arising from architectural decisions, subsystem interactions, or design choices. These are often linked to the presence of faults and their potential to propagate into failures.

Risk assessment, therefore, refers to the structured process of identifying, evaluating, and prioritising risks so that they can be mitigated proactively. In early design phases (Phases 0–A–B), the goal of risk assessment is not to achieve quantitative results, but to highlight critical design issues while flexibility is still high and corrective action is feasible.

2.2. Fault Detection and its link to Risk Assessment

Fault Detection, Isolation and Recovery (FDIR) methods have long played a critical role in ensuring the reliability of space missions. These methods generally focus on detecting and mitigating faults onboard a satellite in the shortest possible time. While the approaches to fault detection have matured over time, they still rely heavily on simulation, testing and telemetry data analysis and are often implemented late in the development cycle [10][9].

Traditional fault detection techniques are often categorised into model-based, data-driven and knowledge-based approaches:

- **Model-Based Approaches**
  These approaches rely on physical or analytical models to replicate the system's behaviour. Methods like parity relations and Kalman filters compare predicted behaviour with telemetry data to identify faults. They are commonly employed in onboard FDIR systems[11][12].
- **Data-driven Approaches**
  These use machine learning and statistical methods that are trained on real-time or historical telemetry data. Methods like support vector machines, neural networks and clustering have been applied in fault detection and require large volumes of labelled data. These techniques focus on late-stage design and operational phases as well [13].
- **Knowledge-Based Approaches**
  These approaches involve the development of an expert- defined system relying on rules, databases or predefined fault libraries. They are more explainable and can work with limited data, making them attractive for early-phase analysis [14]. Despite the effectiveness of FDIR in an operational context, one area of fault detection remains largely untapped. The integration of these existing techniques into system design activities could help detect and manage faults early on. However, most of these are either too heavy for Phases A and B studies or disconnected from the system architecture .
- **Hybrid Fault Detection Approaches**
  A rising trend for incorporating early fault management has been leaning towards Hybrid Fault detection, which blends multiple paradigms to tailor existing approaches to overcome their limitations. For example, Pesola proposed a

framework to integrate the benefits of model-based and data-driven approaches[15]. The fault detection was, however, demonstrated during the hardware-in-loop campaign, which is not always possible at an early stage. Another relevant work proposes a fault prediction system combining rule-based reasoning with structured fault tree logic [16].

These approaches are effective in operations, but offer limited support in the early design phases, which is the focus of this thesis.

2.2.1. Early phase fault detection

In Space missions, the early stages refer to initial phases where the mission is identified, its feasibility is evaluated, and a preliminary design is determined. These are typically designated as Phase 0, A, and B in ESA and NASA project life cycles [17].

Recent studies have emphasised the importance of shifting fault detection activities earlier into the system design phase. Studies conducted at Thales Alenia Space highlight how simpler architectures could be generated if fault management strategies are incorporated during requirements specification and design phases. Delaying this could result in late design changes that impact the cost and planning of the mission [18].

2.3. Risk Assessment methods

In parallel with fault detection, space missions typically apply risk assessment techniques to evaluate and prioritise critical aspects of design. Risk assessment includes identifying, analysing, prioritising and mitigating potential risks to a project or organization. It is performed in an integrated, holistic way, maximizing the overall benefits in areas such as: design, manufacturing, testing, operation, maintenance, and disposal, together with their interfaces control over risk consequences management, cost, and schedule [19].

While performing risk assessment, trade-offs are performed among often competing goals and associated risks are assessed based on their severity, impact and likelihood of occurrence. The assessments of the alternatives for mitigating the risks are iterated, and the resulting measurements of performance and risk trend are used to optimize the tradable resources [20].

Risk assessment methodologies help quantify and qualify risks based on assessment factors and provide a systematic framework for decision-making in an event of risk. This enables implementation of appropriate risk mitigation strategies.

Common risk assessment methodologies include:

Qualitative

Qualitative analysis uses descriptive labels like 'High', 'Medium', 'Low' based on subjective judgment to assess risk likelihood and impact. It is suitable for scenarios where quick assessment is needed or data is limited. It is a simple and fast method, but lacks measurable precision and may be inconsistent [21].

Quantitative

Quantitative analysis assigns numerical values to risks using statistical data or models like Monte Carlo simulations. It is suitable for cases with mature data about risk environments, as detailed financial or statistical modeling is involved. While it is highly precise and transparent, strong data availability and technical modeling is required to perform this assessment [21].

FMEA/FMECA

Failure Modes and Effects Analysis (FMEA) / Failure Modes, Effects and Criticality Analysis (FMECA) is performed on the functional design, physical design and the processes used to realize the final product **Functional FMECA**, **Product FMECA** and **Process FMECA** respectively) [22].

In FMEA assessment, all potential risks and failure modes are identified and classified based on severity, whereas in FMECA, they are classified based on criticality of their consequences. During the analysis, appropriate measures are proposed and introduced in the design to render all such consequences acceptable. [22].

When any design changes are made, the FMEA/FMECA is updated and the assessment is done again to analyse the effects of new failure modes introduced by the changes. This methodology also has provisions for failure detection and recovery actions. In addition, potential failure propagation is also studied.

Hardware-software interaction analysis (HSIA)

HSIA is be performed to ensure that the software reacts in an acceptable way to hardware failure. It is performed at the level of the technical specification of the software. This analysis can be included in the FMEA analysis [22].

Fault tree analysis (FTA)

A Fault Tree Analysis is performed to ensure that the design conforms to the failure tolerance requirements for combinations of failures or event combinations leading to the undesirable end events such as loss of mission. FTA can be performed at subsystem level with respect to the top events like loss of function of the subsystem, and inadvertent activation of the subsystem function [22].

Common-cause analysis

Common-cause analyses are be performed on reliability and safety critical items in to identify the root cause of failures that have a potential to negate failure tolerance levels. They can be accomplished as part of FMEA/FMECA or FTA [22].

Although multiple risk assessment methods are well established in the literature, in practice they are not always applied correctly or at the appropriate stage to realise their intended benefits. Studies highlight that risk assessment is frequently conducted too late in the lifecycle, or is performed superficially, reducing its ability to influence architectural decisions and design trade-offs in the early phases[23].

2.4. Model Based Systems engineering

2.4.1. Systems engineering and shift to MBSE

Systems Engineering is defined by INCOSE as a "transdisciplinary and integrative approach to enable the successful realization, use, and retirement of engineered systems, using systems principles and concepts, and scientific, technological, and management methods"[8]. It addresses and satisfies the needs of stakeholders by developing cost and time-effective solutions.

In the context of space missions, the systems engineering process is quite disciplined and highly focused on mission safety. A space systems engineer coordinates and plays a key role in various tasks including development of the concept of operations (ConOps) and resulting system architecture, defining and allocating requirements, evaluating design trade-off, balancing technical risk between systems, assessing interfaces, and providing oversight of verification and validation activities [9].

Applying systems engineering practices in space missions has helped reduce risks, enhance collaboration, optimize costs and schedules, and improve product quality by providing a structured framework for managing complex problems. However, the disadvantages of these practices can include the high cost of implementing robust systems engineering processes to address the complex nature of space missions. Also, there exists the potential for over-engineering and challenges in integrating innovations into established processes, particularly in lower-cost, risk-tolerant missions.

In the past, systems engineering relied heavily on traditional methodologies to develop complex systems using documents and spreadsheets to manage information and requirements [24]. This required manual updates and posed synchronization challenges, leading to excessive rework and long feedback loops. This Document Based Systems Engineering (DBSE) has been replaced by modern, smart practices which also tackle the disadvantages of systems engineering in space missions. One such practice is Model Based Systems Engineering (MBSE), which has gained considerable traction in recent years.

2.4.2. What is MBSE?

MBSE is an approach that focuses on achieving systems engineering objectives and goals. As defined by INCOSE, "Model-based systems engineering (MBSE) is 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" [25]. The systems' MBSE model aims to understand, communicate, explain, and innovate in the user's interest [26].

The life cycle of a system consists of various activities at different stages that provide the support necessary for efficient modelling. Modelling supports the systems activities, and system engineering models drive these activities among the life cycle process [27].

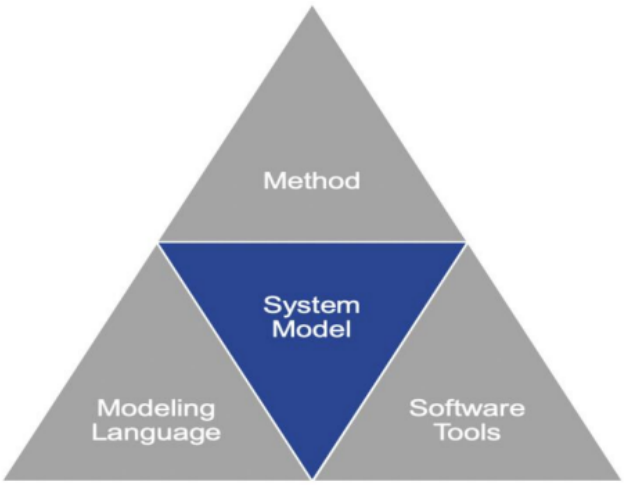


Figure 2.1: MBSE Pyramid [27]

An appropriate MBSE methodology embraces some important elements: a method, a modelling language, and tools essential for successfully executing this processes [28]. As shown in Figure 2.1, these elements provide an overview of the overall MBSE and indicate its capabilities that create the system model [29]. Each of these elements are decribed below:

Method

This indicates the collection of design techniques, systems engineering processes, and approaches used for developing complex system architecture models[29]. It provides details about tasks to be performed on the different elements and how each task and operation occur at different stages. However, each modelling method contains a series of functions taking place in sequence, and the collection of design techniques can be viewed as a process itself [29]. A modelling methodology shall provide all relevant parameters through the necessary sequential stages of the system's life cycle; however, it also depends on the systems engineering team to determine the appropriate parameters and sequence [30].

Modeling Language

This element is used to express the relationship between actions and elements of a system. It is neither a technique nor a framework and is considered a key enabler for MBSE [28]. The language defines what to communicate without determining how a logical sequence of activities and tasks occurs in the system [29]. Therefore, it provides the medium of expression and categorises such relationships and components in the graphical form necessary for the modelling. There are various modelling languages available, implemented through specified standards[30].

Tool

Modeling tools are employed to streamline critical tasks and strengthen the architectural capabilities of a methodology. Each modeling approach illustrates how a system's activities are performed. Through these models, the system architecture can

visually represent various actions within the system, along with their dependencies and both internal and external interfaces. Tools implement the graphical characteristics of modeling methods, and it also ensures compliance with standards and allows the system architecture to be updated when any modification has occurred [30].

2.4.3. Advantages of MBSE

A major advantage of MBSE is credited to its central repository that contains the collection of all system information from which access to any required information is possible. This allows for easy formation and identification of interconnections between the model elements. It also handles any model updates efficiently by ensuring automatic propagation of design changes, performing consistency checks, and error detection. MBSE promises to be a more rigorous and effective means of developing complex systems [31].

Several significant benefits of employing MBSE can be listed as follows:

- **Reduction of Cost and timeline:** Clear definition of stakeholder needs and design assessments help capture risks and uncertainties accurately. Using this, appropriate design decisions are made early on, based on design implications, risks and dependencies. Also, issues are discovered early through the enforced consistency and relationship visibility [32]. This helps reduce the cost and duration of the expensive integration and test phase.
- **Ease of communication among diverse stakeholders:** MBSE helps develop an unambiguous system description which is precise and can be evaluated for consistency, correctness, and completeness [33]. This helps discuss problems and potential solutions clearly, perform holistic trade-off studies and implement risk mitigation steps at the right stage by the appropriate stakeholders.
- **Knowledge capture:** By providing effective means for capturing, assimilating, and retaining design decisions and details in an electronic repository [33]. This information can be reused and updated in the later stages and is retained by the model.
- **Decision-Making:** Since changes in MBSE tools are instantly reflected across the entire design, system design trade-offs based on a set of requirements can be conducted quickly to aid decision-making. Design risks and cost can also be incorporated into the model to enhance the decision-making process.
- **Error Checking and Data Verification:** MBSE tools can validate the model consistency and conformance to standards [33]. Moreover, by reviewing the model execution, developers can affirm that the right system is being built[32]. The identification of inconsistencies in model indicates design flaws. To eliminate them, design parameters can be modified and quick data verification could be performed. Trade studies could include more variables or be more detailed to find precise solutions or reduce errors [33].
- **Documentation and Reuse:** Templates for automatically generating formatted documents from the central repository can be developed using MBSE tools. The documents generated based on the current design parameters can be referred to

for another similar project. Additionally, from a single repository, multiple consistent views can be produced to communicate and analyse designs in different contexts.

These advantages collectively illustrate an important application of MBSE — risk assessment. Identifying and assessing risks, as well as developing appropriate mitigation strategies, are integral components of the MBSE process. This approach supports informed design decisions, reduces project costs, and helps maintain realistic schedules for complex projects.

2.4.4. MBSE and Fault detection

MBSE has emerged as a powerful methodology for handling the increasing complexity of space systems. It enables the creation of structured, graphical models to represent system architectures, requirements, and behaviors from the early stages to implementation. Several MBSE tools and frameworks have been explored in the context of safety and fault analysis.

The COMPASS toolset [34] developed by ESA is a relevant example that combines modelling in the ADL language with formal model-checking capabilities to support system-level verification, fault analysis and design assurance. However, its use and access are largely restricted to formal logic experts and require steep learning curves and complete model definitions.

The SafeSysE project [35] is a similar project that explores the integration of SysML with safety analysis methods such as FMEA and FTA through model transformations. In this approach, safety engineers define metamodels to extend the usability of the SysML models for safety analysis.

An additional tool, MADe (Model-based Analysis and Design Environment) [36] widely used in safety engineering literature and recommended by NASA, supports fault modeling, functional FMEA generation and diagnostic reasoning using a model-based approach. While MADe provides a structure to the safety logic, it is proprietary and may not integrate directly with open-source MBSE platforms like Capella.

In the automotive and aerospace domains, the SysML Safety Profile [37] was developed in line with ISO 26262 and ARP4761 standards. These approaches propose adding safety properties as tagged values or stereotypes and rely on manual rule-checking or tool-augmented validation through plug-ins. The RiskML framework [38] and PRISMA further explore extending MBSE to support quantitative risk modelling.

Studies such as [39] also demonstrate transformations from SysML to fault trees, while [40] proposes MBSE-assisted FMEA methods that balance manual design input with automated risk extraction. Despite these advancements, a gap remains in tools that offer lightweight, explainable, and early-phase fault detection integrated with MBSE modelling, especially suited for conceptual design or resource-limited settings like NEBULA-Xplorer.

2.5. Identified gaps & need for a framework

Building on the state of art, it becomes clear that while MBSE, fault detection and risk assessment have individually advanced, their intersection in the context of early design stages remains underdeveloped. The literature consistently emphasises the potential benefits of integrating failure analysis earlier in the lifecycle, yet also illustrates that existing practices are either too resource-intensive or insufficiently adapted for preliminary modelling.

This indicates a persistent disconnect between what current methods offer and what early-stage system engineering actually requires. To make this distinction explicit, the following key research gaps are identified:

- **Focus on late-stage methods:** Despite increasing interest in early-stage fault analysis, most existing fault detection and management techniques are still concentrated on late phases of system development, such as testing, simulation, or operational telemetry. This means that their value for supporting system modeling and architectural reasoning in the initial design stages remains limited.
- **Restricted applicability of hybrid approaches:** Although hybrid approaches that combine rule-based reasoning with formal transformations or data-driven models have been proposed, their demonstration is typically restricted to verification and validation campaigns. As a result, their potential to assist during initial architecture definition has not been sufficiently explored.
- **Limitations of existing MBSE safety extensions:** While MBSE offers a structured platform for modeling complex system architectures, most existing safety-oriented extensions such as COMPASS and SafeSysE, depend on advanced formal methods, fully matured models, or domain-specific toolchains. This reliance on heavy infrastructure makes them unsuitable for contexts with limited resources and experience, such as student-led missions.
- **Lack of automated and explainable analysis:** Few frameworks provide automated and explainable fault detection that works directly with MBSE outputs. In particular, existing methods do not adequately address the detection of architectural, functional, or interface-level fault patterns during the early design stages.
- **Accessibility and scalability issues in current tools:** Current tools do not provide integrated features for inspecting early-stage design models in a way that is both scalable across different system complexities and accessible to teams without extensive expertise in formal verification. This creates a gap for lightweight solutions that can adapt to varied mission needs.
- **Absence of immediate design feedback:** Existing practices do not seamlessly integrate fault detection into early-stage modelling workflows. There is a lack of mechanisms that allow design teams to receive immediate, actionable feedback on faults during architectural decision-making, which limits the proactive management of risks in the early lifecycle.

- **Limitations of early-stage risk assessment methods:** Traditional risk assessment techniques such as FMEA and FMECA are widely used but typically depend on detailed design information (component reliability data, failure rates, or mature hardware models). At early stages, such data is unavailable, which forces teams to rely on qualitative judgement.

3

Framework development

This chapter outlines a general, tool-agnostic framework for early Fault detection, supporting technical risk assessment during Phases A-B. Building upon the motivation and gaps identified in the previous section, the discussion emphasises the engineering design of the framework, its development process, target objectives, proposed architecture, and potential modes of implementation in a broad context, independent of any specific mission or toolchain.

3.1. Methodology

This section describes the methodological process followed to develop the proposed framework. Risk assessment and System design are both iterative approaches, and hence, the development of this framework also followed an iterative approach. Starting from the identified problem statement and early-phase constraints, requirements were first generated, followed by an MBSE-centred solution to make every design choice traceable to a rationale. Framework development (Figure 3.1) followed a tool and mission-agnostic approach and was divided into the following phases:

1. **Objective and Requirements Definition:**
   From the problem definition and literature-based gaps, a concise set of framework requirements was derived, specifying: (i) the types of issues that must be detectable in early phases, (ii) the form of outputs required to support engineering decisions, and (iii) quality attributes expected of the solution. Each requirement was treated as an acceptance criterion to enable later verification.
2. **MBSE Model development strategy:**
   To support early fault detection without depending on the availability of a mature design, a minimal model content was specified. This MBSE Model concept defines the elements and relations that any MBSE model should provide: Requirements, Functions, Components, Interfaces/Exchanges, and Allocations. In addition, failure-relevant attributes were designated to the model elements. The intent is to enable analysis on developing models while remaining independent of any specific tool.

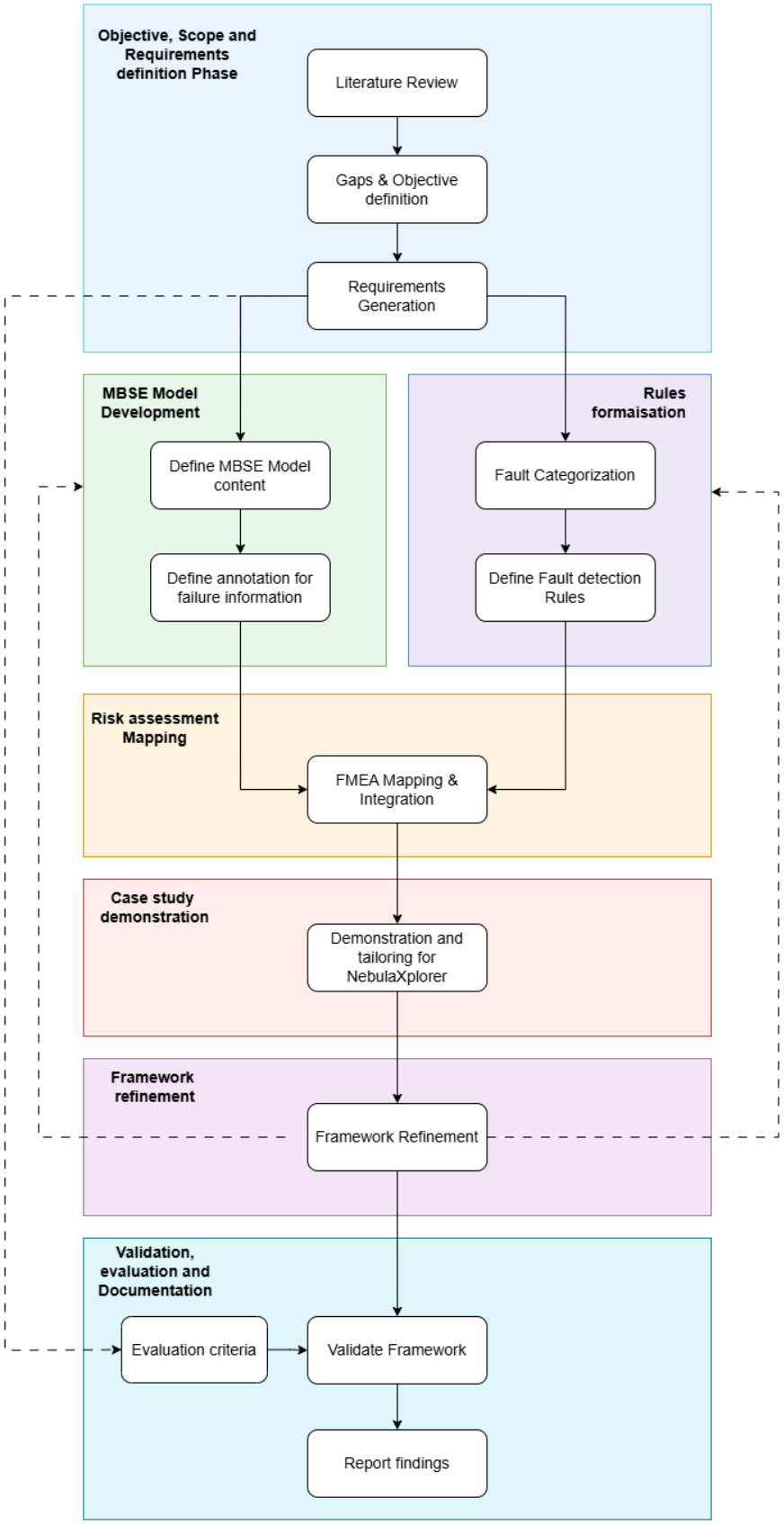


Figure 3.1: Framework Development Methodology

3. **Rules discovery and formalisation:**
   Concurrently with the modelling strategy, a catalogue of detection rules was developed. Patterns were drawn from recurring early-phase concerns (e.g., unallocated critical functions, single points of failure, and missing requirement traces). Each rule was expressed as a checkable statement comprising preconditions, detection logic, and required evidence in the finding.
4. **Risk Assessment integration:**
   To ensure that findings become actionable in design reviews, a fixed mapping from detection findings to FMEA entries was defined. The mapping specifies which fields can be populated directly from the model and rule evidence and which fields remain subject to engineering judgement.
5. **Case application and tailoring:**
   The Framework was applied to a case study: NebulaXplorer mission and tailored to demonstrate its feasibility. Case-dependent modifications were noted.
6. **Framework refinement:**
   Observations from the application step were used to refine the baseline. Revisions focused on clarifying assumptions, improving rule wording to reduce ambiguities, and, where necessary, updating the MBSE model features.
7. **Evaluation plan:**
   Evaluation was planned along two complementary tracks: (i) requirement verification, whereby each framework requirement was revisited and (ii) method assessment using a recognised evaluation framework.

3.2. Requirements

This section specifies the Functional (what the framework shall do) and Non-functional (how it shall behave) requirements to be usable in early phases. These have been derived from the research gaps discussed in Section 2.5. Each requirement is denoted with a unique ID, FR-XX for Functional and NFR-YY for non-functional requirements. They are illustrated in Table 3.1 and Table 3.2

Table 3.1: Framework functional requirements

ID	Name	Description (The framework shall..)
FR-01	Fault detection	Detect defined categories of early-phase (A-B) faults from MBSE models, including allocation issues, interface mismatches, and redundancy gaps.
FR-02	MBSE Model	Operate on MBSE model representations that provide, at minimum, the following architectural information: functions, components, and their allocated relationships; and redundancy attributes where applicable.

Continued on next page

Table 3.1: Framework functional requirements (Continued)

FR-03	Findings dataset	Generate a structured findings dataset that lists each detected fault, affected element(s) and fault types.
FR-04	Failure Analysis	Analyse the effect of potential function or component failures on higher-level system capabilities, given the current architectural design.
FR-05	Risk Assessment	Provide an output dataset suitable for downstream risk assessment, including at a minimum: function/component, failure mode, effect and associated severity and likelihood.

Table 3.2: Framework non-functional requirements

ID	Name	Description (The framework shall..)
NFR-01	Terminology	Use terminology consistent with widely adopted MBSE practices (e.g., allocation, interface, redundancy) to ensure interpretability by system engineers.
NFR-02	Explainability	Provide transparent reasoning for the results.
NFR-03	Determinism	Produce identical outputs, given identical inputs and configurations.
NFR-04	Robustness	Degrade gracefully on incomplete input while still analysing mandatory fields.
NFR-05	Usability	Allow adaptation of detection rules or thresholds to different mission contexts.
NFR-06	Traceability	Ensure traceability of results to the originating model element.

3.3. Proposed Framework Architecture

At its core, the framework operates on a minimal MBSE model that provides just enough detail to enable analysis without requiring a fully matured architecture. This model forms the foundation of the framework, and its specific content is described in Section 3.4. The framework applies four interdependent layers (Figure 3.2), using the model as a foundation. Each layer addresses a distinct step in the transition from system requirements to actionable risk insights.

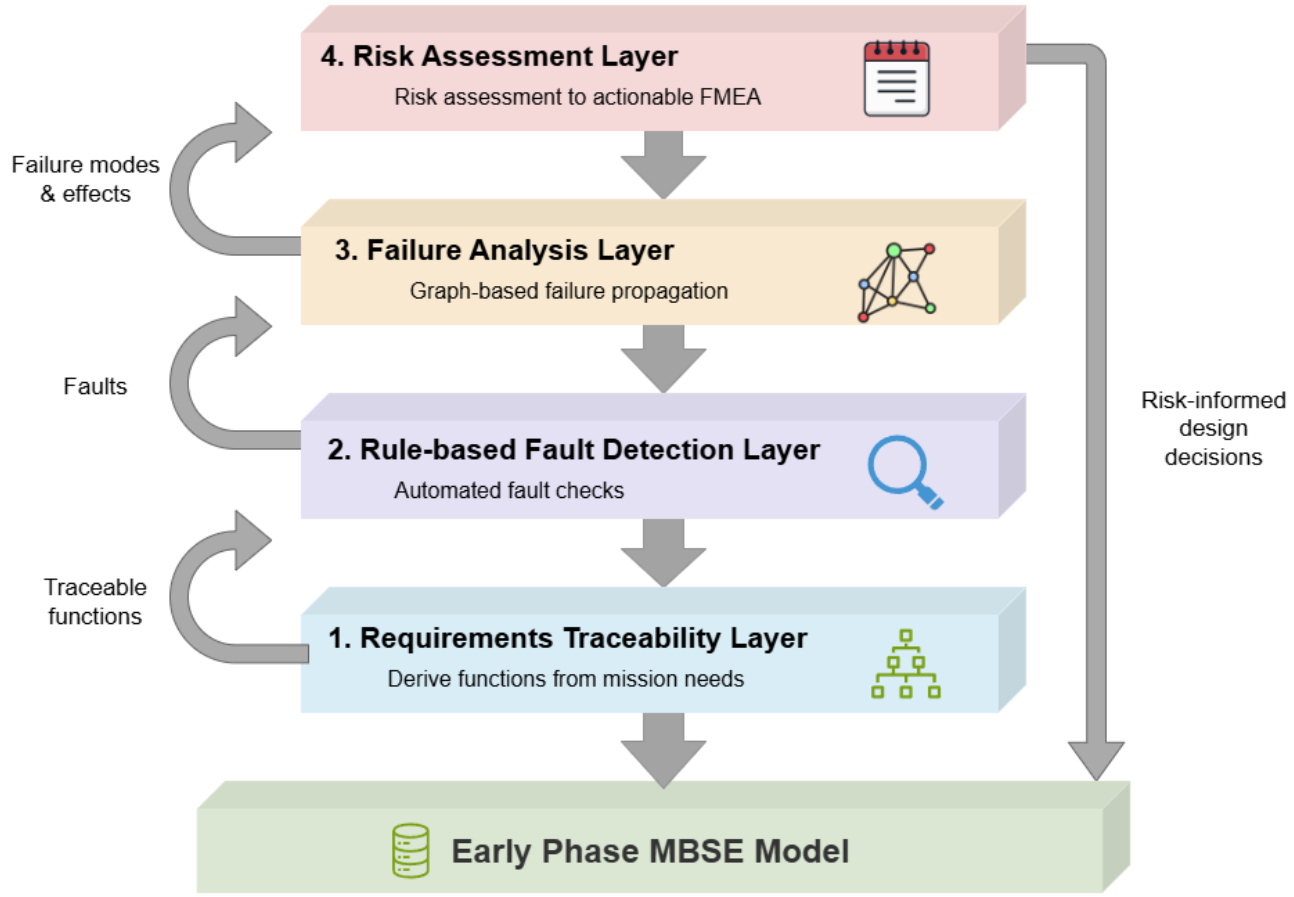


Figure 3.2: Framework Architecture

Requirements Traceability Layer

This layer requires that all system functions be explicitly derived from higher-level functional requirements. This ensures that no "orphan" functions remain unconnected to mission needs, while also providing a baseline for later analysis. Traceability at this stage is treated as a primary rule: every function should have a requirement origin, and every functional requirement should be realised by at least one function.

Rule-based dection Layer

A set of formalised rules is executed on the MBSE model to detect early-phase issues. These include, for instance, consistency between power demand and supply, environmental specifications, and requirement–function allocation checks. Each rule is expressed as preconditions, detection logic, and supporting evidence, ensuring transparency and reproducibility.

Failure Analysis Layer

Interprets the MBSE model as a graph of functions and their interconnections. Failures are represented as degraded or removed nodes, and their effects are propagated through functional chains to assess the impact on higher-level capabilities. The details of this analysis are explained in Section 3.6.

Risk Assessment Integration Layer

Translates detection findings and failure propagation outcomes into a functional FMEA structure according to ECSS standards [22]. Citing the limitations of existing risk assessment for applicability to early phases, **Functional FMEA** was chosen over a hardware approach for the following reasons:

- Captures credible points of failure independent of the final design and integration choices.
- Feeds the technical risk register before part-reliability data is determined.

Model evidence pre-populates fields such as function/component, failure mode, effect, severity and design recommendation, while likelihood remains subject to engineering judgement. In this way, early fault detection results directly contribute to risk-informed decision-making.

3.4. MBSE Model Specification

The framework relies on a minimal MBSE model that captures essential system information. To define this foundation, the Systems Engineering Body of Knowledge (SEBoK) was adopted as a reference [41]. SEBoK identifies four essential system views that together provide a complete representation of a system throughout its lifecycle:

- Requirements View: Capturing stakeholder goals, system-level capabilities, and success criteria.
- Functional/Behavioural View: Describing what the system must do to meet requirements, including functional breakdown and state-dependent behaviours.
- Structural View: Representing the system's architecture in terms of components and their relationships.
- Parametric View: Defining constraints, properties, and quantitative attributes that govern system performance and behaviour.

These views are independent of any particular modelling language or tool and form a widely accepted foundation for system modelling. From these views, the framework adopts a subset of model elements that can be realistically defined in Phases A-B:

- **Requirements (Requirements view):** High-level mission and system requirements, which provide the source for all subsequent functions and structure. They are the starting point for establishing traceability.
- **Functions and exchanges (Functional View):** Represented in a Function Breakdown Diagram (FBD), giving a hierarchical decomposition of system behaviour from the functional requirements. Additionally, exchanges are defined between functions to represent the flow of data, control, or resources. These exchanges are grouped into **functional chains** that describe the main system capabilities. Capturing the functional chains enables the framework to reason about end-to-end capability realisation and to analyse how a particular function failure may disrupt higher-level mission objectives.

- **Components and allocations (Structural View):** Abstract physical building blocks, serving as allocation targets for the established functions. These elements represent the logical and eventual physical breakdown (eg: Subsystems, Assemblies) of the system. They are primarily defined using a Block Definition Diagram (BDD), with their internal composition detailed in an Internal Block Diagram (IBD).
- **Failure-relevant Properties (Parametric View):** Attributes such as redundancy, power requirements, and radiation tolerance, which enable the framework to perform rule-based checks and support risk assessment.

The model content was intentionally kept minimal to avoid dependence on detailed design information that is typically unavailable in Phases A-B, while still sufficient to capture and highlight early-phase design and architectural faults. This structured specification forms the foundational layer for the subsequent framework capabilities: guaranteeing that traceability, rule-based detection, failure analysis, and risk assessment are all based on consistent system data.

3.5. Rule Formalisation

The second layer of the framework is a rule-based detection that enables early identification of architectural issues in Phases A-B. Rules were derived from recurring early-phase concerns reported in literature and design practice, and each was expressed in a formal structure to ensure transparency and reproducibility.

A rule is defined using three elements:

- Preconditions: Model elements and attributes required for the rule to be applicable.
- Detection Logic: Conditional statement that evaluates the model against the intended pattern.
- Supporting Evidence: Specific model elements or attributes that triggered the finding, ensuring the result is traceable and interpretable.

The rules applied in this framework are grouped into three broad categories:

1. **Traceability Rules:** In MBSE practice, requirements, functions, and components are expected to be consistently linked to avoid elements existing in isolation. In early design phases, these traces could be incomplete or omitted. For this reason, the framework treats traceability not as an assumption but as an explicit rule to be verified.

   The two primary checks are:

   - **Requirement-Function Traceability:** Every function must be derived from at least one requirement (Table 3.5)
   - **Function-Component Allocation:** Every function must be allocated to a component that realises it.

Table 3.3: *Illustrative example*: Requirement-Function Traceability rule

Preconditions	Function *F* exists, Requirement *R* exists
Detection Logic	If *F* has no «deriveReq» (or equivalent) trace link from any *R*, trigger finding.
Evidence	The identifier of the orphaned function is reported.

2. **Consistency Rules:** Consistency rules check that quantitative attributes captured in the model do not conflict. Typical examples include resource balance checks and redundancy definitions.

Table 3.4: *Illustrative example*: Power Balance rule

Preconditions	Function *F* is allocated to Component *C*; both have defined power attributes
Detection Logic	If *C.PowerRequired* > *F.PowerAvailable*, trigger finding
Evidence	"Component C1 requires 15 W, but Function F1 generates only 10 W."

3. **Attribute Completeness Rules:** In early phases, many design attributes that are important for risk assessment may not yet be known or specified. Rather than assuming completeness, the framework explicitly checks for the presence of such attributes and flags missing data. This allows the model to highlight 'known-unknowns' parameters that will need to be defined later to enable a comprehensive analysis.

Table 3.5: *Illustrative example*: Radiation Tolerance rule

Preconditions	Component *C* exists in the model
Detection Logic	If *C* does not have a Radiation Tolerance attribute defined, trigger finding
Evidence	"Component C2 has no defined radiation tolerance."

By capturing rules in this structured format, the framework enables systematic detection of incomplete, inconsistent, or untraceable design elements. These findings are then passed forward into the failure analysis and risk assessment layers.

3.6. Failure Analysis

The failure analysis layer builds on the traceability and rule-based checks defined earlier. While those layers ensure that the MBSE model is consistent and complete, this layer investigates the structural robustness of the functional architecture itself. The idea is to move beyond verifying attributes and links, and also ask: *what happens if a function or exchange fails?*

To answer this, the functional block diagram is represented as a directed graph of functions and exchanges, enabling the use of graph theory-based methods to reason about failure propagation and bottlenecks [42]. In this way, the framework introduces an analysis methodology capable of revealing architectural weaknesses during early lifecycle phases, well before detailed design specifications are finalised.

3.6.1. Graph representation of Functional Architecture

To enable automated analysis, the functional block diagram is abstracted into a **directed graph**, where edges have a defined direction and are represented as:

$$G = (V, E)$$

- **Nodes V:** Functions (F)
- **Edges E:** Functional Exchanges (FE)
- **Attributes:** Each FE carries a classification attribute(type:Resource/Core/Utility) while each F carries redundancy/performance properties.

Adjacency representation

The abstract graph, G = (V, E) is transformed into computational representations that enable algorithmic reasoning:

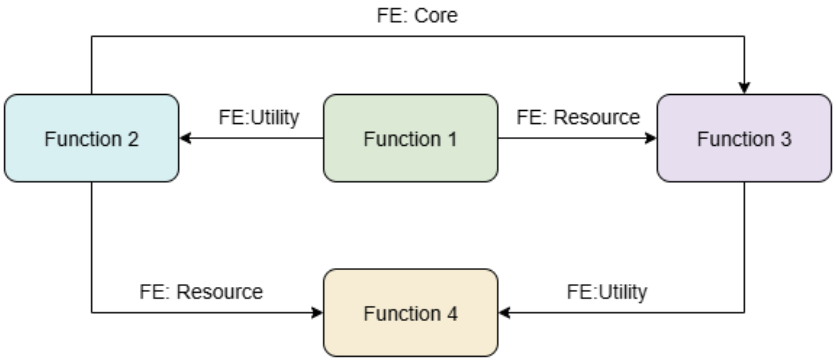
- **Adjacency list:** For every function $u \in V$, its outgoing neighbors $\{ v \in V \mid (u, v) \in E \}$ are stored explicitly in an adjacency list, denoted by:

$$\text{adj}_{\text{out}}(u) = \{v \in V \mid (u, v) \in E\}$$

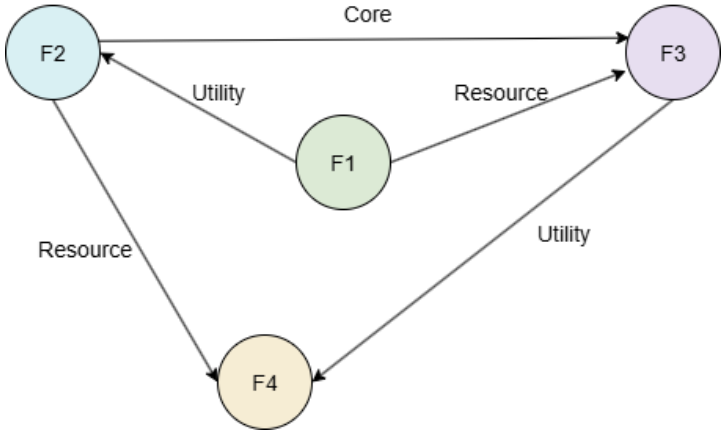
- **Adjacency matrix:** The adjacency matrix A is a binary $n \times n$ matrix defined as

$$A_{ij} = \begin{cases} 1 & \text{if a functional exchange } (i \to j) \in E, \\ 0 & \text{otherwise.} \end{cases}$$

Figure 3.3a shows an example of functional block diagram (FBD) fragment. The same diagram can be abstracted into a directed graph, shown in Figure 3.3b, where functions are represented as nodes and functional exchanges as edges annotated with their type.



(a) Functional Block Diagram (FBD) fragment.



(b) Graph abstraction of the same fragment.

Figure 3.3: From model to graph abstraction: (a) FBD fragment, (b) equivalent directed graph representation.

For illustration, the adjacency matrix A of the example graph in Figure 3.3 is shown below. An entry $A_{ij} = 1$ indicates a functional exchange $F_i \rightarrow F_j$.

$$A = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

This representation is particularly useful for matrix-based metrics (e.g., path counting, centrality). Well-established graph metrics can be applied to these for failure analysis and redundancy evaluation.

3.6.2. Functional Chains and Neighbourhoods

A Functional Chain in MBSE represents a particular end-to-end capability of the system. It typically describes how a mission objective is realised through a sequence of functions and exchanges. The realisation of functions depends not only on the primary functions involved but also on the functions and exchanges that interact with them.

To capture this, the framework analyses each function chain along with its one-hop neighbourhood:

- Primary nodes: The set of functions directly involved in the chain.
- Neighbour nodes: The functions that directly interact with a primary node via a functional exchange.
- Internal edges: Functional exchanges linking two primaries.
- Boundary edges: Functional exchanges, not part of the chain but flowing in or out of primaries.

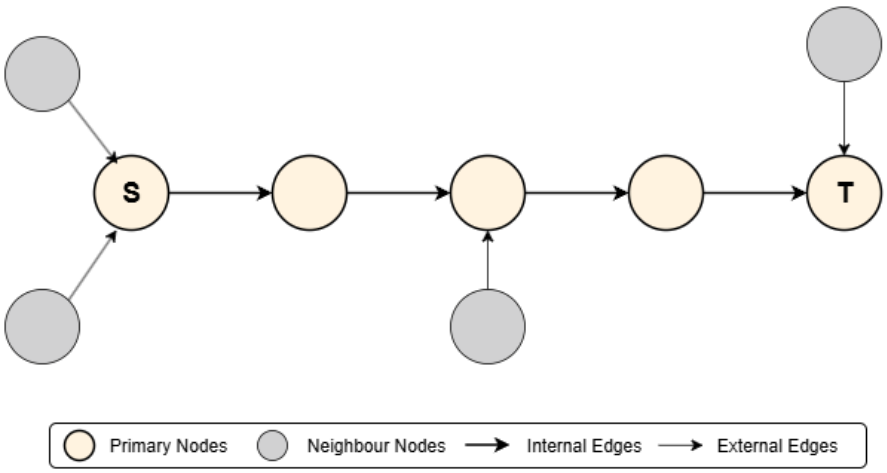


Figure 3.4: Functional chain and one-hop neighbourhood

Endpoints of the chain (Figure 3.4) are denoted as **source S** and **sink T** and are inferred automatically. The rule used is that a source has no inbound exchanges from within the chain, and a sink has no outbound exchanges.

Once the functional chain and its neighbourhood are identified, their structure can also be expressed in a matrix form. The adjacency matrix introduced earlier can be restricted to just the primary nodes of the chain, producing the submatrix:

$$A_c = A[V_c, V_c]$$

where A is the system adjacency matrix and V_c the set of chain nodes. Interactions between the chain and its neighbourhood are described by boundary submatrices:

- Outgoing boundary matrix:

$$B_{C\to N}(u,v) = 1 \quad \text{if} \quad u \in V_C, v \in V \setminus V_C, (u,v) \in E$$

- Incoming boundary matrix:

$$B_{N\to C}(v,u) = 1 \quad \text{if} \quad v \in V \setminus V_C, u \in V_c, (v,u) \in E$$

Together, these boundary matrices capture the external dependencies of the chain, indicating which external functions supply inputs to the chain and which downstream functions depend on its outputs. This representation allows the analysis to highlight two types of issues in architectural design:

1. **Internal fragility:** Potential weak links or single points of failure in the chain
2. **External dependency:** Outside exchanges or functions whose failure could disrupt the capability realisation.

3.6.3. Extended Graph Representation

The adjacency matrix representation introduced in section 3.6.1 can effectively capture the structural connectivity of the functional architecture. To extend the analysis beyond architectural weaknesses and assess the consequences of a failure, additional attributes are introduced:

- **Node capacities:**
  Each function $u \in V$ is associated with a redundancy capacity c(u). This value indicates the number of ways in which a function can be realised, which in turn represents the tolerance of a function to failure. For example, a function realised by two redundant units would have $c(u) = 2$. These values can be represented as a diagonal capacity matrix:

$$C = \text{diag}\big(c(u_1), c(u_2), \ldots, c(u_{|V|})\big)$$

- **Exchange classifications:**
  Each functional exchange $(u, v) \in E$ carries a qualitative label, t and is used to differentiate the impact of the loss of a particular edge on a node.

$$t : E \rightarrow \{Resource, Core, Utility\}$$

The three matrices (A, C, t) together form the basis for all the failure metric calculations introduced in the following sections:

- A captures structure (whether dependency exists)
- C captures redundancy (failure tolerance)
- t captures criticality (type of dependency)

3.6.4. Method and Metrics

The different metrics and the steps to calculate the same are presented in this section.

Connectivity and Reachability

This analysis evaluates whether a functional chain can survive a single functional failure. The central question it evaluates is whether the path from the source S to the sink T remains connected when individual intermediate functions are removed.

Two complementary tests are applied on the functional chain:

1. **Baseline Connectivity**
   An initial connectivity test confirms that a directed path exists between S and T in the intact graph $G = (V, E)$.
2. **Single-Node Path Survivability**
   To assess the functional chain vulnerability, each intermediate function, u is removed in turn and for every modified graph $G - \{u\}$, reachability is re-assessed. If removal of u disconnects S from T, then u is identified as a **critical node** or a single point of failure.

The output of this analysis is the set of Single Point of Failures (SPOFs). This metric forms the foundation of internal fragility assessment. A chain dominated by SPOFs reflects a weak architecture with limited tolerance to single-node failures, whereas an empty SPOF matrix indicates resilience.

Neighbour dependencies

The analysis then evaluates whether the integrity of the functional chain depends on the functions in the one-hop neighbourhood. The question is whether the failure of a neighbour function $N \notin V_c$ can indirectly break the chain by isolating one or more primary functions. To assess this, the following checks are performed:

1. **Gather required inputs per primary:**
   For each primary $P \in V_c$, the inbound exchanges, grouped by name and type are listed.
2. **Uniqueness by type:**
   For each required input(name, type) of P, collect its corresponding nodes. If the provider set reduces to N for that (name, type) then P is uniquely dependent on neighbour N for that typed input.
3. **Cascade removal and reachability:**
   If N fails, the uniquely dependent P is functionally isolated with respect to that input. Remove N, P and re-assess reachability from S to T. If the path is broken, classify N as a supporting-critical neighbour for that chain.

Redundancy and Min-Cut Analysis

This test evaluates the structural redundancy of a functional chain, providing a quantitative measure of its tolerance against single functional failures. It addresses the question: What is the minimum number of simultaneous function failures required to lose a capability?

The redundancy analysis is formulated as a **minimum vertex cut** problem. A vertex cut is the smallest set of nodes whose removal eliminates all directed paths from S to T. To compute this, the capacity matrix introduced earlier is used as follows:

1. **Node transformation:**
   Each function $u \in V_c$ is split into two nodes (u_{in}, u_{out}) connected by an internal edge (u_{in}, u_{out}).

2. **Capacity assignment:**
   The internal edge (u_{in}, u_{out}) is given capacity of 1, representing the cost of removing that function. Functional exchanges are assigned infinite capacity, reflecting the assumption that edges do not fail at this point.
3. **Algorithm execution:**
   A standard max-flow/min-cut algorithm is applied to the transformed graph. By the Max-Flow Min-Cut Theorem, the maximum flow value equals the size of the minimum cut set.

The resulting min-cut value or **Failure Resilience Score (FRS)** provides a measure of the chain's structural resilience:

- min-cut = 1: no redundancy exists, a single function failure interrupts capability realisation.
- min-cut $\geq$ 2: multiple disjoint paths exist and can tolerate at least one failure before loss of capability.

Global Centrality

This analysis moves beyond individual functional chains to identify **global single points of failure (SPOFs)** that act as critical bottlenecks across the entire functional architecture. The central question is: Which functions are global bottlenecks and lie on a disproportionately high number of shortest paths?

To answer this, the **betweenness centrality** ($C_B(u)$) is calculated for every function $u \in V$. It is a measure of how often a function lies on the shortest path between any two other functions s and t:

$$C_B(u) = \sum_{s \neq u \neq t \in V} \frac{\sigma_{st}(u)}{\sigma_{st}}$$

Where:
σ_{st} is the total number of shortest paths between two functions s and t.
$\sigma_{st}(u)$ is the number of those shortest paths that pass through function u.

The computation is performed using a standard betweenness centrality algorithm on the adjacency matrix of the graph, G. The output is a ranked list of functions by their betweenness score. Functions with high centrality values are critical bottlenecks, as failures here risk cascading effects across multiple chains. These nodes are candidates for redundancy, architectural decoupling, or monitoring.

Coupling analysis

The coupling risks of every function are evaluated using the fan-out and fan-in metrics. It helps answer the question: Which functions are vulnerable due to excessive incoming or outgoing exchanges?

For each function $u \in V$:

- **Fan-in:** number of distinct exchanges feeding into v, i.e. $deg^-(v)$
- **Fan-out:** number of distinct exchanges relying on v, i.e. $deg^+(v)$

These values are computed directly from the adjacency list of the graph discussed in the previous section. The output highlights the top-ranked functions by fan-in and fan-out:

- High fan-in nodes are **integration hotspots**. Failure here may affect multiple upstream inputs.
- High fan-out nodes are **cascade risks**. Failure here propagates downstream to many dependents.

Both patterns indicate functions that warrant special design considerations, such as functional splitting or redundancy.

3.7. Risk Assessment

The analyses presented in the previous section yield various metrics. These results are not standalone numerical values. Instead, they provide the building blocks for a structured risk assessment. Each metric maps directly onto the concepts traditionally used in Failure Mode and Effects Analysis (FMEA):

- **Function Failure Analysis**: Generates candidate functional failure modes (e.g., "Loss of Function X").
- **Chain Connectivity & Survivability Tests:** Captures the failure effects, i.e. the loss of end-to-end mission capability.
- **Min-Cut Value:** Provides a measure of resilience to failure.
- **Fan-In / Fan-Out Coupling:** Indicates cascading risk.

The Functional FMEA is automatically derived from the MBSE model: each row corresponds to the failure of a logical function, with its effects, severity, and design recommendations filled in by analysis. Importantly, the framework does not estimate the likelihood of occurrence. In early design phases, reliable failure rate data is unavailable. Likelihood is explicitly left blank, to be populated manually by engineers using judgment or historical reliability data.

3.7.1. Severity assessment

A central aspect of the Functional FMEA is the evaluation of severity. In this framework, severity is determined from two complementary sources:

1. **Functional exchange tag:** Each functional exchange is classified according to its mission relevance and maps to a consequence score as shown in Table 3.6

Table 3.6: Functional Exchange and consequence scores.

FE Tag	Description	Consequence Score
RESOURCE	Irreplaceable system resource	P2 / 4 (Catastrophic)
CORE	Primary mission data or capability enabler (e.g., science data, pointing reference)	P3 / 3 (Critical)
UTILITY	Supporting, safety, or monitoring data (e.g., heater commands, confirmations)	P4 / 2 (Major)

2. **Structural integrity check:** Independent of tagging, the structural analysis identifies whether a single functional failure severs the $S \rightarrow T$ chain. If so, this structural break overrides functional tagging and assigns the highest severity.

   The final severity is therefore assigned according to:

$$\text{Final Severity} = \max\left(P_{\text{Structural}},\ P_{\text{FE Tag}}\right)$$

   Where,
   $P_{\text{Structural}}$ denotes severity derived from connectivity tests
   $P_{\text{FE Tag}}$ is derived from mission relevance tagging

   The resulting severity classes (P1–P5) are defined in Table 3.7.

Table 3.7: Severity classification scheme (aligned with ECSS categories).

Priority (P)	Condition Met	Severed FE Tag	Score	Consequence
P1	Structural Break: $S \rightarrow T$ path severed	Any	4	Catastrophic
P2	Functional Isolation: Path survives, but RESOURCE lost	RESOURCE	4	Catastrophic (Resource Loss Override)
P3	Functional Isolation: Path survives, but CORE lost	CORE	3	Critical (Mission Objective Loss)
P4	Functional Isolation: Path survives, but UTILITY lost	UTILITY	2	Major (Degradation)
P5	No Isolation: Neighbor removed, no unique input lost	N/A	1	Minor (Negligible Impact)

This severity scheme ensures that the most disruptive failures are always ranked above functional degradations. In this way, the framework aligns directly with ECSS severity categories mentioned in [22].

3.7.2. Aggregation of results

The analyses described in section 3.6 are conducted on an individual chain basis. A particular system may have multiple capabilities and hence, may be defined by various chains. To provide a single, system-level view, results from each chain have to be aggregated to provide an overall assessment.

For every function, u a single, comprehensive **Function Risk Profile (FRP)** is consolidated as follows:

1. **Primary Ranking Criteria (Consequence):**
    - **Worst-Case Severity**$(P^*(u))$**:** It is the maximum severity assigned to u across all functional chains, measuring the highest impact of its failure on any mission capability.
    $$P^*(u) = max_{chains\ c} P_c(u)$$
2. **Secondary Ranking Criteria (Exposure and Vulnerability)**: They sort the severity of functions that lie on the same band.
    - **Chain coverage** $N_c(u)$**:** This measures the involvement of the function across the functional chains, thus flagging **system-level risk**
    $$N_c(u) = \#\{\text{chains where } u \text{ appears with non-trivial impact}\}$$
    - **Structural involvement flag** $B(u)$**:** This is a Boolean flag that indicates whether a function is a bottleneck in at least one functional chain. A 'TRUE' value indicates the need for a design action (e.g., increased redundancy) and hence, a higher priority over another function at the same severity level.
    - **Minimum Failure Resilience (**FRS_{min}**):** A function's redundancy risk is governed by the minimum observed Failure Resilience Score (FRS) across all chains containing it. The least resilient value determines the function's overall redundancy requirement.
3. **Tertiary Ranking Criteria (Hubs & Tie-breakers):** They are the final tie-breakers and produce the mitigation priority rank
    - **Maximum Coupling (**Fan_out_{max}**):** Functions with a higher Fan out act as hubs whose failure can cascade and affect many downstream functions. They are, hence, ranked higher.
    - **Global Centrality Rank:** This parameter acts as a final tie-breaker when all other criteria scores are equal for a particular function. This ensures that the function most critical to the overall network structure is addressed before others.

The consolidated F**unction Risk Profile (FRP)** encapsulates the per-chain vulnerabilities into a final **Mitigation Priority Rank**, directly serving as a prioritised list of design recommendations and mitigation actions that engineers can implement to manage risk.

3.7.3. Functional FMEA & Design Recommendations

The final step of the framework is to consolidate the calculated Functional Risk Profile (FRP) into a Functional FMEA in line with ECSS standards. The objective is to produce a document that can guide design and be used for reviews in early phases. The Functional FMEA is a **semi-automated, row-per-function failure analysis** and is mapped as shown in Table 3.8.

Table 3.8: Derivation of Functional FMEA Fields

FMEA Field	Derivation Source	Automation Status
Function	Function name.	Auto-filled
Failure Mode	"Loss of Function ⟨name⟩."	Auto-filled
Effect	Text summarised from the worst-case chain analysis.	Auto-filled
Severity (P)	Determined by consolidated Worst-Case Severity ($P^\star$), using the P1–P5 scale.	Auto-filled
Recommended Action	Rule-based mapping from $P^\star$ and structural flags ($\mathrm{FRS}_{\min}$, $\mathrm{Fan}_{\mathrm{out\,max}}$).	Auto-filled
Likelihood / Detection	Not automatically assessed due to lack of early-phase reliability data.	**Manual**

Design Recommendations

The framework proposes some common mitigation actions based on the consolidated Function Risk Profile, attempting to translate the metric into actionable design recommendations:

Table 3.9: Design Recommendations in FMEA

Primary Evidence	Recommended Action
P1 or $B(u)$ = TRUE	Add redundancy: Alternate S→T route or redundant providers; review common-cause links.
P2 (RESOURCE Loss)	Isolate/Secure: Resource redundancy or fail-safe modes.
P3 (CORE Loss)	Backup/Buffer: Secondary core source/path, buffering/caching, or eliminating single enabler functions.
P4 (UTILITY Loss)	Degrade Gracefully: Monitoring + fallback, controlled degradation, or add a secondary utility source.

Continued on next page

Table 3.9: Design Recommendations in FMEA (Continued)

Primary Evidence	Recommended Action
High $\text{Fan}_{\text{out max}}$ (Hub Risk)	Refactor: Split the function (modularisation), introduce a buffer/mediator, or simplify dependency structure.

By automating the majority of the FMEA construction, the framework accelerates the identification of structural vulnerabilities and aids in early design, well before detailed design commences.

Summary

This chapter presented the design of the proposed framework, structured into four layers: requirements traceability, fault detection, failure analysis, and risk assessment. The design was grounded in ECSS-compliant methods such as FMEA, while extending them with MBSE and graph theory-driven metrics. Key choices made were introduced, including the use of functional chains as analysis objects, the definition of failure characteristics attributes, and the integration of severity scoring and resilience metrics. Together, these design elements provide the foundation for demonstrating the framework in later chapters, ensuring that the analysis outputs can be traced within the MBSE environment.

4

NEBULA-Xplorer mission

This chapter introduces the NEBULA-Xplorer mission and its systems engineering context. It outlines the mission objectives, the role of SRON and partner organisations, the team structure, mission phases, and the approach to requirements generation. These elements provide the organisational and technical background on which the framework from Chapter 3 will be applied and tailored.

4.1. Mission Overview and Objectives

NEBULA-Xplorer: Netherlands Educational SmallSat for Exploring Binary-linked Astrophysics-X-ray Observer aims to become one of the most innovative projects in the Dutch scientific community. Led by **SRON**, Space Research Organisation Netherlands, the mission is primarily student-driven and offers students the opportunity to engage in astrophysics research and contribute to the development of new space technologies.

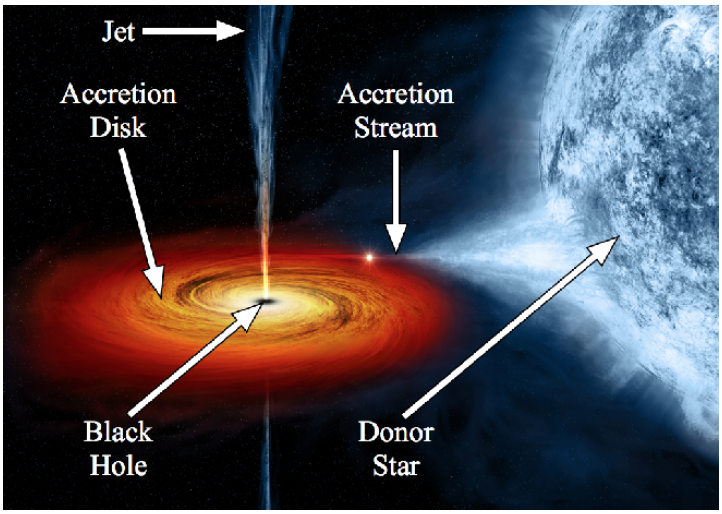


Figure 4.1: Black hole binary system [43]

The primary goal of the NEBULA-Xplorer is to investigate the extreme gravity and relativistic behaviour of binary systems, with a special focus on observing the relativistic jets produced by matter falling onto the black hole or neutron star within these systems.

In black hole–donor star binary systems, matter from the companion star is pulled into the black hole's accretion disk via an accretion stream (Figure 4.1). Well-studied binaries such as Cygnus X-1 and IGR J17091-3624 exhibit distinctive patterns of X-ray variability. By analysing both the energy spectrum and the timing of this X-ray emission, valuable insights can be gained into the dynamics of accretion and the geometry of black holes.

The present mission emphasises the detection of relativistic jets, an aspect of these systems that remains less understood compared to accretion and disk processes. X-ray features typically evolve over timescales of days to months, highlighting the importance of extended monitoring for improved understanding of their temporal and physical development.

Although observatories such as XMM-Newton, Chandra, and NICER already enable valuable X-ray studies, access to these facilities is quite limited and subject to intense competition. NEBULA-Xplorer distinguishes itself by offering the capability to collect continuous, dedicated soft X-ray data for the Dutch scientific community.

The overarching scientific objective of NEBULA-Xplorer is to design and deploy a compact X-ray observatory satellite capable of sustained monitoring of X-ray binary systems. Alongside this, the project also carries an educational mission: to create an environment where students gain direct, hands-on involvement in every stage of a space mission. This dual purpose not only advances scientific knowledge but also equips students with the skills and experience necessary to contribute to the future workforce of the Dutch space sector.

4.2. Systems engineering context

The NEBULA-Xplorer mission is developed within a structured systems engineering framework. This section introduces the main stakeholders, timeline and the team structure, followed by the approach to requirements generation and the digital engineering tools that support the project.

4.2.1. Stakeholders

The primary stakeholder and lead organisation of the NEBULA-Xplorer mission is SRON, Space Research Organisation Netherlands. As the primary research center for astrophysical instrumentation and space science, SRON has a long history of developing X-ray detectors, space instruments, and contributing to international missions with ESA and NASA. In NEBULA-Xplorer, SRON carries full responsibility for mission leadership, systems engineering, and payload development, while at the same time embedding education as a central mission objective.

In addition to SRON, several other stakeholders are involved. Dutch universities supply students from a range of disciplines, who form the main workforce behind the mission. Industrial partners such as Airbus NL, ISISpace, and Royal NLR are expected to support spacecraft bus development, launch preparation, and ground segment operations. The Dutch scientific community represents the end-users of the mission's data, benefiting from continuous soft X-ray observations of binary systems.

4.2.2. Team Structure

The NEBULA-Xplorer mission is structured using several coordinated teams, that reflect the spacecraft's major elements and the mission's educational nature. Overall responsibility rests with SRON, where the Mission Project Leader oversees technical and scientific coherence and resolves ambiguities in requirements or interfaces in consultation with the Principal Investigator (PI) and the Product Assurance (PA) Manager.

The Systems Engineering (SE) team, led by SRON with support from industrial partners (Starion Group), plays a central role in defining requirements, managing interfaces, and ensuring verification and validation in line with ECSS standards. Owing to the educational nature of the project, the composition of this team is flexible, as students join for limited periods and rotate across subsystems.

Three dedicated teams implement the mission design:

- **Spacecraft team:** responsible for the design, manufacturing, and testing of the satellite bus, expected to be led by an industrial partner such as Airbus NL or ISISpace (TBD).
- **Payload team:** led by SRON, tasked with the design, integration, and testing of the X-ray instrument.
- **Ground segment team:** expected to be led by Royal NLR, responsible for the development of ground support equipment and operations.

Each of these subsystem teams reports requirements or interface issues to the SE team, which maintains mission-wide consistency. This structure ensures a balance between expert oversight and student-driven execution, allowing participants to gain hands-on experience while ensuring reliability.

4.2.3. Mission phases and timeline

The mission follows the standard ECSS development lifecycle, with phases 0/A (mission concept and feasibility), B (detailed definition), C/D (design, development, manufacturing, assembly, integration, and verification), E (operations), and F (disposal). Due to its student-driven nature, these phases are not strictly sequential: payload and spacecraft development may progress at different paces, and transitions between phases are often more fluid than in conventional missions.

The mission adopts an EM-PFM model philosophy, in which both the spacecraft and the scientific instrument are first realised as Engineering Models (EM) before transitioning to Proto-Flight Models (PFM). Depending on maturity levels, certain subsystems may also require dedicated qualification, and the use of breadboards or early prototypes is not excluded.

Development of the payload instrument began in September 2024, and since then, three groups of students have contributed to its design and testing. Phase 0/A concluded with a System Requirements Review (SRR) for the spacecraft and a Preliminary Design Review (PDR) for the scientific instrument, both conducted in July 2025. Normally, SRR and PDR take place after Phase B, but for NEBULA-Xplorer these reviews were expedited to reflect the educational and iterative nature of the mission. Subsequent reviews, including a Conceptual Design Review (CoDR) in Phase B and a Critical Design Review (CDR) in Phase C/D, are planned as the project matures.

The mission is currently baselined for a launch in **late 2029 or early 2030**, with a nominal operational lifetime of **five years** before entering end-of-life disposal (Phase F) in line with ECSS guidelines. Over the full course of development and operations, it is anticipated that more than 400 students will participate in the project, reflecting the inherently dynamic and continuously evolving nature of a student-driven mission.

4.2.4. Requirements generation and flowdown

The requirements for NEBULA-Xplorer are organised hierarchically to maintain a clear link between the mission objectives and the technical design. At the highest level, the Mission Requirements Document (MRD) defines all requirements needed to achieve the scientific goals of the mission. These cover the mission timeline, spacecraft and instrument, launcher, ground segment, and operations. The full requirement hierarchy for the MRD is shown in Appendix **??**.

The System Requirements Document (SRD) flows down directly from the MRD. The SRD organisation and requirement categories are illustrated in Figure 4.2. SRD translates high-level mission requirements into system-level specifications and further into subsystem- and equipment-level constraints. Each requirement in the MRD and SRD is assigned a unique identifier, followed by a category code and sequential number:

$$X - YYYY - nnn$$

Where,

- X is the requirement type: ”R” for a requirement or ”G” for a goal
- YYYY is the requirement category consisting of 3 to 4 letters
- nnn is the requirement identifier (sequential number of 3 digits).

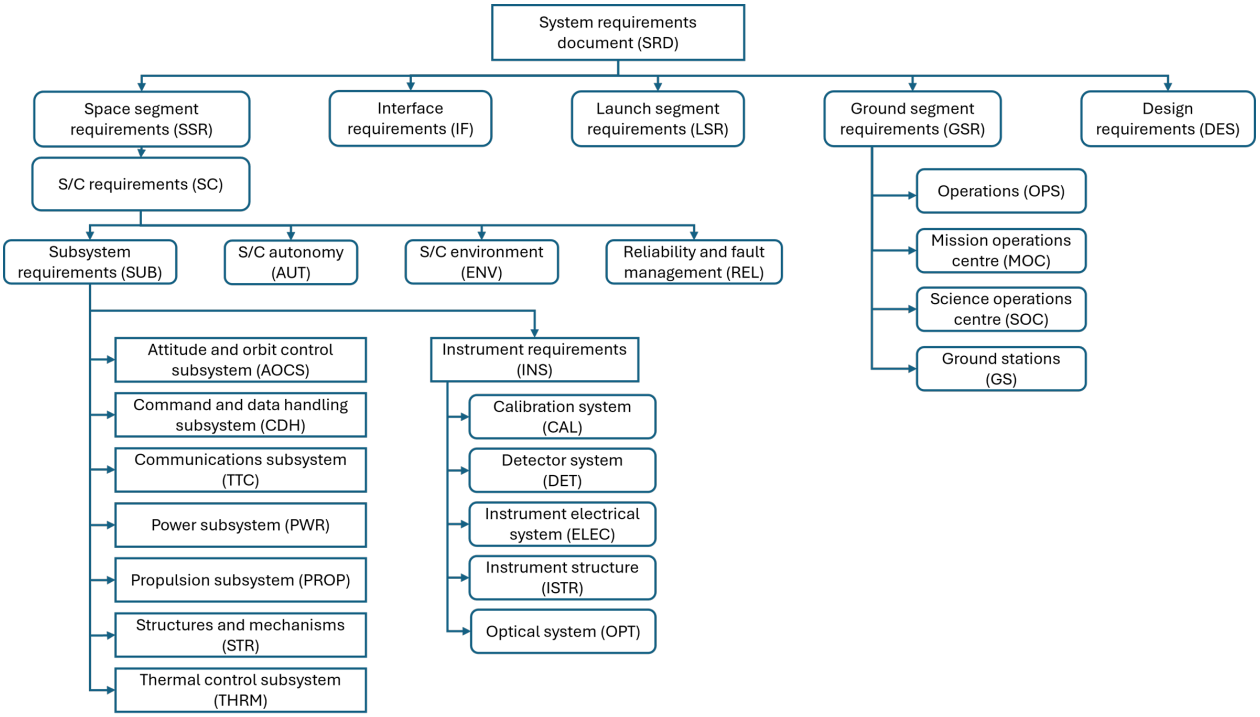


Figure 4.2: System Requirements Organisation [44]

Requirements are mandatory and must be verified, while goals serve as desirable performance targets to maximise scientific output without introducing excessive cost or complexity. Both documents are managed in the CDP4-COMET environment [45], which supports collaboration and traceability across subsystems. Any changes are handled through Redmine and the mission's Change Control Board (CCB).

4.3. Tailoring the Framework to Mission Needs

The previous section describes the overall context of the mission and the various stakeholders involved. They have varying roles, levels of influence, and expectations. Tailoring the proposed framework requires further understanding of stakeholder priorities:

- **SRON (Mission Owner)**: Its interest lies in the credibility of the framework and its alignment with ECSS standards. SRON uses the outputs in system-level reviews and needs confidence that the results are accurate and review-ready.
- **System Engineering (SE) team:** They are the primary users of this framework. Their power is high because they integrate the results into design decisions. Their interest is also high: the framework should integrate seamlessly into current systems engineering practices.
- **Student team (rotating workforce)**: The framework must be usable, explainable, and resilient to incomplete inputs. Their power is low, but their day-to-day interaction with the framework gives them high interest.

- **Industrial Partners (Airbus NL, ISISpace, Royal NLR, etc.)**: Their direct use of the framework, and hence their interest, is limited.
- **Dutch Scientific Community**: They are interested in scientific data continuity, but their interest or power in framework development is low.

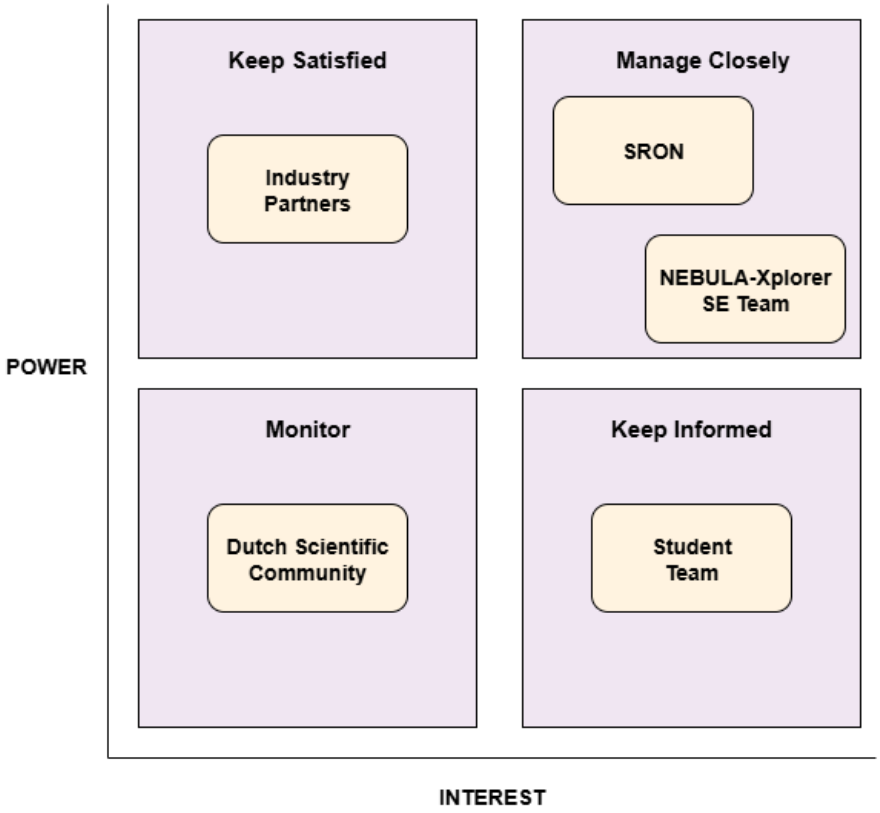


Figure 4.3: Power-Interest Grid for framework

Plotted on a Power-Interest (PI) Grid (Figure 4.3), the Systems Engineering team and SRON emerge as the most powerful stakeholders, with students also playing a critical role due to their high day-to-day involvement. This combination makes explainability, traceability, and lightweight implementation central to tailoring the framework for NEBULA-Xplorer.

Mission specific needs

Stakeholder priorities highlight several mission specific needs that the framework must address to be effective for NEBULA-Xplorer:

- Lightweight implementation: The framework must operate with limited resources, relying on open-source or low-cost tools and minimal design data.
- Continuity despite turnover: The framework must remain explainable, traceable, and resilient to incomplete model data.
- Iterative application: Since mission reviews are iterative and non-linear, the framework must provide outputs that can be updated and reused across phases.

These needs provide the link between stakeholder expectations and the generic framework requirements defined in Chapter 4. They also highlight the constraints under which the framework has to be applied.

Summary

This chapter concludes that the NEBULA-Xplorer mission provides both the complexity and constraints necessary to demostrate the proposed framework. Its scientific ambition, coupled with limited resources and student-led development, underscores the need for lightweight, adaptable methods. The stakeholder analysis highlights that tailoring the framework to mission-specific needs such as open-source usability, resilience to team turnover, and alignment with iterative ECSS reviews as essential. Overall, the chapter establishes that early, MBSE-driven risk management is not just beneficial but critical for ensuring mission reliability in this context.

5

Tool and Method Set-up

This chapter introduces the toolchain used to implement the framework in context of NEBULA-Xplorer mission. Capella with the Arcadia methodology is introduced, supported by CDP4/COMET for requirements management and Python scripts for automation.

5.1. Selection of MBSE Tool and Methodology

The selection of Capella as the MBSE tool for this study was a direct result of the mission-specific needs of the NEBULA-Xplorer (section 4.3). Although the framework is designed to be tool-agnostic, its implementation for the NEBULA-Xplorer required a unique tool selection. Capella is an open-source tool that implements the Arcadia Methodology and addresses the constraints from the mission more effectively than comparable MBSE tools in the following ways:

- **Open-source and lightweight:** Capella can be installed and maintained with minimal effort in a student context where turnover is high. Unlike commercial tools such as Cameo or Enterprise Architect, it does not require licenses or significant setup resources, making it more practical for a resource-constrained mission.
- **Built-in Arcadia methodology:** Capella provides a defined set of viewpoints that guide model development without the need for custom profiles. This built-in structure supports consistency and reduces the learning curve, while also ensuring that evolving models remain interpretable across project phases.
- **Educational accessibility and adoption:** Capella is widely used in academic and industry contexts, supported by tutorials and community resources. Its accessibility makes it particularly suitable for rotating student teams, while still producing models that are robust enough for professional stakeholders such as SRON and industry partners.

The following subsection provides a detailed introduction to the Arcadia methodology, serving as a basis for the model development presented in Chapter 6.

5.1.1. ARCADIA Methodology

Arcadia (ARChitecture And Design Integrated Approach) provides a structured method for the development of complex systems. It introduces a model-based and architecture-centric approach, guiding engineers through the definition, analysis, design and validation of a system [46]. This methodology is highly adaptable, supporting various development strategies such as top-down or bottom-up modelling, particularly through its implementation in tools like Capella [47].

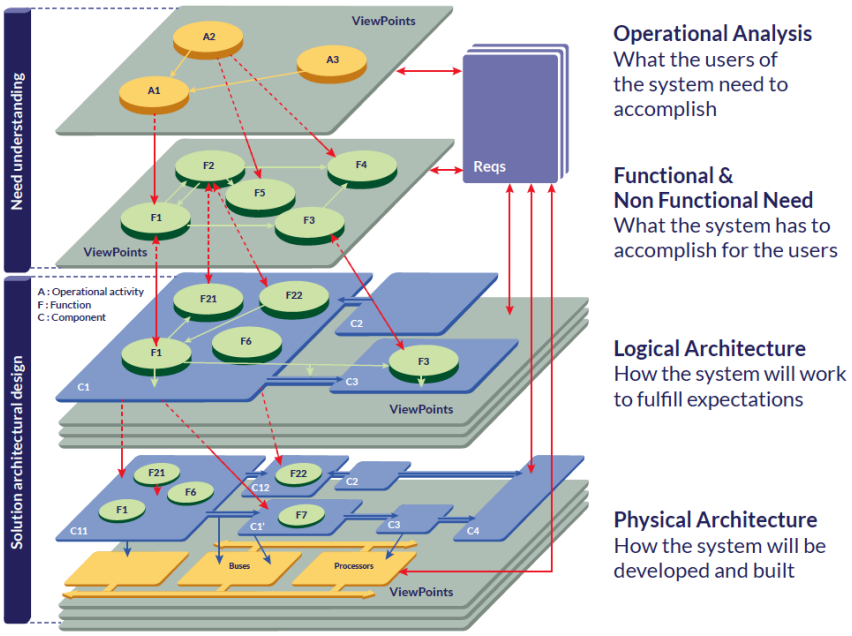


Figure 5.1: ARCADIA Methodology [48]

The methodology follows a sequential approach and enables defining the ”problem space” before the system design. They key phases as shown in Figure 5.1 are:

- **Operational Analysis:** This phase explores the problem space by defining the capabilities and user needs from an operational standpoint. It defines the scope of what the system is expected to do for its user.
- **System Analysis:** This phase builds on the operational layer and defines what the system must do it satisfy user needs. The system is treated as a ’black box’, and both functional and non-functional requirements are identified, without any description of internal structure.
- **Logical Architecture:** This phase shifts attention to how the system will meet the defined requirements. System functions are decomposed into logical components, which are then allocated within a conceptual architecture.
- **Physical Architecture:** This layer addresses the actual implementation of the system. Logical components are realised by specific physical components. The resulting model provides a blueprint for system development and integration.

5.2. Toolchain and Workflow

The practical implementation of the framework requires an integrated tool environment capable of supporting requirements management, system modelling and automated analysis. To achieve this, a toolchain was established that combines Capella, CDP4/COMET and Python scripts, each fulfilling a distinct role within the workflow:

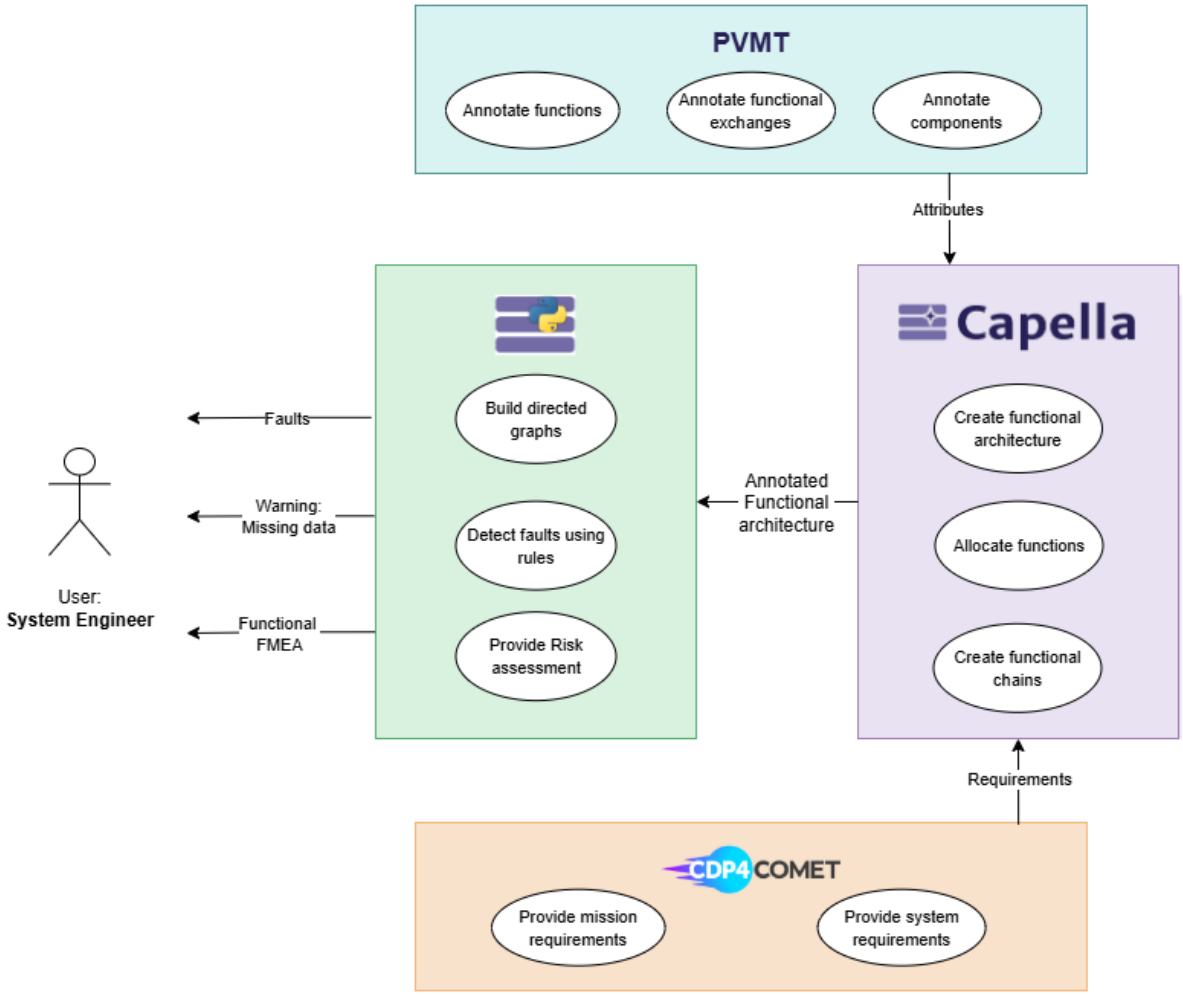


Figure 5.2: Toolchain and workflow

- **Capella:** This forms the backbone of the setup, providing the MBSE environment to model the system. It captures the spacecraft and instrument functions and interactions. The architecture developed in this tool forms the input for all automated analysis. The **Property Value Management Tool (PVMT)** is used to enhance the models with the essential attributes discussed in Chapter 3.
- **CDP4 COMET:** The team uses COMET to manage the Mission Requirements Document (MRD) and the System Requirements Document (SRD). This environment provides collaborative editing capabilities and is integrated with Capella via a COMET-CDP4 plug-in, developed by the Starion group, to enable efficient data transfer between the two platforms.

- **Python:** To automate the analytical processes, Python scripts are integrated to process the Capella model data. These scripts implement the core fault detection and risk analysis logic detailed in Chapter 3, generating outputs that pinpoint potential design faults, identify missing model data, and delineate critical failure modes within the system.

The overall workflow is operated by a user system engineer, who maintains the Capella model, manages requirements in CDP4, executes the analysis scripts, and coordinates changes. This setup is deliberately lightweight and based on open-source or widely available infrastructure, aligning with the mission-specific needs identified in Chapter 4. Figure 5.2 illustrates the interactions between the tools, showing how requirements, models, and analysis results are exchanged within the framework.

Summary

This chapter introduces Capella, supported by the Arcadia methodology, as the most effective balance between accessibility, structure, and robustness for implementing the framework in the NEBULA-Xplorer mission. Its open-source nature and built-in viewpoints address the mission's constraints of limited resources and high student turnover, while ensuring outputs remain consistent and interpretable. The integration of Capella with CDP4/COMET for requirements management and Python scripts for automated analysis establishes a lightweight yet complete toolchain. Overall, the chapter demonstrates that this setup not only satisfies mission-specific needs but also offers a practical way for applying and verifying the proposed framework in the following chapters.

6

Framework Verification

This chapter demonstrates the framework on the NEBULA-Xplorer baseline design, showing how fault detection, failure analysis, and risk assessment are applied in practice. It illustrates the workflow from MBSE modelling to the automated outputs highlighting critical vulnerabilities.

6.1. Baseline design: NEBULA-Xplorer

The NEBULA-Xplorer mission consists of a compact X-ray telescope as the primary payload, supported by a dedicated spacecraft bus that provides the necessary resources for operation. These elements together define the technical baseline of the systems that is represented in the MBSE Model.

6.1.1. Instrument: X-ray Telescope

The payload of NEBULA-Xplorer is a compact X-ray telescope designed to study the quasi-periodic behaviour of binary systems by detecting their X-ray emissions. The telescope has an effective area of 200 cm^2 at 1 keV and 180cm^2 at 6 keV and measures approximately 110 x 85 x 50 cm. This size is unusually compact for an X-ray observation instrument and is achieved through a set of nested cylindrical mirrors that reflect incoming X-rays onto a focal-plane detector.

The instrument design is further organised into the following key subsystems:

- **Optical Assembly:** A concentric arrangement of cylindrical mirrors that provides the necessary grazing-incidence reflection of X-rays. The mirror geometry has been optimised to achieve the effective area, enabling the payload to fit within the volume and mass budget of a small satellite.
- **Detection system:** A commercially available detector unit mounted on the focal plane. It provides the sensitivity required for long-term flux measurements but must be cooled to approximately -35°C to maintain performance.
- **Calibration system:** Mounted near the detector unit, this system provides the reference signals to monitor and verify the detector's performance in orbit.

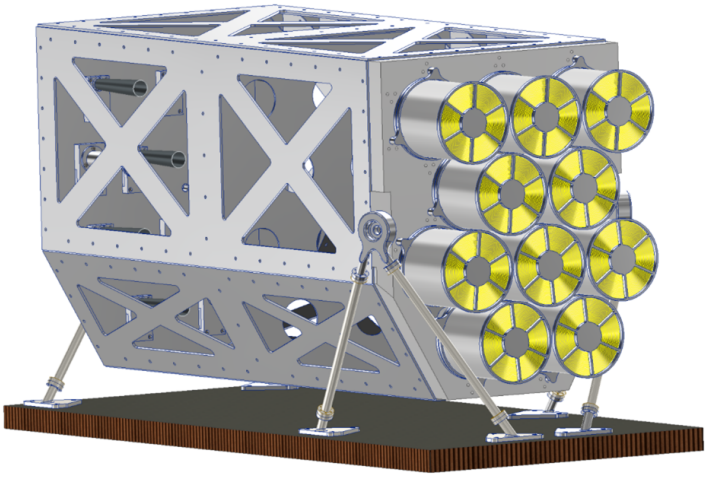


Figure 6.1: NEBULA-Xplorer instrument [49]

- **Read-out electronics:** Dedicated front-end electronics that convert the analogue detector signals into digital data, which is passed to the spacecraft's on-board computer. These electronics also provide housekeeping monitoring and an interface with the spacecraft's main computer.
- **Structure:** A compact housing integrates all subsystems into a stable assembly. It provides optical alignment, mechanical robustness, and thermal interfaces to the spacecraft bus.

6.1.2. Spacecraft bus

The spacecraft bus provides all the supporting resources required to operate the X-ray telescope in orbit and ensures that the mission objectives can be achieved. The bus module integrates all the necessary subsystems for power, thermal management, communication, command handling and attitude control.

The main subsystems of the bus are:

- **Attitude and Orbit Control System(AOCS):** Maintains accurate pointing of the telescope, using reaction wheels, magnetorquers and star trackers.
- **Propulsion:** The role of the propulsion system is limited to providing ΔV for collision avoidance and end-of-life disposal in line with mission and debris-mitigation requirements.
- **Communication:** It includes the downlink of science and housekeeping telemetry, as well as uplink of commands for payload and bus operations.
- **Power:** Provides continuous power through deployable solar arrays and battery storage.
- **Command & Data Handling(CD&H):** On-board computing and data management. It handles instrument commanding, mode switching, mass memory, and routing of science/housekeeping data.

- **Thermal Control:** Ensures stable operating temperatures for both the bus and payload, considering worst-case cold and hot combinations of attitude, orbit and internal dissipation encountered during the mission.
- **Structure & Mechanisms:** The primary platform supporting payload and subsystems; withstands launch loads and provides mounting and mechanisms for deployables (e.g., solar arrays)

In the months leading up to the System Requirements Review (SRR) in July 2025, the NEBULA-Xplorer team conducted a preliminary selection of various components to realise the requirements of each subsystem. Appendix **??** provides an overview of the internal and external configuration of the NEBULA-Xplorer satellite as captured during the SRR. The detailed specifications of these components are documented in [49]. These selections were then captured as attributes within the MBSE model.

6.2. MBSE Model Development

The model development follows ARCADIA methodology and covers its Operational, System and Logical Architecture layer. The Physical Architecture is not elaborated since the framework targets early-phase design, where not all components may be allocated.

6.2.1. Operational Analysis

The operational analysis establishes the link between stakeholder expectations and defines how the NEBULA-Xplorer system will be utilised. It begins with the identification of the key stakeholders and operational entities, represented in the Operational Entity Breakdown Diagram (Figure 6.2). These include the Team (SRON, student contributors, and industry partners), the Mission Operations Team, the Ground Segment, the Launch Service Provider, the Space environment and the Scientific Community, which are the end-users of the data.

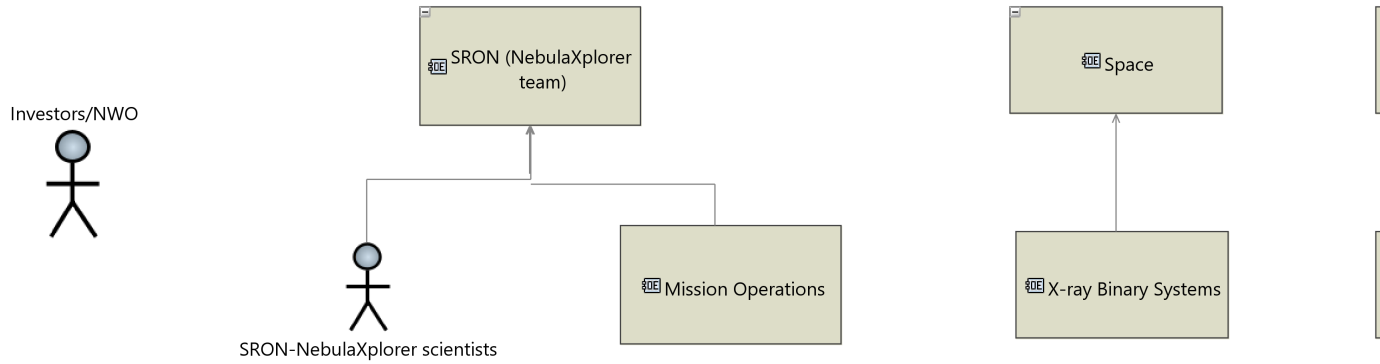


Figure 6.2: Operational Entities Diagram [OEBD]

From this baseline, a set of operational capabilities (Figure B.1 in Appendix B) was defined and associated with the entities. At the highest level, these comprise continuous observation of X-ray binaries, delivering space segment to the target orbit, mission op-

erations and health management, and end-of-life disposal. Other capabilities include providing mission funding and student participation in design and operations.

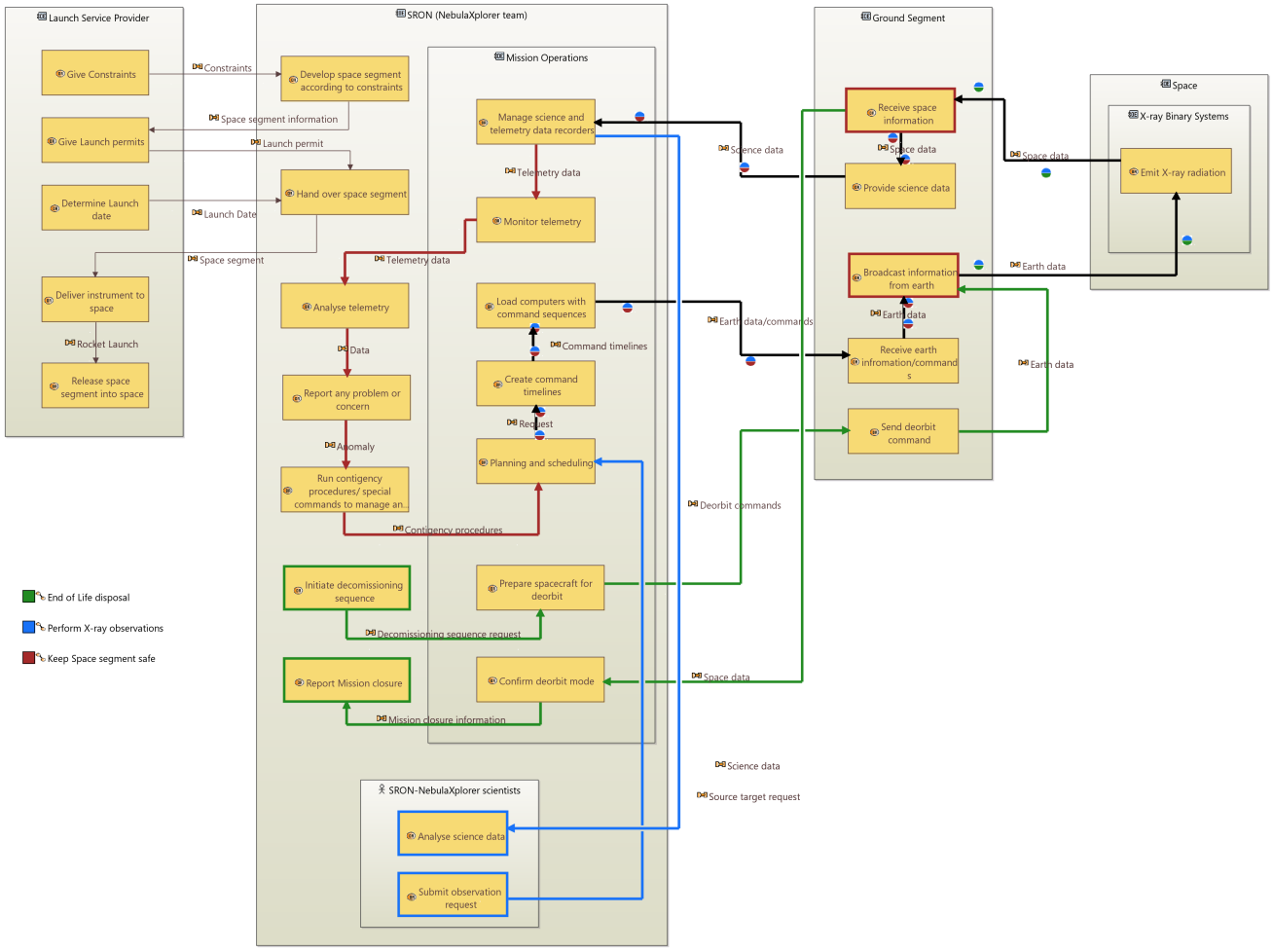


Figure 6.3: Operational Architecture Diagram [OAB]

Each capability was refined into a number of operational activities (e.g., planning observation schedules, collecting and downlinking data, monitoring spacecraft health, anomaly response). These activities were then allocated to the relevant entities: for example, observation planning is performed by the Team, command execution and monitoring by the Ground Segment, and data exploitation by the Scientific Community.

The resulting operational model provides a comprehensive view of how NEBULA-Xplorer is intended to function within its environment. It forms the basis for the subsequent system analysis, where these operational capability of performing X-ray observation is further expressed using system-level capabilities that the spacecraft must fulfil.

6.2.2. System Analysis

In the Arcadia methodology, the System Analysis layer refines the operational view by treating the spacecraft as a black box, defining the system's necessary actions to achieve its mission objective. Unlike the operational analysis, which focused on stakeholders and their needs, this layer introduces system actors, the mission objective, and the resulting system capabilities and interactions.

The identified system actors interacting with NEBULA-Xplorer include:

- **SRON/NebulaXplorer team**, responsible for mission management and science operations
- **Mission operations segment**, ensuring safe and continuous operations
- **Ground segment**, enabling command and telemetry exchange
- **Launch provider**, responsible for orbit insertion and compliance with launch constraints
- **External space environment**, which imposes natural disturbances and degradation factors

From this baseline, the mission objective was captured as:

"Provide X-ray binary measurements, safety and mission compliance solution."

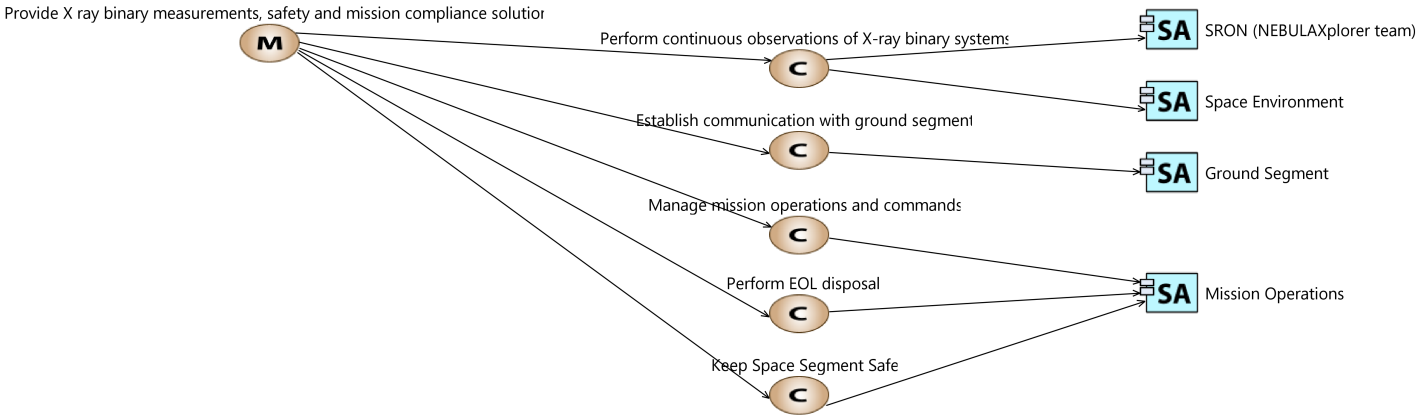


Figure 6.4: Mission Capabilities Diagram

This mission objective was decomposed into a set of system capabilities (Figure 6.4), such as:

- Performing continuous observations of X-ray binary systems.
- Establishing communication with the ground segment.
- Managing mission operations and commands.
- Executing end-of-life disposal in compliance with guidelines
- Ensuring spacecraft safety

After allocating each capability to the relevant system actor, the system functions were defined (Table 6.1). In Arcadia, system functions represent the activities the system must perform to realise the capabilities, while still treating the system as a black box.

Table 6.1: Requirement and mapped functions list.

Sno.	Requirement ID	Mapped Functions
1	R-SC-100	Perform X-ray observation with instrument onboard
2	R-SC-040	Perform end-of-life deorbit manoeuvre (space debris mitigation)
3	R-SC-060 / R-SC-062	Provide spacecraft command and telemetry communication
4	R-SC-061	Ensure secure and reliable communication
5	R-SC-140 / R-GSR-050	Determine orbit knowledge
6	R-GSR-050	Maintain spacecraft orbit
7	R-GSR-010	Enable operational control throughout all mission phases
8	R-SC-080	Provide, manage, and distribute electrical power
9	R-SC-090	Maintain safe thermal environment for payload and sub-systems
10	R-SC-110	Provide structural support and subsystem integration
11	R-IF-020	Enable payload interfaces with other systems
12	R-SC-121	Enable payload operational control
13	R-SC-120	Store and manage mission data onboard
14	R-SC-063	Transmit science data to ground
15	ENV (category)	Ensure mechanical and structural compliance with launch and mission environments
16	R-SC-130	Maintain 3-axis stabilisation
17	R-SC-131	Provide required pointing accuracy
18	R-SC-141	Control the orbit of the spacecraft
19	R-GSR-070 (detect) / R-SC-041 (response)	Detect and respond to external collision risks

For NEBULA-Xplorer, the identification of system functions was guided directly by

the System Requirements Document (SRD) and the operational activities. This mapping exercise implements the **Requirements Traceability layer** of the framework introduced in Chapter 3 and was captured in a System Function Breakdown Diagram [SFBD]. This is crucial because the framework mandates fault detection and risk analysis only on functions explicitly tied to defined requirements. Excluding untraceable functions ensures consistency with the framework's philosophy and its focus on requirement-driven risk analysis.

Development of System Architecture

Following the identification of system functions, the subsequent step in the System Analysis layer was to define the functional exchanges. These exchanges formalise the input-output flows between functions and describe their interactions.

The next activity involved the allocation of functions to either the spacecraft system or external actors (e.g., Ground Segment, Mission operations). This allocation step clarified the distribution of responsibilities and provided the foundation for constructing the System Architecture Blank [SAB] as shown in Figure 6.5.

For clarity and conciseness, this thesis does not present all intermediate System Data Flow Blank [SDFB] diagrams generated during the modelling exercise. These are available in the project repository link in Appendix B. The focus here remains on the consolidated [SAB], which captures the essential exchanges and allocations relevant to NEBULA-Xplorer mission.

Transition to Logical Architecture

Following the completion of the System Analysis layer, which essentially produced a system-level functional block diagram, the modelling was intentionally extended to define the Logical Architecture. While the Framework described in Chapter 3 permits concluding analysis at the System layer for certain applications, the specific objectives of this work, early fault detection and risk assessment for NEBULA-Xplorer, required a higher fidelity of modelling. At this level, functions are decomposed and mapped to subsystem components, enabling the annotation of functional redundancies, interfaces, and dependencies. This ensures the effective implementation of the framework, particularly in performing comprehensive failure analysis.

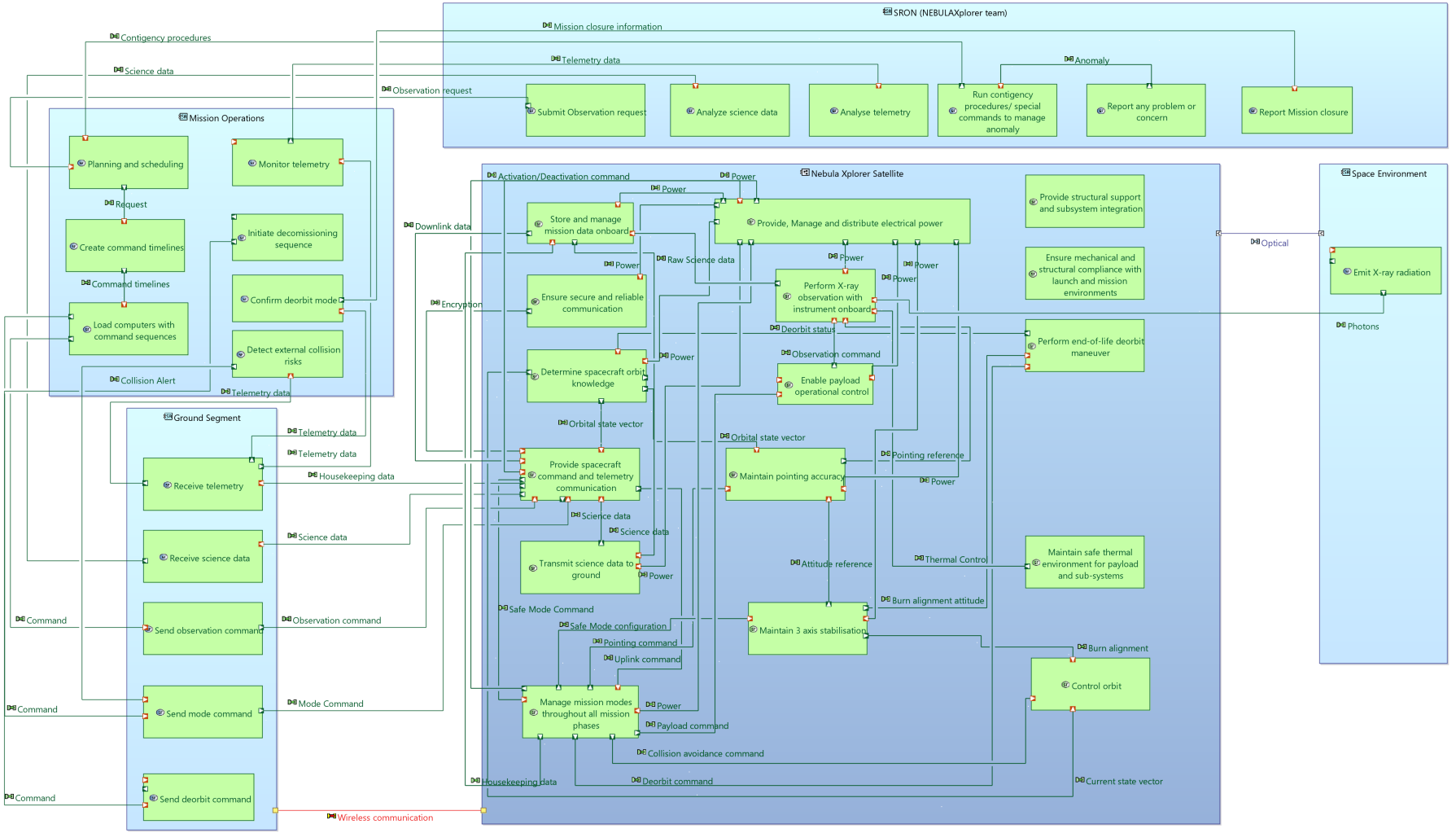


Figure 6.5: System Architecture Diagram — NEBULA-Xplorer

6.2.3. Logical Architecture

The Logical Architecture represents the system's white-box description by defining how the system will work to fulfill its objectives.

Logical Function Decomposition

The system-level functions defined in the previous section were transitioned to the Logical Architecture and subsequently decomposed into logical functions. This refinement ensured that each function achieved a level of detail sufficient to support subsystem or component mapping and subsequent annotation. Every logical function was explicitly traced back to its originating requirement in the SRD, thereby preserving a clear derivation chain:

SRD Requirement ⋯⋯➤ Logical Function ⟶ Sub–Logical Functions

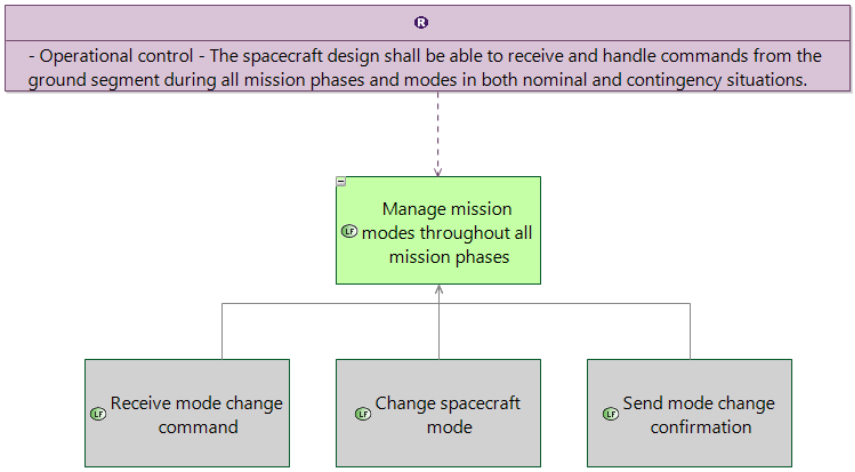


Figure 6.6: *Illustrative example:* Logical Functional Breakdown

An example of the above derivation chain is illustrated in Figure 6.6 and a complete Logical Functional Breakdown Diagram [LFBD] can be found in the repository link (Appendix B). To define the interactions between the functions, Logical Data Flow Blank [LDFB] diagrams were generated, although they are not reproduced here. The complete set is maintained in the repository link in Appendix B.

Development of the Logical Architecture

The system was then decomposed into logical components that reflect the instrument and spacecraft bus as described in section 6.1. Figure 6.7 shows the Logical Component Breakdown for the NEBULA-Xplorer mission. With both functions and components defined, allocation was performed to map each logical function to exactly one

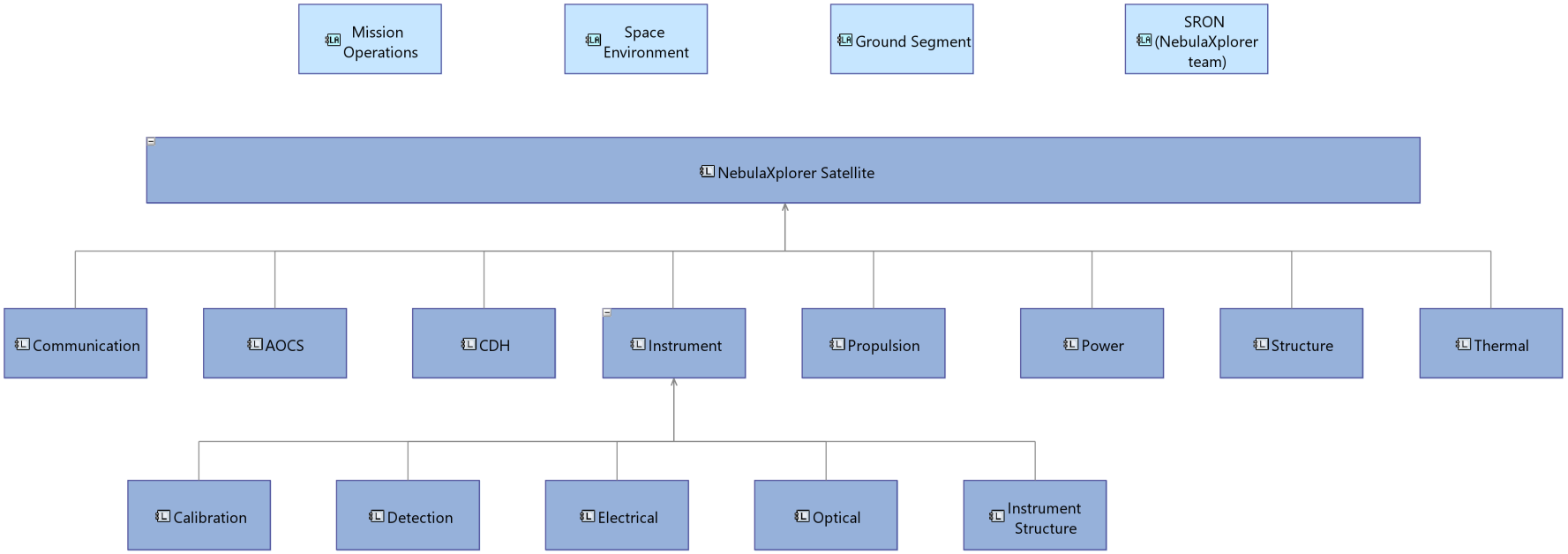


Figure 6.7: Logical Component Breakdown Diagram (LCBD)

logical component. Functional exchanges captured in [LDFB]s were subsequently implemented as component exchanges. This consolidation resulted in a Logical Architecture model as shown in Figure 6.8.

Functional Chains

The next step after creating the Logical Architecture in Capella is to establish functional chains. They are ordered sequences of functions and functional exchanges that depict a specific system capability path within a context [47]. They are represented through a network of functional exchanges and serve as a key input for Failure Analysis in the Framework.

For NEBULA-Xplorer, the establishment of functional chains was guided by two principles:

1. **Coverage of mission-critical capabilities:** Each chain corresponds to a top-level capability essential to mission success.
2. **Completeness of function involvement:** Every logical function identified in the model is covered by at least one chain, ensuring that the risk assessment layer does not overlook any functional element.

Based on these criteria, five primary functional chains (Figure 6.8) were defined:

- **Perform X-ray observations:** Represents the end-to-end path from instrument operations to data collection, storage, and downlink.
- **EOL Disposal:** Captures functions required to ensure the safe disposal of the spacecraft at the end of its life.
- **Collision Avoidance:** Ensures that the spacecraft can detect and respond to collisions.
- **Power Generation:** Represents the production, storage, and distribution of electrical power to all spacecraft subsystems.
- **Thermal Control:** Covers the thermal regulation functions necessary to maintain both spacecraft bus and instrument components within their operating ranges.

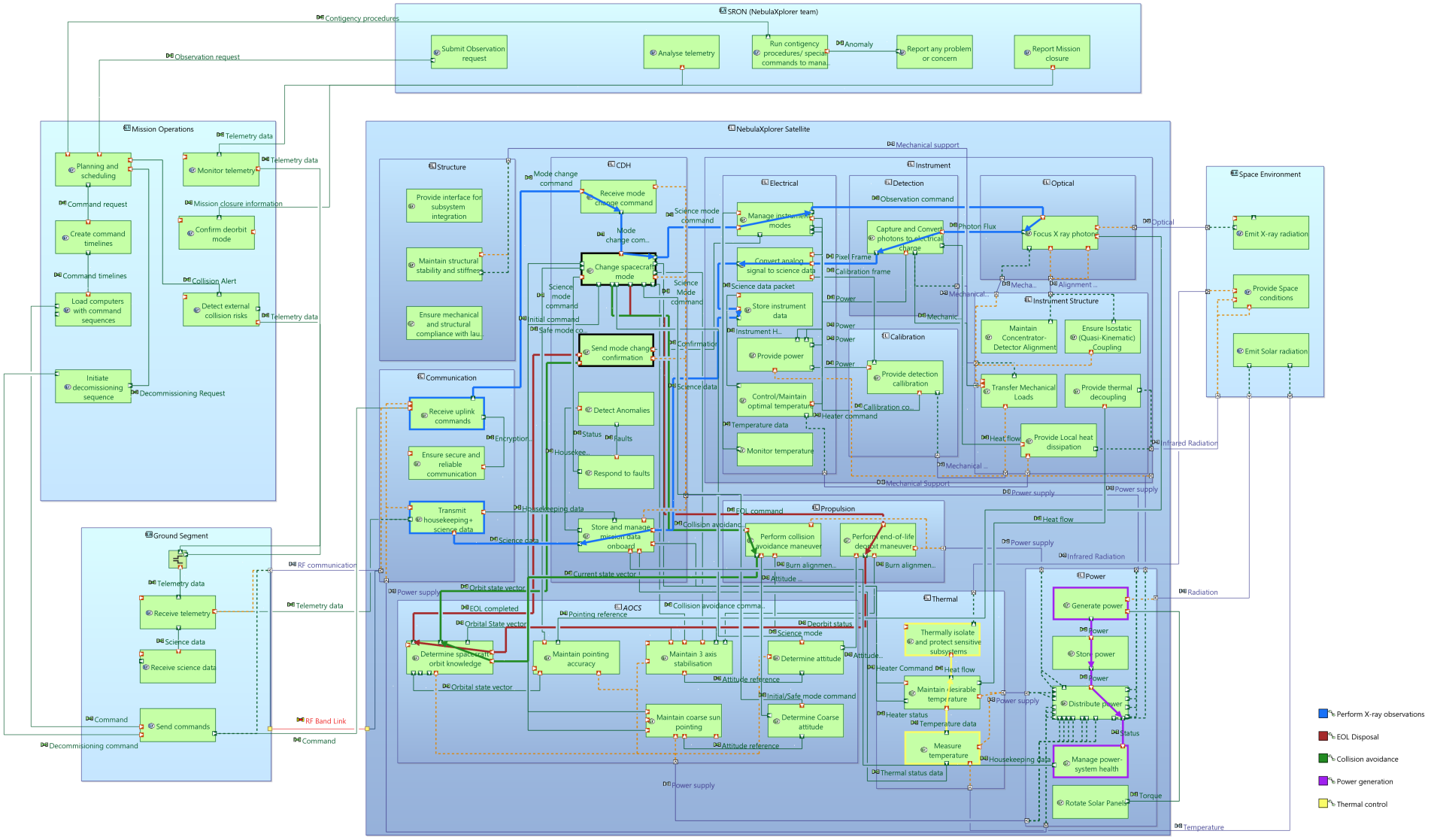


Figure 6.8: Logical Architecture Diagram (LAB)

6.2.4. Attribute Definition and Management

A dedicated attribute structure was developed using Capella's Property Value Management Tool (PVMT) to enable subsequent fault and risk assessment. The attributes, stored as properties in PVMT, were organised into four coherent groups: function attributes, component attributes, exchange attributes, and failure characteristics. This organisation directly maps to the principal elements of the Logical Architecture, ensuring every critical design feature is linked to the risk analysis framework.

The guiding principle behind the structure was as follows:

- **Design-oriented descriptors:** Functions, components, and exchanges are enriched with properties, including redundancy, power capacity, and type, that describe design decisions and performance.
- **Risk-oriented descriptors:** Constraints represent failure modes, each with its own set of risk attributes similar to an FMEA document.

NebulaXplorer | *[NebulaXplorer] Configuration PV ×

Name	Type	Default Value
Function_attributes		
Scope	[LOGICAL]	
EClass Rule	LogicalFunction	
Redundancy_type	Redundancy type	None
Redundancy_level	integerProperty	0
TID_Tolerance	floatProperty	0.0 krad
Radiation_type	Radiation level	None
Functional_exchange_attributes		
Scope	[SYSTEM, LOGICAL]	
EClass Rule	FunctionalExchange	
Functional_exchange_type	Functional exchange definition	Utility
Failure_characteristics		
Scope	[SYSTEM, LOGICAL]	
EClass Rule	Constraint	
Detectability	Detectability	Unknown
Likelihood	Likelihood	Extremely remote
Mitigation type	Mitigation type	None
Failure Mode ID	stringProperty	
Failure Mode Description	stringProperty	
Severity	stringProperty	
Effect	stringProperty	
Mitigation/Design recommendation	stringProperty	
Logical_component_attributes		
Scope	[LOGICAL]	
EClass Rule	LogicalComponent	
Capacity_Data	floatProperty	0.0
Power_consumed	floatProperty	0.0 W
Worst_mission_TID	floatProperty	0.0 krad
Power_characteristics		
Scope	[LOGICAL]	
EClass Rule	LogicalComponent	
Arrays_EOL	floatProperty	0.0 W
PCDU_Max	floatProperty	0.0 W
Battery_Capacity	floatProperty	0.0 Wh
Eclipse_duration	floatProperty	0.0 s

Figure 6.9: PVMT Structure

The PVMT system was implemented hierarchically so that all attributes belonging to one group are clearly distinguished from the others and can be queried automatically during analysis. Figure 6.9 shows the attribute tree as defined in PVMT, illustrating how the four groups are structured and how they relate to the different model entities.

Function Attributes

To extend the model with reliability-related information, a set of custom attributes was defined and attached to logical functions using PVMT. These attributes are designed to capture implementation-relevant properties that directly affect failure analysis and redundancy considerations. Table 6.2 summarises the attributes defined for logical functions.

Table 6.2: PVMT attributes for Logical Functions.

Attribute	Type	Description	Unit
Redundancy level	Integer	Number of parallel implementations of the function (≥ 1).	-
TID tolerance	Float	Total ionising dose the hardware can withstand before degradation.	krad
Radiation type	Enumeration	Radiation resistance level of the implementation (none, tolerant, hardened).	None

Figure 6.10 illustrates their application for the logical function, Convert analog signal to science data, which represents the analogue-to-digital converter of the detection chain. Here, the redundancy level is set to 2 (two ADCs in design), while the TID tolerance is specified as 300 krad. The radiation type attribute was defined in the property structure but not actively used in the current analysis.

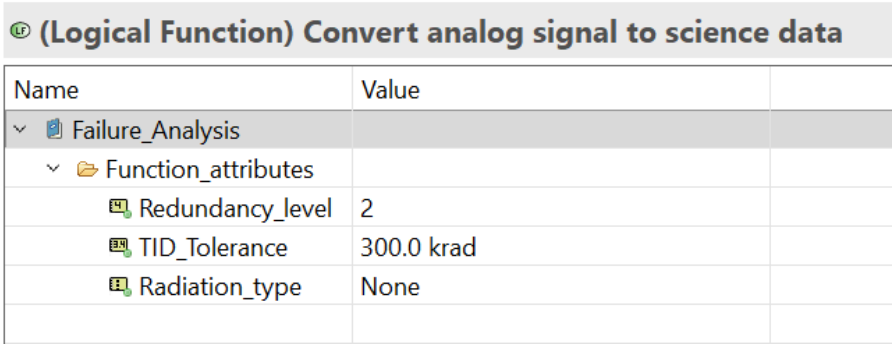


Figure 6.10: Example of PVMT function attributes applied to the logical function

Component Attributes

Logical components were also extended with custom attributes to capture resource usage and environmental limits relevant to failure analysis. These attributes represent

quantitative properties at the subsystem level, enabling the framework to reason about power consumption and radiation limits. The data rate attribute was defined in the property structure but not used in the current analysis, as not all data was known at that stage. Table 6.3 summarises the attributes defined for logical components.

Table 6.3: PVMT attributes for Logical Components.

Attribute	Type	Description	Units
Power consumed	Float	Estimated power demand of the component in nominal operation.	W
Worst mission TID	Float	Maximum radiation exposure expected during mission lifetime.	krad
Capacity data	Float	Maximum throughput or capacity relevant to the component.	Mbps

In addition, the Power subsystem was treated as a special case, as it must support power budgeting and related fault conditions. A separate property group (Power characteristics) was defined and mapped only to the Power logical component. This group captures properties such as end-of-life array output, battery capacity, depth of discharge (DoD), transmission efficiency, and power margin, as shown in Table 6.4.

Table 6.4: PVMT attributes for Power Subsystem.

Attribute	Type	Description	Units
Arrays EOL	Float	End-of-life power generation capacity of solar arrays.	W
PCDU Max	Float	Maximum distribution capability of the Power Conditioning & Distribution Unit.	W
Battery capacity	Float	Available energy storage capacity.	Wh
Eclipse duration	Float	Maximum eclipse time expected during orbit.	s
DoD	Float	Depth of discharge of the batteries.	%
Transmission efficiency	Float	Efficiency of power transfer from source to load.	%
Power margin	Float	Residual margin of available power over required power.	%

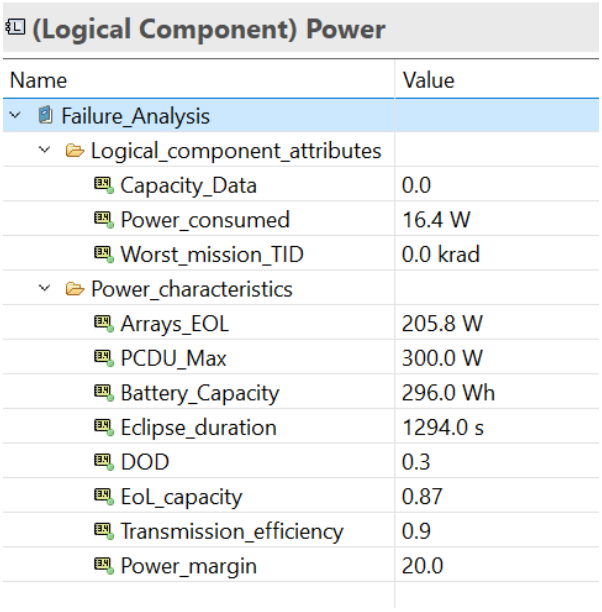


Figure 6.11: Example of PVMT function attributes applied to the logical component

As an illustration, Figure 6.11 shows the attribute assignment for the Power logical component. Only those values that were available were populated in the model. Together, these values enable automated fault detection for power and form the input for the risk assessment scripts described in Chapter 3.

Functional Exchange Attributes

For functional exchanges, the attributes defined earlier (Section 3.7.1) were applied directly within the model. Each logical functional exchange was categorised according to its contribution to mission functionality.

Table 6.5: Example Functional Exchange attributes.

Exchange	Attribute	Value
Pixel Frame	Functional Exchange Type	Core
Callibration command	Functional Exchange Type	Utility
Power	Functional Exchange Type	Resource

Table 6.5 illustrates examples of functional exchange types. The exchange “Pixel Frame”, connecting the detection unit to the read-out electronics, was assigned the type Core, since it is part of the primary observation chain. In contrast, non-critical flows like the Calibration Command were assigned to the Utility category. Exchanges like Power were assigned the Resource category.

Constraint (Failure Mode) Attributes

While design attributes are attached to functions, components, and exchanges, failure analysis requires a different structure. As shown in the attribute tree, a dedicated

attribute set termed Failure Characteristics was defined, applied not directly to logical functions but to constraints representing specific failure modes. This separation ensures that multiple failure modes can be documented in the model if failure analysis is extended in the future. It also allows the documentation of a separate 'Risk Management' architecture diagram to capture the FMEA.

The Failure Characteristics attributes were defined to align with information fields required in an ECSS-compliant FMEA (ECSS-Q-ST-30C). They include identifiers (Failure Mode ID, textual description), risk parameters (Severity, Likelihood, Detectability), and mitigation-related information (Mitigation Type, Design Recommendation).

Unlike the design attributes, risk descriptors are not populated during the modelling process. Instead, they are automatically generated during execution of the **Risk Assessment layer** (see Section 3.7). Their role at this stage is therefore structural: to provide a container for the risk analysis outputs, ensuring that the subsequent FMEA report can be derived consistently from the model without manual re-entry of data. The outputs and an example of the attribute population are discussed in Section 6.2.4.

6.3. Fault Detection Layer demonstration

The Fault Detection layer was applied to the NEBULA-Xplorer model to demonstrate the second step in the framework. Two rules were implemented in Python and executed on the Capella model enriched with PVMT attributes. The purpose was to illustrate how requirement-driven checks can flag potential faults in design decisions or missing data already at the logical architecture stage. Both rules include **attribute completeness checks** as a built-in rule, since the absence of key properties could invalidate the analysis.

6.3.1. Rule 1: Power adequacy

The power adequacy rule verifies that the spacecraft's power generation and storage capacity are sufficient to cover the demand of all logical components, with the design margin applied. The implementation checks three aspects:

- **Total demand completeness**: The rule first sums the attribute `Power_consumed` across all Logical Components (LCs). Missing values are reported individually. The required power is defined as

$$P_{\text{req}} = \left(\sum_{i=1}^{N} P_{i,\text{consumed}} \right) (1 + m),$$

  where m is the design margin (default 20% according to ECSS [50]).

- **Sunlight adequacy**: The available steady-state power is the minimum of the solar array end-of-life output (P_{array}) and the PCDU capacity (P_{pcdu}). The check is satisfied if

$$\min(P_{\text{array}}, P_{\text{pcdu}}) \geq P_{\text{req}}.$$

- **Eclipse endurance** – The usable battery capacity (E_{batt}), corrected for depth of

discharge (DOD), efficiency factors (η), and end-of-life degradation, must sustain the demand during eclipse of duration t_{ecl}. The required energy is

$$E_{\text{need}} = \frac{P_{\text{req}} \cdot t_{\text{ecl}}}{DOD \cdot \eta_{\text{batt}} \cdot \eta_{\text{trans}}}$$

The condition checked is $E_{\text{batt}} \geq E_{\text{need}}$.

The output of the rule provided the following results:

1. If all inputs were present and the adequacy check passed → **OK**.
2. If data was complete but one adequacy check failed → **FAIL**.
3. If key parameters or LC consumption values were missing → **ATTENTION**, with a list of missing attributes.

An excerpt of the output is shown below:

```
================================================================================
Rule R1: Power (sun adequacy + eclipse capacity)
================================================================================

LCs counted (after filter): Total Power_consumed = 74.23 W
Data note: 5 LCs missing Power_consumed:
   Electrical, Instrument Structure, Detection, Optical, Calibration
Array meets power demand: avail 205.80 W >= required 89.08 W (sum 74.2 W, +20%).
Eclipse OK: batt 296.00 Wh >= need 136.31 Wh (t=0.359 h).
Data incomplete - missing PVs: None; LCs without Power_consumed: 5

================================================================================
Overall fault-rules summary
================================================================================

- R1 Power: ATTENTION! Sun OK; Eclipse OK; Data missing: 0 PVs, 5 LCs.

================================================================================
```

The rule identified a total demand of 74.2 W with a 20% design margin, which was within the available array and battery capacity. However, five Logical Components (defined under instrument) lacked `Power_consumed` values, resulting in an "ATTENTION" outcome rather than a full "OK". This check ensures the possibility of running the analysis every time new components are selected across the team.

6.3.2. Rule 2: Radiation Tolerance

The radiation tolerance rule checks compliance with the mission's expected Total Ionising Dose (TID) environment. The rule follows the structure shown below:

1. The `Worst_mission_TID` is defined as a scalar property on the satellite-level logical component.
2. Each logical function allocated to the spacecraft is expected to carry a `TID_tolerance` attribute.
3. Functions without this attribute are flagged as **MISSING**.
4. Functions with tolerance values below the mission TID are flagged as **FAIL**.
5. Only functions with values above the mission TID baseline are considered **OK**.

An excerpt of the output is shown below:

```
================================================================================
Rule R3: Radiation Tolerance Check (TID)
================================================================================

Worst-case mission TID: 7.8 krad.
[MISSING] Focus X ray photons: TID tolerance value not found.
[OK] Capture and Convert photons to electrical charge: 20.0 krad >= mission TID 7.8 krad.
[MISSING] Provide detection callibration: TID tolerance value not found.
[MISSING] Maintain Concentrator-Detector Alignment: TID tolerance value not found.
[MISSING] Transfer Mechanical Loads: TID tolerance value not found.
[OK] Control/Maintain optimal temperature: 30.0 krad >= mission TID 7.8 krad.
[OK] Provide power: 100.0 krad >= mission TID 7.8 krad.
[OK] Manage instrument modes: 30.0 krad >= mission TID 7.8 krad.
[OK] Monitor temperature: 30.0 krad >= mission TID 7.8 krad.
[OK] Convert analog signal to science data: 300.0 krad >= mission TID 7.8 krad.
[OK] Store instrument data: 100.0 krad >= mission TID 7.8 krad.

================================================================================
Overall fault-rules summary
================================================================================

- R3 Radiation: OK: 7 | FAIL: 0 | MISSING: 36

================================================================================
```

It shows that several functions did not have TID values defined, leading to 36 missing entries. Gaps in entries at this point are expected as not all components or associated values are known. Among the functions with values, 7 were correctly validated as having sufficient radiation tolerance. Importantly, no function failed the rule, i.e., none had a tolerance below the worst-case mission value.

The summary result, therefore, highlights an **incompleteness of attribute assignment** rather than design inadequacy. This illustrates the value of the rule: missing or undefined values are flagged as clearly as design violations, ensuring early detection of gaps in the attribute population.

6.4. Failure Analysis Layer demonstration

To demonstrate the implementation of the Failure Analysis layer, the *Perform X-ray observations* functional chain was selected as shown in Figure 6.12. This chain captures the end-to-end flow of mission science data, from photon capture to downlink, and thus represents a critical capability of NEBULA-Xplorer.

The chain has the following scope:

- 10 Primary Logical Functions
- 26 Supporting functions
- 10 internal and 39 Boundary functional exchanges of type: Core, Utility and Resource

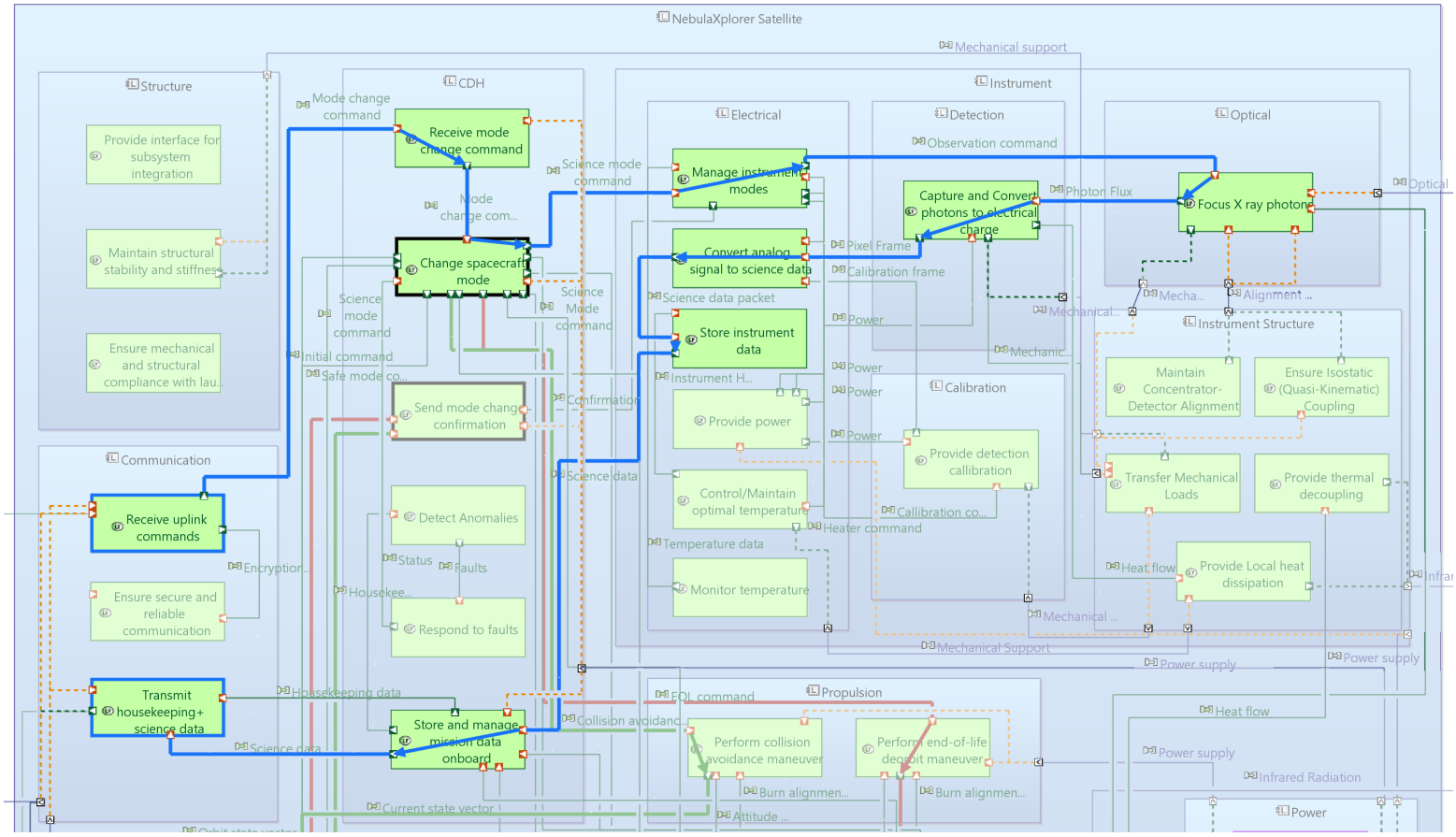


Figure 6.12: Perform X-ray observation chain and neighbourhood

This neighbourhood view ensures that not only functions inside the chain but also supporting functions in other subsystems are considered. The method described in Section 3.6 was implemented using custom scripts developed in Python4Capella. Examples are presented below to demonstrate how different types of supporting and primary functions are analysed in the results:

Example 1: Distribute Power

Within the Perform X-ray Observations neighbourhood, the function Distribute Power was highlighted by the analysis. Single Node Cut test showed that its removal disconnects several primaries (Change spacecraft mode, Receive uplink commands, Store and manage mission data onboard, Transmit housekeeping and science data) from the resource power. This made the baseline path from uplink to downlink unreachable as shown in the excerpt below.

```
  Neighbors Causing Functional Isolation:
    - Neighbor: Control/Maintain optimal temperature
      Breaks S-T: FALSE (Path Survives)
Impacts primary Store instrument data by severing input(s): Instrument Heater status

    - Neighbor: Distribute power
      Breaks S-T: TRUE (Breaks Path)
Impacts primary Change spacecraft mode by severing input(s): Power
Impacts primary Receive mode change command by severing input(s): Power
Impacts primary Receive uplink commands by severing input(s): Power
Impacts primary Store and manage mission data onboard   by severing input(s): Power
Impacts primary Transmit housekeeping+     science data by severing input(s): Power
```

Severity assessment classified this as Catastrophic (P1) when left unmitigated, downgraded to Critical (P3) after considering functional redundancy ($FRS = 2$). This implies that a single failure in the distribution would compromise the science telemetry and control. It presents itself as a system-level bottleneck and a design recommendation to introduce resource redundancy or a fail-safe mode.

Example 2: Maintain Pointing Accuracy

The 'Maintain pointing accuracy' function, which supports the science chain, did not fail entirely but suffered a degradation (a loss of pointing reference input). This degradation impaired the performance of the subsequent function, 'Focus X-ray photons', resulting in reduced science data quality.

```
    - Neighbor: Maintain pointing accuracy
      Breaks S-T: FALSE (Path Survives)
Impacts primary Focus X ray photons by severing input(s): Pointing reference
```

According to the severity rules, this failure was classified as Critical (P3) upon cut analysis. Upon correction for redundancy, the severity was corrected to Major (P4), not mission-ending, but significantly impacting mission success. This case demonstrates how the framework distinguishes between catastrophic failures and degradation.

Example 3: Provide Detection Calibration

The calibration function supported the 'Convert analog signal to science data' primary. Its removal did not isolate the chain but removed calibration frames, affecting data quality.

```
    - Neighbor: Provide detection callibration
      Breaks S-T: FALSE (Path Survives)
Impacts primary Convert analog signal to science data by severing input(s): Calibration frame
```

The severity was classified as Major (P4) but reduced to Minor (P5) upon correction for implemented mitigation actions. This indicates an impact on quality with no material effect on end-to-end delivery. That is, the chain remains available but data quality or accuracy may drift until the next calibration. The recommendation output was that no immediate redundancy was required, but that periodic recalibration or ground-based correction could manage the risk.

Example 4: Store and Manage Mission Data Onboard

Among the 10 primary functions in the Perform X-ray Observations chain, 'Store and Manage Mission Data Onboard' was identified as a bottleneck. Removal of this function severed the downstream transfer of science and housekeeping data, breaking the uplink–downlink objective.

The single-node cut test identifies five primaries that individually break the chain when removed, including 'Store and manage mission data onboard':

```
[Test A : S-T single-node cuts (chain+neighbors)]
  S: Receive uplink commands
  T: Transmit housekeeping+     science data
  Baseline reachable: True

  Mandatory primaries (by rule : all primaries must exist):
    - Capture and Convert photons to electrical charge
    - Change spacecraft mode
    - Convert analog signal to science data
    - Focus X ray photons
    - Manage instrument modes
    - Receive mode change command
    - Receive uplink commands
    - Store and manage mission data onboard


    - Store instrument data
    - Transmit housekeeping+     science data

  Single-node cuts (5):
    - Change spacecraft mode
    - Receive mode change command
    - Receive uplink commands
    - Store and manage mission data onboard


    - Transmit housekeeping+     science data
```

Consistently, the chain-level redundancy calculation reports FRS = 2 (min-cut) and names Store and manage mission data onboard as a SPOF for science/telemetry continuity. A failure here prevents science data persistence and onward transmission even if upstream functions are healthy. Possible design recommendations would include adding distributed storage and separate HK/TT&C storage paths where feasible.

Single Chain Analysis

For *Perform X-ray observations* chain, the analysis surfaces:

- **Path-breaking neighbours functions** (e.g., Distribute power)
- **Performance-degrading supporting functions** (e.g., Maintain pointing accuracy).
- **Benign utilities functions** (e.g., Provide detection calibration) that can be handled during operation.
- **Intrinsic chain SPOFs** (e.g., Store and manage mission data onboard) confirmed by FRS=2.

6.5. Risk Assessment Layer demonstration

The methodology described in Section 3.7 was implemented using the results of the Failure Analysis layer, applied across all the defined functional chains of the NEBULA-Xplorer. While the Failure Analysis Layer highlighted the severity of failures within a single chain (eg.,Perform X-ray Observations), the current layer consolidates these results at a system-level.

Severity Aggregation

The results of the Failure Analysis layer provide chain-specific severities that are then consolidated into a worst-case severity for each logical function. This step ensures that the resulting risk assessment is comprehensive and accounts for a function's role across different chains.

To illustrate this, a subset of functions from *Perform X-ray Observations* chain were evaluated across other chains using the method described in section 3.7.1.

Example 1: Distribute Power

- In the X-ray observation chain, its loss was classified as Critical (P3), since it breaks the path from 'Receive uplink commands' to 'Transmit housekeeping and science data.' Without this function, several primaries (e.g., Change spacecraft mode, Store mission data, Transmit data) lose their power input.
- In the Power generation chain, the same function again appears as a bottleneck, as all generated power must eventually be distributed to downstream users.
- As a result of aggregation, 'Distribute power' remains **Critical (P3)**, highlighting it as a cross-chain vulnerability.

Example 2: Store and Manage Mission Data Onboard

- In the observation chain, failure of this function impacts the science data collection. This was scored as Critical (P3).
- In the thermal control chain, it continues to play a vital role, but without creating additional structural breaks.
- Maintain desirable temperature is hence consolidated as **Critical (P3)**, emphasising its importance for spacecraft health and data integrity.

Example 3: Rotate Solar Panels

- This function is only present in the Power generation chain, where failure reduces solar flux capture and limits power availability. It was classified as Critical (P3) in that chain.
- Since it does not appear elsewhere, the aggregated severity remains Critical (P3).

Example 4: Determine coarse attitude

- In both the observation chain and the collision avoidance chain, its failure does not sever end-to-end connectivity, but reduces pointing quality. This was consistently rated as Major (P4).
- Determine coarse attitude remains at Major (P4), a degradation but not a 'loss of mission' condition.

Assigning a single severity value reveals that functions such as Distribute power, Manage instrument modes, and Maintain desirable temperature are identified as system-level critical functions, whereas others like Determine coarse attitude or Provide detection calibration are localised risks. This constitutes the first step in building the **Function Risk Profile (FRP)** that underpins the risk assessment layer.

Aggregation into Function Risk Profile

The next step in the layer is to aggregate results across all functional chains to identify functions that persistently pose a risk at the system level. To achieve this, each logical function was assigned a Function Risk Profile (FRP) as described in section 3.7. The FRP consolidates:

- Worst-case severity observed across any chain in which the function appears.
- Chain coverage (i.e., number of independent chains where the function is required)
- Minimum redundancy score (FRS_{min}), reflecting whether the function is a bottleneck or supported by parallel paths
- A bottleneck flag, denoting whether the function repeatedly appears as a single point of failure.

The following examples illustrate some of the use cases to demonstrate this process:

Example 1: Distribute Power

- Among all functions, Distribute Power emerged as the most critical system-level risk. It appeared in four separate chains (Perform X-ray observations, Collision Avoidance, EOL Disposal, Thermal Control) and reached a worst-case severity of Critical (P3).
- Its ($FRS_{min} = 2$) indicated low redundancy, and failure of this function consistently propagated to at least five critical primaries, including Change spacecraft mode, Transmit housekeeping data and Store mission data.
- This persistence across chains flagged Distribute Power as a system-level bottleneck and high on the top risk table.

Example 2: Change Spacecraft Mode

- The function 'Change Spacecraft Mode' was also consistently highlighted. It appeared across multiple operational chains, with a worst-case severity of Critical (P3) and an FRS_{min} of 3.
- Although some redundancy was available, the function acted as a hub: upstream failures in Receive mode change command or Respond to faults directly compromised mission continuity.
- Its repeated occurrence across chains marked it as a mission-critical dependency requiring attention and mitigation.

Example 3: Measure temperature

- Thermal monitoring functions also featured prominently. Measure Temperature appeared in two chains (Thermal Control and Perform X-ray observations) and reached a worst-case severity of P3 (Critical).
- Its $FRS_{min} = 1$ confirmed the absence of redundancy, and its failure directly affected data storage and thermal regulation.

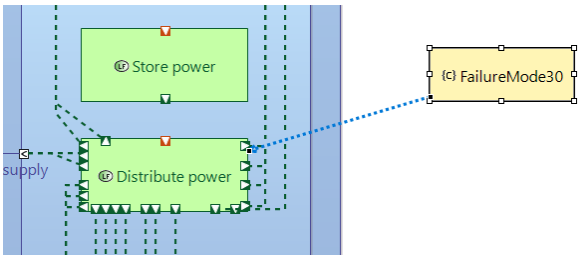
- Although its impact was more localised than Distribute Power, it was classified as a high-risk single-point failure in the thermal subsystem.

Example 4: Maintain Concentrator–Detector Alignment

- Certain functions emerged as mission-specific risks. Maintain Concentrator–Detector Alignment appeared only in the Perform X-ray observations chain, but with a severity of P3 (Critical). Misalignment immediately compromised photon focusing, with no redundancy available.
- Although limited in scope, this function represents a critical vulnerability for the scientific mission objective.

FMEA Population

The Framework generates a populated Functional FMEA directly from the MBSE Model. Each row corresponds to a logical function and its failure mode, effect and severity derived from the risk assessment layer. As an example, Figure 6.13a shows the failure mode representation as constraints, and its corresponding populated entry for 'Distribute Power' is shown in Figure 6.13b. The failure of this function affects several other functional chains, and the corresponding values from the analysis are shown. Fields like 'Detectability' and 'Likelihood' are not populated and must be done manually by the user as discussed in Section 3.7.3



(a) Failure Mode constraint connected to "Distribute power" function.

(Constraint) FailureMode30

Name	Value
Failure_Analysis	
Failure_characteristics	
Detectability	Unknown
Likelihood	[Enum]
Mitigation type	None
Failure Mode ID	FM30
Failure Mode Description	Failure of Distribute power
Severity	3 Critical
Effect	Loss of objective: core science/control input lost to a primary.
Mitigation	Add a secondary core source/path for the impacted primaries (...

(b) Property view of `FailureMode30` with characteristics.

Figure 6.13: Failure Mode modeling in Capella: (A) graphical linkage and (B) detailed attributes.

The same process is repeated for all functions in the model, yielding a complete Functional FMEA consistent with ECSS standards. The demonstration confirms that the framework is able to automatically generate multiple design-related fields from analysis outputs, reducing manual effort while remaining consistent across chains.

Summary

Chapter 6 demonstrates the framework in practice by applying it to the NEBULA-Xplorer MBSE model and progressively layering fault detection, failure analysis, and risk assessment. The results show that the framework can automatically highlight critical vulnerabilities such as power efficacy, bottlenecks, and alignment risks, while producing consistent outputs like functional FMEA tables. By embedding risk-relevant attributes directly into the MBSE model and coupling them with automated Python analyses, the framework reduces manual effort and ensures traceability across functional chains. Overall, the chapter confirms that the approach delivers actionable, ECSS-aligned insights early in the design phase, proving both its feasibility and value for decision-making.

7

Framework Validation

This chapter validates the framework by assessing it against the defined requirements and through the FEMMP evaluation. It establishes the framework's strengths, identifies limitations, and positions it for refinement and broader applicability.

7.1. Requirements Coverage

Validation of the proposed framework is performed against the set of functional and non-functional requirements defined in Chapter 3. These requirements form the basis of what the framework is expected to deliver (FR-01 to FR-05) and how it should deliver those results (NFR-01 to NFR-06).

Table 7.1: Functional Requirement Compliance

ID	Req. Name	Compliance	Section	Implementation Evidence
FR-01	Fault detection	**Yes**	6.3	Rule-based detection layer identified redundancy gaps, and interface dependencies in the Capella logical architecture.
FR-02	MBSE Model	**Yes**	6.2	Operated directly on Capella logical model; processed 74 Logical Functions and 691 ports. Demonstrated integration with Python4Capella.
FR-03	Findings dataset	**Yes**	6.5	Generated structured outputs (console and MBSE Model) listing element, type, effect, and rationale. Includes persistent IDs for traceability to functions.
FR-04	Failure Analysis	**Yes**	6.4	Performed SPOF detection, chain survivability tests, and cascade dependency analysis for the 5 functional chains.
FR-05	Risk Assessment	**Yes (partial automation)**	6.5	Severity aggregation, mitigation ranking, and FMEA population demonstrated. Likelihood left manual, consistent with early-phase data availability.

Requirement coverage is demonstrated by mapping each requirement to concrete evidence obtained from the implementation and case study results. Coverage is shown through a compliance matrix, linking each FR/NFR to the specific layer of the framework and to the outputs presented in Chapters 3 and 6. This process ensures that all previously defined requirements are addressed by the framework in practice. Tables 7.1 and 7.2 discuss the requirement compliance.

Table 7.2: Non-Functional Requirement Compliance

ID	Req. Name	Compliance	Implementation Evidence
NFR-01	Terminology	**Yes**	Uses Capella/ECSS terms (function, exchange, redundancy, SPOF). Ensures interpretability by system engineers.
NFR-02	Explainability	**Yes**	Each finding cites the rule and evidence element (e.g., "Power severed → disables Change spacecraft mode").
NFR-03	Determinism	**Yes**	Multiple re-runs on the same model yield identical outputs, confirming determinism.
NFR-04	Robustness	**Yes (validated)**	Missing radiation rule was flagged as a warning without halting execution, showing graceful degradation.
NFR-05	Usability	**Yes**	Rule set configurable for mission context; extensible in Python.
NFR-06	Traceability	**Yes**	Findings map to originating Capella element IDs; retained in structured dataset.

The framework satisfies all defined functional and non-functional requirements. Together, these results provide evidence that the framework operates as intended and delivers an ECSS-aligned, MBSE-integrated risk analysis suitable for early design phases.

7.2. FEMMP Evaluation

The previous sections evaluated the framework's compliance against two critical criteria: its ability to satisfy functional requirements and its effectiveness in addressing non-functional requirements. However, the usefulness of such a framework also depends on its suitability as part of an MBSE methodology. In other words, it is not enough that the framework produces analyses; it must also integrate with processes, tools, and practitioners in a coherent manner.

To assess this, the **Framework for the Evaluation of MBSE Methodologies and Processes (FEMMP)** developed by Weilkiens et al. was adopted [51]. FEMMP provides a structured catalog of evaluation criteria that allow MBSE end-users to benchmark methodologies against a common set of expectations. These criteria cover aspects such as essentials, practicality, efficiency, usability/experience, and support. FEMMP takes a system-level view, focusing on the overall methodology, its process integration, its implementation in tools, and its applicability to case studies.

Although FEMMP is originally meant for established MBSE methodologies, its use here is justified on two grounds:

- The framework developed in this thesis is not a standalone methodology but an **MBSE extension**. Applying FEMMP allows its strengths and limitations to be highlighted relative to a recognised standard.
- FEMMP's broad coverage ensures that evaluation is complete and transparent, helping to identify both the capabilities as well as the areas beyond its current scope.

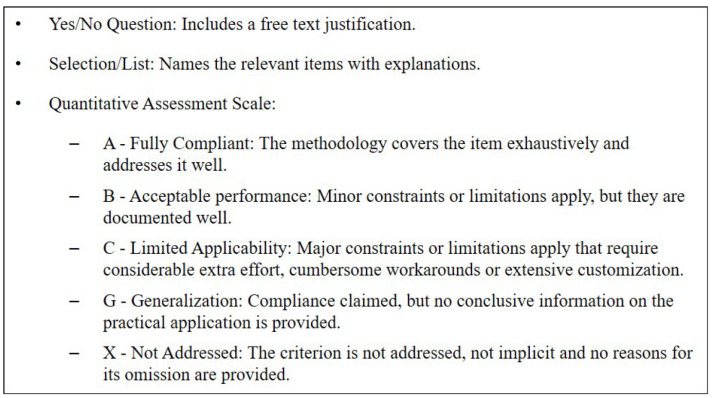


Figure 7.1: FEMMP Criteria types [52]

The FEMMP criteria are weighted based on four different metrics, depending on relevance (Figure 7.1) [51]. For this thesis, the focus is on evaluation criteria most applicable to process integration, model quality, and tool support in early design. This ensures clarity while avoiding unnecessary complexity at this stage of validation. Table 7.3 presents the evaluation of the framework using FEMMP. A complete overview of the FEMMP evaluation criteria and questions can be found in Appendix C

Table 7.3: Evaluation of Framework using FEMMP

Title	ID	Response Type	Response	Evaluation
Learning Curve	G-A01	Scale	B	Requires MBSE and Python knowledge; Manageable for users with some experience.
Suitability for Beginners	G-A02	Scale	C	Not suitable for complete beginners; assumes prior system modelling experience.
Training Effort	G-A03	Statement	–	Needs training in MBSE + custom scripts; effort moderate for first-time users.
Industry Domains	G-B01	Statement	–	Tailored for space missions, adaptable to other domains by changing the severity scale standard.
Support of Standards	G-B03	Statement	–	Aligns with ECSS severity & FMEA; partial ISO coverage.
MBSE–SE Information Exchange	I-B02	Scale	A	Results exportable outside using CSV; seamless cross-tool exchange.
Philosophy (distinction of elements)	L-B01	Yes/No	Yes	Functions, exchanges, and failure modes clearly separated in Capella.

Continued on next page

Table 7.3: Evaluation of Framework using FEMMP (Continued)

Modelling Features	M-B01	List	Functional chains, failure modes, severity, FMEA	Covers major failure/risk constructs needed for system-level RAMS.
Modelling ISO 15288	M-B02	Scale	B	Covers architecture & analysis; lifecycle coverage partial.
Meta-Model	M-B04	Yes/No	No	Relies on existing MBSE tool meta-model; no independent metamodel developed.
ISO Standard Coverage	P-B01	List	ECSS FMEA, ECSS phase A/B	Provides partial alignment with standards but not certified.
Reference Frameworks	P-B02	List	ECSS FMEA/FMECA, Risk-informed design	Leverages existing frameworks in methodology.
Precision	T-B01	Scale	A	Deterministic, rule-based analysis with traceable evidence.
Expert Perspectives	T-E01	Scale	B	Expert opinions to be integrated through change in design decisions and likelihood data; semi-automated.
Reporting	T-E02	Scale	A	Structured reports and failure tables.
Consistency Checks	T-E04	Yes/No	Yes	Validates data-completeness, flags missing or broken links.
Navigation	T-E05	Scale	A	Clear in the MBSE tool; outputs from scripts offer clear recommendations.
User Interface	T-U03	Scale	B	Outputs are text/console-based; no interactive GUI beyond Capella.
Integration	L-P02	Scale	C	No integration with external reliability/thermal/RAMS tools.
Scalability	M-P01	Scale	B	Handles small-to-medium spacecraft models; large-scale untested.
Redundancy	M-P04	Statement	–	Prevents duplication by storing outputs within the same Model.
Consistency	P-P01	Yes/No	Yes	Direct mapping of failure modes to originating model elements.
Connectivity (API/Exchange)	T-P01	Scale	B	Uses Python4Capella API; can be generalized beyond Capella.
Reusability	T-P02	Yes/No	Yes	Scripts are currently mission-specific; require tailoring for reuse.
Documentation	G-S01	Scale	B	Documented in thesis, similar to a user guide.
Training	G-S02	Scale	C	No structured training; self-study required.
Support	T-S01	Scale	C	No vendor support; only community forums.

Discussion of FEMMP Evaluation Using the FEMMP Evaluation, several important insights can be derived regarding its capability and limitations. The evaluation shows that the framework performs strongly in the Essentials category. These include the criterion, Learning Curve, Suitability, Training Effort, and Standards. Since the framework can be used with any MBSE tool and supports risk aggregation across mul-

tiple functional chains, foundational capabilities expected from a risk management methodology are realised.

From a practicality perspective, the framework is clearly aligned with ECSS FMEA standards, but coverage of ISO 15288 [53] is partial. It is currently tailored to small-to-medium satellite missions, meaning scalability to large or multi-satellite constellations remains unproven but should be possible. Reusability to any domain is easily possible but requires tailoring of the scripts.

The Efficiency criterion of Precision, Redundancy, Consistency, and Connectivity is also evaluated for the framework. Reporting is structured, connectivity is supported through Python4Capella APIs, and redundancy of outputs is avoided. From a usability and experience lens, the evaluation is more mixed. Navigation inside Capella is straightforward, but the framework doesn't have a dedicated GUI apart from the Risk Assessment constraint diagram, which may limit accessibility for non-expert users. The learning curve is manageable for MBSE users but not suitable for those without prior modelling knowledge.

Lastly, the support category is the weakest at this point, which was expected. Since this is a thesis project, its documentation in current form is sufficient for research use but would require further development to support widespread industrial adoption. Overall, the FEMMP evaluation highlights a framework that is strong on methodological rigour and precision, but limited in usability, scalability, and support. The evaluation also identifies specific improvement areas, which are taken forward into the conclusions and recommendations chapter.

Summary

Chapter 7 validates the framework by mapping its performance against the functional and non-functional requirements defined earlier, showing that all core objectives were met. The framework was further benchmarked using the FEMMP evaluation, which confirmed strengths in methodology and traceability, while identifying limitations in scalability, and support. These results establish that the framework is not only technically feasible, providing structured, explainable, and repeatable risk insights within an MBSE workflow. At the same time, the evaluation highlights areas for refinement, particularly in extending applicability beyond the current mission context.

8

Conclusion

The objective of this thesis project was to fulfill the following:

To develop a risk management framework integrated with MBSE to improve early-phase design fault identification and risk assessment for the NEBULA-Xplorer mission.

The research questions defined in Chapter 1 are addressed by the work as follows.

1. How can MBSE models be developed to represent fault-critical information necessary for early-stage fault identification?

This work demonstrated that fault-critical information can be embedded directly within the MBSE model through the definition of dedicated attribute sets. This has been described in sections 3.4 and 6.2. Design attributes such as power demand or radiation tolerance were attached to functions and exchanges, while a parallel "Failure Characteristics" attribute family was introduced to capture failure modes, severity, and mitigation. This ensured that the architecture contained both design-oriented and risk-oriented descriptors. By maintaining a clear separation between functional architecture and failure annotations, the model could support both system design and failure analysis without duplication or inconsistency.

2. What patterns of faults or vulnerabilities can be identified early in the design process?

Through the development of automated fault rules, the framework identified three categories of early vulnerabilities:

i Data incompleteness (missing attributes)

ii Performance inadequacy (power and radiation tolerance shortfalls)

iii Structural vulnerabilities (single points of failure, hubs in functional chains)

Demonstrations on the Perform X-ray Observations chain revealed critical dependencies such as reliance on 'Distribute Power' and 'Store and manage mission data onboard.' The method showed that early-phase modelling can highlight bottlenecks and fragility, well before all hardware design details are available (Section 6.3 and 6.4).

3. How can external analysis methods be integrated with MBSE for failure and risk assessment?

The framework integrated external analysis using Python4Capella scripts that extracted, processed, and reintegrated analysis results back into the MBSE environment (Section 5.2). Graph theory-based metrics such as min-cut values, fan-in/out coupling, and resilience scores were computed externally but written back as attribute values within the model. This enabled the semi-automatic generation of a Functional FMEA aligned with ECSS standards. The integration preserved the existing MBSE model, and external tools helped implement the different layers of the framework.

4. How effective is the proposed framework in identifying and prioritising system-level risks?

The demonstration showed that the framework is effective in revealing and prioritising risks in a structured manner (Section 3.7). Critical functions and dependencies were consistently flagged across chains, with severity aggregated into system-level profiles. The automated prioritisation produced a mitigation ranking. For example, identifying power distribution and thermal control as high-risk enablers, while the calibration function was classified as low-impact. Validation using FEMMP further confirmed that the methodology satisfies essential MBSE evaluation criteria, though limitations remain in terms of likelihood estimation, GUI maturity, and scalability (Section 7.2). By addressing the four research questions, the work showed that MBSE can extend beyond descriptive architectures to provide actionable evidence for risk-informed decision-making. The demonstration confirmed that the approach is capable of uncovering single points of failure, and producing structured recommendations in the early design phases.

While the framework is not without limitations, it constitutes a step toward bridging system modelling and Risk Management practices in a unified environment. The following chapter discusses these limitations in detail and outlines recommendations for future work and improvements.

9

Recommendations

The current work has shown that it is feasible to embed fault detection, failure analysis, and risk assessment directly in an MBSE workflow. However, the current implementation represents a first step, and several aspects can be improved to broaden its applicability and value.

Firstly, the analysis is currently limited to single-failure modes as used in traditional Failure FMEA. While this provides useful early insights, it does not capture cascading or a combination of multiple failures. Future work should therefore extend the approach to include this, enabling a more comprehensive analysis of failures.

Secondly, the framework presently models function loss as a binary condition. Many failures manifest instead as degraded or partial performance. Representing such graded states of functionality would make the analysis outcomes more realistic.

Thirdly, the current analysis is static in nature, as is expected with the design maturity in early phases. Timing, latency, and dynamic failure propagation are not taken into account. Integrating time-based parameters into functional exchanges, or coupling with simulation environments, would allow assessment of delay-critical behaviours. Similarly, capacity and load aspects can be incorporated to extend rule checks for performance validation.

Fourthly, while the framework automatically consolidates a Functional FMEA, the likelihood of occurrence and detectability remains a manual entry. Introducing historical reliability data or expert methods could support more complete early-phase risk assessments and extend them to FMECA.

Finally, the FEMMP evaluation highlighted gaps in usability and integration. Documentation and training material should be expanded, a more accessible user interface should be developed, and extend the analysis to would ensure smoother transition to detailed design stages.

These recommendations provide a roadmap for maturing the framework. Each would enhance its ability to support system design in early phases, moving it further closer to a methodology that balances MBSE with risk-informed design.

References

[1] Joseph W Hamaker. "But what will it cost? The history of NASA cost estimating". In: *Readings in Program Control* 6103 (1994), p. 25.

[2] Stephen A Jacklin. *Small-satellite mission failure rates*. Tech. rep. 2019.

[3] Michael A Johnson et al. "The small satellite reliability initiative: A public-private effort addressing smallsat mission confidence". In: *AIAA Small Satellite Conference*. GSFC-E-DAA-TN58616. 2018.

[4] Pierre De Saqui-Sannes et al. "A taxonomy of MBSE approaches by languages, tools and methods". In: *IEEE Access* 10 (2022), pp. 120936–120950.

[5] Louis Thomas et al. "Integration of reliability, availability, maintainability and safety in model-based systems engineering". In: *CEAS Space Journal* 16.2 (2024), pp. 251–261.

[6] SRON Netherlands Institute for Space Research. *NEBULA-Xplorer Mission*. `https://www.sron.nl/en/missions/in-development/missies-in-development-nebula-xplorer/`. Accessed: 2025-10-10. 2025.

[7] European Cooperation for Space Standardization (ECSS). *ECSS-S-ST-00-01C Rev.1: Glossary of terms*. `https://ecss.nl/standard/ecss-s-st-00-01c-rev-1-glossary-of-terms-11-october-2023/`. Accessed: 12 September 2025. 2023.

[8] INCOSE. *INCOSE systems engineering handbook*. John Wiley & Sons, 2023.

[9] Steven R Hirshorn, Linda D Voss, and Linda K Bromley. *Nasa systems engineering handbook*. Tech. rep. 2017.

[10] Technique Apply techniques such as Built. "Fault-Detection, Fault-Isolation and Recovery (FDIR) Techniques". In: ().

[11] Julien Marzat et al. "Model-based fault diagnosis for aerospace systems: a survey". In: *Proceedings of the Institution of Mechanical Engineers, Part G: Journal of aerospace engineering* 226.10 (2012), pp. 1329–1360.

[12] Alireza Alikhani and Ghasem Sharifi. "Application of a model-based fault detection approach on a spacecraft". In: *International Journal of Reliability, Risk and Safety: Theory and Application* 3.2 (2020), pp. 19–26.

[13] Mahsa Azadmanesh et al. "Data-Driven Fault Diagnosis in Spacecraft: an Overview". In: (), p. 48.

[14] Fei Teng et al. "A knowledge service framework for fault diagnosis of low-earth orbit satellite constellation". In: *2023 IEEE International Conference on Web Services (ICWS)*. IEEE. 2023, pp. 669–676.

[15] Eric Pesola et al. "A Hybrid Model-Based and Data-Driven Framework for Automated Spacecraft Fault Detection". In: *Annual Conference of the PHM Society*. Vol. 15. 1. 2023.

[16] Ganhua Li et al. “A fault prediction system for the complex satellite management system based on rule and fault tree”. In: *2022 IEEE 11th Data Driven Control and Learning Systems Conference (DDCLS)*. IEEE. 2022, pp. 1196–1200.

[17] *ECSS-M-ST-10C Rev.1 – Project Planning and Implementation*. Tech. rep. ECSS-M-ST-10C Rev.1. 6 March 2009. Noordwijk, The Netherlands: European Cooperation for Space Standardization (ECSS), Mar. 2009. URL: https://ecss.nl/standard/ecss-m-st-10c-rev-1-project-planning-and-implementation/.

[18] Lorenzo Bitetti et al. “Model Based approach for RAMS analyses in the Space domain with Capella open-source tool”. In: *International Symposium on Model-Based Safety and Assessment*. Springer. 2019, pp. 18–31.

[19] C Preyssl, R Atkins, and T Deak. “Risk management at ESA”. In: *ESA bulletin* 97 (1999), pp. 64–68.

[20] J Marcoux. “The ECSS Product Assurance Standards for Space Products”. In: *Product Assurance Symposium and Software Product Assurance Workshop*. Vol. 377. 1996, p. 57.

[21] Terje Aven. “Risk assessment and risk management: Review of recent advances on their foundation”. In: *European journal of operational research* 253.1 (2016), pp. 1–13.

[22] European Cooperation for Space Standardization (ECSS). *ECSS-Q-ST-30C Rev. 1 – Space product assurance – Dependability*. Standard. Date: 15 February 2017. Noordwijk, The Netherlands, Feb. 2017. URL: https://ecss.nl/standard/ecss-q-st-30c-rev-1-space-product-assurance-dependability-15-february-2017/.

[23] Alexander Kossiakoff et al. *Systems engineering principles and practice*. Vol. 83. John Wiley & Sons, 2011.

[24] Morayo Adedjouma et al. “From document-based to model-based system and software engineering”. In: *Proceedings of the OSS4MDE* (2016).

[25] Albert Solberg. *Model based systems engineering (MBSE)*. 2018.

[26] Joe Gregory et al. “The long and winding road: MBSE adoption for functional avionics of spacecraft”. In: *Journal of Systems and Software* 160 (2020), p. 110453.

[27] Irfan Javid. “Evaluating ARCADIA using the FEMMP framework”. PhD thesis. Technische Hochschule Ingolstadt, 2020.

[28] Mohammad Chami et al. “Towards solving MBSE adoption challenges: the D3 MBSE adoption toolbox”. In: *INCOSE International Symposium*. Vol. 28. 1. Wiley Online Library. 2018, pp. 1463–1477.

[29] Jeff A Estefan et al. “Survey of model-based systems engineering (MBSE) methodologies”. In: *Incose MBSE Focus Group* 25.8 (2007), pp. 1–12.

[30] Lenny Delligatti. *SysML distilled: A brief guide to the systems modeling language*. Addison-Wesley, 2013.

[31] Nadia A Tepper. “Exploring the use of Model-based Systems Engineering (MBSE) to develop systems architectures in naval ship design”. PhD thesis. Monterey California. Naval Postgraduate School, 2010.

[32] Loyd Baker et al. “Foundational concepts for model driven system design”. In: *INCOSE Model Driven System Design Interest Group* 16 (2000), pp. 15–16.

[33] Brian London. “A model-based systems engineering framework for concept development”. PhD thesis. Massachusetts Institute of Technology, 2012.

[34] Marco Bozzano et al. “Spacecraft early design validation using formal methods”. In: *Reliability engineering & system safety* 132 (2014), pp. 20–35.

[35] Faïda Mhenni, Nga Nguyen, and Jean-Yves Choley. “SafeSysE: A safety analysis integration in systems engineering approach”. In: *IEEE Systems Journal* 12.1 (2016), pp. 161–172.

[36] Nancy J Lindsey, Mahdi Alimardani, and Luis D Gallo. “Reliability analysis of complex NASA systems with model-based engineering”. In: *2020 Annual reliability and maintainability symposium (RAMS)*. IEEE. 2020, pp. 1–8.

[37] Geoffrey Biggs, Takeshi Sakamoto, and Tetsuo Kotoku. “A profile and tool for modelling safety information with design information in SysML”. In: *Software & Systems Modeling* 15.1 (2016), pp. 147–178.

[38] Matteo Camilli et al. “Towards risk modeling for collaborative AI”. In: *2021 IEEE/ACM 1st Workshop on AI Engineering-Software Engineering for AI (WAIN)*. IEEE. 2021, pp. 51–54.

[39] Arthur Henrique de Andrade Melani and Gilberto Francisco Martha de Souza. “Obtaining fault trees through SysML diagrams: A MBSE approach for reliability analysis”. In: *2020 Annual reliability and maintainability symposium (RAMS)*. IEEE. 2020, pp. 1–5.

[40] Zhaofeng Huang et al. “MBSE-assisted FMEA approach—Challenges and opportunities”. In: *2017 Annual Reliability and Maintainability Symposium (RAMS)*. IEEE. 2017, pp. 1–8.

[41] SEBoK Editorial Board. *The Guide to the Systems Engineering Body of Knowledge (SEBoK), v. 2.12*. Ed. by N. Hutchison. `https://www.sebokwiki.org`. BKCASE is managed and maintained by the Stevens Institute of Technology Systems Engineering Research Center, the International Council on Systems Engineering, and the IEEE Systems Council. Accessed: 2025-10-11. Hoboken, NJ, 2024.

[42] Bryan M O’Halloran et al. “A graph theory approach to predicting functional failure propagation during conceptual systems design”. In: *Systems Engineering* 24.2 (2021), pp. 100–121.

[43] Kristina Salgado. “Accretion Disk Spectra of Black Hole X-ray Binaries”. PhD thesis. University of Colorado at Boulder, 2015.

[44] SRON. *NEBULA-Xplorer System Requirements Document (SRD)*. Tech. rep. SRON-NEBULA-SP-2024-011. Internal report. Utrecht, The Netherlands: Netherlands Institute for Space Research (SRON), 2024.

[45] Paloma Maestro Redondo, Peter Dubock, and Gwendolyn Kolfschoten. “A TOOL TO SUPPORT CONCURRENT REVIEWING OF CONCURRENT ENGINEERING PRODUCTS”. In: (2024).

[46] Jean-Luc Voirin. *Model-based system and architecture engineering with the arcadia method*. Elsevier, 2017.

[47] Pascal Roques. *Systems architecture modeling with the Arcadia method: a practical guide to Capella*. Elsevier, 2017.

[48] Capella MBSE Tool. *The Arcadia Methodology*. Accessed on 27 October 2023. 2023. URL: https://mbse-capella.org/arcadia.html (visited on 10/27/2023).

[49] NEBULA-Xplorer TEAM. *NEBULA-Xplorer Design Description Document*. Tech. rep. SRON-NEBULA-SP-2025-024. Internal Document; Prepared for the NEBULA-Xplorer mission. Checked by Martin Grim, PA agreed by Nathalie Gorter, Authorised by Martin Grim. SRON Netherlands Institute for Space Research, July 2025.

[50] ESA ESTEC. "Margin philosophy for science assessment studies". In: *ESA UNCLASSIFIED* (2012).

[51] Marco Di Maio et al. "Evaluating MBSE methodologies using the FEMMP framework". In: *2021 IEEE International Symposium on Systems Engineering (ISSE)*. IEEE. 2021, pp. 1–8.

[52] Shashank P Alai. *Evaluating arcadia/capella vs. oosem/sysml for system architecture development*. Purdue University, 2019.

[53] ISOI EC. *INTERNATIONAL ISO/IEC STANDARD 15288*. 2002.

A

Appendix: NEBULA-Xplorer Mission

The requirements in the MRD are categorised according to the following structure.

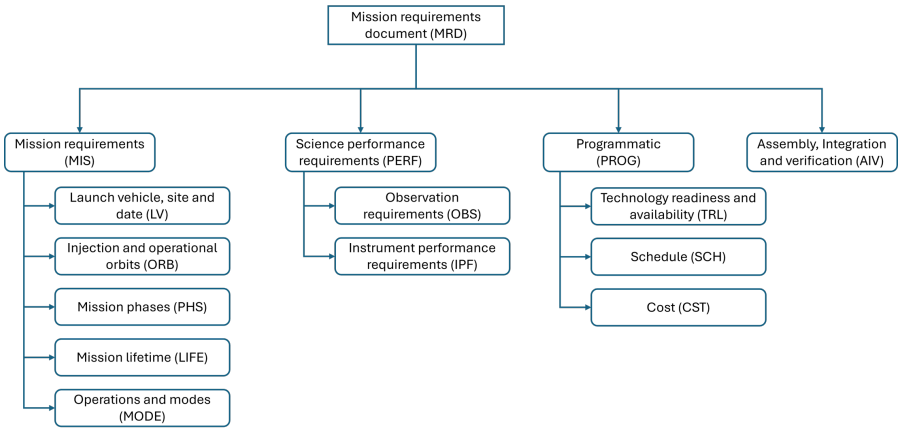


Figure A.1: MRD Organisation [44]

An overview of the internal and external configuration of the NEBULA-Xplorer satellite as captured during the SRR are presented below:

Table A.1: System Components

ID	Component
1	X-ray instrument
2	2x Battery packs
3	Star tracker module
4	Bus subsystem mounting rack: PDU, ACU, PMU, and 2x S-band transceivers
5	Dawn 4U Cubedrive
6	S-band Antenna
7	GNSS Receiver
8	Sun sensor
9	Instrument electronics module
10	2x Magnetometers
11	SADM
12	Reaction wheel module
13	OBC, TCM, and inertial measurement unit

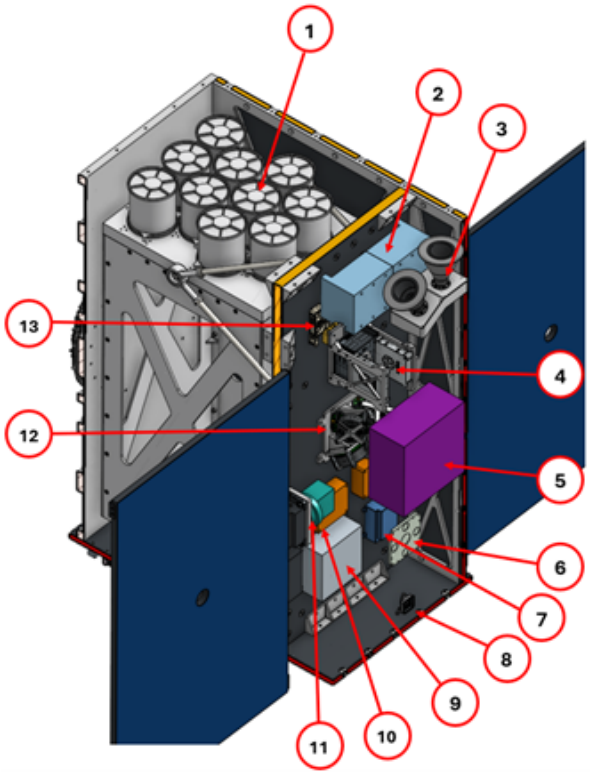


Figure A.2: NEBULA-Xplorer Internal Configuration

Table A.2: Externally Mounted System Components

ID	Component
1	Star tracker module
2	Propulsion module
3	2x S-band antennas
4	4x Coarse sun sensors
5	3x Magnetorquers

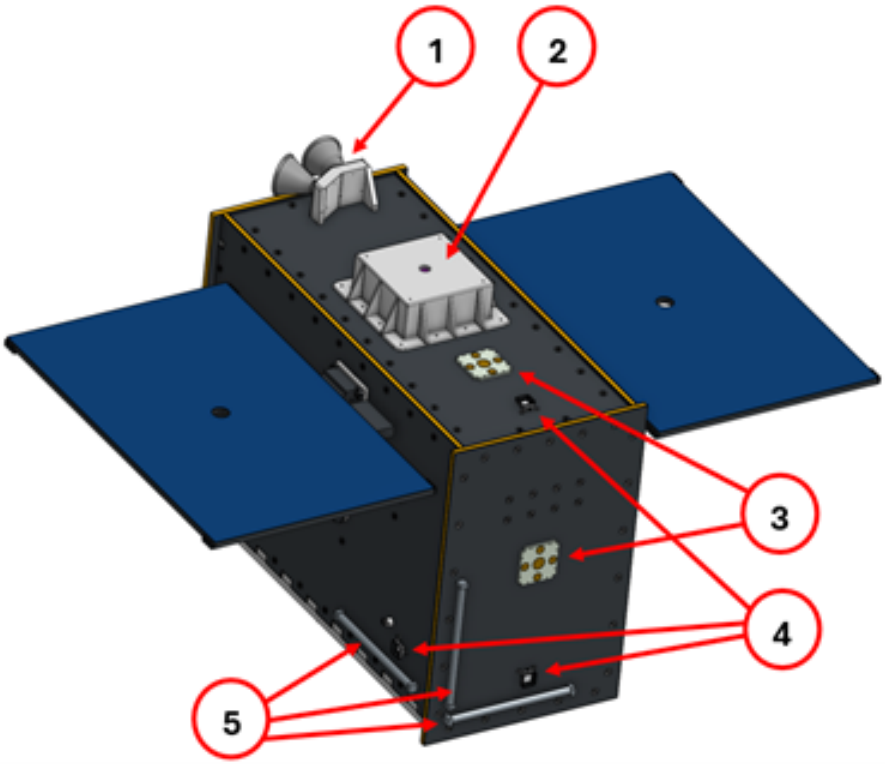


Figure A.3: NEBULA-Xplorer External Configuration

B

Appendix: MBSE Model &Diagrams

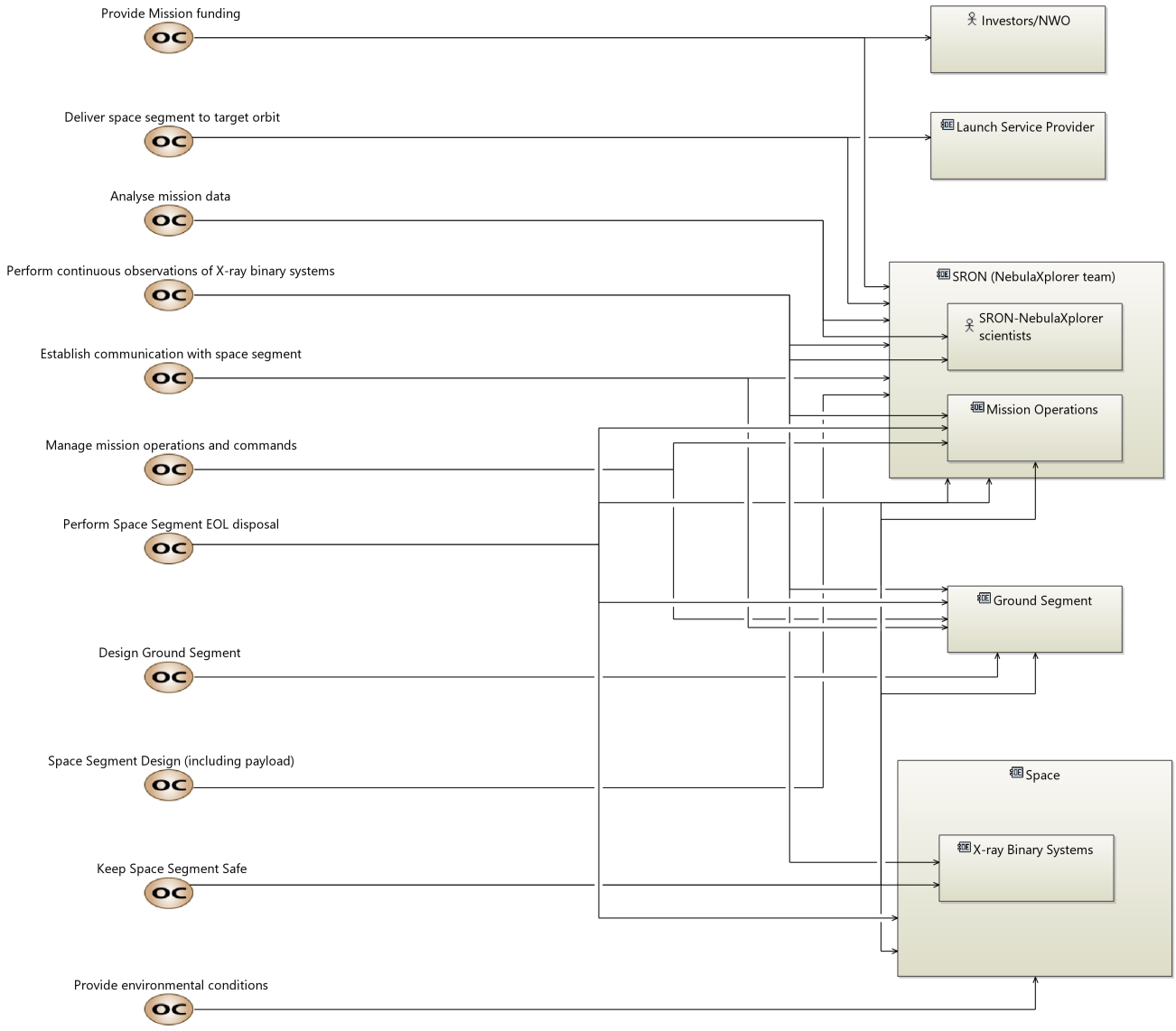


Figure B.1: Operational Capabilties Diagram [OCB]

Repository Access

All scripts, models, and analysis workflows developed during this thesis are maintained in a Git repository. The repository contains:

- Python scripts for framework automation
- Capella model files

The repository link: `https://github.com/gargipantoji/Nebulaxplorer.git`
The repository is private and access can be provided upon request to the author.

C

Appendix: FEMMP Criteria

Title	Description	ID	Wgt	Aspect	Category	Response Type	Former ID
Learning Curve	How steep does the learning curve feel?	G-A-01	3	General	Adopting	Scale	
Suitability for Beginners	How easy to understand is the methodology for (MB)SE novices?	G-A-02	3	General	Adopting	Statement	
Training Effort for SEs	How much training does it take for experienced SEs to implement the methodology correctly?	G-A-03	2	General	Adopting	Statement	
Industry Domains	Which industry domains does the methodology support particularly well/ has it been developed for?	G-B-01	1	General	Basics		
Creativity Support	How does the methodology foster a creative environment for the systems engineering team?	G-B-02	2	General	Basics	Statement	
Support of Other Standards	Does the methodology also support other standards or norms? If so, how well?	G-B-03	1	General	Basics	Statement	
Process Info Capture	How well does the tool capture the information generated throughout the process?	I-B-01	3	Information	Basics	Scale	
MBSE-SE Info Exchange	How well does the tool facilitate the exchange of the information between the MBSE domain and the non-MBSE tools	I-B-02	3	Information	Basics	Scale	
Philosophy	Are Model Elements clearly distinguished from Diagram Elements (separation of content from representation)?	L-B-01	3	Language	Basics	Yes / No	A-02-L
Modelling Features	What modelling features and approaches are included from the reference framework ISO 42010 (Architecture)	M-B-01	2	Model	Basics	List	
Modelling ISO 15288	How well does the MBSE methodology support modelling of the ISO 15288 processes?	M-B-02	3	Model	Basics	Scale	
Abstraction	How well does the model support keeping it abstract through multiple levels?	M-B-03	2	Model	Basics	Scale	
Meta-Model	Does the methodology provide a (semantic) meta-model, ontology or similar?	M-B-04	3	Model	Basics	Yes / No	
ISO Standard	What process steps of ISO 15288 are covered?	P-B-01	3	Process	Basics	List	A-00-P
Framework	What views from the reference framework are used (MODAF, DODAF...)?	P-B-02	3	Process	Basics	List	A-01-P
Precision	How precisly is the process implemented in terms of semantics and sequence, i.e. is it strongly enforced, or can 'wildcard' be used as 'work arounds' that would reduce the model quality?	T-B-01	3	Tool	Basics	Scale	A-03-T
Information Security	How secure is the data/information exchange between parties	T-B-02	2	Tool	Basics	Scale	
Perspectives	To what level is the creation of experts' perspectives automated	T-E-01	1	Tool	Efficiency	Scale	C-00-T
Reporting	How quickly are standard/custom reports, is design documentation created	T-E-02	1	Tool	Efficiency	Scale	C-02-T
Admin	How well does the tool help to minimise work that isn't creating any value	T-E-03	1	Tool	Efficiency	Scale	C-03-T
Checking	Does the tool support consistency checking of the model? (Automated detection of wrong content and/or formats, flagging of , 'loose ends' etc.)?	T-E-04	2	Tool	Efficiency	Yes / No	C-01-T
Navigation	How easy is it to find the correct model element (are elements links, users guided in the process, information well aggregated, need to 'jump' screens)?	T-E-05	2	Tool	UX	Scale	D-00-T
Intuitition	How intuitive is the tool to work with (compliance with UX conventions, standard tool reactions e.g. tool tips, double/right click, drag&drop, delete, Keyboard shortcuts, spell check, familiar operations e.g. as MS-Office)?	T-U-01	2	Tool	UX	Scale	D-01-T
View	How easy is it to configure the UX dynamically (define a matrix with sorting & filtering of columns and rows, store customised view, annotation, comment)?	T-U-02	1	Tool	UX	Scale	D-02-T
UI	How readable is the UI (good use of screen estate and colour, zoom, can fonts and sizes be changed, is information well presentedâ€¦)?	T-U-03	1	Tool	UX	Scale	D-03-T
Multi-User Environment	How well does the tool support the distribution of the work among different parties.	T-U-04	2	Tool	UX	Scale	
Scope	For what engineering purpose is the methodology suited(innovation, improved products, refactoring, reverse engineering, etc.)?	G-P-01	2	General	Practicality	List	B-02-M
Tailoring	How easy is it to tailor the methodology (add, delete or change processes or process steps, object definitions or toggle tool features on and off)?	G-P-02	3	General	Practicality	Scale	B-03-M

Complexity	How often is the methodology 'interrupted' by relying on external processes and/or non-integrated tools, e.g. by a 'paper-review'?	G-P-03	2	General	Practicality	Scale	B-06-M
Simulation	How well does the methodology provide for an integrated simulation of the various model abstractions?	G-P-04	2	General	Practicality	Statement	B-09-M
Language	What Modelling Language is used (If NOT SysML: How well does it define the real-world semantics of the engineering, are elements strictly typed, is their meaning unambiguous, do they have a defined purpose etc.)?	L-P-01	3	Language	Practicality	List	B-00-L
Integration	How well can the model be integrated with specialty engineering models	L-P-02	1	Language	Practicality	Scale	B-08-L
Scalability	How well does the model scale (suitable for large projects, 'grows' with time without becoming cumbersome, does it require partitionaing e.g. in a tree)?	M-P-01	3	Model	Practicality	Scale	B-01-M
Variants	How well does the methodology support the variant management?	M-P-02	3	Model	Practicality	Scale	B-05-M
Independent View Generation	How well does the methodology support the generation of views that can be read in another MBSE language?	M-P-03	2	Model	Practicality	Scale	
Redundancy	How well does the methodology prevent duplication (of work, model elements, artefacts, communications and reports)?	M-P-04	2	Model	Practicality	Statement	B-10-M
Consistency	Is the process self-contained (are in-/outputs to all steps connected)?	P-P-01	3	Process	Practicality	Yes / No	B-04-P
Connectivity	How easily can the information be exchanged with other tools (what standard API are provided by the tool, what API can be added, Is im-/export open protocols & guided by a wizard, can it be rolled--back, what the quality control mechanism etc.)?	T-P-01	3	Tool	Practicality	Scale	B-07-T
Reuseablity	Does the tool allow to reuse any type of Modelling Element across projects	T-P-02	2	Tool	Practicality	Yes / No	C-04-T
Documentation	How well is the methodology supported (books, manuals, case studies, on-line help, community, websites, interactive support, user feedback etc.)?	G-S-01	3	General	Support	Scale	E-00-M
Training	How well is training supported (availability, consultants, coaches, e-training, background knowledge required)?	G-S-02	1	General	Support	Scale	E-01-M
Support	How well is the tool supported (vendor response times, 24/7 helpline etc.)?	T-S-01	1	Tool	Support	Scale	E-02-T