MODEL-BASED PRODUCT LINE ENGINEERING METHODOLOGY FOR VARIABILITY MANAGEMENT IN SYSTEM ARCHITECTURE MODELS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF INFORMATICS OF
THE MIDDLE EAST TECHNICAL UNIVERSITY
BY

TUANA GÜZEL

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE
IN
INFORMATION SYSTEMS

AUGUST 2024

Approval of the thesis:

MODEL-BASED PRODUCT LINE ENGINEERING METHODOLOGY FOR VARIABILITY MANAGEMENT IN SYSTEM ARCHITECTURE MODELS

Submitted by **TUANA GÜZEL** in partial fulfillment of the requirements for the degree of **Master of Science in Information Systems, Middle East Technical University** by,

Prof. Dr. Banu Günel Kılıç
Dean, **Graduate School of Informatics** _____

Prof. Dr. Altan Koçyiğit
Head of Department, **Information Systems** _____

Prof. Dr. Buyurman Baykal
Supervisor, **Electrical and Electronics Engineering Dept., METU** _____

Examining Committee Members:

Asst. Prof. Dr. Özden Özcan Top
Information Systems Dept., METU _____

Prof. Dr. Buyurman Baykal
Electrical and Electronics Engineering Dept., METU _____

Assoc.Prof. Mustafa Sert
Computer Engineering Dept., Başkent University _____

Date: 06 August 2024

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last name : Tuana Güzel

Signature : _____

ABSTRACT

MODEL-BASED PRODUCT LINE ENGINEERING METHODOLOGY FOR VARIABILITY MANAGEMENT IN SYSTEM ARCHITECTURE MODELS

GÜZEL, Tuana

MSc., Department of Information Systems

Supervisor: Prof. Dr. Buyurman BAYKAL

August 2024, 121 pages

The aerospace industry faces significant challenges in maintaining efficiency and quality in product development due to the increasing complexity of systems, driven by variations arising from competitive market dynamics and diverse customer demands. The development and management of these complex systems, characterized by variants necessitates advanced modeling methodologies such as Model-Based Systems Engineering (MBSE) and Product Line Engineering (PLE). Extending MBSE with PLE approaches, Model-Based Product Line Engineering (MBPLE) uses models to define and manage variability, emphasizing systematic asset reuse. However, practical support for MBPLE is currently limited, particularly outside software-intensive systems. A consistent and scalable approach for an effective variability management is essential, facilitating visualization and traceability of variability across different abstraction levels and development artifacts and detailing this variability in systems architecture models enhance the accuracy of variants. This thesis investigates MBPLE's role in managing variability for large aerospace systems, exploring benefits, challenges, and best practices. The research aims to develop a robust methodology for MBPLE by adapting variability management techniques from PLE to the MBSE context by providing methodological support, modeling techniques, and efficient tools for supporting the variability models to ensure alignment with requirements, design dependencies and features. This methodology is exemplified through its implementation in an unmanned aerial vehicle system and validated against established requirements for effective variant management. By systematically addressing variability across multiple abstraction levels, this thesis aims to optimize business processes, improve product quality, and reduce engineering efforts within the aerospace industry.

Keywords: Model Based Systems Engineering, Product Line Engineering, Model Based Product Line Engineering, Variability Management

ÖZ

SİSTEM MİMARİSİ MODELLERİNDE DEĞİŞKENLİK YÖNETİMİ İÇİN MODEL TABANLI ÜRÜN HATTI MÜHENDİSLİĞİ METODOLOJİSİ

GÜZEL, Tuana

Yüksek Lisans, Bilişim Sistemleri Bölümü

Tez Yöneticisi: Prof. Dr. Buyurman BAYKAL

Ağustos 2024, 121 sayfa

Havacılık endüstrisinde ürün geliştirme süreci, rekabetçi pazar dinamikleri ve farklı müşteri taleplerinden kaynaklı artan sistem karmaşıklığı nedeni ile verimliliğini ve kalitesini kaybetmektedir. Değişkenlerle karakterize edilen bu karmaşık sistemlerin geliştirilmesi ve yönetimi, Model Tabanlı Sistem Mühendisliği (MTSM) ve Ürün Hattı Mühendisliği (ÜHM) gibi ileri modelleme metodolojilerine ihtiyaç duymaktadır. MTSM'yi ÜHM yaklaşımlarıyla genişleten Model Tabanlı Ürün Hattı Mühendisliği (MTÜHM), sistematik varlık yeniden kullanımını vurgulayarak değişkenliği tanımlamak ve yönetmek için modeller kullanır. Ancak MTÜHM'ye yönelik uygulamalar, özellikle yazılım yoğunluklu sistemler dışında şu anda sınırlıdır. Etkili bir değişkenlik yönetimi için tutarlı ve ölçeklenebilir bir yaklaşım esastır; farklı soyutlama düzeyleri ve geliştirme unsurları arasındaki değişkenliğin görselleştirilmesi, izlenebilirliğinin kolaylaştırılması ve bu değişkenliğin sistem mimarisi modellerinde detaylandırılması ile varyantların doğruluğu amaçlanmalıdır. Bu tez, kapsamlı havacılık sistemlerinin değişkenlik yönetiminde MTÜHM'nin rolünü, faydalarını, zorluklarını ve uygulamalarını araştırır. Araştırma, MTÜHM için etkin bir metodoloji geliştirmeyi hedefleyerek, gereksinimlerin, tasarım bağımlılıklarının ve sistem özelliklerinin tutarlığını göz önünde bulundurur, ÜHM değişkenlik yönetimi tekniklerini MTSM bağlamına uyarlar ve değişkenlik modellerini desteklemek için metodolojik destek, modelleme teknikleri ve etkili araçlar sağlar. Önerilen metodoloji, etkin değişken yönetimi için gereksinimlerin belirlenerek, örnek bir insansız hava aracı sisteminde uygulanması ve doğrulanmasıyla gösterilmektedir. Çoklu soyutlama seviyelerindeki değişkenliği sistematik olarak ele alan bu tez, havacılık endüstrisindeki iş süreçlerini optimize etmeyi, ürün kalitesini iyileştirmeyi ve mühendislik çabalarını azaltmayı amaçlamaktadır.

Anahtar Sözcükler: Model Tabanlı Sistem Mühendisliği, Ürün Hattı Mühendisliği, Model Tabanlı Ürün Hattı Mühendisliği, Değişkenlik Yönetimi

To my cats

ACKNOWLEDGEMENTS

I would like to express my deepest gratitude to all those who have supported and contributed to the completion of this thesis.

First of all, I would like to express my great appreciation to Prof. Dr. Buyurman Baykal for all his trust, inspiration, and guidance. His aid and suggestions have been fundamental to this research.

To my family, your love, support, and belief in me have been my foundation throughout this journey. I am deeply grateful for everything you have done to help me reach this milestone.

I also wish to acknowledge A. Serhan Hakkoymaz for encouraging me to start this journey, Davut Çıkrıkcı for teaching and fostering my love for systems engineering, and Mehmet Yılmaz for his unconditional support. Your mentorship has been invaluable, offering wisdom, insight, and a steady hand in times of uncertainty.

To my friends, thank you for pretending to understand my research when I explained it to you for the hundredth time. Your blank stares were a constant source of motivation. I am very fortunate to have wonderful friends who provided endless encouragement, laughs, and distractions just when I needed them the most. Special thanks to Beyza Margi and Ferhat Henden for being there through the ups and downs of this journey. Beyza, your constant encouragement and timely reminders to take breaks have been immensely appreciated and have played a crucial role in maintaining my well-being and focus. Ferhat, my heartfelt thanks go to you for the countless hours spent exploring and brainstorming new ideas, and for our in-depth discussions that spanned from research and career to family and life. Your unwavering support and intellectual companionship have been invaluable.

I am incredibly fortunate to have a wonderful partner, Eril Balcıoğlu. Your limitless patience, enduring love, and belief in me have been my greatest sources of strength. Thank you for standing by my side through every step of this journey, for being my rock during challenging times, and for preparing the delicious meals that fueled both my body and mind. Your presence made the entire process far more enjoyable.

Last but not least, I want to thank me, I want to thank me for believing in me, I want to thank me for doing all this hard work, I want to thank me for having no days off, I want to thank me for, for never quitting, I want to thank me for always being a giver, and trying give more than I receive, I want to thank me for trying do more right than wrong, I want to thank me for just being me at all times.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

ARCADIA	Architecture Analysis & Design Integrated Approach
ATA	Air Transportation Association
CC	Core Capabilities
CR	Core Requirements
CS	Core Subsystems
CVS	Core Variant Subsystems
DoDAF	Department of Defense Architecture Framework
DRSM	Design Science Research Methodology
DSM	Design Structure Matrix
E/O	Electro-Optical
FM	Feature Model
FODA	Feature Oriented Domain Analysis
GCS	Ground Control Station
GPS	Global Positioning System
ICD	Interface Control Document
IEEE	Institute of Electrical and Electronics Engineers
INCOSE	The International Council on Systems Engineering
ISO	International Organization for Standardization
LA	Logical Architecture
LAB	Logical Architecture Blank
MBPLE	Model Based Product Line Engineering
MBSE	Model Based Systems Engineering
NFM	Need Feature Model
NFR	Non-functional Requirement
OA	Operational Analysis
OAB	Operational Architecture Blank
OC	Operational Capability
OOSEM	Object-Oriented Systems Engineering Method
PA	Physical Architecture
PAB	Physical Architecture Blank
PLE	Product Line Engineering
pvSCL	Pure:variants expression language
ROI	Return on Investment
SA	System Analysis

SAB	System Architecture Blank
SCV	Scoping, Commonality, Variability
SE	Systems Engineering
SFM	Solution Feature Model
SM	System Model
SoS	System of Systems
SysML	Systems Modeling Language
UAV	Unmanned Aerial Vehicle
UML	Unified Modeling Language
VC	Variant Capabilities
VR	Variant Requirements
VS	Variant Subsystems

CHAPTER 1

INTRODUCTION

The aerospace industry stands as a cornerstone of modern engineering, characterized by its intricate blend of cutting-edge technology, mandatory regulatory standards, and dynamic global markets. To maintain a competitive edge, aerospace manufacturers must continuously enhance the efficiency of their aircraft development processes while addressing diverse customer needs and demands. As systems grow in scale and complexity, the traditional approach of developing each system from scratch or as isolated components becomes increasingly inefficient.

Product Line Engineering (PLE) has emerged as a strategic approach to mitigate these inefficiencies by leveraging commonalities within a family of products while managing their variabilities (Pohl et al., 2005). PLE involves the systematic use of shared assets and the implementation of a common platform to produce a variety of related products. The product line in aerospace is remarkably diverse, encompassing commercial and military aircraft, unmanned aerial vehicles (UAVs), and an array of subsystems and components. Variability can occur in various system artifacts, including requirements, hardware/software components, subsystems, and interfaces. Effective management of this product line and its variabilities is crucial to ensure consistent quality, cost-efficiency, and compliance with safety and performance standards.

Model-Based Systems Engineering (MBSE) employs systems engineering practices by employing models to deliver a comprehensive architectural representation of the entire system, encompassing system requirements, elements, their interrelationships, behaviors, and functions (INCOSE, 2023). By utilizing these advanced descriptive characteristics, Model-Based Product Line Engineering (MBPLE) offers a structured framework for addressing these challenges, leveraging models as primary artifacts to define and manage variability at different levels of abstraction. MBPLE aims to capture, represent, and effectively manage the variability inherent in a product line, emphasizing the systematic reuse of assets to streamline development processes and enhance product quality. However, the general application of this methodology lacks standardization and predominantly focuses on software-intensive systems. Representing and managing variability in system architecture models at every level of abstraction requires a robust and flexible methodology.

1.1 Motivation of the Thesis

The motivation for this thesis arises from the critical challenges faced in managing variability within complex systems, particularly in the aerospace industry, where precision, reliability, and safety are of utmost importance. Inadequate management of variability can lead to severe consequences, including design flaws, increased costs, and compromised

system integrity. While system architecture models play a crucial role in managing this complexity, MBPLE remains insufficient without a structured and systematic implementation.

System architecture models, which encompass requirements, design specifications, and technical solutions, are inherently complex. This complexity is amplified when variability is introduced, not only due to technical challenges but also because of the diverse concerns and expectations of multiple stakeholders. Variations in the system architecture often reflect these differing perspectives, making it essential for these models to be designed with the capacity to incorporate such variations. Therefore, for effective management of variants, system architecture models must be designed to incorporate these variations with consideration for the various aspects of stakeholders from different backgrounds. These models must ensure that the variability information is accessible and understandable, thereby reducing the risk of errors resulting from this complexity (Voirin, 2018).

Another challenge for variability management arises from more technical aspects like systems performance, reliability, and safety concerns. In the aerospace industry, these concerns coming from non-functional requirements significantly influence system design, leading to variations (Kossiakoff et al., 2011). Both functional and non-functional requirements can lead to the emergence or elimination of different system variants. Early identification and structured management of these variabilities are essential to streamline business processes and minimize unnecessary engineering efforts.

The motivation behind this thesis is to introduce a variability modeling methodology in MBPLE. This methodology seeks to bridge the gap between product features and component specifications by integrating effective variability management. The goal is to enable engineers to design systems with reuse in mind, encompassing subsystems, components, specifications, requirements, and system architectures. By doing so, this methodology aims to enhance the efficiency and accuracy of system development in complex engineering domains like aerospace.

1.2 Significance of the Study

The significance of this study lies in its potential to the field of MBPLE by addressing the critical challenges associated with managing variability in system architecture models in complex systems. This study introduces a robust methodology for variability modeling that integrates seamlessly with existing MBSE practices. By offering a structured approach to capturing, representing, and managing variability across all levels of system architecture, the study provides a valuable framework for enhancing the traceability, consistency, and overall effectiveness of MBPLE.

The Capability-based Product Line Architecture (CAPLA) methodology, developed and presented in this thesis, is designed to significantly enhance the effectiveness of MBPLE in managing variants within complex systems. CAPLA is designed to improve the

traceability and consistency of system models by incorporating mechanisms like variability identification, capability-based management, end-to-end traceability, and feature modeling. The methodology leverages existing modeling frameworks and tools, including the ARCADIA framework, Capella, and Pure::variants, to provide a structured approach to variability management across all abstraction layers.

In the MBSE framework, ARCADIA's structured approach, comprehensive coverage of engineering perspectives, and focus on functional analysis make it a critical tool for developing effective MBPLE methodology, ensuring that variability is managed systematically and comprehensively throughout the entire product lifecycle (Voirin, 2018). To maintain continuity and address both needs and solutions effectively, ARCADIA employs a clear separation between purpose, behavior, and structure while ensuring a seamless link between problem analysis and solution design. Its adaptable approach, which aligns with PLE practices, offers a comprehensive and precise system modeling framework for integrating needs, variabilities, and system architecture across different abstraction levels in feature and system architecture models. By having an integration with the feature-oriented modeling tool Pure::variants, ARCADIA facilitates the systematic and coordinated development of models. This integration ensures that both feature models and system architecture models are developed and aligned cohesively, promoting consistency and coherence throughout the modeling process.

This thesis aims to explore and evaluate the role of MBPLE in variant management for complex systems. Through a literature review, it seeks to uncover the benefits, challenges, and best practices associated with implementing MBPLE for effective variant management in system architecture models. By leveraging experience from software product line engineering and adapting modeling techniques for the MBSE context, this methodology seeks to address the challenges associated with variability management and system development. The following points highlight the key contributions brought by this methodology:

- **Enhanced System Understanding and Development Efficiency**
  - The methodology improves the overall understanding of the system and streamlines the development process.
- **Improved Variability Management**
  - It offers mechanisms for managing variability, ensuring that different system variants are effectively handled.
- **Capability-Based Variability Management**
  - By using capabilities to represent variability, the methodology enhances traceability and minimizes errors arising from model transformations.
- **Comprehensive Identification of Variability**
  - Through multiple analyses and structured modeling, the methodology ensures that variability is comprehensively identified across the system.
- **Adaptive Framework for System Changes**
  - It provides an adaptive framework that can manage system changes efficiently, supporting the dynamic nature of system development.

    - Effective management of changes within the system is facilitated, reducing the complexity and potential for errors during the development lifecycle.
- **Integration of Feature and System Architecture Models**
    - This integration guarantees the quality of both the development process and the final products, ensuring that the system architecture is aligned with the intended features.

These contributions collectively result in a robust methodology that not only improves the development process but also ensures the creation of high-quality, reliable products. The developed methodology's structure is demonstrated through its implementation in an unmanned aerial vehicle system. Based on the literature review and research, a set of requirements for a successful methodology for variant management in system architecture models are established and verified through the applied approaches and mechanisms.

1.3 Research Methodology

The research methodology employed in this study is based on the Design Science Research Methodology (DSRM) framework, a recognized approach in Information Systems research (Vaishnavi et al., 2004; Hevner et al. 2004). This methodology was selected for its emphasis on creating and assessing innovative solutions to complex, real-world problems. The research process followed five distinct stages of DSRM, which are described in detail in this chapter and illustrated in Figure 1.

The *Awareness of the Problem* phase identified the need for a structured methodology in Model-Based Product Line Engineering (MBPLE). This need emerged from a literature review, leading to the formulation of initial research questions focused on improving MBPLE practices through enhanced variability management. During the *Suggestion* phase, a systematic literature review was conducted to gather detailed knowledge and establish the initial requirements for a new methodology. This phase resulted in the preliminary design of the CAPLA methodology, integrating capability-based management with MBSE techniques. The *Development* phase focused on creating the key artifacts that constitute the core contributions of this research. The primary artifact developed was the CAPLA methodology, which was designed to address the limitations identified in the Suggestion phase. CAPLA incorporates elements from the ARCADIA method and the Capella tool for system architecture design, along with the Pure::variants tool for feature-oriented design and analysis. Additionally, a set of guidelines were developed to support the practical implementation of the CAPLA methodology. The *Evaluation* phase assessed the CAPLA methodology through a pilot study involving an unmanned aerial vehicle system. The pilot study demonstrated CAPLA's effectiveness in managing variability and ensuring traceability in system development. Finally, the *Conclusion* phase compiled the research findings and discussed the contributions of the CAPLA methodology to MBPLE, highlighting its potential impact on the practice of systems engineering.

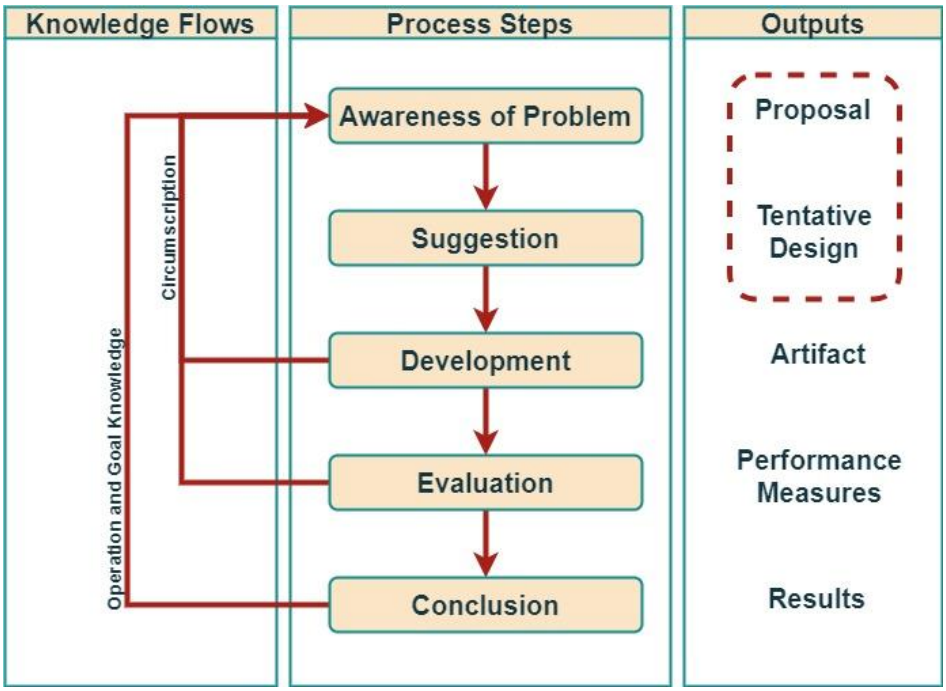

Figure 1: Research methodology of the study (Adapted from Vaishnavi et al., 2004)

In the Awareness of the Problem phase, a thorough literature review led to the identification of key challenges in MBPLE, resulting in the formulation of three primary research questions. These questions became the guiding force throughout the subsequent phases of the research. To effectively address the identified challenges, the following research questions were developed, each focusing on a crucial aspect of MBPLE. These questions formed the requirements that guided the design, development, and evaluation of the CAPLA methodology, ensuring that it met the complex demands of variant management in large-scale systems.

Research Question 1: How is MBPLE currently implemented and optimized for managing variability?

Research Question 1.1: What methodologies and strategies are being used in MBPLE to address variability within product lines?

Research Question 1.2: How do current tools and frameworks facilitate the implementation of MBPLE in managing system variability?

Research Question 1.3: What are the underlying challenges and effective practices in handling complex variability within large-scale MBPLE implementations?

Research Question 2: What are the most effective ways to model variability and commonality across a product line in MBPLE?

Research Question 2.1: How can existing modeling languages be extended or adapted to effectively support MBPLE?

Research Question 2.2: What tools and frameworks are necessary to support the implementation of an MBPLE methodology?

Research Question 3: How can MBPLE effectively manage variability and maintain quality as product lines evolve?

Research Question 3.1: How can variability be traced across different stages of development to ensure consistency in MBPLE?

Research Question 3.1: How does MBPLE scale with increasing product complexity and number of variants?

Research Question 3.2: How can MBPLE ensure that all products within the product line meet the required quality standards?

1.4 Structure of the Thesis

The content of this thesis is structured as follows:

- **Chapter 2:** Reviews the existing literature on Product Line Engineering and Model-Based Systems Engineering approaches,
- **Chapter 3:** Outlines the method development process and details of the proposed methodology,
- **Chapter 4:** Presents the implementation of the methodology, including a pilot study to demonstrate the applicability and benefits of the proposed approach,
- **Chapter 5:** Evaluates the methodology based on the identified requirements,
- **Chapter 6:** Concludes with a summary of findings, contributions to knowledge, and recommendations for future research.

CHAPTER 2

BACKGROUND

In this chapter, a comprehensive review of the foundational methodologies relevant to the study is provided. The principles, applications, and relevance of MBSE, PLE, and MBPLE are examined in detail. This background sets the stage for understanding how these methodologies interrelate and contribute to the effective management of variability in complex systems.

2.1 Model Based Systems Engineering

As a textbook definition, Systems Engineering (SE) is an interdisciplinary approach to establishing the successful creation of systems while identifying customer needs and essential functions early in the development process (Blanchard & Fabrycky, 2014). SE documents the requirements and proceeds with design integration and system validation while considering the overall problem, including operations, performance, testing, manufacturing, cost, and scheduling (Blanchard & Fabrycky, 2014). It brings together several engineering fields and highlights the entire life cycle of a system from the start of the initial concept through production to operation, decommissioning, and disposal (INCOSE, 2023). It includes technical and managerial processes, making sure all parts of the system work well together to achieve the desired results. (Kossiakoff et al., 2011). The methodology involves using models and simulations, managing risks, and conducting trade studies to improve system performance while staying within budget, schedule, and technological constraints (Sage & Rouse, 2011).

When first introduced, SE relied on document-based processes where the system requirements, design specifications, and verification procedures were all captured by large-scale documentation effort (Blanchard & Fabrycky, 2014). This process often encountered problems in maintaining consistency, traceability, and integration as the systems of interest became more complex (Blanchard & Fabrycky, 2014). Therefore, MBSE was raised as a significant approach to manage the increasing complexity in the design, development, and administration of modern systems. MBSE provides higher integration and efficiency to handle the diverse features of systems engineering by utilizing model-centralized practices rather than document-based approaches (INCOSE, 2023). Early efforts to incorporate modeling into SE demonstrated the potential advantages of using formalized models to represent and analyze systems in detail. Over time, advancements in computer-aided design, simulation tools, and standardized modeling languages facilitated the move to MBSE, marking a significant evolution in systems engineering practices (Kossiakoff et al., 2011).

MBSE revolves around several fundamental principles that distinguish it from traditional SE methods. Primarily, MBSE uses models as the primary means of capturing,

communicating, and managing system information. These models serve as the single source of truth throughout the system lifecycle. Models are developed and refined iteratively, allowing for continuous validation and improvement as the system evolves (INCOSE, 2023). Throughout modeling, MBSE emphasizes the integration of various perspectives of the system, including functional, structural, and behavioral views, to ensure a holistic understanding and analysis (Estefan, 2007). Furthermore, MBSE employs standardized languages such as SysML (Systems Modeling Language), which provide a structured and uniform way to represent system components, relationships, and behaviors. It leverages tools for automated analysis and simulation, enabling early detection of design issues and validation of system performance. Centralized model storage and formalized languages ensure consistency across different system components and facilitate traceability from requirements to implementation. (Friedenthal et al., 2014; Kossiakoff et al., 2011).

Various methodologies and frameworks have been developed to support the implementation of MBSE. Some of the prominent ones include the Object-Oriented Systems Engineering Method (OOSEM), which integrates object-oriented techniques with traditional SE processes, focusing on the use of SysML for system modeling (Friedenthal et al., 2014). Another is the V-Model, which graphically represents the system development lifecycle, illustrating the relationship between development phases and corresponding validation activities (INCOSE, 2023). Additionally, architecture frameworks such as Department of Defense Architecture Framework (DoDAF) provide structured approaches for developing and managing system architectures (Buede, 2009).

The implementation of the MBSE approach offers several advantages. Firstly, it enhances communication and collaboration as models act as a common language among stakeholders, improving interaction and teamwork across interdisciplinary groups (Estefan, 2007). Secondly, MBSE improves consistency and traceability by using centralized models to ensure that all system artifacts remain consistent and traceable, thereby enhancing the integrity of the engineering process (Friedenthal et al., 2014). Lastly, it allows for early detection and resolution of issues through automated analysis and simulation capabilities, enabling the identification and correction of design flaws early in the process, which reduces the risk of high-cost late-stage changes (Kossiakoff et al., 2011).

Despite its advantages, MBSE implementation faces several challenges. One major issue is overcoming cultural and organizational barriers. The shift from traditional document-based approaches to model-centralized practices demands extensive changes in organizational culture and mindset (Estefan, 2007). Additionally, ensuring seamless uninterrupted integration and interoperability among various MBSE tools and platforms can be quite challenging and often requires substantial effort (Friedenthal et al., 2014). Another key challenge is the need for skill development and training, as engineers must acquire new competencies in modeling languages and MBSE methodologies, which necessitates detailed training programs.

In summary, MBSE introduces a revolutionary method in systems engineering, overcoming the limitations of traditional document-based approaches. It improves the efficiency, consistency, and effectiveness of the engineering process, ensuring that systems are designed, developed, and validated to comply with requirements. (INCOSE, 2023; Friedenthal et al., 2014).

2.2 Product Line Engineering

In complex sectors, such as aerospace, the development of products involves numerous variations and configurations to meet diverse customer requirements and operational needs. These variations range from different models to various model configurations, each with specific characteristics, equipment and regulatory compliance demands. Managing such complexity through traditional single-system development approaches is increasingly impractical and inefficient (Laguna & González-Baixauli, 2008).

PLE addresses these challenges by enabling manufacturers to develop a core set of common assets—such as design specifications, requirements, test plans/cases, software modules, and hardware components—that can be reused across multiple product variants. Variable assets are assets that change depending on the product in which they are used. These assets correspond to variation points within the product line, allowing for customization and differentiation among product variants. They play a critical role in managing variability, ensuring that each product variant meets specific requirements while maintaining a common core structure. The diversity in product line is managed with variation points that uses mechanisms like model and text transformations, conditionals and macros in codes, filters, feature mappings (Krueger & Clements, 2013). This approach not only reduces development time and costs but also ensures higher consistency and quality across the product line (INCOSE, 2023). By systematically managing commonality and variability, PLE facilitates the rapid customization of products to meet specific customer needs without starting from scratch for each new variant.

The process of constructing a family of products instead of a single system requires a more complex and challenging management and design process. According to The International Standard ISO/IEC 26550:2015 Software and systems engineering – Reference model for product line engineering and management (2015), there is a recognized need for distinct management methodologies in product line engineering, differing from those used in single-system development. This standard identifies two critical life cycle processes in product line engineering: domain engineering and application engineering.

Domain Engineering involves identifying and developing the parts that will be reused across the product family by systematically analyzing and defining commonalities and variabilities. This process ensures that core assets, such as design specifications, software modules, and hardware components, are robust and adaptable to various product configurations. Application Engineering encompasses the activities required to develop individual products using the reusable artifacts created during domain engineering. This

process focuses on assembling and tailoring these core assets to construct specific products that meet customer requirements and operational needs. (Pohl et al., 2005).

Considering the dynamically changing development environment, these two continuous lifecycles—domain engineering and application engineering—interact and feed each other at various design stages. Changes in reuse information or discarding of an asset can significantly impact both lifecycle processes. Managing this variability information is crucial and is effectively handled by modeling the variability in PLE.

Variability Modeling

Variability modeling in PLE enables the systematic identification and management of variabilities and commonalities across the product family. According to the survey made by Berger et.al. (2013), in defense and automotive domains, 26% of the variability models contain 10,000 units of variability, which these units correspond to the features and variation points in majority. This modeling process becomes crucial especially when handling such largescale systems since it ensures that any modifications in the reusable components or core assets are accurately reflected and integrated across all affected products.

In domain engineering, many modeling technique are proposed in academia and industry, with many different tools and techniques but feature modeling is the most widely known and used modeling notation for describing the variability and commonality in a product line. These commonalities and variabilities are modeled as the product features which can be defined as a system property that is relevant to some stakeholder (Capilla et al., 2013). Figure 2 taken from the study of Meinicke *et al*. (2017) represents the features and feature models place in the previously mentioned the domain and application engineering lifecycle in software product lines. In domain engineering lifecycle, domain analysis phase identifies, organizes and represent the commonality and variability in the domain and constructs feature models, domain implementation phase, using the feature models, identifies and creates the reusable artifacts (Kang et al., 1990; Meinicke et al., 2017). Creation of a specific product takes place in application engineering lifecycle, by selection of the features according the user requirements and mapping them to the artifacts (Apel et al., 2013). Supporting the lifecycles, PLE employs the concepts of problem and solution spaces to represent the development phases. The problem space defines the system requirements and components established during domain analysis, while the solution space delineates the architecture that addresses these requirements (Anquetil et al., 2008).

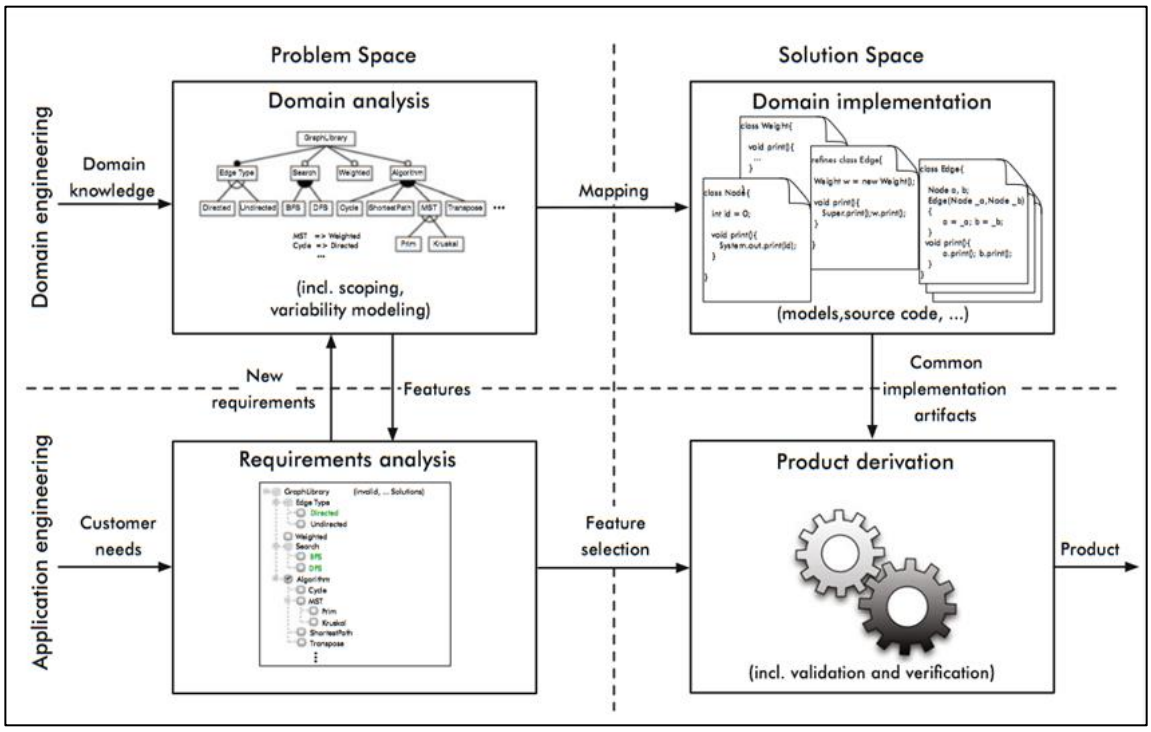

Figure 2 : An overview on software product line engineering (Meinicke et al., 2017)

The feature modeling approach is first introduced by Kang et al.(1990) as a part of the Feature Oriented Domain Analysis (FODA). FODA feature models contain information about the hierarchical feature relationships, alternativeness, optionality and mutual dependencies of the product features for the application domain. These relations and dependencies are used for visualizing and restrict the possible product configurations. The graphical representation of a feature model called feature diagram (Kang et al., 1990) and a simple example of a feature diagram given in Figure 3.

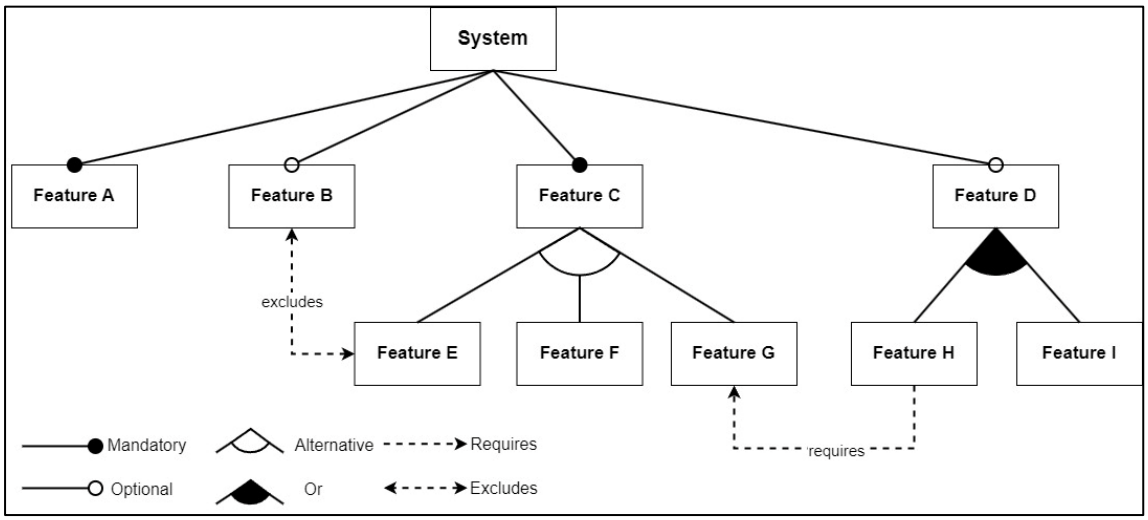

Figure 3 : Example feature diagram

As can be seen in Figure 3, the features can be separated into four groups:

- Mandatory: the feature must be included in the description of its parent feature.
- Optional: the feature may or may not be included depending on the choice of the customer
- Alternative: one and only one of the features from the sub-features or child features can be included in the parent description.
- Or: one or more features from the sub-features or child features can be included in the description of the parent feature.

Besides these types, for constructing the product line configurations constrains between the features in the feature diagrams represented by using the following feature notations:

- Requires: where if a feature requires another feature, it implies the inclusion of the second feature whenever the first feature is chosen.
- Excludes where if a feature excludes another feature, they cannot be applicable at the same time for the same product, so both features exclude each other.

2.3 Model Based Product Line Engineering

In PLE, variations can occur across all system assets, including hardware and software. Managing these variations requires an approach that integrates the holistic system perspective of systems engineering (Gronniger et al., 2008). Traditional systems engineering often does not account for the diversity inherent in PLE. However, with the realization of improved reuse in terms of time, money, and effort, systems engineers have evolved to consider potential variants, customer features, and emerging technologies (Hause et al., 2015; Li et al., 2016). In this context, the INCOSE Systems Engineering Vision 2035 articulates the need for PLE practices in systems engineering as: "Model-based reuse practices will effectively leverage enterprise investments. These practices include reference architectures and composable design, product line engineering, and patterns" (INCOSE, 2021).

As first introduced by Góngora *et al.* (2015) and Hummell & Hause (2015), MBPLE has emerged as a systems engineering practice that combines MBSE and PLE to enhance the efficiency of both practices. MBPLE provides clear and understandable guidance for development to all stakeholders. In MBPLE, supporting variability is achieved by introducing an additional abstract viewpoint and formalism to the MBSE framework, which includes information on variability, constraints, dependencies of needs, requirements, and architecture (INCOSE, 2015; Góngora et al., 2015). The general technique for merging both concepts consist of three models: a variability model, a model expressed in a modeling language (such as SysML, UML), and a realization model that maps and transforms the elements of the other two models through variation points (Goos et al., n.d.; Heidenreich et al., 2010; Pohl et al., 2005; Satyananda et al., 2007; Seidl et al., 2012).

For variability modelling, feature models provide a view for describing product lines by scoping which features are supported by the product line and which are not. Specialization of PLE, such as Feature-Based Systems and Software Product Line Engineering (feature-based PLE), emphasizes the importance of feature models for capturing variability. It describes the automation-supported configuration of engineering assets by mapping features to modelling assets and transforming specific products via a configurator (ISO 26580, 2021). This transformation is defined as converting a 150% MBSE model to a 100% MBSE model through features and variation points. A product line architecture modeled with MBSE that contains shared and variant assets for all products in the product line is called a 150% MBSE model. This shared model, based on the creation of feature configurations, transforms into 100% MBSE model that represent a single variant (Góngora et al., 2015; Hummell & Hause, 2015).

The 150% MBSE model can be thought of as a master model that covers all abstraction layers in line with the system hierarchy levels. 100% MBSE models can be derived with different industrial scopes for any lifecycle. Especially for complex systems like aircraft, a handling mechanism for the synchronization of the two models is needed. Consistency management is crucial when designing MBPLE models, as all modelling assets—such as

requirements, behaviours, and interfaces—contained in the 150% MBSE model, model need to be updated whenever there are additions or subtractions of any modelling element, regardless of whether the element belongs to the base or variant model. Changes must be made in the master model to ensure that all 100% models are notified of the changes. This requires efficient configuration management in MBPLE (Forlingieri, 2022; Hummell & Hause, 2015).

The overall implementation methodology of MBPLE is described in Figure 4.

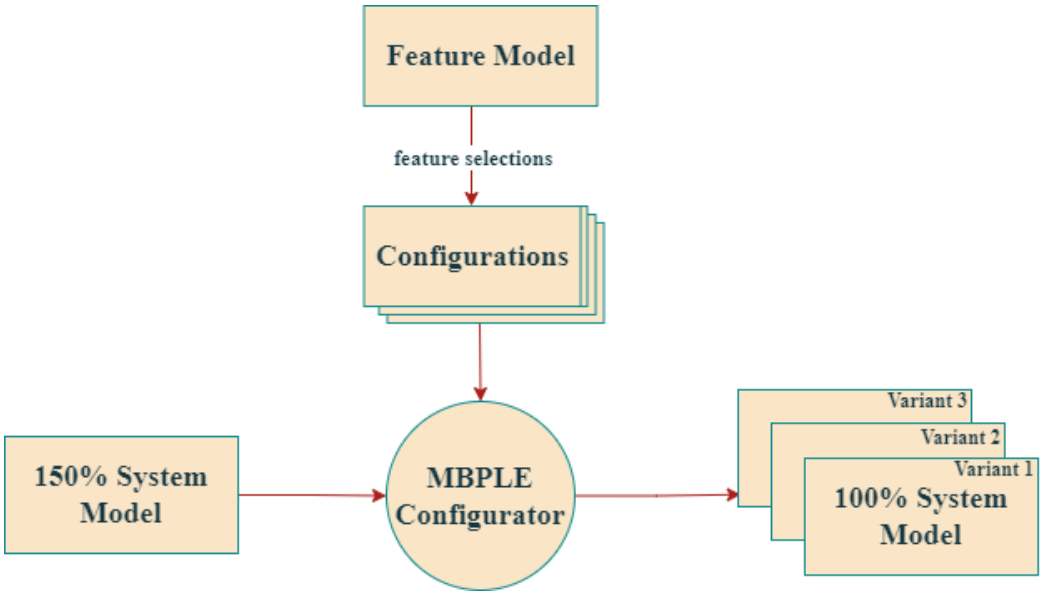

Figure 4 : Implementation of MBPLE

Tool Support for MBPLE

In this thesis, integration of variability management with the model-based systems engineering approach addressed. To support this approach and modelling activities, efficient tools for automatic configuration adapted for two modelling aspects.

Many tool solutions and approaches are introduced in the industry for product line engineering; i.e. GEARS (Krueger, 2007), Pure::variants (Beuche, 2004) and FeatureIDE (Thüm et al., 2014) assist the engineers in modelling the variability. The usage of the tools based on the feature modelling is dominating the industry and Pure::variants is currently the most preferable tool since it provides not just a feature model but a variant management methodology with having several integration interfaces with MBSE tools (Berger et al., 2013).

MBSE tools are critical enablers of modern systems engineering practices, offering powerful capabilities for modelling, simulation, and analysis. These tools play a vital role

in ensuring that complex systems meet their requirements and perform as expected. Supporting various modelling languages and frameworks, such as SysML (Systems Modeling Language), UML (Unified Modeling Language), and others tailored to specific industries or applications, MBSE tools provide a versatile platform for system design and development. Industry-leading tools like IBM Rational Rhapsody, MagicDraw, Simulink, and Capella are widely utilized across various sectors to tackle the challenges of designing and managing intricate systems.

The ARCADIA (Architecture Analysis & Design Integrated Approach), modeling method and Capella[1] tool are chosen for the MBSE modeling implementation of the MBPLE methodology presented in this thesis. ARCADIA offers an approach that unites all needs, information, and product descriptions from many stakeholders into a single model. Capella's common and easily understandable interface, with its multi-perspective approach, allows even specialty engineering constraints to be easily implemented into the model. Describing requirements and solutions in the same model also supports the analysis of architectural alternatives and verification of the properties of the system architecture (Voirin, 2018). For variant modeling, Pure::variants [2] tool is chosen. Integration between Capella and Pure::variants enables the derivation of variants from Capella models by linking relevant features and modeling elements and removing or keeping modeling elements based on variability information.

ARCADIA/Capella

ARCADIA, developed by Thales, is a model-based methodology designed to define and verify systems, software, and hardware architectures (Voirin, 2018). As defined in IEEE 1220 standard, systems engineering is a process that begins with requirements analysis and continues to the definition of the physical system, establishing a clear distinction between the problem space (represented by requirements) and the solution space (encompassing architecture and design activities) (IEEE, 2005). Within these spaces, requirement and functional analyses are key activities for system architecture design. ARCADIA adopts this methodology, defining and verifying system architecture based on the formalization of needs analysis, encompassing operational, functional, and non-functional characteristics. Voirin emphasizes this focus on functional analysis in ARCADIA, stating, "Functional analysis constitutes the major support for the understanding and the expression of need in ARCADIA, as well as for the definition of the expected behavior of each system component during the design stage" (Voirin, 2018). By constructing well-defined input, output, dependency, and exchange mechanisms

[1] Open source MBSE tool. Capella. (n.d.). https://mbse-capella.org/

[2] Pure:Variants: Pure-systems. pure-systems. (n.d.). https://www.pure-systems.com/purevariants

between functions at every level, it translates functional requirements into a formalized set of system functions (Voirin et al., 2016).

Capella is the recommended workbench for implementing the ARCADIA method, as it provides models for each ARCADIA perspective and links between them as given in Figure 5.

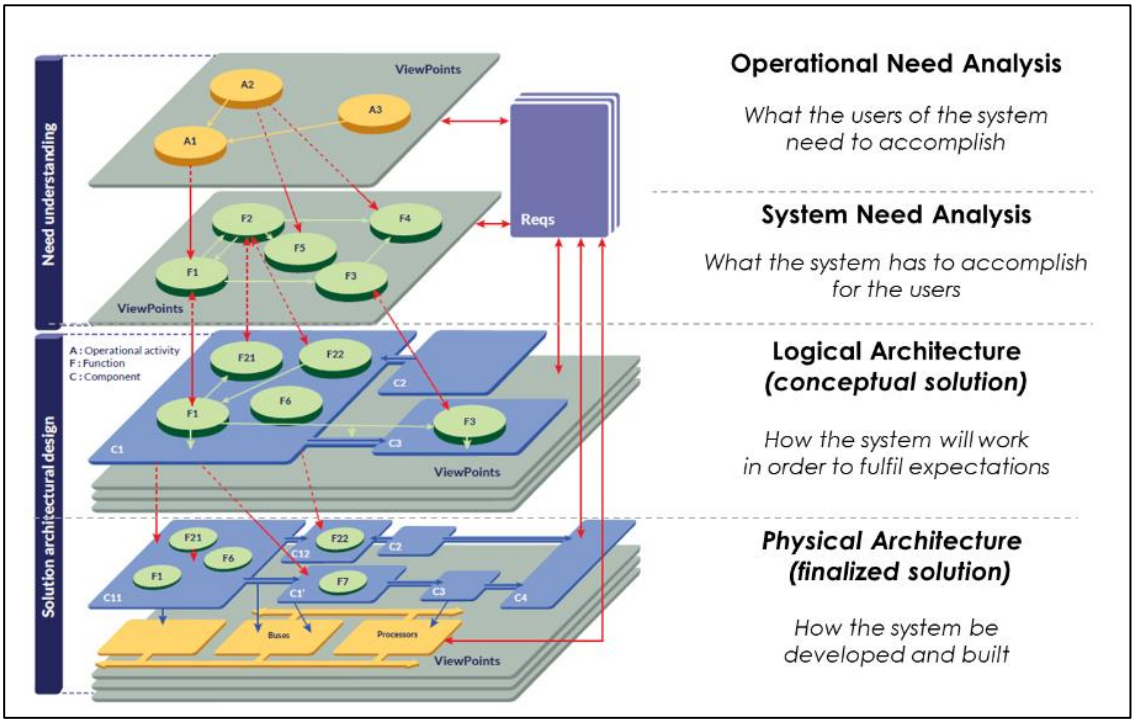

Figure 5 : Arcadia Methodology

In the problem space, abbreviated as the "Need Understanding Space" in ARCADIA, models are created under two perspectives: Operational Analysis (OA) and System Analysis (SA). OA focuses on the needs and goals of customers and users, identifying and characterizing them in the system model without mentioning the system itself. SA, on the other hand, focuses on the system as a black box, defining the expectations from the system as functions based on the needs identified in the OA level.

In the solution space, abbreviated as the "Solution Architectural Design Space" in ARCADIA, two new perspectives are introduced: Logical Architecture (LA) and Physical Architecture (PA). The LA provides an abstract representation of the system architecture by defining logical components and associating them with the corresponding functions. This coarse-grained representation offers foresight for system design decisions that can be easily understood by stakeholders. The PA level develops the technical solution defined in the OA, incorporating physical system components, their behaviors, and interactions,

including technical and technological aspects in detail. These four perspectives, along with their many views, ensure requirements traceability to the components.

Pure:: Variants

The Pure::variants tool suite, developed by Pure-Systems (*Pure::variants User's Guide*, 2023), is a commercial product that offers a set of integrated tools supporting various phases of the software product line development and derivation process. As an MBPLE configurator, it provides a clear variability modeling language and supports the automatic derivation of 100% MBSE models based on feature configurations. Additionally, it offers automatic conflict detection and resolution during variant definitions, based on the dependencies configured in the variability model.

Pure::variants uses four types of models for variability modeling as given in Figure 6: feature models, family models, variant description models, and result models. In line with PLE development spaces, feature models are used to model variability in the problem space. It classifies features as mandatory, optional, alternative, and or as defined in FODA and extending the concept of dependencies between the features, more complex dependencies can be formed besides requires and conflicts. Family models handle variability in the solution space by mapping the problem space to architectural elements, it holds the information of variation points. Variant description models express the feature configurations needed to create a specific variant, controlling these configurations by the feature types and dependencies defined in the feature model. Finally, based on the valid feature configurations formed in the variant description models, result models are created through transformation. These result models are free of the variations defined in the feature and family models, describing a single variant (Beuche, 2004; Capilla et al., 2013; Sinnema & Deelstra, 2007).

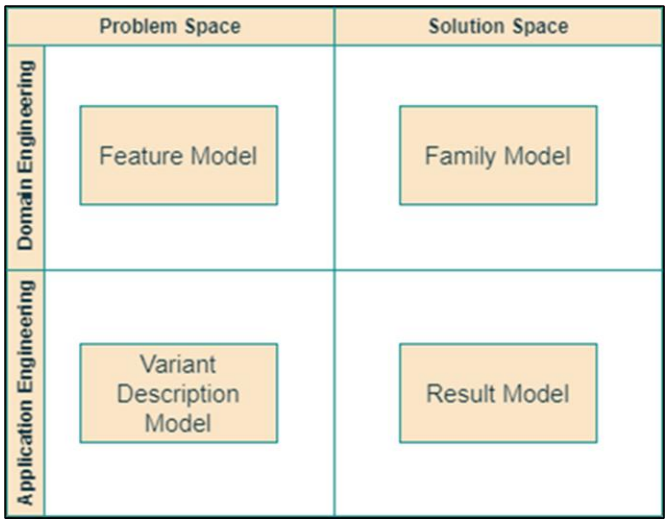

Figure 6 : Overview of the models used in variability modeling(Adapted from *Pure::Variants User's Guide,* 2023)

CHAPTER 3

DEVELOPMENT AND DESCRIPTION OF THE METHODOLOGY: CAPLA

In this chapter, we conduct a comprehensive review of the method development process, offering a detailed examination of the steps and specifics involved in developing the CAPLA methodology. Additionally, key requirements that shape the development and validation of the CAPLA methodology are identified, ensuring its alignment with the intricate demands of modern system engineering. This analysis provides insights into the approaches employed to achieve effective variability management within system architecture models.

3.1 Method Development Process and Related Work

To acquire the necessary insights into the current state of MBPLE, we first conduct a thorough review of existing literature. This review focuses on addressing the following research questions:

- *Research Question 1.1:* What methodologies and strategies are being used in MBPLE to address variability within product lines?
- *Research Question 1.2:* How do current tools and frameworks facilitate the implementation of MBPLE in managing system variability?
- *Research Question 1.3:* What are the underlying challenges and effective practices in handling complex variability within large-scale MBPLE implementations?

Through the literature review to answer the research questions, we were able to gain an in-depth understanding on the current situation, especially regarding to the notable advancements and expanding applications MBPLE is experiencing across various industries. This trend underscores its increasing importance in managing complex product variations and enhancing development efficiency. Both industry and academia have introduced several approaches to manage variation in system architecture. However, these approaches predominantly focus on software-intensive systems. Czarnecki & Ankiewicz (2005) used superimposed UML 2.0 models and mapped them with feature models and derive possible products. To further support the derivation process, several works have proposed the use of decision models. Loughran *et al.*(2008) defined Variability Modelling Language (VML) to express variability and decisions that manage feature configurations to compose software components. Separating the variability information from the problem and solution space, Berg & Bishop (2005) used decision models to express features, variation points, their relationships, and dependencies to maintain traceability and consistency of derived software products.

Additionally, numerous papers have concentrated on variability in system architecture. For instance, Gaeta & Czarnecki (2015) proposed a methodology using SysML diagrams to model variability in aerospace systems. Addressing the complexity of aerospace systems, Li *et al.* (2016) developed a modeling methodology for physical variability in aircraft using SysML diagrams. Grönniger *et al*. (2008) employed feature and variant views to express variability in logical architectures of automative systems using function nets . Also, presenting variability viewpoints, Tekinerdogan & Sözer (2012) offer a methodology that addresses variability information in separate views by making the variability visible within the related architecture views themselves, thus removing the need to translate variability models into architectural models. Focusing on constraints to represent variation points and variants in SysML diagrams at different levels, Dumitrescu *et al.*(2013) presented an integration of variability management into the MBSE approach. Although these works answer questions like, "how to express variability in the architecture model?" and "how to integrate variant models to system architecture models?", there is still no standardized method for handling variation in every phase of the development process.

System architecture encompasses the entire behavior of the system, including functional, structural, and dynamic descriptions. These behaviors reflect various stakeholder perspectives and must be represented in a structured manner since they do not share the same abstraction levels. Utilizing views for different abstraction levels is a strategy to manage this complexity throughout the systems engineering process.

Integrating MBSE approach into PLE processes allows involvement at every abstraction level, from requirement definition to subsystem definitions and component design. Given that variability can exist at various levels of the development cycle, it is crucial to contribute to variability across different views to execute MBPLE successfully. Model transformations and mappings enable the integration of variability and system modeling. However, managing traceability and dependencies between features and modeling elements at every abstraction level, especially for complex and large systems, requires a well-structured method. The method proposed in this study addresses the establishment of transformation rules that guide product configurations and ensures that as models progress through each transformation stage, application domain models are automatically adjusted to integrate new considerations from higher levels of abstraction or additional implementation specifics from lower levels of abstraction.

The CAPLA methodology presented in this study, builds upon the frameworks introduced by Navas *et al.*(2021) and Voirin (2018), expanding and adapting their principles to effectively manage variability in system architecture models across all stages of development. Navas *et al.*(2021) and Voirin (2018) highlight the underutilization of MBPLE in practice and propose an approach to efficiently align business objectives with customer needs. Using ARCADIA and Pure::variants, the method categorizes each type of variability distinctly within problem and solution spaces using dedicated feature models. This approach aims to ensure precise definition of product variabilities and configurations, as well as their integration into the architecture.

Utilizing the aforementioned approaches, the CAPLA offers a more comprehensive study, elevated by systems engineering principles. This approach provides a detailed characterization of architecture variability, encapsulating all system assets.

To establish a robust method for managing variability within system architecture, it is essential to address both academic and industrial challenges. This involves identifying specific requirements that will guide and validate the proposed methodology. The following research questions are pivotal in specifying these requirements:

- *Research Question 2.1:* How can existing modeling languages be extended or adapted to effectively support MBPLE?
- *Research Question 2.2:* What tools and frameworks are necessary to support the implementation of an MBPLE methodology?
- *Research Question 3.1:* How can variability be traced across different stages of development to ensure consistency in MBPLE?
- *Research Question 3.1:* How does MBPLE scale with increasing product complexity and number of variants?
- *Research Question 3.2:* How can MBPLE ensure that all products within the product line meet the required quality standards?

Addressing these questions provides the foundation for transforming critical aspects of MBPLE into actionable requirements. These requirements will guide the selection of appropriate tools and approaches for implementing and validating the developed method.

Methodology Requirements

Requirement 1:	*The proposed methodology shall support the application of Model-Based Systems Engineering approaches for designing system architecture, facilitating a model-driven development process.*
Requirement 2:	*The proposed methodology shall support feature-oriented design and analysis by creating, visualizing, and managing system variant features, including modeling interdependencies between features.*

The foundations of the MBPLE approach are encapsulated in Requirements 1 and 2. MBSE provides advanced visualization and modeling capabilities to effectively represent system architectures. These architectures include requirements, subsystems, functions,

components, and interfaces. By integrating variants into the architecture, MBSE can significantly enhances product line quality and productivity.

In addition to visualizing and modeling the system architecture, feature-oriented variant modeling is a highly effective approach for structuring variants and feature relationships. This method involves modeling the dependencies and interrelations between features, ensuring a clear understanding of feature interactions. This approach supports the feature selection process for each variant, ensuring that the combination of components and features is both valid and optimal for each specific variant.

Requirement 3: *The proposed methodology shall include automated mechanisms to derive valid configurations based on predefined configurations and rules for each product variant, reducing manual efforts and errors.*

This requirement also aligns with the MBPLE approach, specifically addressing the transformation of 150% system models into 100% models. It supports the valid configuration creation process outlined in Requirement 2, ensuring that the tools facilitate the automated creation of system models for each variant according to feature configurations.

Requirement 4: *The proposed methodology shall offer a platform ease of use with clear, readable representations of technical architecture and variant configurations accessible to architects and stakeholders at all levels.*

This requirement provides guidance for selecting the tools and approaches to be utilized in the methodology. Given the involvement of numerous stakeholders with diverse perspectives—including executives, business analysts, system architects, design engineers, and product line engineers—readability and usability are paramount. Additionally, offering clear and comprehensible representations of the system options and alternatives is crucial for achieving consensus with the customer.

Requirement 5: *The proposed methodology shall include tools and techniques to comprehensively manage system variability across different abstraction levels, ensuring that all aspects of variability are captured and represented.*

As mentioned previously, variability can occur at diverse development stages. To model the system architecture completely and accurately, and to represent variability effectively, both the system model and feature model should be designed to represent different levels

of abstraction. This approach supports the joint and collaborative construction of both models, resulting in a coherent system in all aspects. Variability information should be present even in the early stages, represented at a high level of abstraction. This early representation is critical for providing a decision-making and development environment for the company. For instance, in the early stages of development, it offers insights into the return on investment (ROI) of different variants. During the solution design stages, it facilitates more structured trade-off analyses, aiding in making well-informed decisions.

Requirement 6: *The proposed methodology shall incorporate end-to-end traceability mechanisms that form clear relationships between high-level requirements, technical solutions, and architectural elements across variants.*

This requirement ensures that the system -as a whole- remains consistent and retains all information related to variant decisions and design. Traceability between variants at different levels of abstraction is crucial, as it facilitates verification, maintenance, and realization of the system. The ability to trace the impact of requirements on the system architecture and variants enhances understandability, thereby improving overall management.

Requirement 7: *The proposed methodology shall be scalable to handle the complexity associated with large and diverse product lines, ensuring efficient management of multiple variants.*

Regardless of the system's size and complexity, variability in the system architecture should be manageable. The proposed method should be effective in large and complex systems, ensuring that it does not require additional manual effort due to the system's size.

Requirement 8: *The proposed methodology shall improve change management processes, providing clear traceability and impact analysis for evolving product features.*

Supported by *Requirement 6*, the methodology should include a change management mechanism to minimize potential errors resulting from changes in the system. Regardless of the scope of changes, the impacted elements should be easily traceable, preventing users from creating incorrect models. Additionally, the methodology should be flexible enough to accommodate an evolving product line and incorporate changes at any development stage.

By meeting these requirements, the CAPLA methodology aims to enhance system understanding, manage variability efficiently, improve development efficiency, and

support effective change management. These improvements represented as the capabilities of the CAPLA and the requirements that serve these capabilities given below.

Capability 1: Improved System Understanding- Requirement 4,5,6

Capability 2: Enhanced Variability Management- Requirement 2,5

Capability 3: Improved Development Efficiency- Requirement 1,3,7

Capability 4: Effective Change Management- Requirement 7

To enhance the readability and comprehensibility of the entire thesis, the methodology is visualized through modeling. Figure 6 illustrates the relationships between the operational capabilities (OC) and actors.

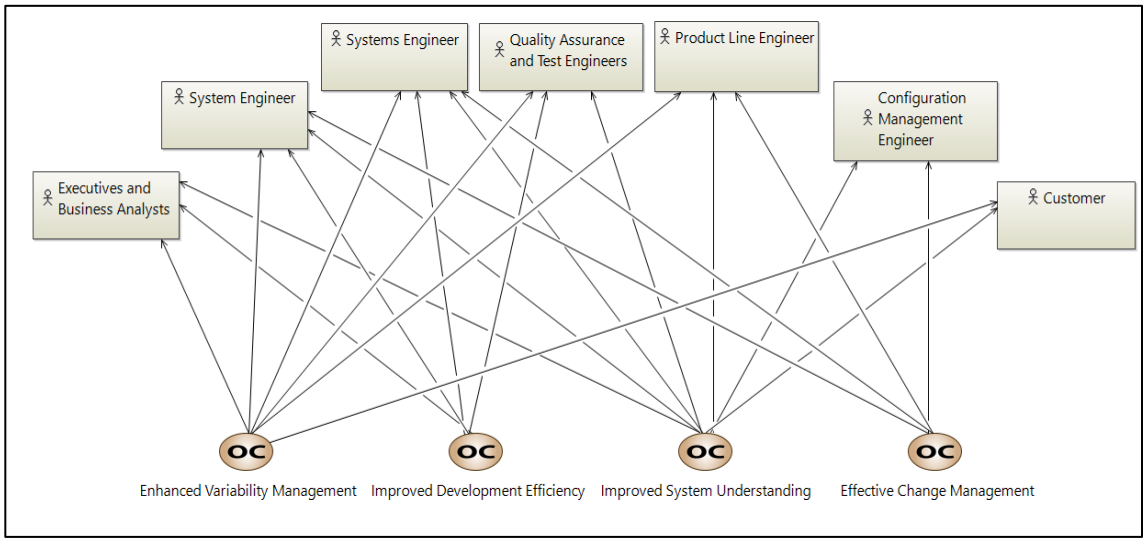

Figure 7 : Thesis Capabilities and Actors

3.2 Variant Management Method

A scalable PLE methodology is deployed to systematically represent variations across diverse levels of abstraction and all developmental stages, regardless of the system's complexity, to structure a variant management method. The problem space within this framework encapsulates comprehensive system specifications and requirements. Conversely, the solution space is dedicated to delineating technical architecture specifics across the architecture, design, and implementation phases. However, despite the comprehensive coverage provided by these spaces, they are found to be lacking in efficiently defining the interconnections among features and products (Berg & Bishop, 2005). Recognizing this deficiency, Berg & Bishop (2005) propose the introduction of another space, named the decision space. This decision space serves as a pivotal construct

that not only captures variability information but also encapsulates and integrates the contents of the problem and solution spaces. Within the decision space, the relations between user dynamic and variant-specific features are defined, giving a more nuanced understanding of the relationship between artifacts within the problem and solution spaces. This approach projected with the mapping concept that the tools provide between the feature and system model in this study. Both feature and system modeling elements in the need space and problem space mapped to each other to construct products. In this chapter, we present the methodology for managing the variations in these three spaces.

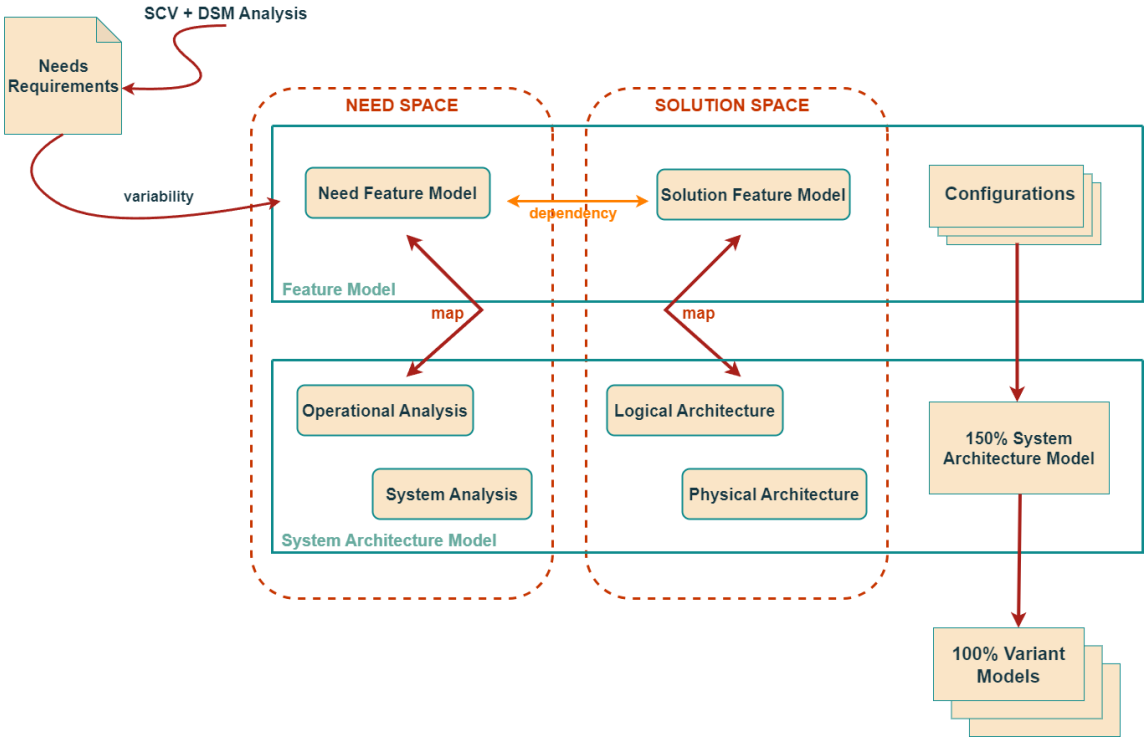

Figure 8: Overview of CAPLA Methodology

3.2.1 Variability Identification

Need Space

Effectively managing variability within a system requires thorough identification and classification of needs and requirements, a crucial aspect in product line engineering. These variabilities derive from various stakeholders, customers, development, and business entities. For instance, business considerations, such as assessing reuse rates and return on investment (ROI), can transform a system to a product line (Alves et al., 2010). Customer needs, on the other hand, heavily influence product definitions and architectural decisions. With contributions from multiple stakeholders, the Scoping, Commonality, and Variability (SCV) analysis method become instrumental in identifying commonalities and variabilities within requirements, laying the foundation for constructing product families (Coplien et al., 1998).

Ensuring alignment between business and user requirements is crucial for accurately modeling the context, operational goals, and variability within a system. However, relying solely on natural language descriptions often falls short in capturing the full scope and nuances of these requirements. Capability-based management offers a more comprehensive approach to specifying system requirements, addressing this limitation (Frank et al., 2014; Link & Trujillo, 2016).

Despite being underutilized, capabilities play a pivotal role in the systems engineering process as they provide an abstract, high-level perspective of a product, system, or organization (Antunes & Borbinha, 2013). Capabilities serve as fundamental drivers for capturing variability within a system, particularly when the product definition and objectives are not clear from customer and business requirements alone. Therefore, the identification of commonalities and variabilities is grounded in the capabilities derived from these requirements.

In the ARCADIA methodology, a capability is defined as "the ability of the system to provide a service that supports the achievement of high-level business objectives" (Bonnet et al., 2019). Within ARCADIA, capability elements play a significant role, especially in need modeling, aligning with the capability-based management approach that we have adopted. This approach ensures a coherent representation of requirements, facilitating effective system development and alignment with overarching business goals.

By identifying common and variant capabilities, we gain clarity on the functional and non-functional requirements associated with each capability. This enables us to classify these requirements more easily. Moreover, at this level, we can distinguish product families based on these capabilities. This division allows for a more structured approach to system development, where products within the same family share common features and capabilities while accommodating variant requirements as needed.

With the initial commonality and variability analysis, capabilities and requirements are divided into two as “Core” and “Variant”. Core Capabilities (CC) represent the common essential abilities of the aircraft and the corresponding requirements will be called Core Requirements (CR) from now on. Variant Capabilities (VC) and Variant Requirements (VR) are the capabilities and requirements that are essential for the specific mission of a variant.

Solution Architecture Space

Within the solution phase, the identification of interfaces between solution components (logical and physical architecture components) is conducted using a systematic method. The focus of the thesis revolves around a drone system, comprising numerous subsystems, each encompassing a significant amount of model elements. Given the complexity inherent in modeling the entire system within a single model to reduce complexity and enhance model readability, an approach is developed for representing subsystems as logical components.

During the creation of the 150% model, the system comprises numerous subsystems belonging to various variants. It is imperative to model variant-specific subsystems separately from core subsystems to effectively manage changes resulting from variants within the system. Thus, aligning with the classification of capabilities and requirements, commonality and variability analysis was also conducted for subsystems.

The first type consists of subsystems without any variability, termed Core Subsystems (CS) throughout this paper, which remain unaffected by variant changes in the system and offer greater modeling flexibility. On the other hand, Variant Subsystems can take two forms: those unique to specific variants (Variant Subsystems, VS), and Core Subsystems that incorporate variability internally (Core Variant Subsystems, CVS). To manage these subsystem types effectively, the Design Structure Matrix (DSM) model is employed to facilitate comprehensive analysis and interface management. The DSM matrix is particularly utilized for identifying Core Variant Subsystems (CVSs), revealing variability that may not be readily apparent in capabilities. In the drone model presented, certain CSs include variant-specific subcomponents, which originate from either functional or non-functional requirements. If a variant necessitates a unique interface or function within a CS that cannot be accommodated by the existing design, it results in the formation of a CS to CVS.

Design Structure Matrix

In systems engineering, the seamless integration of subsystems to ensure advanced functionality relies heavily on effective interface management (Browning, 2016). Interfaces are initially identified during the architecture definition process as architectural models are developed. Various analysis methods and tools are employed to analyze these interfaces, aiding in their identification and comprehension within the system (INCOSE, 2023). DSM modeling method is chosen for interface analysis due to its compactness, scalability, and intuitive readability.

The DSM is a versatile modeling technique utilized in the design, development, and management of complex systems. It offers network modeling tools that depict system elements and their interactions, effectively illustrating the system's architecture or designed structure. While resembling the N^2 diagram in appearance and application, the DSM typically employs a different input and output convention, with inputs represented on horizontal rows and outputs on vertical columns(Eppinger & Browning, 2012). The DSM finds applications across various project fields such as organization, scheduling, parameters, and architecture. In our case, the Architecture DSM is utilized to represent interactions between subsystems.

This matrix serves as a clear visualization of subsystem interfaces and their respective types. In aviation projects, matrices of this nature are commonly utilized as inputs for design and Interface Control Documents (ICD). Even at early stages, matrices are created for systems, and if the system encompasses variants, these matrices must be extended to include variant-specific interfaces or integrated into existing matrices. Variant matrices are required to include information on:

- Interfaces between Variant Subsystems (VS) and Core Subsystems (CS).
- Interfaces between Variant Subsystems (VS) and Core Variant Subsystems (CVS).
- Interfaces between Core Subsystems (CS) and Core Variant Subsystems (CVS).

In each variant, numerous specific interfaces with various subsystems can increase complexity and potentially decrease the reuse rate, contradicting the goals of product line engineering. The primary goal in modeling a system with its variants is to enhance profitability and reduce engineering effort and errors by maximizing reuse and minimizing complexity. If the reuse rate of system variants is low, separate modeling of each variant may be logical for the company. DSM can also provide an early insight for analyzing these concerns. However, in this study, maximizing the reuse rate is prioritized to ensure that the 150% model holds significance within the context of product line engineering.

3.2.2 Variability Modeling

The variabilities within the system manifest across various stages of modeling. As previously delineated, our modeling approach is divided into two key facets: feature modeling and system modeling.

In the context of ARCADIA, the division of the system model into need understanding and solution architecture carried through to the feature modeling. This conceptual framework represents abstract layers along the vertical axis, spanning from the system needs and functionalities to the working principles and construction specifics of the system. Such separation between the problem domain and solution domain serves to mitigate complexity within both models while providing a clean traceability between them.

The System Model (SM) is initially created as a 150% model, encompassing all variants within a single model. However, to operationalize product configuration and create 100% models, a integration between System Model elements and Feature Model (FM) elements is established. This linkage is facilitated by a variant management tool, which automates the removal of non-related elements from the SM, thereby refining it into a cohesive 100% models. This iterative process ensures that the resultant models accurately reflect the specified configurations while optimizing efficiency and precision in product development. Overview of the feature models structure in development spaces given in Figure 9.

Feature Modeling

Need Feature Model

Need Feature Model (NFM), represents capabilities, the highest-level abstraction used to represent the system functional requirements and non-functional requirements. This feature model supports the variability identification process. The identified variant capabilities implemented as the parent features and functional requirements implemented as child features of that realized capability the to the model. This NFM will be mapped to the SM at Operational Analysis (OA) and System Analysis (SA) levels which are the levels that represent the understanding of needs in ARCADIA.

Solution Feature Model

Solution Feature Model (SFM) represents the details of the variability in the system in solution level. This variabilities in the system functional structure, component structure and software/hardware architecture implemented as features in the system. SFM will be mapped to the SM at Logical Architecture (LA) and Physical Architecture (PA) levels which are the levels that represent the system solution in ARCADIA.

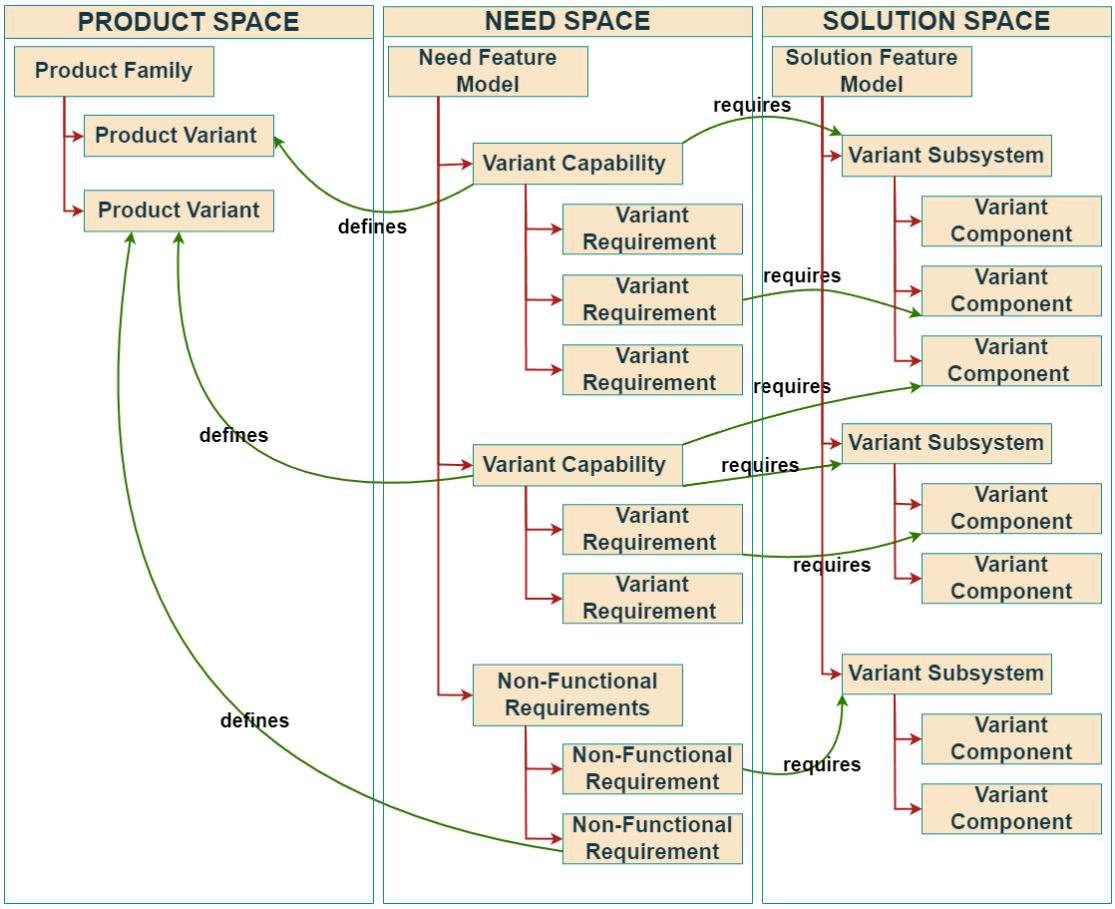

Figure 9 : Detailed Structure of Feature Models in Need and Solution Spaces

3.2.3 System Modeling

System Modeling of Variability in the Need Understanding Space

Variability in Operational Analysis Level

In ARCADIA methodology, constructing the SM starts with the Operational Analysis (OA). At this level, to express the customer needs, system architects can model the required high-level operational capabilities of the mission without initially defining the system of interest which is valuable in cases dealing with multiple customers and products. This enables a more comprehensive operational needs analysis, facilitating the definition of alternatives to satisfy customer needs and allowing for a more informed decision-making process during development (Voirin, 2018).

Within SM elements, the variation points in the operational analysis materializes through the representation of actors and capabilities. These capabilities are subsequently modeled as operational capability elements, complete with associated actors and operational entities. The variant capability features inherent within the NFM are implemented into operational capabilities within the SM, with the inclusion of common capabilities. Actors represent the product families that exist in the system, regardless of the number of capabilities for a specific variant. This design enables the flexible management of product families, empowering the allocation of diverse user roles across the system architecture.

Variability in System Analysis Level

System Analysis (SA) establishes the functions that the system must perform to fulfill the operational capabilities defined in the Operational Analysis and identifies the system's behavior with external system interfaces and interactions (Voirin, 2018). Operational capabilities, derived from higher-level requirements during the OA, provide a generalized definition of the system's objectives. However, at the SA level, a more detailed analysis is conducted towards achieving a solution, leading to a further breakdown of operational capabilities into more granular elements.

Expanding the definition of customer needs, the SA level articulates expectations from the system through system functions that validate each operational capability. By revealing the impact of these capabilities on the system, it provides an overview of system constraints stemming from requirements and system characteristics, resulting in various function alternatives. This supports trade-off analysis and business analysis by formalizing system needs (Voirin, 2018). Each capability's design load and distinction become clearer, driving design decisions, compromises, and the elimination of expected characteristics or variabilities in the system.

This level requires breaking down the capabilities into a hierarchical structure, identifying the child capabilities needed to achieve the desired capabilities. The capabilities realized requirements represented with the child capabilities. In the NFM, this corresponds to the capabilities child functional requirements features, this detail provides us to model the

variabilities in a family, if exists. The variation in the system carries through the capability realized system function and actor elements in the SA.

System Modeling of Variability in the Solution Architecture Space

Variability in Logical Architecture Level

The logical architecture level signifies the transition from need definition to solution definition within the system architecture. Here, the Logical Architecture delineates how the system will operate to fulfill its requirements/ capabilities. This phase involves breaking down the system into its logical elements, which define the system's logical architecture and execute the system's functional architecture. This level allows for the refinement of functions to provide a more detailed expression of the solution without unnecessary complexity, thereby enabling stakeholders to make informed decisions about the solution.

The breakdown of a system into logical components aligns with the concept of System-of-Systems (SoS) within the domain of systems engineering. Feature modeling enables hierarchical structure of the product lines across development boundaries, this supports the system SoS architectural concept (Krueger & Clements, 2013). SoS is characterized as "large-scale integrated systems that are heterogeneous and independently operable on their own but are networked together for a common goal" (Jamshidi, 2008). This paradigm emerges as a solution for managing the complexity inherent in large-scale systems by providing a framework for examining the interactions, behaviors, and capabilities of each constituent system.

In environments marked by evolving concepts of operations and technologies, maintaining traceability between the SoS architecture and capabilities is paramount. This traceability serves to mitigate the risks associated with the removal of redundant requirements, as such redundancies may still hold relevance within the context of individual systems (Azani, 2008). By preserving this traceability, organizations can prevent faulty limitations on capabilities, thereby ensuring the continued efficacy and adaptability of the system architecture in dynamic operational landscapes. This aspect is important since we are dealing with a system architecture with many variable elements.

Dividing a large system to smaller subsystems, to ensure the reduced complexity in the constraints, interactions and performance is important for an successful SoS design (Keating, 2008). Additionally, with the identification of the solution, new variation points for solutions may emerge at this stage, alongside existing variations, and variation points. Managing these variations, especially resulting from the interfaces of the subsystems, can be inherently complex, particularly in projects with operational variants, necessitating meticulous attention to ensure the accuracy of the resultant model. Interface management techniques play a vital role in managing this complexity, beginning with the identification of interfaces. It is a crucial aspect of system development, and there are many analysis methods to handle this complexity. In this study, the DSM modeling technique was

selected to identify interfaces between subsystems, facilitating effective interface management throughout the solution definition phase.

In the LA level, variations are inherently associated with the logical functions and actors which are aligned with capability realizations crafted at the SA level. However, this integration does not inherently encapsulate the logical components-which represent the subsystems- crucial for a robust solution architecture. At this point, the SFM proves invaluable, facilitating the linkage of essential logical elements required for the development of a comprehensive and accurate 100% model. By leveraging the SFM, organizations can effectively bridge the gap between the logical components and the overarching solution architecture, ensuring a cohesive and well-structured model that aligns seamlessly with project objectives and requirements.

Variability in Physical Architecture Level

At the Physical Architecture (PA) level, the development and build details of systems are specified, encompassing the physical components that comprise the system. This level of architecture studies the physical aspects of the system, providing comprehensive information about the physical composition, configuration, and layout of its components. It defines the specific hardware/software elements, materials and connections required for constructing the system.

In essence, the PA level serves as a blueprint for the physical realization of the system. The focus is on selecting the appropriate components that, when combined, provide the necessary structure and behavior to fulfill the functionality requirements. This process necessitates exploring alternative solutions to attain desired capabilities, thereby introducing variation points in the solution landscape.

Due to capability realization in Capella, physical functions and actors represents the variation in the physical level. Like in logical components, physical components also not affected by this involvement and the variations in the physical components managed by creating links to SFM features.

3.2.4 Traceability Between Development Spaces

In intricate systems characterized by both commonalities and variabilities, effective management necessitates the establishment of mapping across diverse perspectives and levels of abstraction. The identification of interrelations among modeling elements across varying levels of abstraction within product variant models yields critical insights into the development and maintenance of a product line. Facilitating an enhanced comprehension of the system and enabling the tracking of changes, traceability links are pivotal in establishing connections between these models.

Variant management is based on the establishment of two distinct types of traceability links. Firstly, the links formed between the feature model and the system model serve to encapsulate the variability inherent within the system model. This is particularly crucial

given that variant information within the system model alone may not be visible. Secondly, the traceability between feature models, encompassing both the need feature model and the solution feature model, holds significance in managing relationships among variant features. These links provide invaluable insights into the relations and constraints of features across different variants, thereby enhancing the efficacy of variant management. A general overview of the constructed relations between feature models and system model in need and solution spaces given in Figure 10.

Traceability Between Feature and System Models

While maintaining traceability from upper levels to lower levels is a customary practice in system engineering, there is often insufficient emphasis placed on forming the relationship between features and requirements. Across various approaches, focus tends to be on establishing connections among features to capture the effects of feature changes and uphold consistency. However, this approach often overlooks the feature changes implications to the requirements, or vice versa.(Yu et al., 2012) Given that requirements directly correspond to user needs, such oversight can inadvertently yield undesired outcomes in the final product.

As previously noted, the process of defining system variation encompasses the identification of commonalities and the execution of variability analyses rooted in system capabilities derived from requirements. Recognizing the pivotal role played by requirements in shaping system functionality and aligning it with user needs is crucial. By adopting a perspective that integrates the linkage between features and requirements, organizations can effectively mitigate the risk of delivering products that fall short of user expectations or exhibit inconsistencies in functionality.

This objective is realized through the establishment of need level feature model, wherein variant capabilities and requirements are identified as features and mapped accordingly to corresponding system model capabilities in OA and SA levels. Managing functionality descriptions within the need level mapping, solution level involves the utilization of solution feature models to encapsulate the variations in behavior and the specification of components. This necessitates a harmonized construction of feature and system model elements by mapping the variant solution architecture components in LA and PA to solution feature components in the SFM. Feature mapping facilitates the transformation of variation points delineated within the FM into technical representations through variant management tools, thereby facilitating the creation of a family model in the SM.

Traceability Between Feature Models

In addressing the challenge of information deficiency within the system model related to variant relations, a strategic approach involves establishing relationships within feature models and their constituent elements. While feature models are categorized into need and solution spaces, it is imperative to study traceability links between these distinct FMs. This traceability mechanism is pivotal in safeguarding crucial insights of solution features. Furthermore, studying feature relations within the feature models is indispensable for

crafting a precise and comprehensive model. These relationships are realized through the creation of dependencies among features within the feature models. These dependencies serve to regulate the selection of conflicting features simultaneously or automatically select dependent features. Key dependencies, such as require and exclude, or conditions formulated in the pvSCL language, form the cornerstone of constructing these relations. They provide a structured framework for managing feature interactions and ensuring coherence within the model, thereby facilitating effective variant management and system development processes.

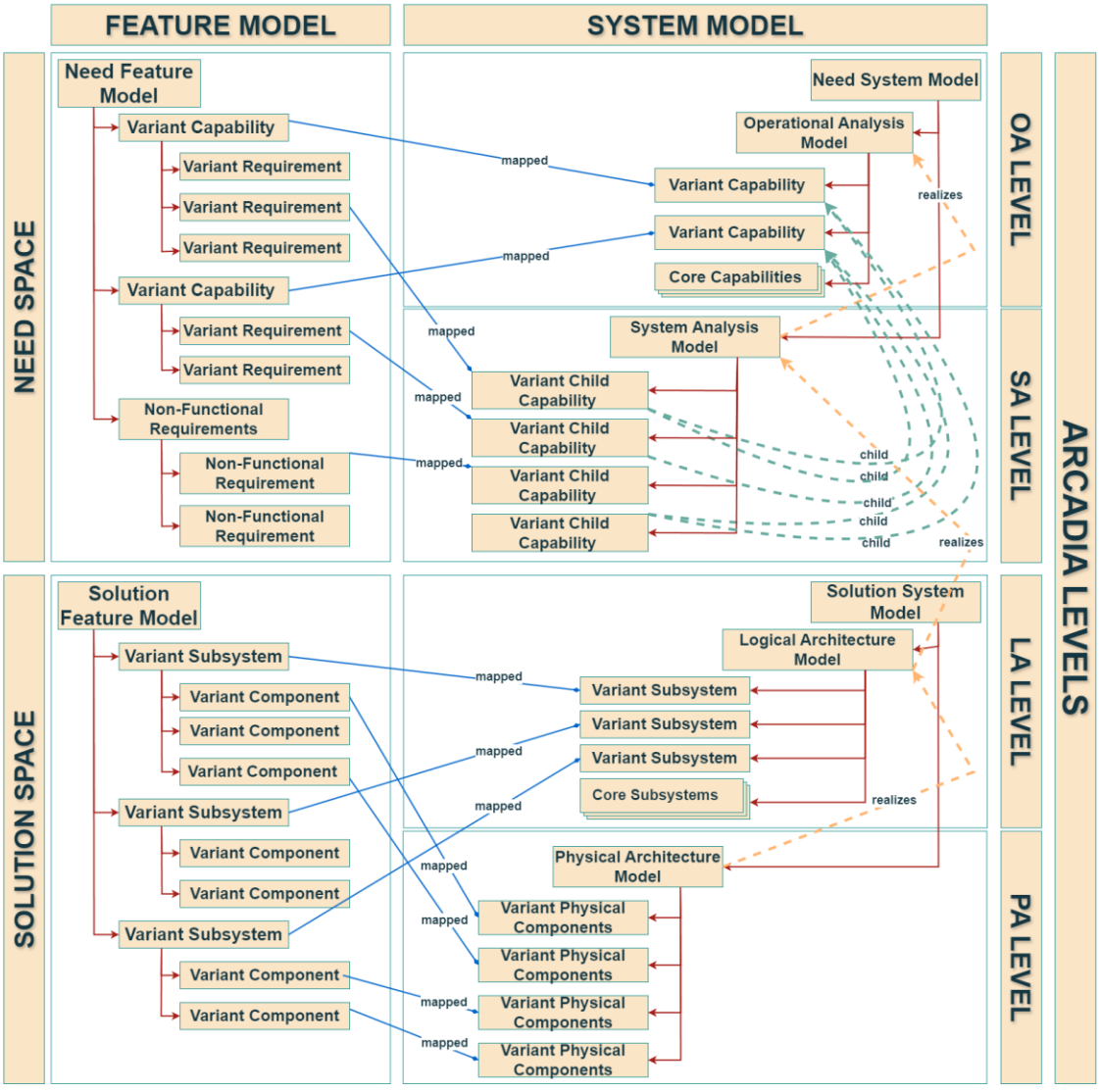

Figure 10 : Overview of relations between Feature Models and System Model in development spaces

CHAPTER 4

AN ILLUSTRATIVE APPLICATION OF THE CAPLA METHODOLOGY

In this chapter, we present the implementation of the CAPLA methodology developed using Pure::variants and Capella tools. Building upon the methodology outlined in the previous chapter, we demonstrate its application to an example technical architecture. This chapter offers a detailed examination of the outcomes and valuable insights derived from this implementation process.

4.1 System of Interest

The workflow of this study is oriented towards the development of feature models and system models, with a primary focus on establishing a robust linkage between them. Initially, the SM is conceptualized as a comprehensive 150% model, encompassing all potential variants within the system. Subsequently, through the bridge mechanism, 100% models are derived to represent specific variants, streamlining the modeling process for practical application.

To illustrate the CAPLA in a tangible context, a proof-of-concept approach is adopted, wherein an example system is defined. This example system is intentionally kept as simple as possible to optimize time and efforts, allowing for a concentrated exploration of the methodology's intricacies. By emphasizing clarity and conciseness, the example system serves as a vehicle for effectively conveying the methodology's underlying principles and processes.

In representing an aerospace vehicles product line, military drones are chosen as the focal example system, from which three distinct product families have been constructed based on differing operational concepts. Product line of a drone refers to a series of drones offered by a manufacturer that share common design elements, features, and capabilities but may vary in terms of size, payload capacity, range, and specialized functions to fill the different market needs or applications. The example drone variants in the study are meticulously designed to meet the varying demands of military operations; encompassing surveillance, communication infrastructure provision, and precision strike capabilities.

The development of such product lines may diverge depending on the adopted business model. The decision to design and model these products within a single cohesive product line or across multiple lines is dependent upon the strategic objectives of the company. However, the overarching aim in modeling a system and its variants is to boost profitability while mitigating engineering effort and errors, achieved through the maximization of reuse and minimization of complexity. However, in this thesis, maximizing the reuse rate is prioritized to ensure that the 150% model holds significance within the context of product line engineering thus all these drone variants covered in a single model. The reuse rate kept high with assumptions and constraints defined in the model for all stages of the development process.

A foundational assumption that supports this approach is the modularity of the drone, enabling customization through the integration of diverse payloads and sensors to meet specific operational requirements while preserving a standardized body structure. This modular design promotes agility and adaptability, facilitating efficient development processes and ensuring the overarching coherence of the product line.

Distinct product families emerge based on variances in operational concepts. These families are the "Bomber Drone," typically deployed in military contexts to execute strategic strikes against enemy targets; the "Surveillance Drone," utilized for aerial surveillance, monitoring, and data collection; and the "Relay Drone," fulfilling the crucial role of establishing communication relay nodes within a network framework. Through these three product families, apart from their operational concept variance, variations in users, functions, components, and physical interface are supported with examples and their variance in occurrence in the design process. Several characteristics are common across all three product families, with each evolving at various stages of the development process. However, nuanced differences arise to address specific performance and functionality requirements unique to each drone variant.

For the Surveillance Drone, the standard E/O Camera does not meet the aerial imaging performance expectations, necessitating the use of a more advanced E/O Camera to ensure high image quality. In case of the Bomber Drone, conflicts between munition payload and performance requirements arise. As a compromise, adjustments in endurance were made to achieve a balance between payload capacity and operational capabilities.

Conversely, the Relay Drone presents users with options to customize its capabilities. Users can choose mesh networking capability and success rate of adaptive modulation, tailoring the drone's functionality to suit their specific needs. Furthermore, this dynamic formation of the Relay Drone allows for the creation of three additional configuration options, bringing the total to six distinct product configurations. This includes one for the Surveillance Drone, one for the Bomber Drone, and four configurations tailored to the Relay Drone, each offering unique features and capabilities to meet diverse user requirements. All configurations are given in Table 1.

Table 1 : Drone Configurations

Configuration Name	**Specific Functionality**
Surveillance Drone	Advanced Camera
Bomber Drone	Payload+ Reduced Endurance
Relay Drone_00	Basic Relay
Relay Drone_01	Mesh Relay
Relay Drone_02	High reliability for adaptive modulation
Relay Drone_03	Mesh Relay + High reliability for adaptive modulation

The system of interest decomposed into subsystems and these subsystems are defined using the ATA numbering system, a common standard for referencing subsystems in aviation (Greenough & Williams, 2007). In this study, subsystems include the ATA 22 Auto Flight Subsystem, ATA 23 Communication Subsystem, ATA 24 Electrical Power Subsystem, ATA 34 Navigation Subsystem, ATA 72 Engine Subsystem, ATA 93 Surveillance Subsystem, ATA 94 Weapon Subsystem, and ATA 43 Tactical Communication Subsystem. In the model, functions are limited to missions representing variability in the system, thus excluding core capabilities like landing and structural integrity. Consequently, ATA chapters such as ATA 32 Landing Gear Subsystem and ATA 53 Fuselage Subsystem are not included to maintain simplicity in expressing the methodology, assuming that other subsystems remain unaffected by variability in the system.

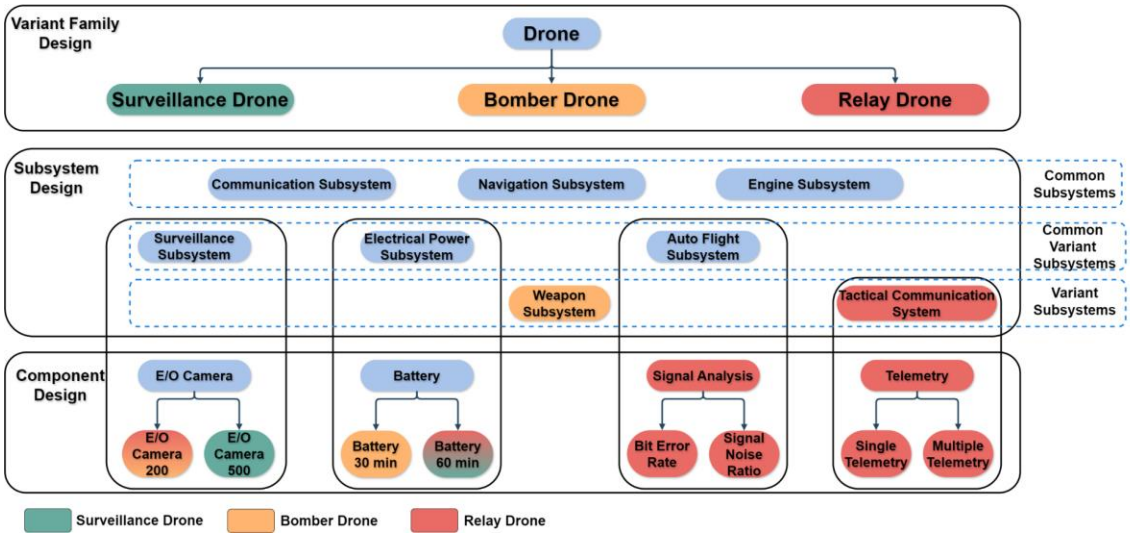

Figure 11 : Subsystem and component variability for Drone Technical Architecture in design levels

Figure 11 shows the variance in subsystems and components across different design levels due to variations. For example, as previously mentioned, the conflict between the munition payload and performance in the Bomber Drone creates a variance in the battery component within the system. While the Surveillance Drone and Relay Drone can use the same battery (shown in both red and green), the Bomber Drone requires a different battery (shown in yellow) to meet its specific needs.

4.2 Variability Management in Need Space

In the need space, management begins with the identification of variabilities and continues with the joint development and mapping of the SM and the NFM. This comprehensive approach enables a better understanding of how different variabilities impact the system, providing a robust foundation for analyzing and managing the complexity inherent in meeting diverse customer needs. In Figure 12, the overview of activities and their interactions within the developed methodology for managing variability in the need space

is modeled. This figure illustrates the comprehensive process of identifying variabilities, and how these variabilities are managed through the simultaneous development of the SM and the NFM. By capturing the operational activities and their interactions, the figure provides a clear visual representation of the methodology, highlighting how the coordinated development of SM and NFM supports effective decision-making and trade-off analysis in addressing diverse customer needs.

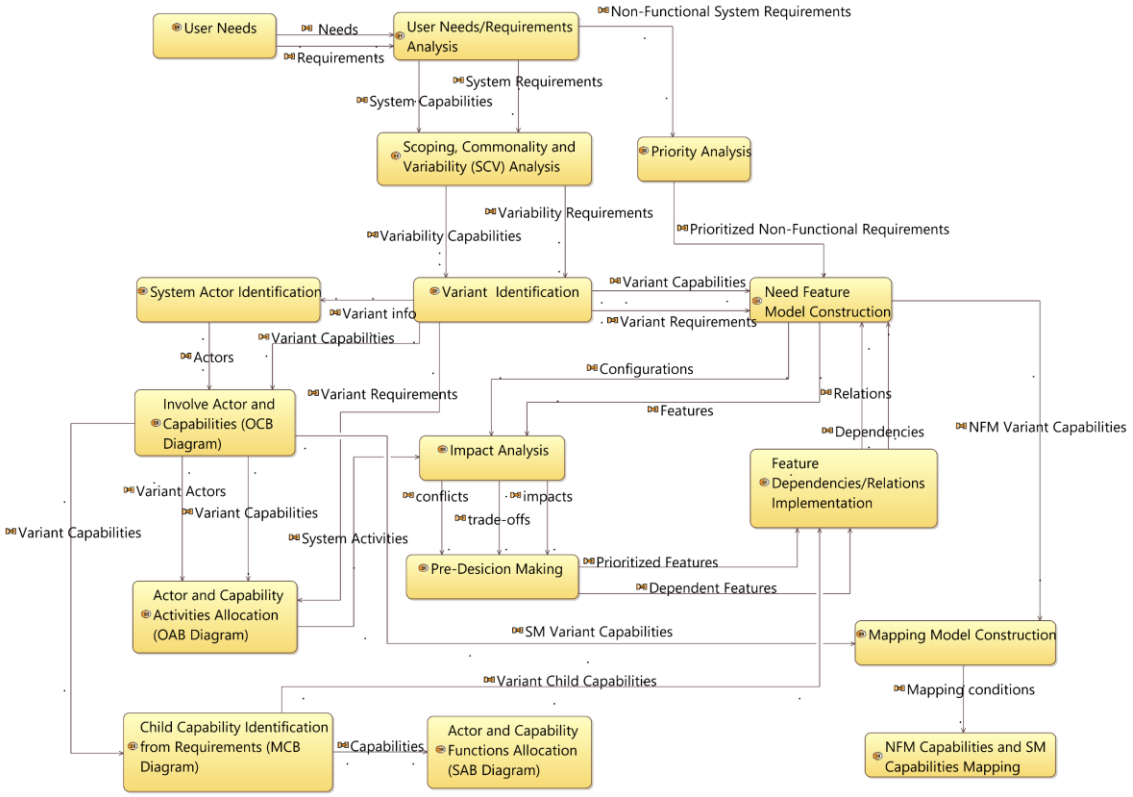

Figure 12: Overview of the activities for variability management in need space

4.2.1 User Needs and Requirement Analysis

Conducting initial SCV analysis, capabilities and requirements classified as “Core” and “Variant”, as given in Table 2 and Table 3 respectively.

Table 2 : Core Capabilities and Requirements

Core Capabilities and Requirements		
CC1:	Perform Controlled Flight	
	CR1:	The drone shall be capable of taking off vertically and landing safely.
	CR2:	The drone shall be able to maintain stable flight and hover at a specified altitude.
	CR3:	The drone shall respond to user inputs for controlling flight direction, speed, and altitude adjustments.
	CR4:	The drone shall incorporate flight stabilization features to compensate for external factors such as wind gusts and turbulence.
CC2:	Obstacle Avoidance	
	CR5:	The drone shall be equipped with obstacle avoidance sensors to prevent collisions with objects during flight.
CC3:	Remote control via Ground Station	
	CR6:	The drone shall establish communication links with a ground control station (GCS) for remote operation and monitoring.
	CR7:	The drone shall support bidirectional communication for receiving commands from the GCS and providing status updates and feedback to the operator.
	CR8:	The drone camera system shall be equipped with a high-resolution camera capable of capturing images with a minimum resolution of 12 megapixels.
CC4:	Real-time data transmission	
	CR9:	The drone shall transmit telemetry data to the GCS in real-time, including GPS coordinates, altitude, battery status, and sensor readings.

Table 3 : Variant Capabilities and Requirements

Variant Capabilities and Requirements		
V1_VC1:	Provide situational awareness	
	V1_VR1:	The surveillance drone system shall be capable of conducting surveillance missions in both day and night conditions.

Table 3 (continued).

	V1_VR2:	The surveillance drone system shall be equipped with a high-resolution camera capable of capturing still images with a minimum resolution of 20 megapixels.
	V1_VR3:	The surveillance drone system shall support real-time video streaming with a minimum resolution of 1080p at 30 frames per second.
	V1_VR4:	The camera shall have zoom capabilities (optical or digital) for close-up inspection of targets or areas of interest.
V2_C1:	Destroy target	
	V2_VR1:	The bomber drone system shall be capable of delivering munitions accurately to designated targets with precision-guided capabilities.
V3_C1:	Act as a communication relay	
	V3_VR1:	The relay drone shall have the capability to establish and maintain communication links with ground-based users, other drones, or satellite networks.
	V3_VR2:	The relay drone system shall be equipped with adaptive signal modulation capabilities to dynamically adjust modulation schemes based on environmental conditions, interference levels, and link quality
	V3_VR3:	The drone system shall provide communication coverage over a minimum area of 100 square kilometers with a range of at least 50 kilometers from the drone's location.
	V3_VR4:	The relay drone shall be equipped with mesh networking devices to establish and maintain communication links with other drones and ground stations within the network.

For the non-functional requirements (NFR) the distinction is not necessary since the non-functional requirements validation is going to be valid for all products if not specifically written for a specific variant. Stakeholder expectations prioritized since conflicts in the system requirements can result in variation points in the system which will be studied later. NFRs are given in Table 4.

Table 4 : Non-functional Requirements

Non-functional requirements		**Surveillance Drone**	**Bomber Drone**	**Relay Drone**
NFR_1:	The system will have a maximum weight of 10 kg.	M	M	M
NFR_2:	The drone shall have a maximum speed of 50 kilometers per hour.	M	M	M
NFR_3:	The drone shall be capable of achieving a maximum altitude of 500 meters above ground level.	M	M	M
NFR_4:	The drone shall have a minimum flight endurance of 60 minutes to support extended operations.	M	N/A	M
NFR_5:	The drone shall have a carrying capacity of 2 kg munitions.	N/A	M	N/A
NFR_6:	The drone shall have a 80% rate of success of signal modulation.	N/A	N/A	O
NFR_7:	The bomber drone shall have a minimum flight endurance of 30 minutes to support extended operations.	N/A	M	N/A
M:Mandatory O:Optional				

The VC1, VC2, and VC3 capabilities correspond to variances in the operational concepts outlined earlier, defining the three product families: Surveillance Drone, Bomber Drone, and Relay Drone, respectively. To explore the potential variance types throughout the product line design process, certain constraints and assumptions are established.

For instance, the V3_VR4 requirement serves as an optional child capability for the Relay Drone, exemplifying a variation point emerging from a system functionality requirement. NFRs are also crucial considerations for each product family, as outlined in Table 4. NFR_6, expressing the reliability of adaptive modulation (V3_VR2), is deemed optional due to the trade-off analysis conducted during development stages, considering its impact on workload and cost.

Similarly, performance analysis during development revealed that fulfilling NFR_1, NFR_2, NFR_3, and NFR_4 simultaneously, along with V2_VR1, is untenable for the system. Consequently, a compromise is made from endurance, with NFR_1, NFR_2, and NFR_3 deemed essential performance requirements, and V2_VR1 designated as mandatory. Moreover, an additional endurance non-functional requirement, NFR7, is introduced specifically for the Bomber Drone.

For the Surveillance Drone, a variation point emerges due to the definition of an advanced functionality for the surveillance aspect of the system. While the core system necessitates an electro-optic camera requirement (CR2), conflicts arise with V1_VR1/VR2/VR3/VR4, which represent similar functionalities. Since these cannot coexist, a variation point for the electro-optic camera solution emerges from this conflict.

4.2.2 Feature Modeling in Need Space

With the identification of capabilities and requirements, it is feasible to construct the need feature model (NFM). The variability tree can be initially seeded, in its early phases, from the previously conducted business analysis or the common and variant capabilities identified at the project's inception.

The variant capabilities and requirements are modeled using Pure::variants. The CC's and CR's were not modeled in the system since they have no representation for variability in the system and it will create unnecessary load to the variability model.

The VCs are modeled as alternative features since their operational concepts are differentiated in the business and user analysis. The corresponding functional requirements are modeled as mandatory child features of the capabilities.

Modeling the NFRs can be bit trickier than the functional requirements since in terms of system performance, reliability and safety, system must be considered as a whole and variants in the system can affect these requirements. Also, variants can have other non-functional requirements that conflict with the core non-functional requirements. To manage these requirements, unlike core requirements, core non-functional requirements modeled in the NFM as mandatory features. The variant NFRs and core NFRs are not separated under capabilities or variants in the model, the distinction is managed through the relations formed between the features.

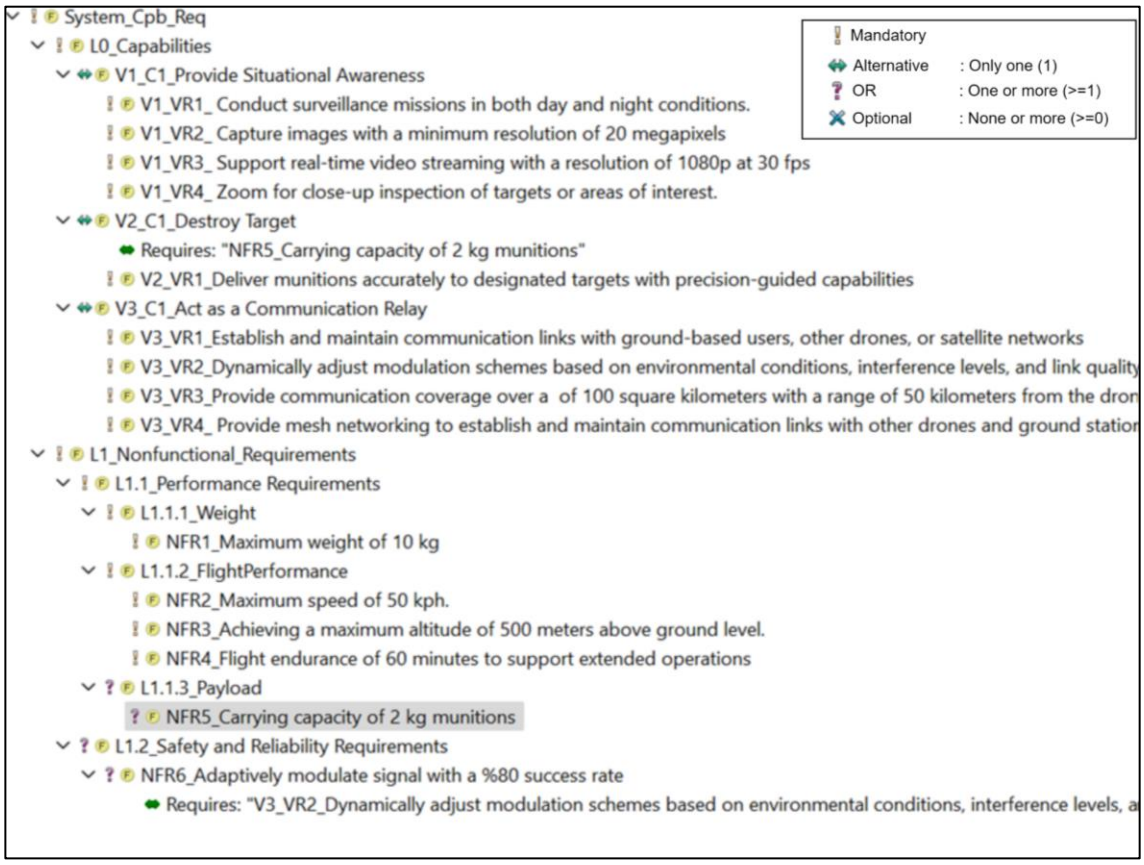

Figure 13 : Initial Need Feature Model

For the NFRs as can be seen in Figure 13, the categories like performance and safety are implemented for the readability of the model. The feature types of the categories are important for the correctness of the model. A general rule can be said for cases like these, even if a single mandatory child feature exists under a parent feature, that parent feature must be mandatory also. In other cases, if a single child exists, the parent and child can be same type but for multiple different type child containing parents each case must be analyzed separately.

NFR5 is a variant specific NFR, it is implemented as an optional feature so that we can link the requirement to our desired capability and the other capabilities are not affected by the exitance of the feature. The required relation formed between the NFR6 and V2_C1 supports the automatic configuration, by selecting V2_C1, NFR6 automatically will be selected in the model. The selection of this feature can also be prohibited to the other features but since we are forming a feature model that can be managed through capabilities, with selection of the capabilities and correct linking, the wrong selections in the model prohibited without too much effort

NFR6 s also a variant specific NFR, but as can be seen, the relation is formed differently to show the variation possibilities that one encounters in a project. In the need analysis,

this requirement was defined as a desired requirement by the user. NFR5 is defined as a must requirement for that variant so it can be linked to the capability. NFR6 is a desirable feature, so the feature will be presented to the user as a choice, and it is implemented as an optional feature. It is directly related to V3_VR2 requirement, for traceability in design decisions in the system, a relation is formed with requirement instead of capability. Unlike the previous case, the requirement does not need the NFR to validate itself but on the contrary the NFR feature needs the requirement to validate itself, it cannot be present if the requirement is not selected so the relation is linked to the NFR itself. This way, while forming configurations, the relation prohibits the selection of NFR6 if the V3_C1 is not selected therefore V3_VR2 is not selected, which is examined in detail with other relations in the following sections (Figure 33).

4.2.3 Impact Analysis and Pre-decision Making in Need Space

In the previous phase, the interrelations among features are established based on the initial needs analysis conducted. As these models take shape, a more comprehensive understanding of the overall system evolves, facilitating further analysis. This step is not necessary but strongly recommended for projects seeking a robust modeling foundation.

During this stage, how the requirements impact the entire system are analyzed. The features corresponding to the variants may conflict with business objectives and system requirements. Conducting preliminary performance, cost, and risk analyses helps generate more realistic NFMs.

In this specific case, such analysis led the system to adjust the NFRs for certain variants. It was revealed that the bomber system failed to meet performance requirements due to weight constraints, necessitating a trade-off. Emerging from mandated uniform weight across all variants, priority was assigned to specific NFRs—NFR2, NFR3, and NFR5—. Any deviation from this weight constraint would result in both a cost escalation and a misalignment with business goals. Consequently, a variation point emerged from these decisions. Given that a 2kg payload capacity was mandatory, compromises were made from the endurance, NFR4, and a lighter battery chosen to align with performance requirements and a new requirement, NFR7, added to endurance requirement set. The updated NFM reflecting these decisions is depicted below.

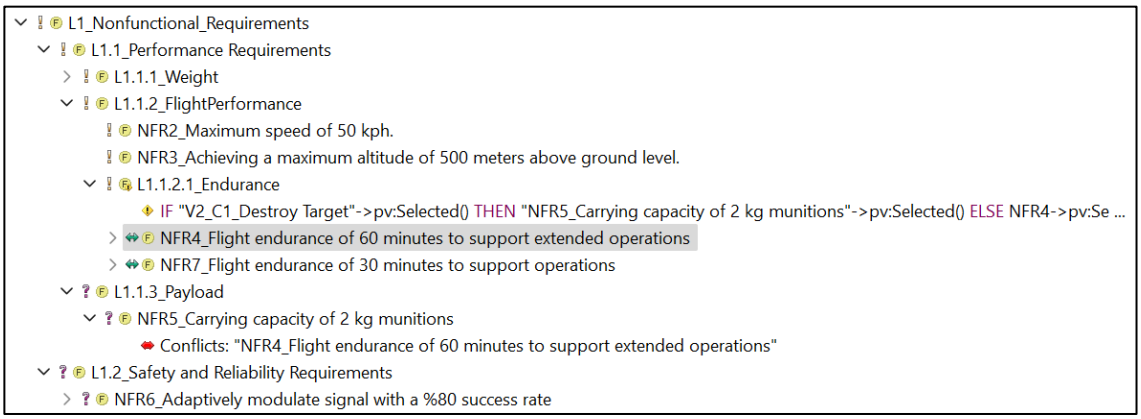

Figure 14 : Updated Need Feature Model (Addition of NFR7)

Within the system, the endurance requirement presents two possibilities: "30 minutes" and "60 minutes". Given their mutual exclusivity, these NFR features are implemented as optional features. However, the parent category feature implemented as a mandatory feature to maintain the necessity of making a choice for the endurance. To maintain traceability and rationale behind these choices, relations and constraints are enforced.

It was analyzed previously that NFR4 and NFR5 are conflicting features because of the weight restriction so a conflicting relation is implemented to NFR5. Following the methodology, features are predominantly controlled via capabilities, ensuring that a single capability choice automatically selects all associated features, thereby self-assembling the feature model for that variant. To achieve this, the endurance alternatives are managed through restrictions. In the pvSCL language, the following restriction is formulated for automatic alternative selections:

IF V2_C1->pv:Selected() THEN NFR5->pv:Selected() ELSE NFR4->pv:Selected() ENDIF

This logic ensures that if V2_C1 is not selected, NFR4 will be automatically selected for all other variants in the NFM.

4.2.4 System Modeling in Need Space

This section offers an overview of the strategic approach to integrating the previously constructed product line into the system model at the earliest possible stage. The features and decisions are implemented into the various levels of Capella, taking into consideration the representations unique to each level. The NFM developed up to this point is particularly relevant to the Operational and System Analysis Levels in Capella, as it effectively represents the system's needs. Moving forward to address system solutions, FM and SM are jointly developed.

Operational Analysis Level

Operational capabilities serve to refine operational needs and are often described through various operational activities undertaken by actors or entities. These activities illustrate how operational capabilities are realized in practice and provide insights into the interactions and workflows involved in achieving a particular capability. Unlike NFM, SM was constructed to define the system, therefore the SM includes the whole system behavior and the variations, and this results in a 150% model.

While variant capabilities are represented by actors in the system, core capabilities—relevant across all variants—are linked to an operational entity termed "Drone". Once operational capabilities are allocated to actors, entities, and the environment in Operational Capabilities Blank (OCB) diagram, their interactions are modeled alongside the allocated activities aimed at fulfilling the mission objectives.

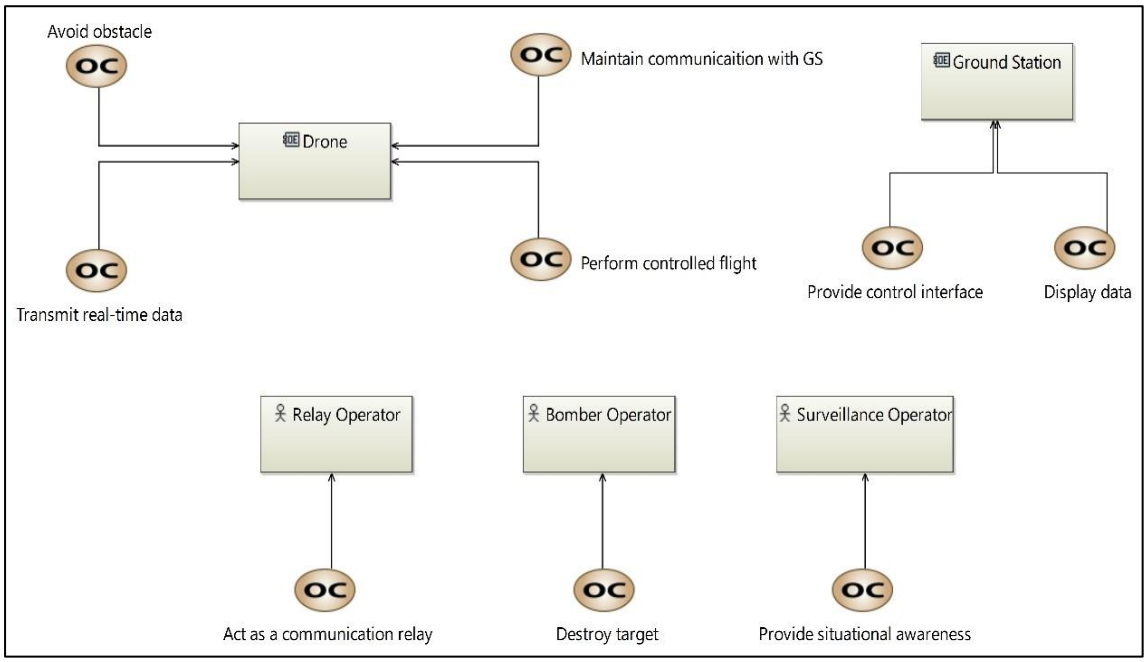

Figure 15 : Operational Capabilities Blank (OCB) diagram of 150% Drone system

At the operational level, the NFM primarily resides at the capabilities and requirements level with their relationships. Moreover, relevant activities supporting these capabilities are linked with the involvement relation in Capella to ensure traceability between variation-related and core activities of the system.

Effectively managing variations through capabilities ensures that the integrity of other diagrams, within the OA, remains intact despite potential changes in system variability. In the Capella environment, Architecture Blank diagrams hold a significant importance across all engineering levels as they comprehensively define the system architecture, encompassing actors and capabilities activities along with their interactions. In Figure 16,

the activities necessary to achieve the capabilities are allocated to actors and drone system. This diagram serves as a key tool, offering a holistic representation of system architecture by allocating the activities necessary to achieve the capabilities to actors and drone system.

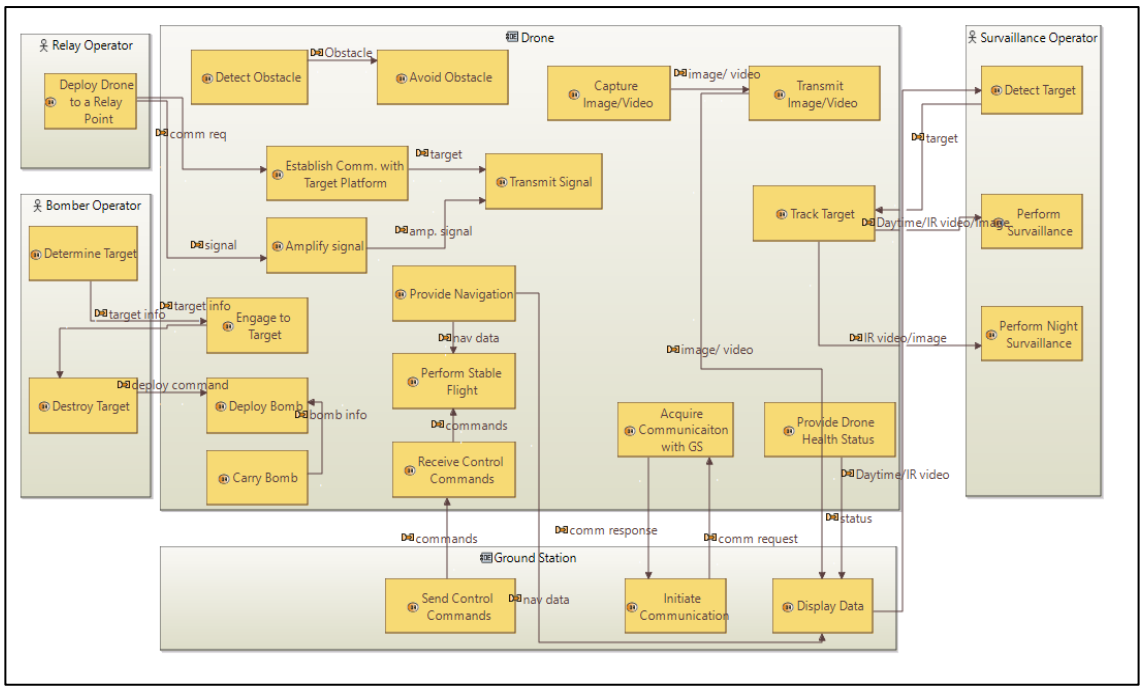

Figure 16 : Operational Architecture Blank (OAB) diagram of 150% Drone System

The OAB diagram is a complex diagram compared to other diagrams in the OA level, provides a consolidated view of all activities and actors, accommodating variations within the system. In the OCB diagram, the variants distinction is clear but in OAB this distinction is only visible in terms of actors. As it can be seen, the activities allocated to the Drone system contains many activities belong to various capabilities including the common ones. Making sure that every activity involved with a capability, in OAB variability information is still traceable. Through testing and validation, the methodology has demonstrated its efficacy to maintain the variability information across various diagrams within the OA level including the OAB.

Another facet of traceability is established by mapping operational capabilities between the SM and NFM, thus ensuring that each variation point corresponds to an operational analysis modeling element. This linkage between models is facilitated through the creation of a mapping model, wherein Pure::variants add-on acts as a bridge between the SM and FM, mapping SM elements to NFM features. In this mapping model (Figure 17), capabilities in the FM are integrated as conditions, and operational capabilities are attached to these conditions to cover a broad spectrum of modeling elements across various levels and diagrams.

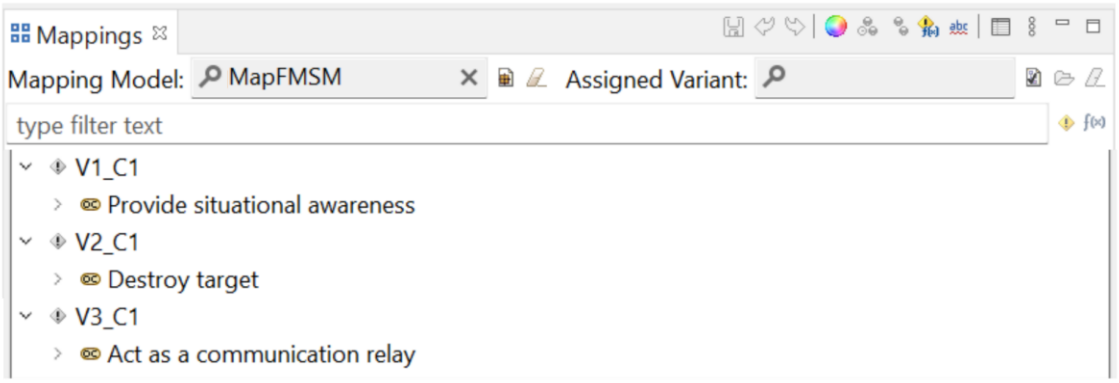

Figure 17 : Mapping model between FM and SM

The management of SM elements' coverage is governed by the propagation rules established within Capella. These propagations are facilitated through realization and involvement links between Capella elements. At the Operational Analysis (OA) level, capabilities are interconnected with actors and operational activities through capability involvement links. By mapping these SM capabilities to NFM elements, the involvements are automatically extended to the corresponding NFM element. An additional benefit of this approach is that if an actor or element is involved with another capability aside from the mapped one, their involvement is removed from the mapping model. This ensures the creation of a correct 100% model post-transition without any loss of elements.

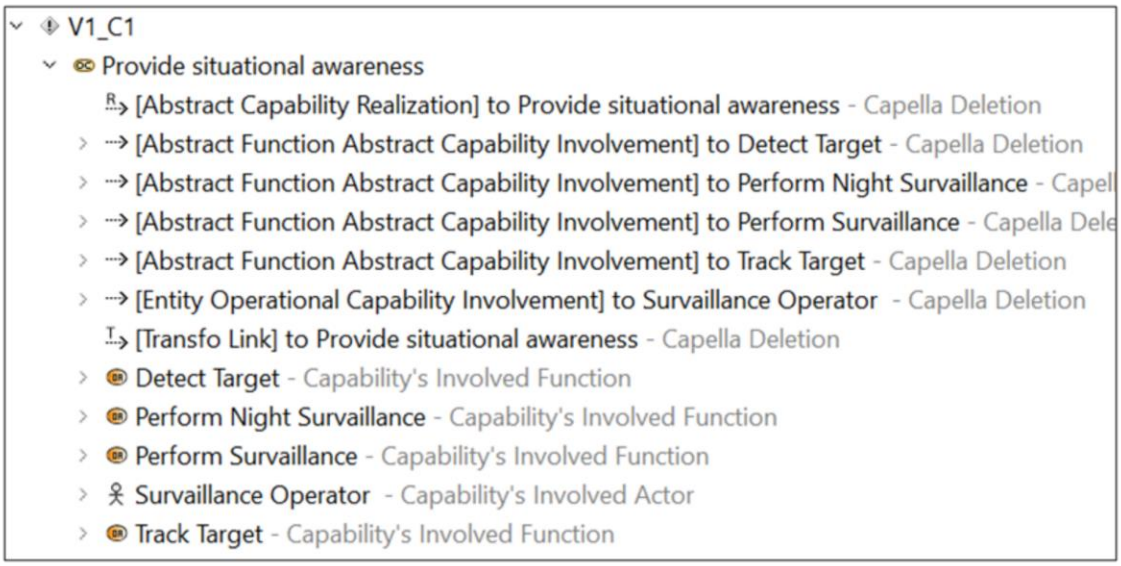

Figure 18 : Example of automatic mapping of elements through capability involvement

Another essential diagram is the Scenario diagram across all levels, which depicts a sequence of messages exchanged between model elements. These scenarios must be linked to specific capability to be created. In scenarios where a capability is absent in the 100% model, the corresponding Scenario diagram will also be omitted. Additionally, in instances where a model element is shared across multiple capabilities, it remains present in other scenarios due to its multiple involvements.

At this level, it is not necessary to create configuration space in the Pure::variants and assign the features in the NFM to the variants, but it is a helpful practice for controlled modeling. Six variant model is created in the configuration space according to the configurations determined in Table 1 as “Bomber Drone”, “Relay Drone 00/01/02/03” and “Surveillance Drone”, as given in Figure 19.

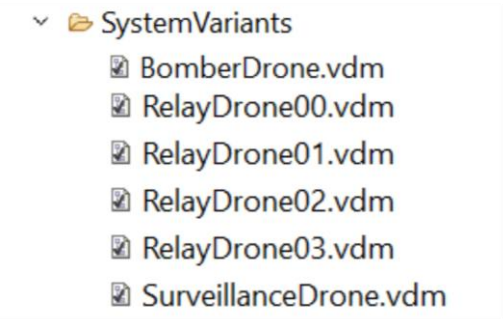

Figure 19 : System Variants

For each variant, NFM is a blank canvas and with the constructed dependencies by selecting a capability, the NFM automatically configures itself. As an example, in Figure 20 shows the feature configuration of Surveillance Drone. By selecting only V1_C1 capability, the other capabilities and non-relevant NFRs selection automatically prohibited in the NFM.

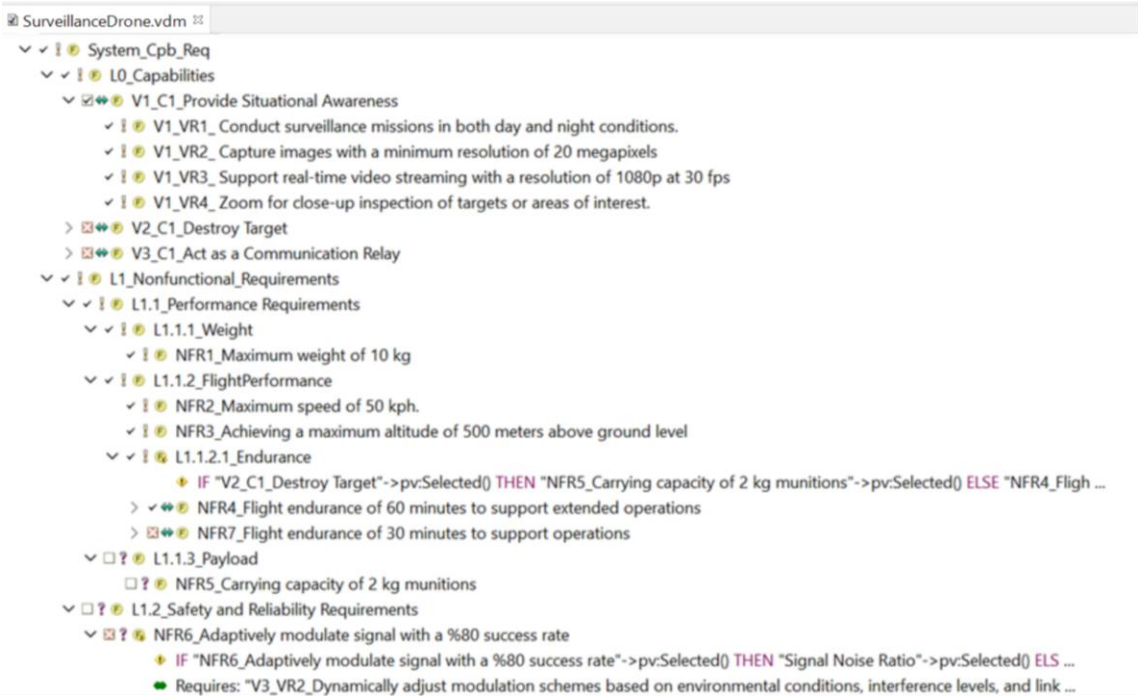

Figure 20 : Surveillance Drone NFM configuration

With the conditions in the mapping, the result models in OA for the selected variant model can be seen in grey-out mode or hide mode. When transition made from 150% model to 100% model at the end, unrelated SM elements will be removed and not be visible. In Figure 21 and Figure 22, according to the FM configuration of Surveillance Drone given in Figure 20, the modeling elements specific to other variants are in grey-out mode, the 100% resultant models can be seen in APPENDIX A.

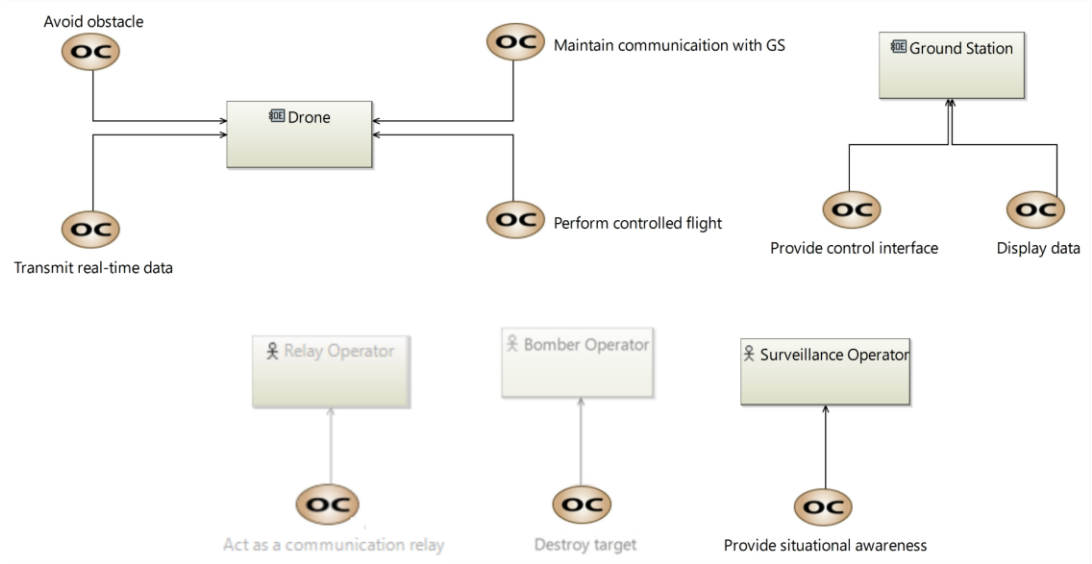

Figure 21 : 100% OCB diagram of Surveillance Drone (Grey-out mode)

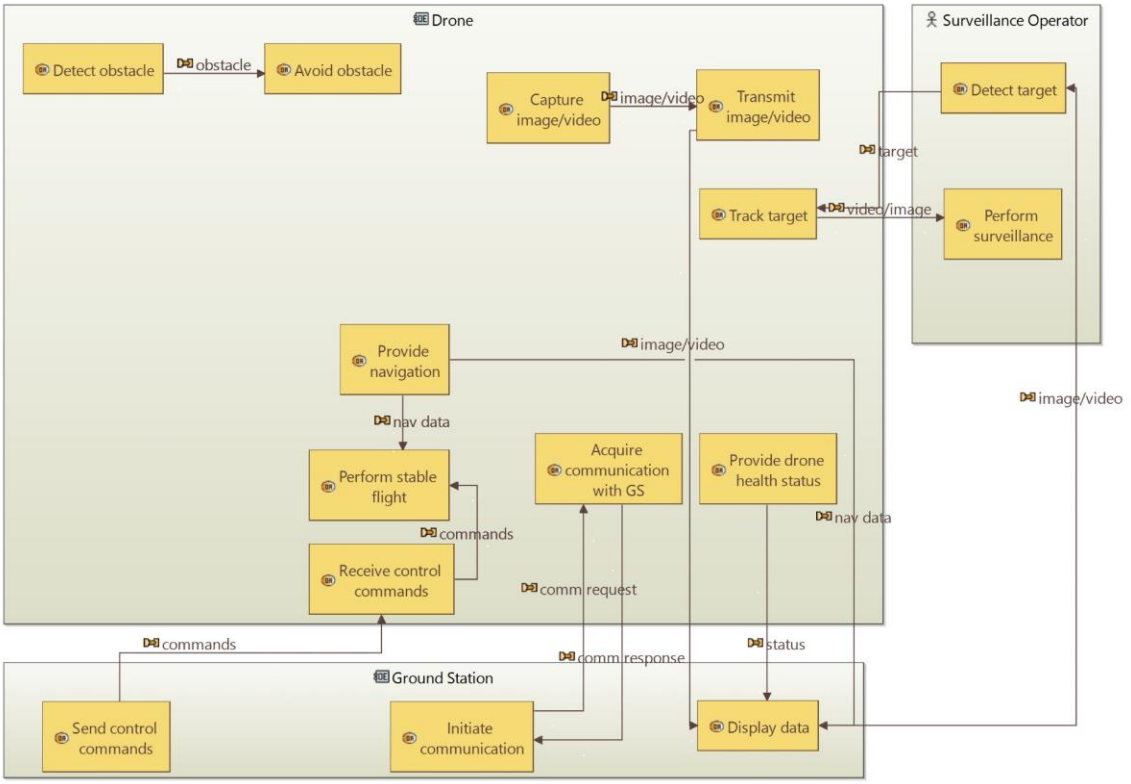

Figure 22 : 100% OAB diagram of Surveillance Drone (Hide mode)

System Analysis Level

To streamline the modeling and traceability processes, Capella enables automatic transitions of certain modeling elements between levels. Operational capabilities transition into system capabilities, operational activities evolve into system functions, and operational actors are transformed into system actor elements during the Capellas automatic transition from OA to SA. Notably, since the Drone operational actor represents the system in the operational analysis, it remains unchanged during actor transitions.

In SM, child system capabilities are created that corresponds to the requirements constructed in the NFM and include relation is formed with their related system capabilities derived from operational capabilities in Mission Capabilities Blank (MCB) diagram (Figure 23). The core capabilities are also a part of this diagram and transitions but to maintain the simplicity to express the methodology, in specific diagrams core capabilities excluded.

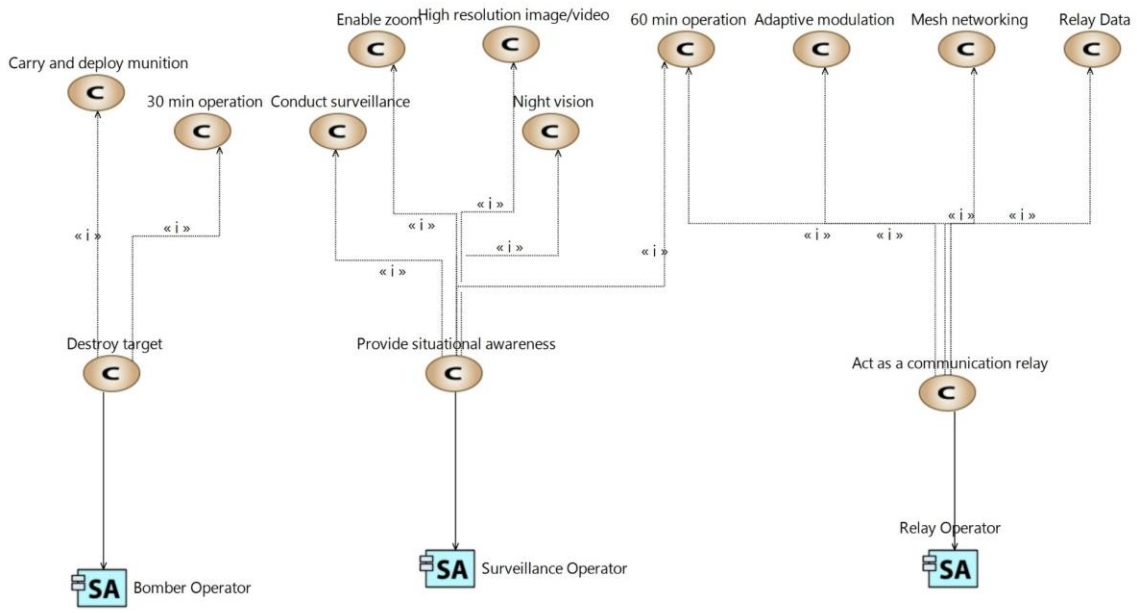

Figure 23 : Mission Capabilities Blank (MCB) diagram of 150% Drone System

Variability in the SA phase can be further delineated by assessing customer importance, system essentiality, and feasibility. Requirements specified in the NFM are re-evaluated based on these criteria, leading to the classification of variant requirements as necessary, desirable, or unnecessary. For instance, in this study, the requirement of mesh networking (V3_VR4) is deemed desirable due to its design complexity, resulting in additional costs for the customer. Consequently, the feature type of V3_VR4 is updated to optional in the NFM, while the V3_C1 remains unchanged, as other requirements continue to define this capability (Figure 21).

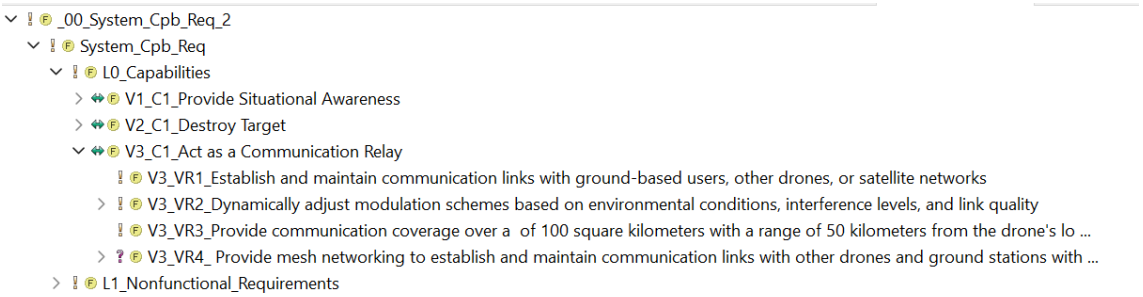

Figure 24 : Updated Need Feature Model (Type change of V3_VR4)

These changes can be observed in the System Architecture Blank (SAB) diagram (Figure 25). This diagram serves as an instrumental tool in defining system functions and delineating exchanges between them, thereby facilitating the realization of capabilities alongside system actors and entities. While detailing system functions at this stage can augment model maturity, it is paramount to uphold traceability between capabilities and

functions, ensuring that new functions are aligned with the operational capabilities they serve.

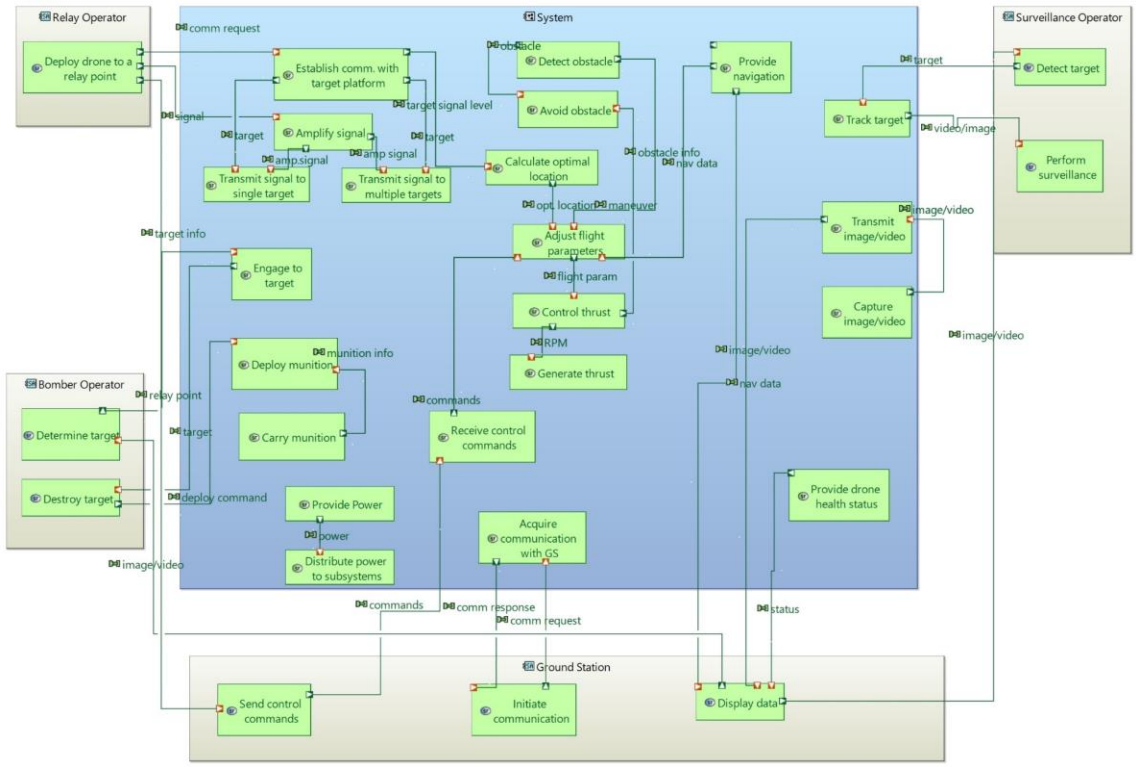

Figure 25 : System Architecture Blank (SAB) diagram of 150% Drone system

For instance, within an OA framework, the function "Transmit signal", due to its association with the "Mesh Networking" child capability, is decomposed into "Transmit signal to single target" and "Transmit signal to multiple targets". At this juncture, a Capellas involvement link is established between the "Transmit signal to multiple targets" function and the "Mesh Networking" child capability in the SM (Figure 26). This way, if the "Mesh Networking" capability is not selected in the NFM, the "Transmit signal to multiple targets" function will not be included in the resultant model.

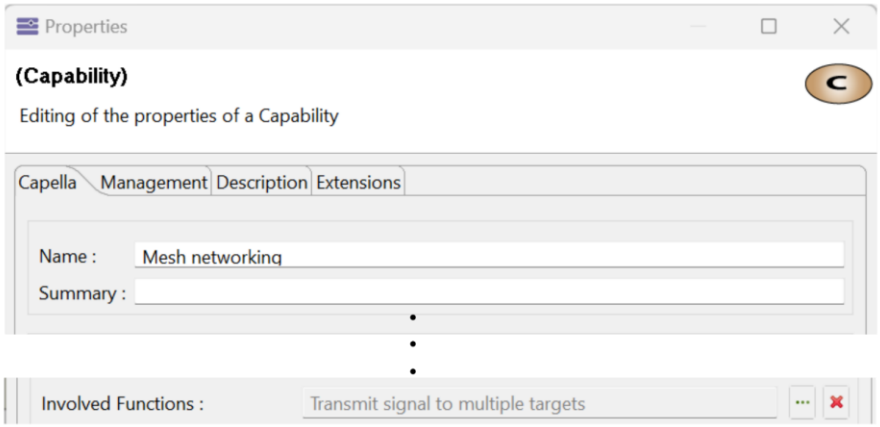

Figure 26 : Capability and function involvement in Capella

In cases where a variant encompasses variation points within its child capabilities, an additional mapping must be incorporated into the mapping model. For instance, even if the parent Relay Drone capability is deemed valid, its requirement may not be valid due to selections made in the NFM. For this, V3_VR4 requirement feature added as a condition to the mapping model and "Mesh Networking" child capability attached to the condition. This ensures that the system modeling elements of the Relay Drone model, excluding those associated with the "Mesh Networking" child capability, are present in the resultant model (Figure 27). This meticulous mapping approach ensures coherence and fidelity in model realization, safeguarding against discrepancies resulting from feature selections.

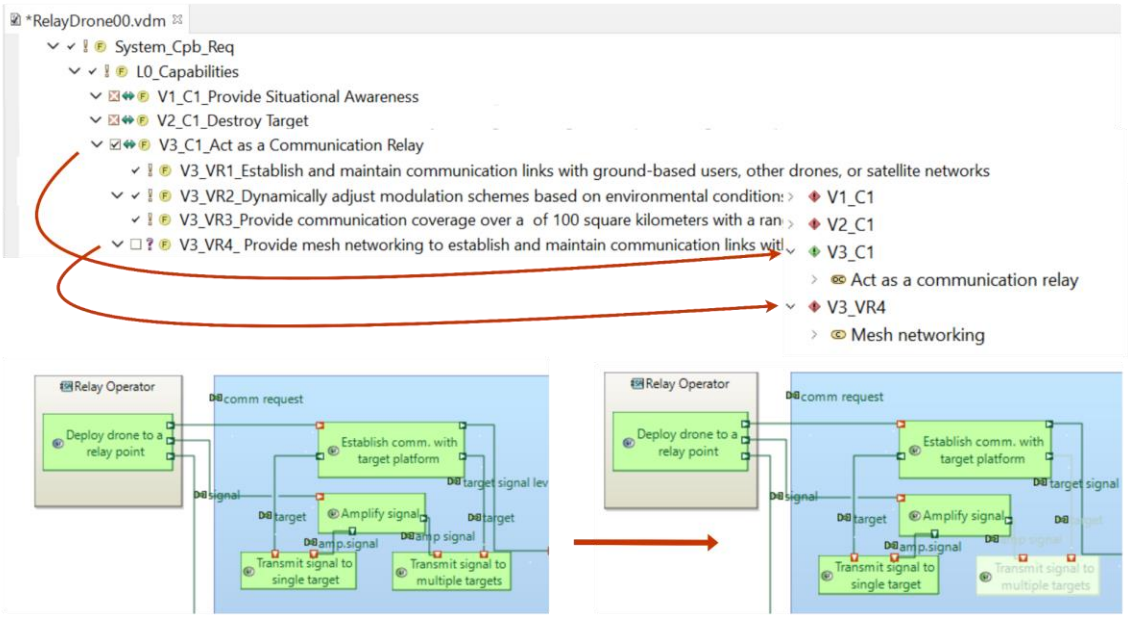

Figure 27 : Removal of the SM element in 100 % resultant model of Relay Drone_00

In this level, system functions carry representation of the variation points in the NFM. The mapping established previously influences system functions through the realization of operational activities. If a system function is realized by a variability-related operational activity, propagation rules map capabilities involved functions to the capabilities condition, ensuring that the resulting 100% model only includes functions realized by the capability related to that variant.

Figure 28: System functions propagation rules

Based on the rules in the mapping model, Figure 29 shows the resultant SAB model of Surveillance Drone and the removal of the functions related to other variants capabilities

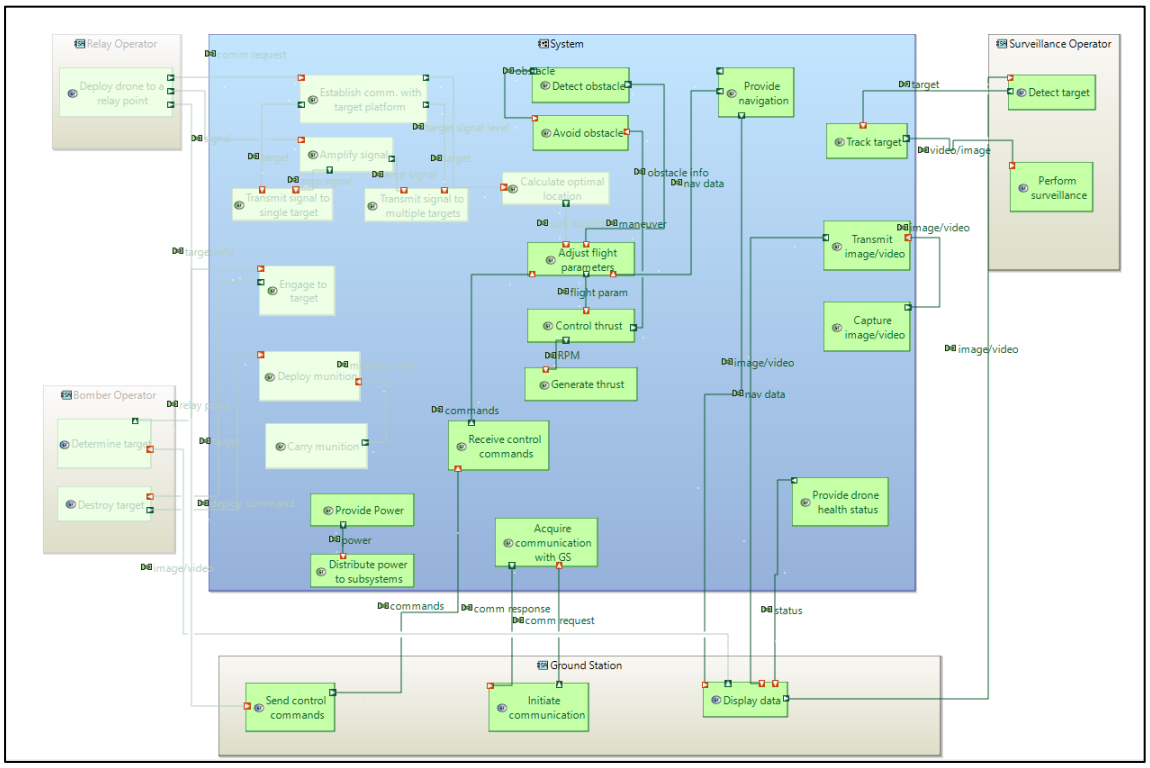

Figure 29 : 100% SAB diagram of Surveillance Drone (Grey-out mode)

Scenarios and mode state machine diagrams are frequently utilized to enhance the model's comprehensiveness for system analysis. As observed in the OA level, scenarios remain dependent on capabilities, thus preserving variations through capabilities and facilitating straightforward management of these diagrams. Mode state machine diagrams, on the other hand, depict the behavior of an actor or entity with distinct modes or states. These diagrams are crafted for each actor individually, acknowledging the variability inherent in actors. Consequently, transitions to the 100% model also impact actors realized by capabilities, ensuring that unrelated mode state machine diagrams and their elements are systematically removed from the resultant model.

4.3 Variability Management in Solution Space

In the solution space, management begins with the identification of subsystems through systems engineering practices and progresses with the joint development and mapping of the SM and SFM. Regarding the representation of the solution for the system, the previously defined functions become more structured and solidified in terms of variables. Consequently, the possibilities and feasibilities of the solutions become apparent. The simultaneous development of these two models allows them to complement each other, resulting in a comprehensive solution architecture encompassing all variables.

Figure 30 overviews activities and interactions within the developed methodology for managing variability in the solution space, showing how subsystems and variabilities are identified and managed through SM and FM.

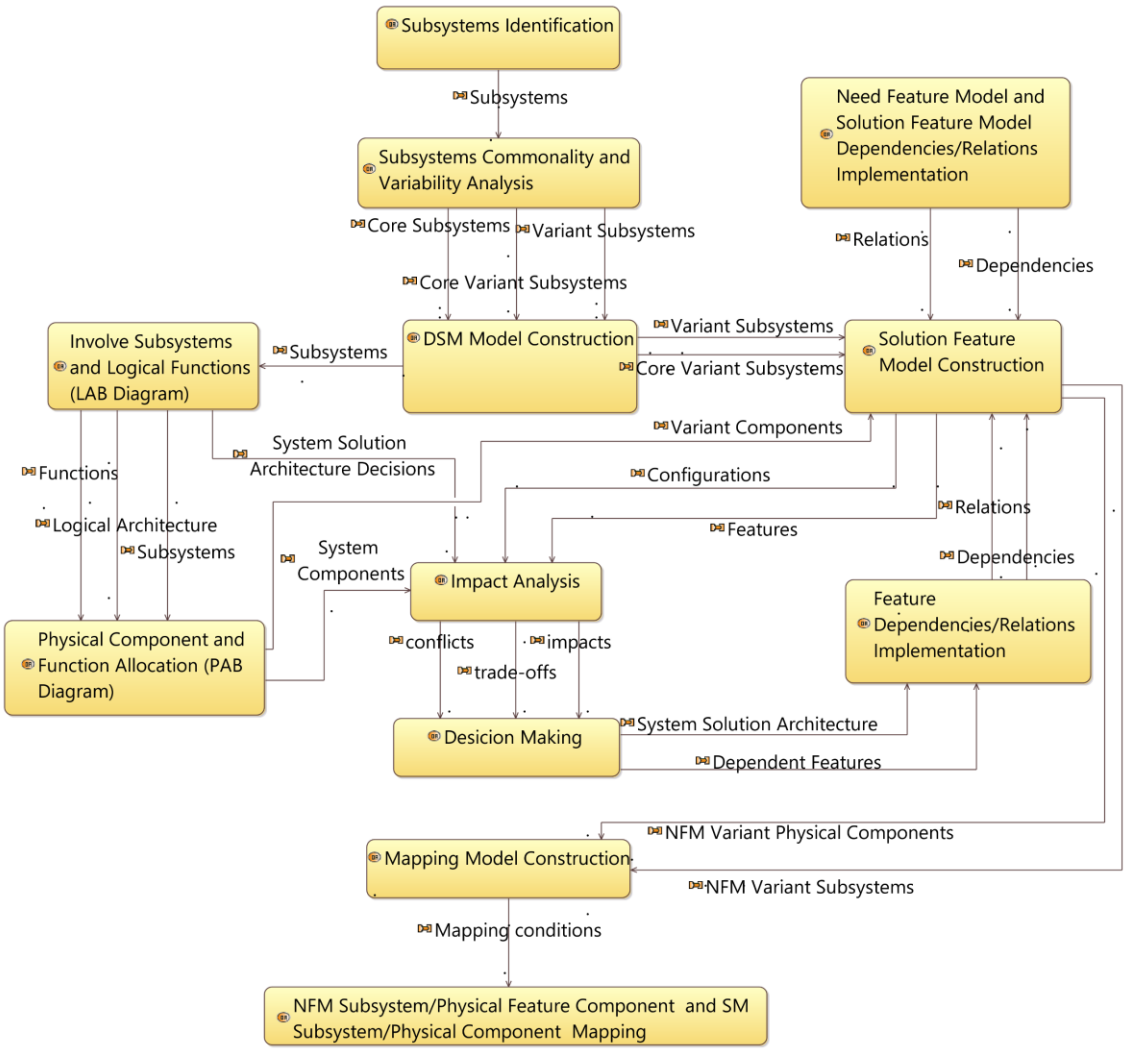

Figure 30: Overview of the activities for variability management in solution space

4.3.1 Subsystem Identification Analysis

New variation points for solutions may emerge at this stage, alongside existing variations, and variation points. Managing these variations can be inherently complex, particularly in projects with operational variants, necessitating meticulous attention to ensure the accuracy of the resultant model. Interface management techniques play a vital role in managing this complexity, beginning with the identification of interfaces. With DSM modeling technique, the interfaces between subsystems identified, facilitating effective interface management throughout the solution definition phase. This analysis structures the second feature model, Solution Feature Model (SFM). Through the DSM, subsystems

affected by variations become evident through their interactions. Consequently, it is imperative that the affected subsystems are included in the solution feature model to facilitate the modeling of variant-specific subsystem-to-subsystem restrictions.

The initial step involves decomposing the system of interest into subsystems that realize logical functions. In the subsequent step, interactions between subsystems are identified, and a DSM matrix is constructed using interaction types.

Analyzing the logical functions and solutions, using the method for interface management that previously stated, subsystems classified as;

Core Subsystems: ATA 23, ATA 34, ATA 72

Variant Subsystems: ATA 43, ATA 94

Core Variant Subsystems: ATA 22, ATA 24, ATA 93

The interactions between Core Subsystems (CS), Core Variant Subsystems (CVS), and Variant Subsystems (VS) offer valuable insights and serve as a guiding framework for the SM to shape functional exchanges. However, the DSM primarily serves as the cornerstone for the SFM development. In this study, the possibilities for subsystem variability, classifying them as CS, CVS, and VS, were thoroughly examined and strategies for their management were explored.

To enhance clarity in expressing the methodology, assumptions are also made for interactions of the subsystems. Specifically, interactions of VS and CVS are primarily with the ATA 22 Subsystem. It is assumed that variability within these subsystems does not significantly impact the solution for ATA 22, except for its interaction with the ATA 43 VS. As evident, as solutions are formulated, new variation points emerge at the solution architecture space, encouraging the creation of a DSM to capture these changes effectively.

Table 5, shows the data and electrical interfaces between the CSs (in white), CVSs (in blue) and VSs(in orange). The columns indicate the subsystems that provide inputs, while the rows indicate the subsystems that receive those inputs. For instance, in the electrical interface between ATA 24 and ATA 22, ATA 24 supplies electrical power to the ATA 22 subsystem.

Table 5 : DSM of Drone System

	ATA 22	ATA 23	ATA 24	ATA 34	ATA 43	ATA 72	ATA 93	ATA 94
ATA 22		D	E	D				
ATA 23	D			D			D	
ATA 24	D							
ATA 34	E							
ATA 43	E, D		E					
ATA 72	D		E					
ATA 93			E					
ATA 94	E D							
E: Electrical Interface D: Data Interface								

4.3.2 Feature Modeling in Solution Space

In the SFM, like the NFM, core subsystems are omitted from the model as they lack significance in terms of variation. VSs and CVSs are represented as features within the system. These two types of subsystems are managed in the SFM by delineating the feature types of the subsystems. Essentially, if a variation point projects a system element for all variants, the CVS becomes a mandatory feature. However, if the variant system element is unique and lacks representation elsewhere in the system, the CVS is designated as an optional feature. Meanwhile, VSs are consistently treated as optional features owing to their reliance on variation choices.

In essence, the primary distinction lies in the nature of the functions. In the former case, the "Provide Power" and "Capture Image/Video" system functions are initially considered valid for all variants. However, at the solution level, these functions are split into two logical functions—one for the resulting function of the variability and another for the remaining variants. For instance, "Capture Image/Video" system function at SA level is detailed enough for representing the expectation from the system but in LA level, to represent the systems solution, it decomposed into “Capture HD Daytime/IR Image /Video” and “Capture Daytime Image /Video”. Surveillance Drone requirement V1_VR1/VR2/VR3/VR4 calls for a more advanced features for imaging, this variation point represented in the LA with “Capture HD Daytime/IR Image /Video” function. Th rest of the variants do not require a advanced imaging so “Capture Daytime Image /Video” function used for representing their imaging functionality. Since this imaging function have representation for all variants, ATA 93 CVS becomes a mandatory feature. The same is valid for the “Provide Power” function for ATA 24 CVS. Consequently, the decomposed system functions project a function for all variants at the solution space system model.

In contrast, the function representing variability within the ATA 22 subsystem begins at the SA level, wherein it reflects the system's variability. During the SA level, the "Calculate Optimal Location" function was introduced to address the "Adaptive Modulation" system capability. In the solution space, a design decision was made to implement software for calculating the optimal location within the ATA 22 Auto Flight Subsystem. Notably, this function is specific to the Relay Drone variant and is not relevant to other variants. These variation point designate the ATA 22 Subsystem as an optional CVS.

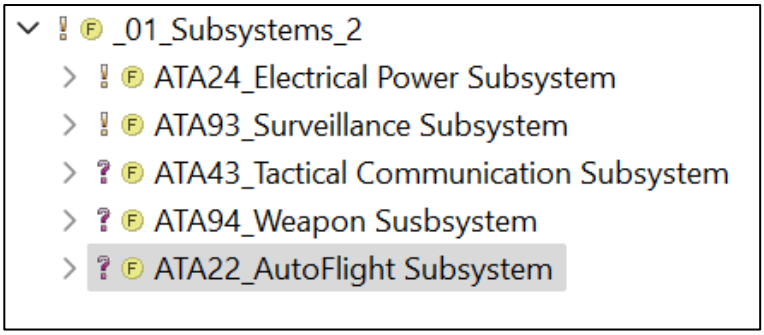

Figure 31 : Variant and Core Variant Subsystem features of Drone System

In the SFM, the allocation of physical component features is structured such that they are considered child features of their respective subsystem features. This ensures a hierarchical organization wherein essential components, such as a battery and E/O camera, are encompassed within mandatory parent features, despite the components themselves being presented as alternative selections.

Within the context of ATA 43 VS, the decision to incorporate it hinges upon the desired capability of relaying, with the Relay Drone variant introducing its own set of variation points. Specifically, the inclusion of the "Mesh Relay" capability necessitates the addition of the "Tactical Communication Module" physical component feature to the ATA 43 VS. Notably, the presence of this feature becomes contingent upon user preference, so feature implemented as an optional feature. Consequently, even in cases where the mesh relay capability is not selected, the presence of the ATA 43 VS remains within the system architecture. Conversely, ATA 94, while also classified as a VS, lacks any variation points within its variant, thereby resulting in the absence of child features. In the case of ATA 22 CVS, within the Relay Drone variant, variation arises from the user's dynamic choice regarding the reliability of the dynamic modulation capability. This choice significantly impacts the analyzed signal type, thereby establishing variation points as physical link alternatives between the ATA 43 and ATA 22 Subsystems. With the identficaiton of subsystems and components along with the variation points, SFM is created (Figure 32).

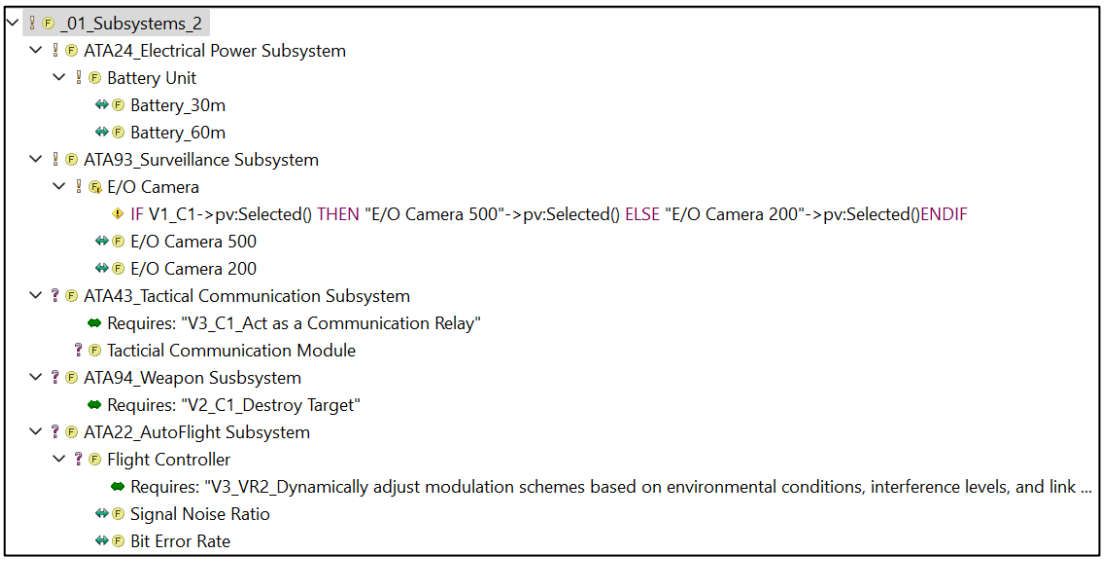

Figure 32 : Solution Feature Model of Drone System

Also, for the coherence and automatic creation of the feature model, the dependencies are formed between and within the NFM and SFM.

In the NFM, managing dependencies, such as the selection of the E/O Camera 500 being contingent upon the selection of V1_C1, while the E/O Camera 200 remains valid for other variants, poses challenges in terms of variant management. To address this issue, a pvSCL condition is created to handle E/O Camera selection. In the absence of V1_C1, the E/O camera feature is automatically selected for all variants, streamlining the selection process and ensuring consistency.

IF V1_C1->pv:Selected() THEN EO_500->pv:Selected() ELSE EO_200->pv:Selected() ENDIF

Moreover, the creation of component features in the SFM allows for dependencies to be established between the NFM and SFM, facilitating the automatic formation of the model. Selections of battery features are linked with corresponding NFRs, ensuring that either one is selected based on system requirements.

For VSs, dependency relations are formed with their associated capabilities. This ensures that when a related capability is selected, the dependent VS is also automatically selected. For instance, in the ATA 43 VS, the addition of a telemetry mesh at the component level for the V3_VR4 capability is presented to the customer as a choice, and dynamic formation for the relay drone is subsequently conducted.

Furthermore, another dynamic formation is managed through the selection of NFR6 feature in the NFM. A “requires” dependency ensures that the selection of the requirement is only possible if the V3_VR2 feature is selected. Consequently, the selection of V3_VR2 necessitates the selection of a microcontroller in the SFM, which in turn forces the

selection of interfaces. This complex interplay is managed through pvSCL conditions, ensuring that the model accurately reflects system requirements. If NFR6 is selected, the Signal Noise Ratio feature must be selected; otherwise, the Bit Error Rate feature is automatically chosen in the feature model.

IF NFR6->pv:Selected() THEN SNR->pv:Selected() ELSE BER->pv:Selected()ENDIF

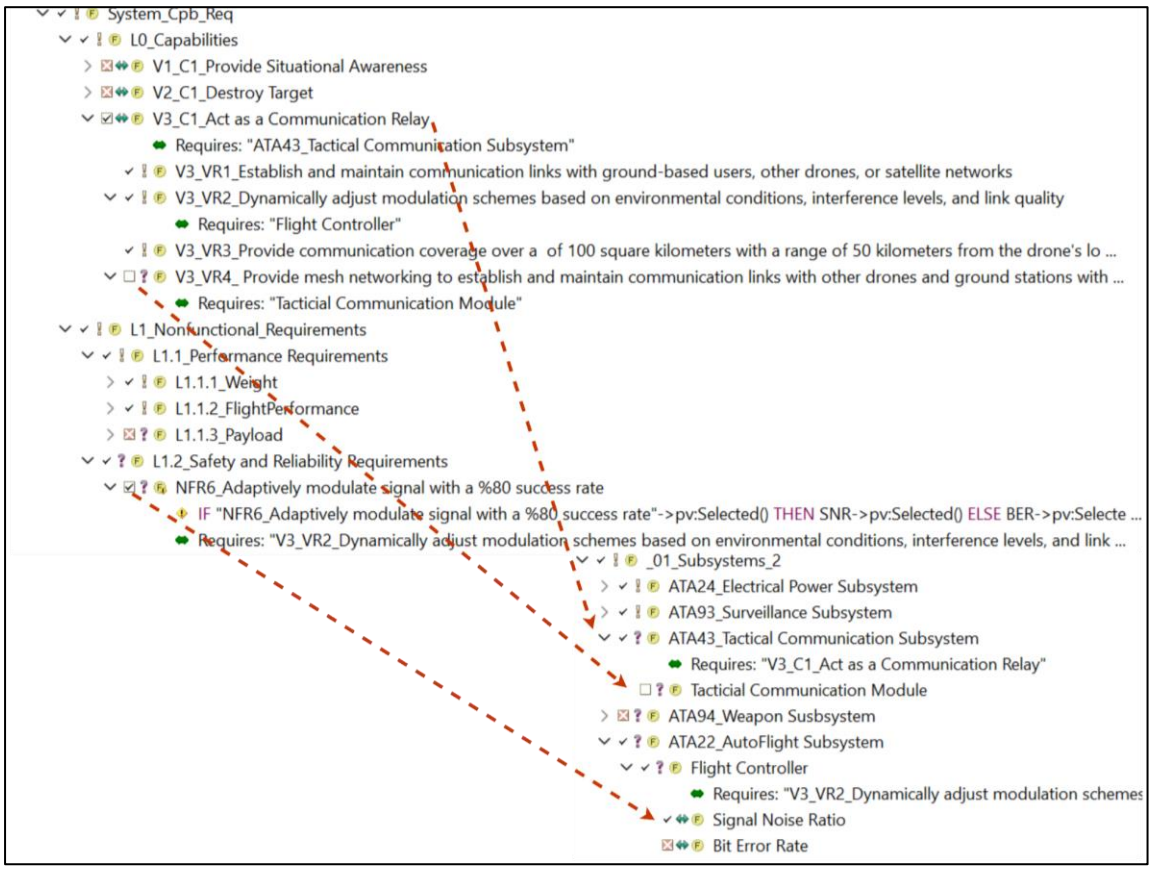

Figure 33 : Automatic configuration of Relay Drone with NFR6

4.3.3 System Modeling in Solution Architecture Space

This section offers an overview of the solution architecture in system models. The solution specific variabilities and decisions are implemented into the various levels of Capella, taking into consideration the representations unique to each level. Although the NFM constructed in the previous section, while developing the NFM, the functionalities and system model elements taken into the consideration, which highlights the importance of the simultaneous development of FM and SM for accurate system architecture. The NFM developed up to this point is particularly relevant to the Logical and Physical Architecture Levels in Capella, as it effectively represents the system's solution.

Logical Architecture

At the Logical Architecture (LA) level, the functions and capabilities developed at the SA level undergo transformation into logical functions and capabilities, while preserving all exchanges and relationships among them. Also at this level, certain decision analyses and assumptions were made to shape both models in a manner that enhances potential variability. While alterations in service time requirements for Bomber Drone during the OA level may not directly impact SA functions, they can significantly influence the system function of "Provide Power" at the Logical Architecture level, given its representation of the solution-level architecture. Consequently, it becomes imperative to decompose the functions "Capture Image/Video" and "Provide Power" function, and to realize them with their corresponding logical capabilities.

This underscores the necessity of accommodating functional variations even at the LA level through capability relations. Unlike functional variations example at the SA level, there is no need to add these child capabilities to the mapping model at the LA level. This is because these variations do not inherently create variation points within a product family; rather, their functional existence in the model is managed through parent capability propagations. This approach ensures that the LA model remains flexible and adaptable to varying requirements and scenarios, while maintaining coherence and consistency across different levels of abstraction.

Logical Architecture Blank (LAB) diagrams serve as vital constructs for delineating the logical components within a system, along with their assigned functions and exchanges. This model embodies complexity as it provides a comprehensive representation, 150% model, by modeling solution variations and allocating function variations to subsystem components.

By meticulously detailing the logical components and their associated functions and exchanges, LAB diagrams offer a clear and structured depiction of the system architecture. Moreover, they facilitate the identification of solution variations, enabling stakeholders to grasp the nuances of different system configurations. Furthermore, LAB diagrams play a crucial role in delineating the responsibilities within the organization, particularly concerning the allocation of function variations to subsystem components. This ensures clarity and accountability, enabling effective decision-making and system management throughout the development lifecycle. In Figure 34, %150 LAB diagram of Drone System that contains all variability points in solution with functions is represented.

A similar mapping methodology is employed between the SFM and SM at the LA level. Mapping operational capabilities directly influences logical functions, fostering coherence within the 100% model concerning actors and functions. However, this propagation does not extend to logical components, i.e., subsystems.

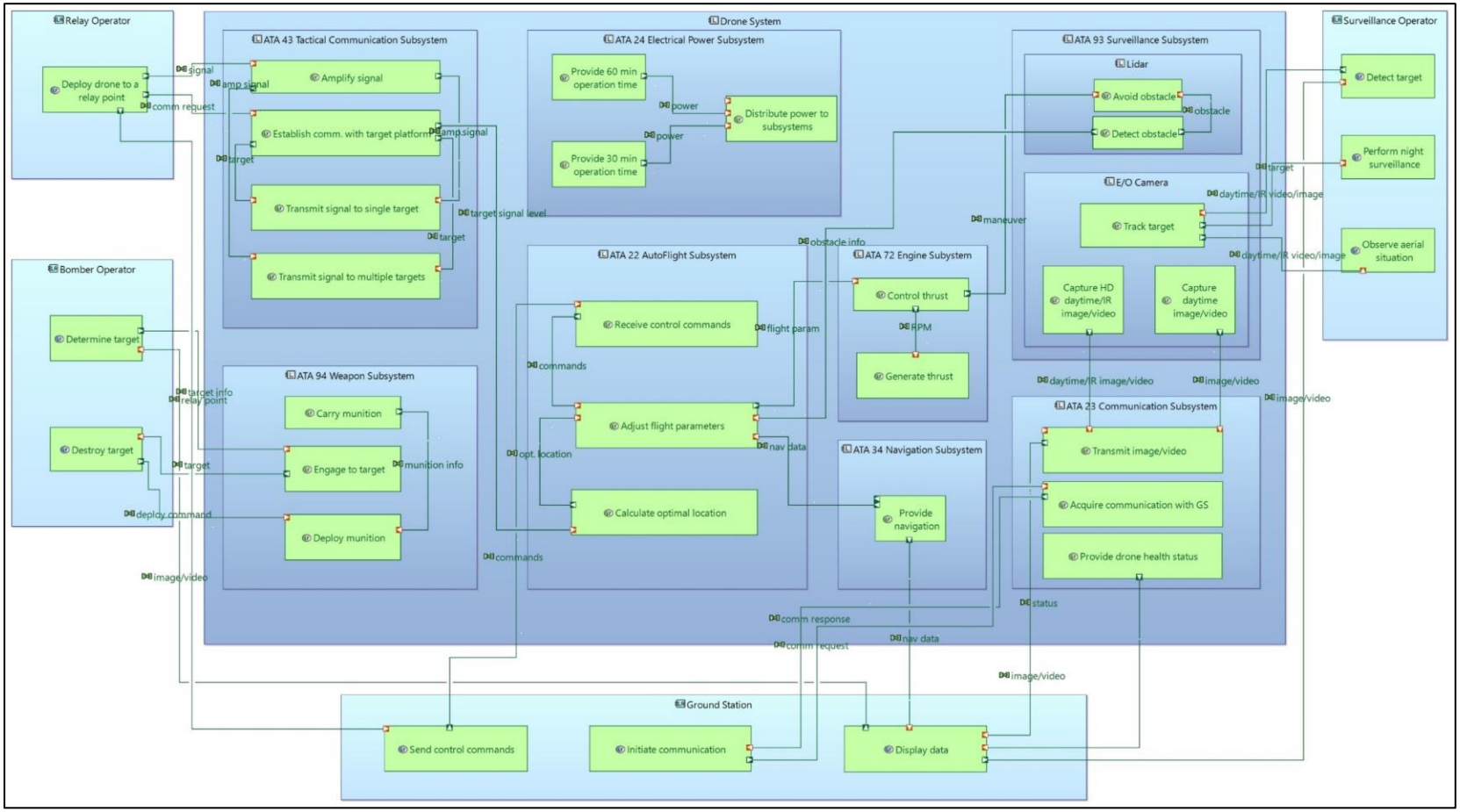

Figure 34 : Logical Architecture Blank diagram of 150% Drone system

Variation in the LA level encompasses both logical functions and components. For CVS's logical functions represent the variation points but for VS's both logical functions and components are the representation of the variation points. To uphold model coherence, VS components are mapped to SFM features. Subsystem features within the SFM are incorporated as conditions, with subsystem logical components linked to these conditions. Consequently, during LAB diagram generation, VSs are either retained or omitted in the resultant model based on variant selections. This method ensures consistency and alignment between the SFM and SM, facilitating a comprehensive depiction of system variations and their effects on subsystems.

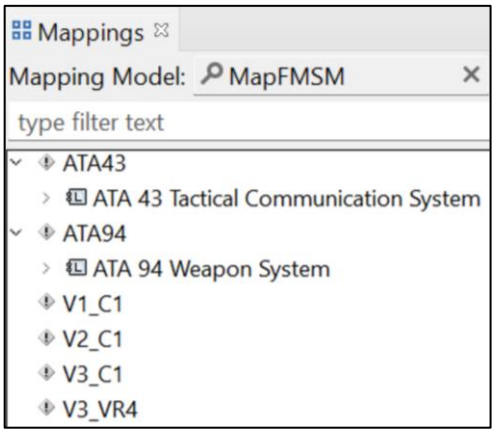

Figure 35 : Updated Mapping Model with Variant Subsystems

As an example, in Figure 36, Surveillance Drone's mapping model according to the conditions can be seen. Since ATA 43 and ATA 94 subsystems are not valid in the NFM, conditions become false, resulting in the removal of the attached SM elements in the diagram (Figure 37).

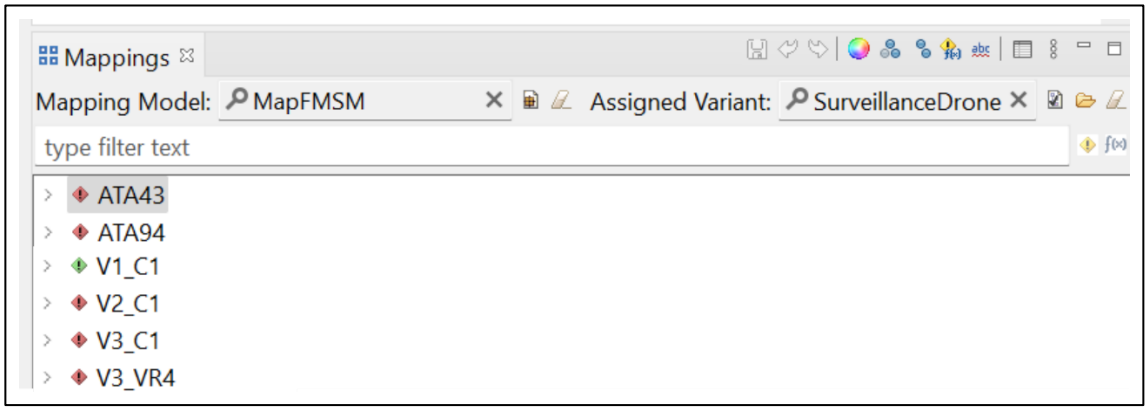

Figure 36: Initial mapping model of Surveillance Drone

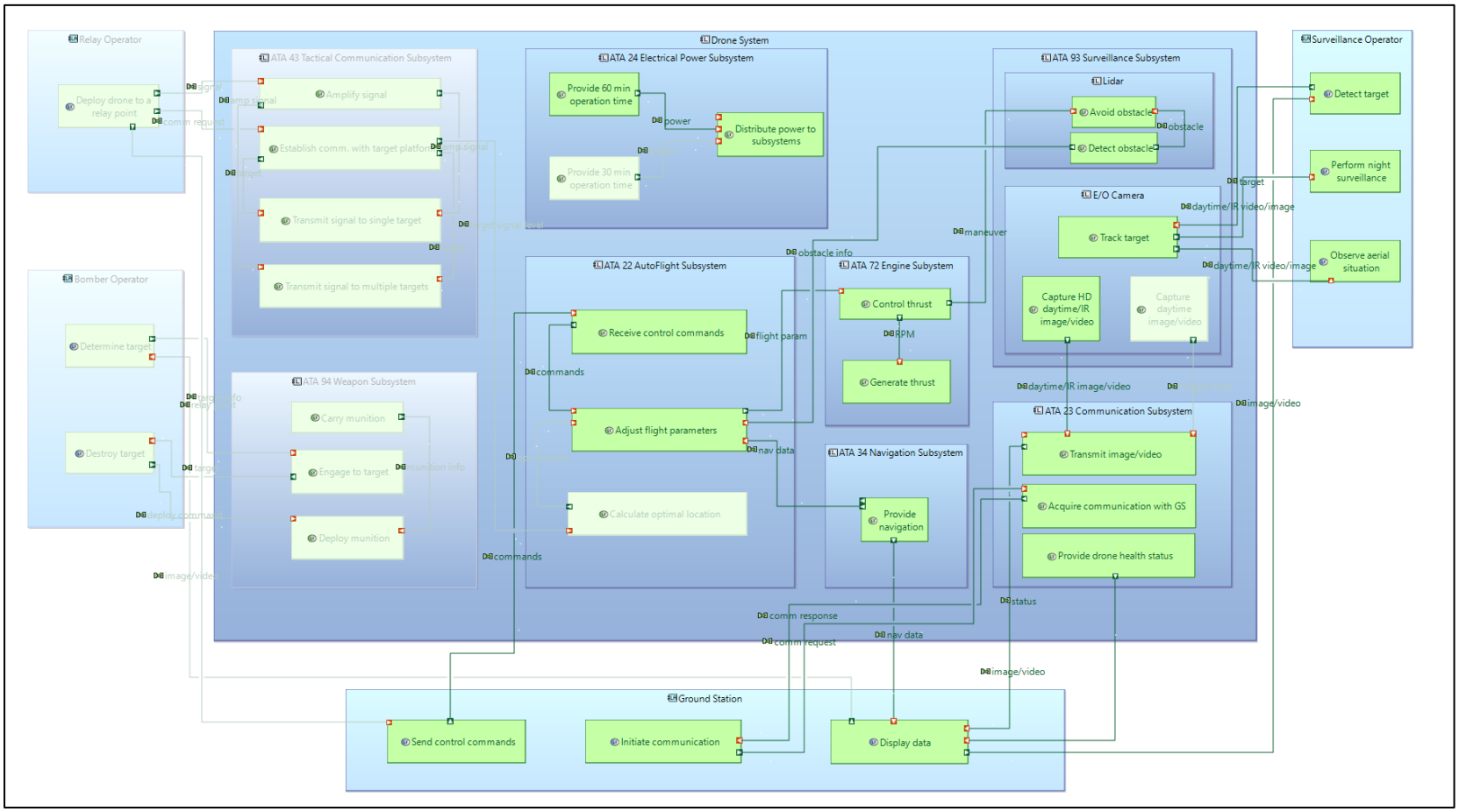

Figure 37 : 100% LAB diagram of Surveillance Drone (Grey-out mode)

Physical Architecture

At the Physical Architecture (PA) level, similar to previous stages, automatic transitions are applied to functions and actors. However, in PA, the system solution becomes significantly more intricate and detailed compared to the LA. This stage encompasses detailed aspects such as equipment and technological choices, all meticulously modeled to refine the system's design.

While Capella facilitates the automatic transformation of logical components into behaviour physical components, this approach may not efficiently capture the desired level of detail in the system. Therefore, automatic transition is not applied to logical components in this context. Within the Physical Architecture Blank (PAB) diagram, Subsystems and their components are imported as physical components, while equipment-level details are imported as behavior components. Physical links between these components are established, and the detailed functions allocated to the relevant behavior components are assigned (Figure 38).

Propagation rules for functions and actors remain valid at the PA level. Capability-related functions and actors are either retained or omitted based on the capability mapping conducted between the NFM and operational capabilities. But in PA, the variations are also represented by the physical components that functions are allocated. Similar to logical components, the propagation rules do not apply to the realized physical and behavior components of the capabilities or functions at the PA level. Consequently, to ensure model accuracy, it is imperative to map the physical and behavior components to the related subsystems or requirements in the mapping model. This mapping facilitates the integration of physical and behavior components into the broader system architecture, ensuring coherence and alignment between the various modeling elements. The VS physical components mapped to the preexisting condition of the subsystems. The physical components that represent the hardware and software components mapped directly to the related child component feature created under the subsystem features in the NFM (Figure 39).

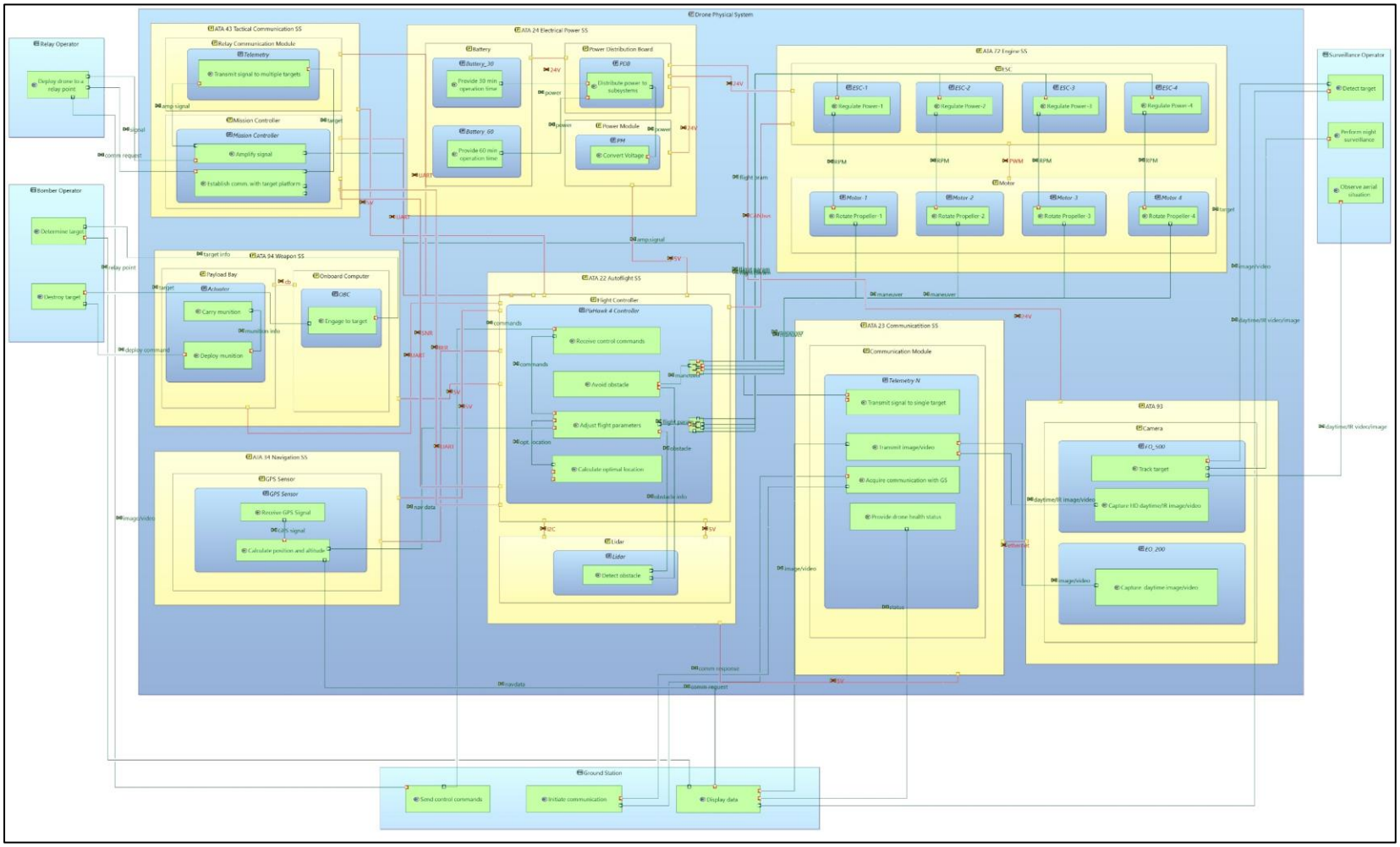

Figure 38 : Physical Architecture Blank diagram of 150% Drone system

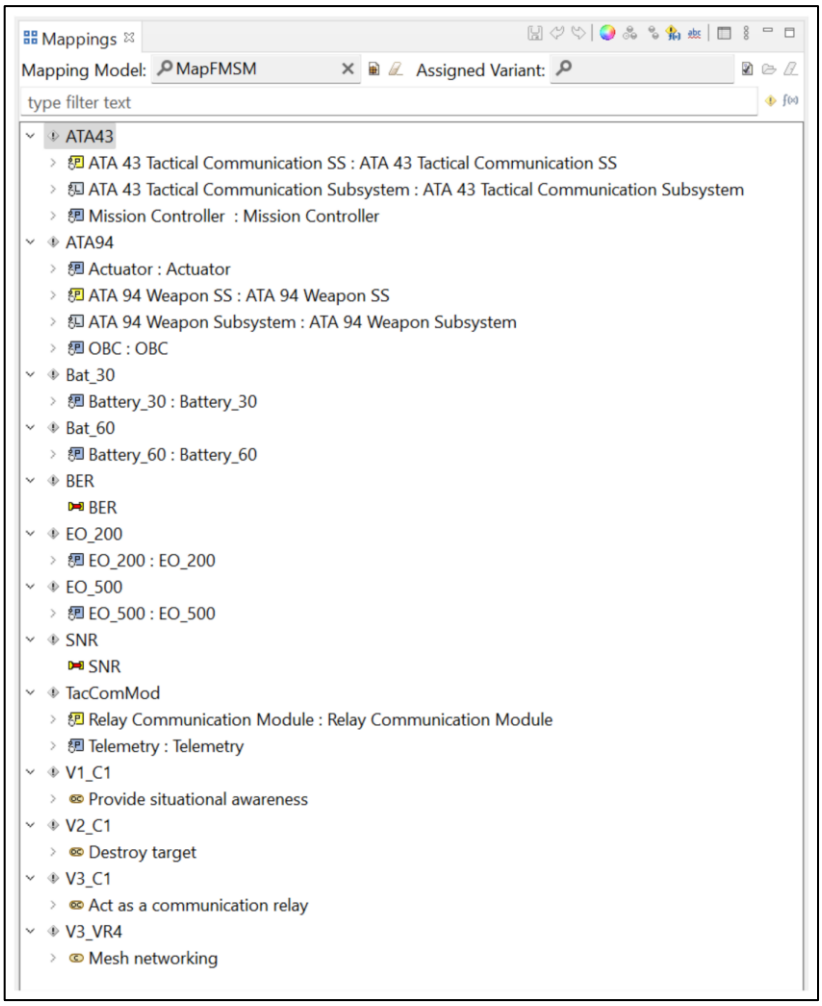

Figure 39 : Updated Mapping Model with Variant Physical Components

In Figure 40, Surveillance Drone's mapping model based on the configuration in the SFM (Figure 41) is shown. The SM elements under the false conditions removed from the resultant model, including the physical interfaces and components (Figure 42).

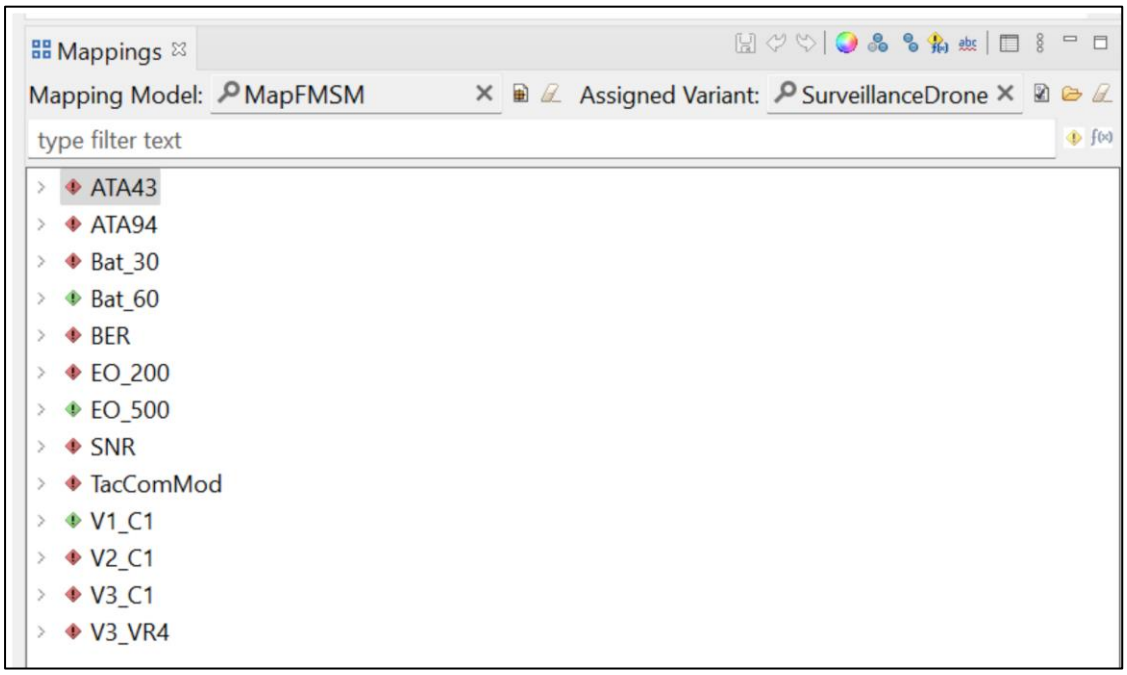

Figure 40: Final mapping model of Surveillance Drone

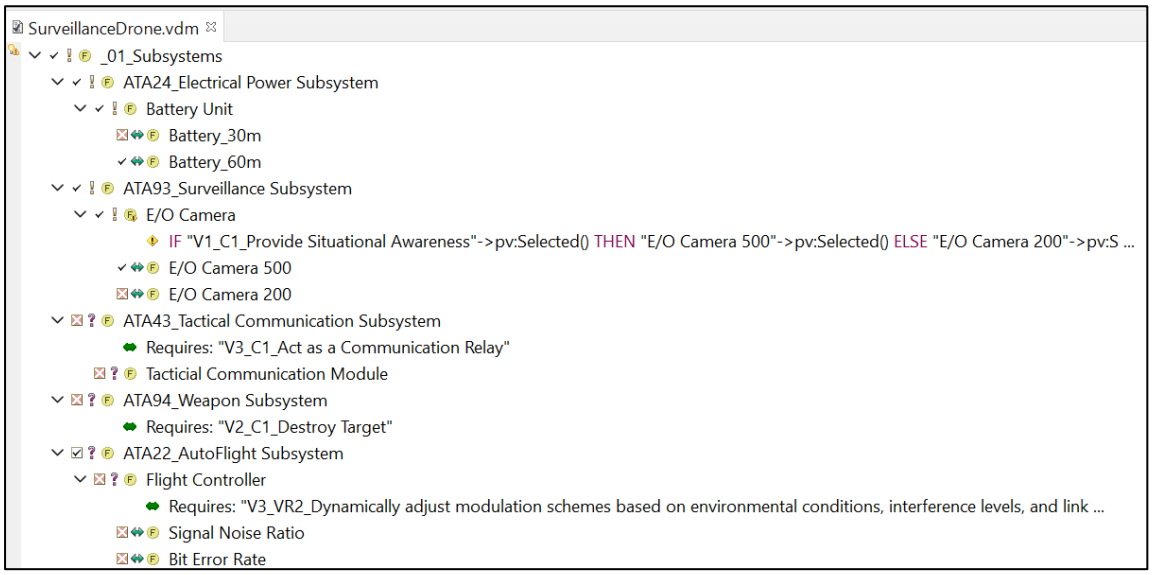

Figure 41: Surveillance Drone SFM Configuration

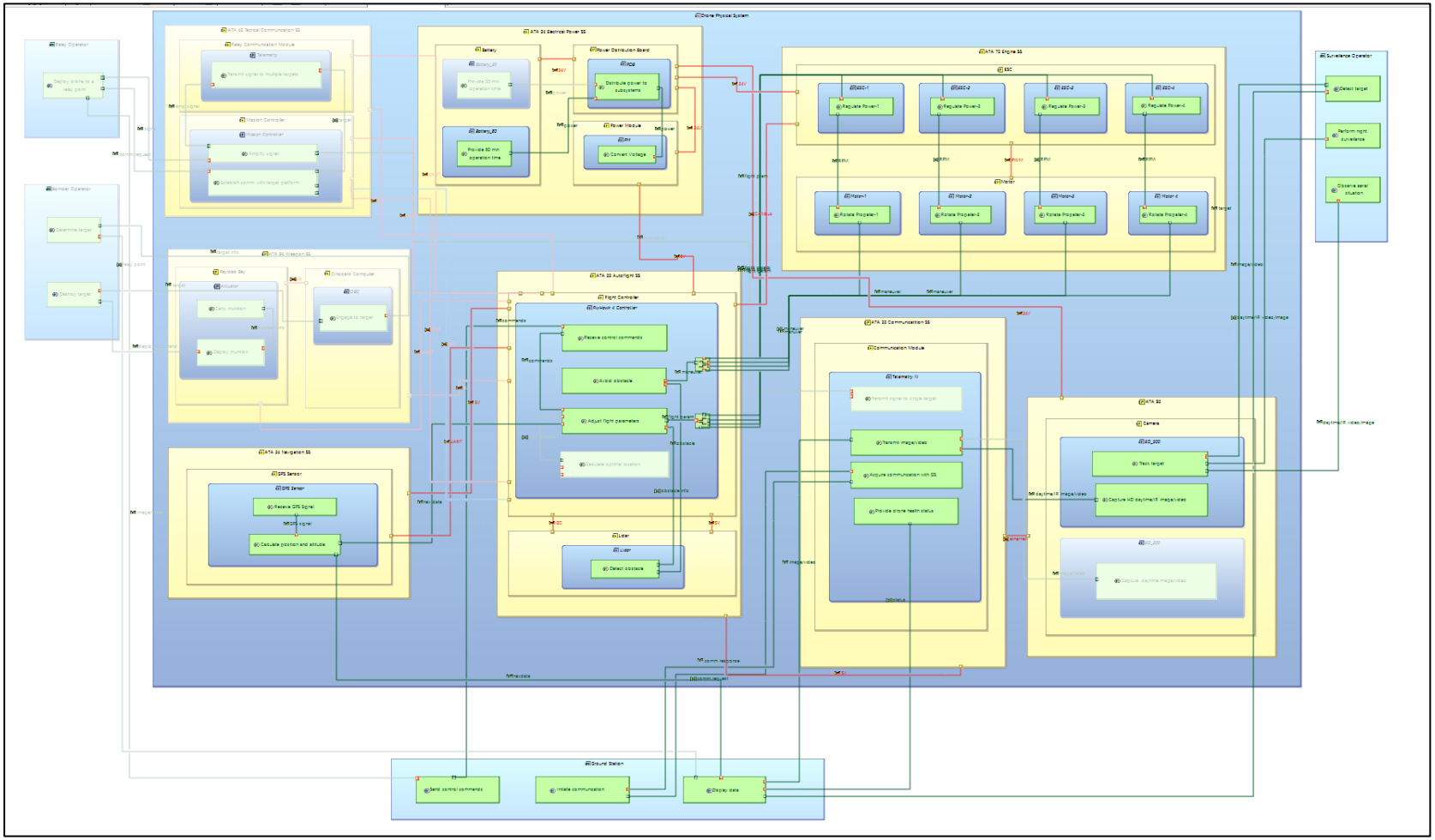

Figure 42 : 100% PAB diagram of Surveillance Drone (Grey-out mode)

CHAPTER 5

VALIDATION OF THE CAPLA METHODOLOGY

5.1 Validation of the Requirements

In this chapter, a comprehensive evaluation of the CAPLA methodology for management of variability in the system architecture is undertaken. The objective of this evaluation is to verify the methodology against the predetermined requirements established in Chapter 3. By systematically assessing how the methodology meets these requirements, it is aimed to demonstrate its effectiveness, reliability, and overall utility in managing complex product lines.

The evaluation process is structured to ensure a thorough analysis, utilizing both qualitative and quantitative approaches. To start with, the specific requirements are outlined that the methodology was designed to address, which encompass critical aspects such as system understanding, design efficiency, change management, collaboration, scalability, reliability, precision, innovation, decision-making, integration, and configuration optimization.

Subsequently, the core components of the methodology are explored in depth, with a focus on the approaches and solutions it offers. Each capability is rigorously assessed to evaluate its effectiveness in meeting the relevant requirements. This assessment includes a comprehensive analysis of the tools, techniques, and processes incorporated into the methodology, emphasizing their strengths as well as any potential limitations.

With the CAPLA methodology presented in this paper, the following improvements are aimed to achieve:

Requirement 1:	*The proposed methodology shall support the application of Model-Based Systems Engineering approaches for designing system architecture, facilitating a model-driven development process.*

In contemporary engineering and other advanced disciplines, the use of abstract descriptions of systems before constructing a complex system is essential. The MBSE approach has emerged as a solution to visualize and structure these descriptions effectively. To leverage the advantages of MBSE, as discussed in previous chapters, the ARCADIA method and the Capella tool are employed for designing system architecture.

Supporting the idea of representing variability at every level of abstraction, ARCADIA accommodates different engineering stages and provides perspectives in both the problem and solution spaces, offering a link between all modeling views. A drone system is modeled in Capella to illustrate the system architecture. In Capella, system architecture

modeling begins with the solution space levels, including operational and system analysis levels. Capabilities and system actors are identified, and related activities and preliminary system functions are constructed, along with the interactions between them. Utilizing Capella's traceability features, capabilities are realized by functions, and a coherent problem space architecture is constructed.

Transitioning from the problem space to solution space, the logical and physical architecture levels modeled. The information from the problem space is preserved by Capella's automated transformations of functions, interactions, and actors. Logical functions derived from the system functions and newly introduced logical functions are integrated into the system. Subsystems are created as logical components, and the logical functions characterizing the subsystems are allocated accordingly. As the level of detail increases, physical functions are derived from logical functions, and new ones are introduced. Physical components that realize the subsystems and functions are allocated to the physical components, and physical links are formed.

The system architecture is constructed by considering all levels of abstraction throughout the development stages, starting from requirements to functions, subsystems, and software/hardware components.

Requirement 2: *The proposed methodology shall support feature-oriented design and analysis by creating, visualizing, and managing system variant features, including modeling interdependencies between features.*

Feature-oriented design fundamentals are integral for managing system variants. This process begins with the identification of variants through SCV analysis (Coplien et al., 1998). Requirements and capabilities are identified from initial requirements and categorized as either "Core" or "Variant." The variant requirements and capabilities are defined as features, and a hierarchical feature tree is constructed. Types are assigned to features—alternative, optional, or, and mandatory. Additionally, constraints and dependencies such as "requires" and "excludes" are defined to model the interdependencies between features.

In CAPLA, as detailed in Section 3.2, the solution and problem spaces are identified in line with PLE applications and system architecture. This distinction leads to the creation of two feature models. The Need Feature Model, representing the problem space, identifies high-level variant requirements through SCV analysis and represents them as capabilities in the feature model. Each capability corresponds to a variant, such as V1_C1—Provide Situational Awareness capability for a Surveillance Drone. Capabilities are represented as alternative features to restrict the selection of multiple capabilities belonging to different variants. The same management techniques, including feature types, dependencies, and constraints, are applied throughout the feature modeling process.

The requirements that realize specific capabilities are defined as child features of those capabilities.

The Solution Space Feature Model (FM), aligning with the System Model, represents the logical and physical components. A similar analysis is conducted to identify subsystems that help in identifying variant requirements and capabilities. Three types of subsystems are identified: Variant Subsystems, Core Variant Subsystems, and Core Subsystems. Variant and Core Variant Subsystems are incorporated into the Solution Space FM as features, and the physical components containing variants are inserted as child features. These two FMs are linked through interrelations, connecting the problems represented by requirements to their solutions represented by subsystems and components. Variant configurations are formed by selecting features from both feature models based on their relationships. Advanced relations enable the automatic configuration of feature models, an important aspect of this thesis.

Constructing two FMs representing different development spaces offers an advantage for managing overall variants and ensures alignment with the System Model. Features representing system variability correspond to system model elements.

For demonstrating these concepts, the Pure::variants tool is used for modeling feature models. It provides a clear visualization and fundamental feature modeling capabilities for forming feature trees and relations effectively. Pure::variants supports advanced relation construction, facilitating automatic and correct variant configurations.

Requirement 3:	*The proposed methodology shall include automated mechanisms to derive valid configurations based on predefined configurations and rules for each product variant, reducing manual efforts and errors.*

As mentioned above, feature relations are crucial for managing variant configurations. To enhance this capability, a structured approach has been developed to ensure valid and automated variant configurations for predefined configurations.

Starting with capabilities, if all requirements are essential for a capability, the child requirements are inserted as mandatory features. This guarantees that when the capability is selected, all associated requirements are automatically selected. If a capability includes a variation point, such as a Mesh networking requirement, it will be classified as an optional requirement, necessitating manual selection. Additionally, by incorporating non-functional requirements into the system, conflicting requirements can be managed with restrictions. For example, endurance and the carrying capacity of Bomber Drones conflict with each other. This information is crucial for system design. In complex and large-scale systems, such conflicts can be overlooked, so visualizing and presenting these conflicts is important for valid configurations.

Automatic configuration becomes crucial in these situations. Using the Pure::variants pvSCL language, advanced relations are formed to automate the selection process. For instance, when the capability driving the Bomber Drone is selected, the system is forced to select the correct endurance requirement. Besides automatically selecting the correct non-functional and functional requirements, selecting a capability also affects the selection of subsystems and components, either directly or through its requirements.

A direct effect can be seen in the selection of Variant Subsystems (VSs). If a capability requires a subsystem that other variants do not, the "requires" relationship between the subsystem and capability enables its automatic selection. An example of an indirect effect is the Bomber Drone capability driving the corresponding endurance requirement. This forms a "requires" relation with the non-functional requirement and battery component, ensuring the automatic selection of the component. Furthermore, dependencies and constraints, such as "requires" and "excludes," model the interdependencies between features and enforce correct configurations

To further streamline the configuration process, advanced feature selection mechanisms are implemented. These mechanisms analyze the impacts of optional features in advance and automatically adjust related features based on predefined rules. For example, if a feature related to a specific subsystem is selected, the system automatically includes all necessary components, reducing the need for manual intervention. This approach not only saves time but also minimizes the risk of errors caused by overlooked dependencies.

In this thesis, automated variant configurations are seen as crucial for preventing mistakes resulting from the system's scale and complexity. Constructing the variant configuration process to minimize manual labor ensures consistency and accuracy. Even in optional feature selection, the other impacted features must be analyzed in advance, and relations must be formed to support automated configuration.

For demonstration, the Pure::variants tool is used for modeling feature models. It provides clear visualization and fundamental feature modeling capabilities, forming feature trees and relations effectively. Pure::variants supports advanced relation construction, facilitating automatic and correct variant configurations, and ultimately ensuring alignment with the overall system model.

Requirement 4: *The proposed methodology shall offer a platform ease of use with clear, readable representations of technical architecture and variant configurations accessible to architects and stakeholders at all levels.*

To provide clear and easily understandable system architecture diagrams and views for different stakeholders at every level of abstraction—such as high-level system overviews for executives and business analysts and technical architecture details for system

engineers—the Capella tool is selected for system architecture modeling. Capella is user-friendly and supports users with built-in guidance for developing system architecture rather than providing a blank canvas. This makes the tool accessible for both novice and experienced users. Given the diverse range of stakeholders with various engineering specialties involved in our development process, easy access to information for everyone is crucial for maintaining the system.

Pure::variants is a tool specifically designed for variability modeling and management. It provides clean visualization and advanced relationships for variant configurations. With a focus on usability, understandability, and flexibility, Pure::variants effectively meets these concerns.

A significant concern of this thesis is to provide structured variant management for system architecture. To demonstrate the proposed approaches and methods, the two modeling tools must work together seamlessly. Having integration with Capella, Pure::variants supports incorporating variability information into system models. Using the mapping feature of Pure::variants, multiple system models can be derived from the 150% models. These 150% models, which contain all variant-specific and common system model elements, transform into 100% models that include only the selected variants and common model elements. Pure::variants enables this transformation by removing elements that do not belong to the selected variant. The conditions for this transformation are derived from the feature model elements and system modeling elements mapped to these conditions. Pure::variants evaluates these conditions based on the established variant configurations and, according to their validity, removes or retains the modeling elements in the system architecture. Another advantage of this integration is the creation of result models. The tool automatically generates 100% Capella models for each variant, which are free from variability and can be worked on individually. The tool selection process is also identified in detail in Section 2.3.

Requirement 5: *The proposed methodology shall include tools and techniques to comprehensively manage system variability across different abstraction levels, ensuring that all aspects of variability are captured and represented.*

The primary concern for the development of the CAPLA derived from this requirement to manage variability across different development stages and abstraction levels. As previously mentioned, the same approach used for building the system model is applied to variability modeling. Feature models are divided into two spaces, mirroring the system model, and these models are developed concurrently. To capture all relevant aspects, modeling is divided into two spaces: Need Space and Solution Space.

Within the modeling framework ARCADIA, this distinction provides perspectives at different abstraction levels. In the Need Space, operational and system analysis

perspectives offer an overview of the system's objectives by identifying actors, their requirements represented as capabilities, and the system functions that realize these capabilities. Feature Models (FMs) support these actors and capabilities by identifying variants. In Need Feature Models (NFMs), variant-specific requirements and capabilities are identified as features. These two models support the definition of the product portfolio and customer options.

In Solution Space, the logical architecture level defines working principles by modeling subsystems, logical functions, and interactions. At a more detailed level, physical architecture models the physical components and links. The architectural details at these levels lead to a technically detailed FM. Architectural choices, dependencies, constraints, and reusable components are identified in the Solution Feature Model (SFM), ensuring that the system and feature models complement each other when linked.

This joint development of System Models and Feature Models facilitates easier verification of FMs as they remain coherent with the System Models, reducing the size and complexity of the FM. Since the architecture description is provided in the System Model, the FM can focus solely on variability-related elements. The System Model offers an overview of product outcomes with selected configurations, highlights dependencies not visible in the FM, and identifies unrealistic solutions, ensuring the feasibility of each configuration.

Requirement 6: *The proposed methodology shall incorporate end-to-end traceability mechanisms that form clear relationships between high-level requirements, technical solutions, and architectural elements across variants.*

When developing the methodology, traceability is considered at every step, resulting in multiple traceability links. In the System Model, traceability between perspectives is established through automatic transitions of modeling elements. Capability-based modeling distinguishes variants and ensures that every modeling element can be traced back to its variant's capability. At all abstraction levels, every common and variant element in the System Model can be traced to a capability.

Traceability between the Need Feature Model (NFM) and the Solution Feature Model (SFM) is constructed through defined relations. The identification of SFM features relies on features in the NFM, ensuring justification and traceability. As identified in the automatic configuration process verification, the components realizing capabilities are linked through various relationships.

The traceability between Feature Models and System Models ensures that both models complement each other through joint development. By utilizing the mapping feature of Pure::variants, FM features and corresponding System Model elements are mapped to

each other. Capability elements are mapped to capability features, ensuring that variant-specific functions, actors, and interactions are realized by the capabilities at every abstraction level. This mapping includes subsystem elements, physical components, and links at the solution level, maintaining traceability throughout the solution space. Detailed traceability mechanisms are provided in Section 3.2.4.

This traceability ensures correct transitions from 150% to 100% models, preventing the faulty occurrence or elimination of modeling elements not directly mapped to a variant. By linking variant-specific modeling elements, the traceability framework guarantees coherence and accuracy in the models.

Additionally, constructing traceability links effectively involves developing a Design Structure Matrix (DSM). Variability realization within the system can be complex and not immediately apparent; for example, the occurrence of a system might necessitate changes in another system. A DSM matrix facilitates the analysis of subsystem interfaces and their interactions. If a variant system (VS) interfaces with another system requiring changes, this can be directly analyzed and addressed in the DSM matrix. This ensures that traceability and justification links are maintained at each level of variability modeling.

Requirement 7: *The proposed methodology shall be scalable to handle the complexity associated with large and diverse product lines, ensuring efficient management of multiple variants.*

In complex and large systems, managing variants and their elements within models presents significant challenges, beginning with the identification of these variants. Variant information is not always explicitly clear in the provided needs or requirements. Although SCV analysis can streamline this process, it does not address variants that emerge during development phases. Particularly in complex systems with numerous stakeholders, design information, constraints, and decisions can be overlooked or lost. Even in well-guided development environments with high levels of communication and management, emerging variants during development stages can still be missed.

To address this issue, we employ a widely used industry approach to capture variabilities and their effects within the system: the Design Structure Matrix (DSM). As identified in Section 3.2.1, DSM is a modelling method useful for interface analysis, and the constructed DSM matrix highlights the interfaces between subsystems, making the impact of variant subsystems on other systems clearly visible. Core variant subsystems, which are common across all variants but may require new interfaces or design changes for specific variants, are particularly important for correct design. These variation points need to be managed effectively, and DSM helps identify these points at the solution level.

Another approach for managing variations in large and complex systems is the mapping approach we use. Typically, mapping directly to variation point modeling elements within

the system requires considerable manual effort and can result in errors. Variation point elements can be functions, interfaces, or physical components, all interconnected through realization or interaction links in the System Model. But since it is known that variations occur starting with the identification of capabilities, addressing variation from this point reduces complexity.

By constructing traceability for modelling elements with capabilities, we can manage variability information more structurally. In CAPLA, the System Model capabilities are mapped to Need Feature Model (NFM) capabilities and requirements, which are inserted as conditions in the mapping model. Realization links between modelling elements and propagation rules ensure that all related elements are mapped to those conditions. This approach allows for managing the capabilities, realized functions, interactions, and actors within the system together during transitions from the 150% to 100% models. In the solution space, subsystems and physical components are similarly inserted as conditions and mapped to each other, reducing manual labour significantly in systems with numerous variation points. This mapping approach, supported by the joint development of Feature Models and System Models, reduces the size and complexity of the FM, making variation points clearer and simplifying the mapping process.

Additionally, improving the trade-off process can further reduce complexity. Candidate solutions can be easily modeled with their constraints and dependencies as variants. This approach allows all concerns from different perspectives to be clearly seen, enabling the identification of the best solution more easily.

Requirement 8: *The proposed methodology shall improve change management processes, providing clear traceability and impact analysis for evolving product features.*

All the aforementioned approaches support the change management process effectively. Traceability, DSM modeling, and the mapping methodology manage variability information across all levels. In a dynamic environment, changes in the product line can occur at various development stages. Any addition or subtraction of an element in the system can significantly affect the correctness of the models. By implementing a capability-based variant management approach, any change that the system faces must be realized by a common or variant capability. This ensures that, regardless of the change's scope, it will be realized by a capability, maintaining the accuracy of the resulting model.

The impact of the change must also be considered, and DSM modeling provides a solution for this. By analyzing the impact, affected subsystems can be easily detected. The constructed DSM matrix highlights interfaces between subsystems, making the effects of changes on variant subsystems and their interactions with other systems clearly visible. This comprehensive approach ensures that the system adapts correctly to changes, maintaining model integrity and correctness.

In summary, the integration of traceability, DSM modeling, and mapping methodologies in the proposed approach ensures robust change management. Capability-based variant management guarantees that changes are systematically realized, while DSM modeling facilitates the identification and analysis of affected subsystems. Together, these methods support the dynamic adaptation of complex systems, ensuring their models remain accurate and reliable during changes.

5.2 Validity Threats and Limitations

In this chapter, we discuss the potential threats to the validity of the research presented in this thesis. While the CAPLA methodology was developed to enhance the effectiveness of MBPLE in managing variants within complex systems, it was tested in a pilot project and not in a real-world application. This limitation introduces several validity threats that must be acknowledged and addressed.

The CAPLA methodology was applied within a controlled, pilot environment. Despite efforts to incorporate complexities and challenges into the pilot study, the inherent limitations of this controlled setting may prevent the results and conclusions from fully reflecting the intricacies and unforeseen challenges that could arise in real-world applications. This constraint may impact the internal validity of the findings, as the pilot may not encompass all potential variables that could influence outcomes in a more dynamic and unpredictable environment. Since the methodology was applied in a pilot under the researcher's direct supervision, there is also a risk of bias. Decisions may have been unconsciously influenced by the researcher's expectations, potentially compromising the objectivity of the results.

Although an aerospace system was chosen as the pilot project to encompass a broad range of applications due to its high complexity and stringent regulations, the pilot's context and scale may not adequately represent the diverse systems and industries where the CAPLA methodology could be applied. Furthermore, the absence of a real-world application poses a significant threat to external validity, as the methodology's effectiveness and robustness have not been tested in an operational setting. In such environments, additional factors like organizational constraints, budget limitations, and real-time decision-making pressures could significantly influence the outcomes.

While the CAPLA methodology shows promise in a controlled pilot environment, several validity threats arise due to the absence of real-world application. These threats highlight the need for further research and testing in diverse, real-world scenarios to fully validate the methodology's effectiveness and generalizability. Future work focus will also be on applying CAPLA in operational environments to address these validity concerns and refine the methodology based on real-world feedback.

CHAPTER 6

CONCLUSIONS

This thesis aims to support the tasks of managing variability in MBPLE by providing a structured method for the product development cycle. Along with advancements in the aerospace industry, variant management has become crucial for protecting efficiency and quality. The development of a variability management method for MBPLE represents a significant advancement in addressing the complexities and challenges inherent in aerospace product development.

The proposed variability management method is grounded in a comprehensive analysis of existing techniques and practices, identifying critical gaps and opportunities for innovation. By integrating MBSE and feature modeling approaches with variability management principles, this method provides a cohesive approach to managing the diverse and dynamic requirements of product lines. The feature modeling approach defines variation points and creates product configurations. MBSE is also an excellent approach for presenting system architecture with models at various abstraction levels that support every development level in systems engineering. Incorporating these two advanced modeling approaches, MBPLE techniques are effective in capturing and managing variability in system architecture but in terms of variability in different levels and abstractions, they lack flexibility.

This framework harmonizes ARCADIA and feature modeling languages and tools, ensuring seamless integration and interoperability across different stages of product development. Additionally, taking systems engineering processes as a guideline, capability-based management, identification of separate but interconnected development spaces, and DSM models are used. With capability-based management, feature models aligned with system architecture and created for need and solution spaces, providing traceability between all system and feature artifacts. Incorporating capabilities, requirements, and subsystems as features into feature models and forming relations with related space artifacts, the method ensures that all design choices, variation points, and features are traceable up to requirements. It facilitates the consistent representation of variability information, enabling precise and unambiguous communication among stakeholders.

Recognizing that all variabilities cannot be visible from requirements, using DSM models and the SCV analysis methods, variabilities are captured for every development phase asset, i.e., requirements, subsystems, components, interfaces. Implementing a structured mapping methodology between system architecture and feature models, along with forming dependencies and relations between need and solution space feature models, enhances the creation of resultant system architecture models for every variant and enables automatic configuration. Automation in variability analysis significantly reduces the manual effort required to identify, document, and manage variability. This not only

enhances productivity but also minimizes the risk of errors and inconsistencies, leading to higher quality outcomes. Capability-based management is another essential approach used for minimizing errors in resultant system architecture models. This approach reduces manual effort in mapping feature and system architecture models by using capabilities as the representation of variability in the system. By managing system elements through their associated capabilities, variation points are linked to their existence reasoning via these capabilities. Even if an element is realized by multiple capabilities, the element and its related components remain in the system as long as at least one capability exists. This method addresses the issue of creating inconsistent models due to the presence of unrelated elements or faulty elimination of related elements, as it effectively manages variability through capability-based management rather than linking it to a single variant-related element.

Providing a hierarchical relationship and classification techniques across all modeling elements, the method is designed to scale with the growing complexity of product lines, accommodating a wide range of products and variations. Its flexible architecture allows for easy adaptation to evolving requirements and technological advancements, ensuring long-term applicability and relevance. Our approach emphasizes seamless integration with existing development processes, promoting a holistic view of product line engineering. This ensures that variability management is not an isolated activity but an integral part of the overall development lifecycle.

The empirical validation of our method, conducted through a pilot study and evaluation of the established requirements for an effective variant management methodology in system architecture models based on comprehensive literature reviews and research, demonstrates its efficacy in enhancing variability management in MBPLE. The findings highlight substantial enhancements in development efficiency, product quality, and stakeholder satisfaction. In conclusion, the proposed variability management method addresses crucial challenges in MBPLE, providing a comprehensive and practical solution that enhances the management of complex product lines effectively. By promoting greater consistency, efficiency, and scalability, this method contributes to advancing practices in product line engineering and lays the foundation for more innovative and high-quality products.

Future Work

Building upon the foundations laid in this thesis, several expansions and refinements emerge, focusing on further automation, evaluation of system performance, and business analysis of variants. In this work, with capability-based management, mapping was not fully extended to solution-level system modeling artifacts. Logical and physical architecture-level modeling elements, except for functions and interactions, were incorporated into the mapping model due to the restrictions of realization and propagation rules between modeling elements in tools. Future work can expand these realization and propagation rules to encompass all elements at the solution level, leveraging the open-source development space provided by the tools. Another significant enhancement

involves the automatic evaluation of variability's effect on system performance. Although this work considered non-functional requirements, it did not use system performance metrics to conduct advanced engineering analysis. Future research should integrate diverse system properties for every product family into variation points, allowing for automatic analysis of system performance. By incorporating system performance metrics, the variability management method can provide more detailed and actionable insights into the impact of variability on system effectiveness and efficiency. Lastly, future research should define and establish quantitative metrics like variant similarities to measure business impact of variability. Conducting a comprehensive business analysis will provide valuable insights for productivity gains, cost efficiencies, revenue generation from new variants, and overall enhancement of market competitiveness.

REFERENCES

Alves, V., Niu, N., Alves, C., & Valença, G. (2010). Requirements engineering for software product lines: A systematic literature review. *Information and Software Technology*, *52*(8), 806–820. https://doi.org/10.1016/j.infsof.2010.03.014

Antunes, G., & Borbinha, J. (2013). Capabilities in Systems Engineering: An Overview. In J. Falcão E Cunha, M. Snene, & H. Nóvoa (Eds.), *Exploring Services Science* (Vol. 143, pp. 29–42). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-36356-6_3

Apel, S., Batory, D., Kästner, C., & Saake, G. (2013). *Feature-Oriented Software Product Lines: Concepts and Implementation*. Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-37521-7

Azani, C. (2008). An Open Systems Approach to System of Systems Engineering. In M. Jamshidi (Ed.), *System of Systems Engineering: Innovations for the 21st Century* (1st ed.). Wiley. https://doi.org/10.1002/9780470403501

Berg, K., Bishop, J., & Muthig, D. (2005, July). Tracing software product line variability: from problem to solution space. In *Proceedings of the 2005 annual research conference of the South African institute of computer scientists and information technologists on IT research in developing countries* (pp. 182-191).

Berger, T., Rublack, R., Nair, D., Atlee, J. M., Becker, M., Czarnecki, K., & Wąsowski, A. (2013). A survey of variability modeling in industrial practice. *Proceedings of the Seventh International Workshop on Variability Modelling of Software-Intensive Systems*, 1–8. https://doi.org/10.1145/2430502.2430513

Beuche, D. (2004). *Demonstration: Variant and Variability Management with pure::variants*.

Blanchard, B. S., & Fabrycky, W. J. (2014). *Systems Engineering and Analysis* (5th ed.). Pearson Education.

Bonnet, S., Voirin, J.-L., & Navas, J. (2019). Augmenting requirements with models to improve the articulation between system engineering levels and optimize V&V practices. *INCOSE International Symposium*, *29*(1), 1018–1033. https://doi.org/10.1002/j.2334-5837.2019.00650.x

Browning, T. R. (2016). Design Structure Matrix Extensions and Innovations: A Survey and New Opportunities. *IEEE Transactions on Engineering Management*, 63(1), 27–52. https://doi.org/10.1109/TEM.2015.2491283

Buede, D. M. (2009). *The engineering design of systems: Models and methods* (Second edition). John Wiley & Sons.

Capilla, R., Bosch, J., & Kang, K.-C. (Eds.). (2013). *Systems and Software Variability Management: Concepts, Tools and Experiences*. Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-36583-6

Coplien, J., Hoffman, D., & Weiss, D. (1998). Commonality and variability in software engineering. *IEEE Software*, *15*(6), 37–45. https://doi.org/10.1109/52.730836

Czarnecki, K., & Antkiewicz, M. (2005). Mapping Features to Models: A Template Approach Based on Superimposed Variants. In *Generative Programming and Component Engineering*.

Dumitrescu, C., Mazo, R., Salinesi, C., & Dauron, A. (2013). Bridging the gap between product lines and systems engineering: An experience in variability management for automotive model based systems engineering. *Proceedings of the 17th International Software Product Line Conference*, 254–263. https://doi.org/10.1145/2491627.2491655

Eppinger, S. D., & Browning, T. R. (2012). *Design structure matrix methods and applications*. MIT Press.

Estefan, J. A. (2007). *Survey of model-based systems engineering (MBSE) methodologies*. INCOSE MBSE Focus Group,25(8), 1-12.

Forlingieri, M. (2022). The four dimensions of Variability and their impact on MBPLE: How to approach variability in the development of aircraft product lines at Airbus. *Proceedings of the 16th International Working Conference on Variability Modelling of Software-Intensive Systems*, 1–4. https://doi.org/10.1145/3510466.3511275

Frank, U., Loucopoulos, P., Pastor, Ó., & Petrounias, I. (Eds.). (2014). *The Practice of Enterprise Modeling: 7th IFIP WG 8.1 Working Conference, PoEM 2014, Manchester, UK, November 12-13, 2014. Proceedings* (Vol. 197). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-662-45501-2

Friedenthal, S., Moore, A., & Steiner, R. (2014). *A Practical Guide to SysML: The Systems Modeling Language* (3rd ed.). Morgan Kaufmann.

Gaeta, J. P., & Czarnecki, K. (2015). Modeling aerospace systems product lines in SysML. *Proceedings of the 19th International Conference on Software Product Line*, 293–302. https://doi.org/10.1145/2791060.2791104

Góngora, H. G. C., Ferrogalini, M. F., & Moreau, C. (2015). How to Boost Product Line Engineering with MBSE – A Case Study of a Rolling Stock Product Line. *In Complex Systems Design & Management: Proceedings of the Fifth International Conference on Complex Systems Design & Management CSD&M 2014*. Springer International Publishing. https://doi.org/10.1007/978-3-319-11617-4

Goos, G., Hartmanis, J., van Leeuwen, J., Hutchison, D., Kanade, T., Kittler, J., Kleinberg, J. M., Kobsa, A., Mattern, F., Mitchell, J. C., Naor, M., Nierstrasz, O., Rangan, C. P., & Steffen, B. (n.d.). *Lecture Notes in Computer Science*.

Greenough, R. M., & Williams, D. (2007). Investigating the transfer of techniques for electronic technical support documentation from aerospace to machine tools. *The International Journal of Advanced Manufacturing Technology*, *32*(7–8), 774–779. https://doi.org/10.1007/s00170-005-0380-7

Grönniger, H., Krahn, H., Pinkernell, C., & Rumpe, B. (2008). Modeling variants of automotive systems using views. *Modellbasierte Entwicklung von eingebetteten Fahrzeugfunktionen (MBEFF), Informatik Bericht*, *1*, 76-89.

H. Alan, T. S. March, J. Park, and S. Ram, "Design science in information systems research," *MIS Q.*, vol. 28, no. 1, pp. 75–105, 2004.

Hause, M., Korff, A., & Apperly, H. (2015). *Using Model-based Product Line Engineering for Decision Making and to Guide Product Development*.

Heidenreich, F., Sánchez, P., Santos, J., Zschaler, S., Alférez, M., Araújo, J., Fuentes, L., Kulesza, U., Moreira, A., & Rashid, A. (2010). Relating Feature Models to Other Models of a Software Product Line: A Comparative Study of FeatureMapper and VML*. *In S. Katz, M. Mezini, & J. Kienzle (Eds.), Transactions on Aspect-Oriented Software Development VII (Vol. 6210, pp. 69–114).* Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-16086-8_3

Hummell, J., & Hause, M. (2015). Model-based Product Line Engineering—Enabling product families with variants. *2015 IEEE Aerospace Conference*, 1–8. https://doi.org/10.1109/AERO.2015.7119108

IEEE Computer Society (IEEE). (2005). IEEE-STD-1220-2005—*IEEE standard for application and management of the systems engineering process.*

INCOSE. (2015). Systems Engineering Handbook: A Guide for System Life Cycle Process and Activities. Fourth Edition. D. Walden, G. Roedler, K. Forsberg, R.D. Hamelin, and T. Shortell (Eds.). San Diego, CA: International Council on Systems Engineering. Published by John Wiley & Sons.

INCOSE. (2021). *INCOSE systems engineering vision 2035*. INCOSE.

INCOSE. (2023). *Systems Engineering Handbook: A Guide for System Life Cycle Processes and Activities* (5th ed.). International Council on Systems Engineering.

ISO (2015), *ISO/IEC 26550:2015*, Software and systems engineering -- Reference model for product line engineering and management.

Jamshidi, M. (Ed.). (2008). *System of Systems Engineering: Innovations for the 21st Century* (1st ed.). Wiley. https://doi.org/10.1002/9780470403501

Kang, K. C., Cohen, S. G., Hess, J. A., Novak, W. E., & Peterson, A. S. (1990). *Feature-Oriented Domain Analysis (FODA) Feasibility Study:* Defense Technical Information Center. https://doi.org/10.21236/ADA235785

Keating, C. B. (2008). Emergence in System of Systems. In M. Jamshidi (Ed.), *System of Systems Engineering: Innovations for the 21st Century* (1st ed.). Wiley. https://doi.org/10.1002/9780470403501

Kossiakoff, A., Sweet, W. N., Seymour, S. J., & Biemer, S. M. (2011). *Systems Engineering Principles and Practice* (2nd ed.). Wiley.

Krueger, C. W. (2007). BigLever software gears and the 3-tiered SPL methodology. *Companion to the 22nd ACM SIGPLAN Conference on Object-Oriented Programming Systems and Applications Companion*. https://doi.org/10.1145/1297846.1297918

Krueger, C., & Clements, P. (2013). Systems and software product line engineering. *Encyclopedia of Software Engineering*, *2*, 1-14.

Laguna, M. A., & González-Baixauli, B. (2008). *Product Line Requirements: Multi-Paradigm Variability Models*.

Li, M., Guan, L., Dickerson, C., & Grigg, A. (2016). Model-based systems product line engineering with physical design variability for aircraft systems. *2016 11th System of Systems Engineering Conference (SoSE)*, 1–6. https://doi.org/10.1109/SYSOSE.2016.7542933

Link, S., & Trujillo, J. C. (Eds.). (2016). *Advances in Conceptual Modeling* (Vol. 9975). Springer International Publishing. https://doi.org/10.1007/978-3-319-47717-6

Loughran, N., Sánchez, P., Garcia, A., & Fuentes, L. (2008). Language Support for Managing Variability in Architectural Models. *In C. Pautasso & É. Tanter (Eds.), Software Composition (Vol. 4954, pp. 36–51)*. Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-540-78789-1_3

Meinicke, J., Thüm, T., Schröter, R., Benduhn, F., Leich, T., & Saake, G. (2017). *Mastering Software Variability with FeatureIDE*. Springer International Publishing. https://doi.org/10.1007/978-3-319-61443-4

Navas, J., Bonnet, S., Voirin, J., & Chalé Gongora, H. G. (2021). A value-driven, integrated approach to Model-Based Product Line Engineering. *INCOSE International Symposium*, *31*(1), 95–110. https://doi.org/10.1002/j.2334-5837.2021.00828.x

Open source MBSE tool. *Capella*. (n.d.). https://mbse-capella.org/

Pohl, K., Böckle, G., & Linden, F. van der. (2005). Software product line engineering: Foundations, principles, and techniques (1st ed). Springer.

Pure::variants:Pure-systems.pure-systems.(n.d.).https://www.pure-systems.com/purevariants

Pure::variants User's Guide. (n.d.).

Sage, A.P., & Rouse,W.B. (2011). *Handbook of systems engineering and management*. John Wiley & Sons.

Satyananda, T. K., Lee, D., Kang, S., & Hashmi, S. I. (2007). Identifying Traceability between Feature Model and Software Architecture in Software Product Line using

Formal Concept Analysis. *2007 International Conference on Computational Science and Its Applications (ICCSA 2007)*, 380–388. https://doi.org/10.1109/ICCSA.2007.59

Seidl, C., Heidenreich, F., & Aßmann, U. (2012). Co-evolution of models and feature mapping in software product lines. *Proceedings of the 16th International Software Product Line Conference - Volume 1*, 76–85. https://doi.org/10.1145/2362536.2362550

Sinnema, M., & Deelstra, S. (2007). Classifying variability modeling techniques. *Information and Software Technology*, 49(7), 717–739. https://doi.org/10.1016/j.infsof.2006.08.001

Tekinerdogan, B., & Sözer, H. (2012). Variability viewpoint for introducing variability in software architecture viewpoints. *Proceedings of the WICSA/ECSA 2012 Companion Volume*, 163–166. https://doi.org/10.1145/2361999.2362033

Thüm, T., Kästner, C., Benduhn, F., Meinicke, J., Saake, G., & Leich, T. (2014). *FeatureIDE: An extensible framework for feature-oriented software development.* Science of Computer Programming, *79*, 70-85.

Trujillo, J. C. (Eds.). (2016). *Advances in Conceptual Modeling* (Vol. 9975). Springer International Publishing. https://doi.org/10.1007/978-3-319-47717-6

Vijay Vaishnavi and Bill Kuechler And, “Design science research in Information systems (DSRIS),” 2004. [Online]. Available: http://desrist.org/design-research-in-information-systems/. [Accessed: 10-Aug-2024].

Voirin, J., Bonnet, S., Exertier, D., & Normand, V. (2016). Simplifying (and enriching) SysML to perform functional analysis and model instances. *INCOSE International Symposium*, *26*(1), 253–268. https://doi.org/10.1002/j.2334-5837.2016.00158.x

Voirin, J.-L. (2018). *Model-based system and Architecture Engineering with the arcadia method*. ISTE Press ; Elsevier.

Yu, D., Geng, P., & Wu, W. (2012). Constructing Traceability between Features and Requirements for Software Product Line Engineering. *2012 19th Asia-Pacific Software Engineering Conference*, 27–34. https://doi.org/10.1109/APSEC.2012.135

APPENDICES

APPENDIX A

In this section, we present the comprehensive structure of the 150% and resultant 100% models for variants of the Drone Systems. This includes detailed feature configurations and their associated mappings. Each variant is meticulously documented to ensure clarity and accuracy in representing the diverse capabilities and functionalities inherent in the system.

%150 DRONE MODEL

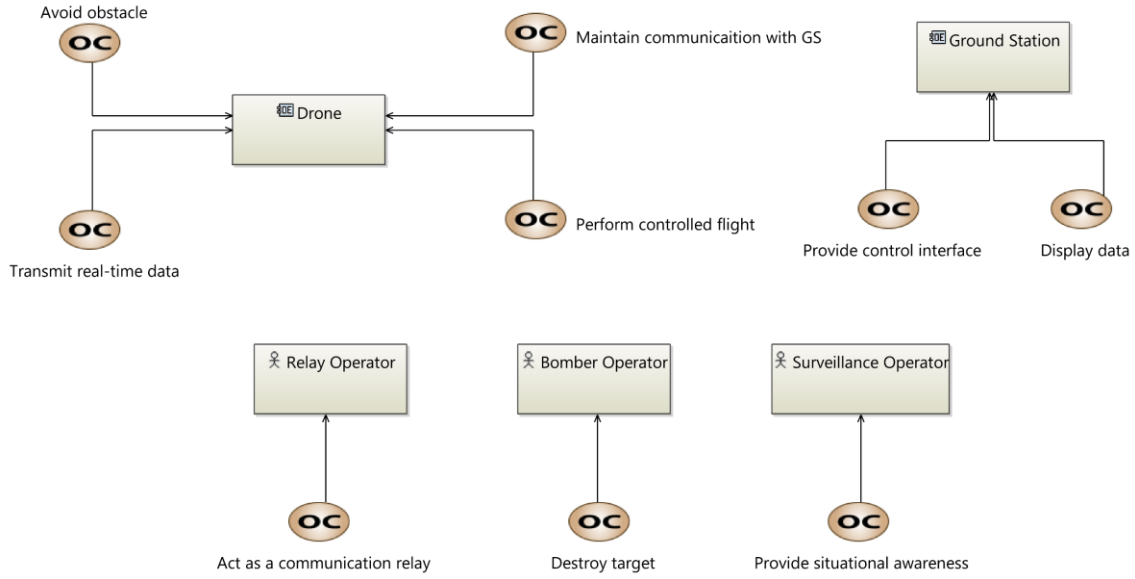

Figure 43: OCB Diagram of 150% Drone System

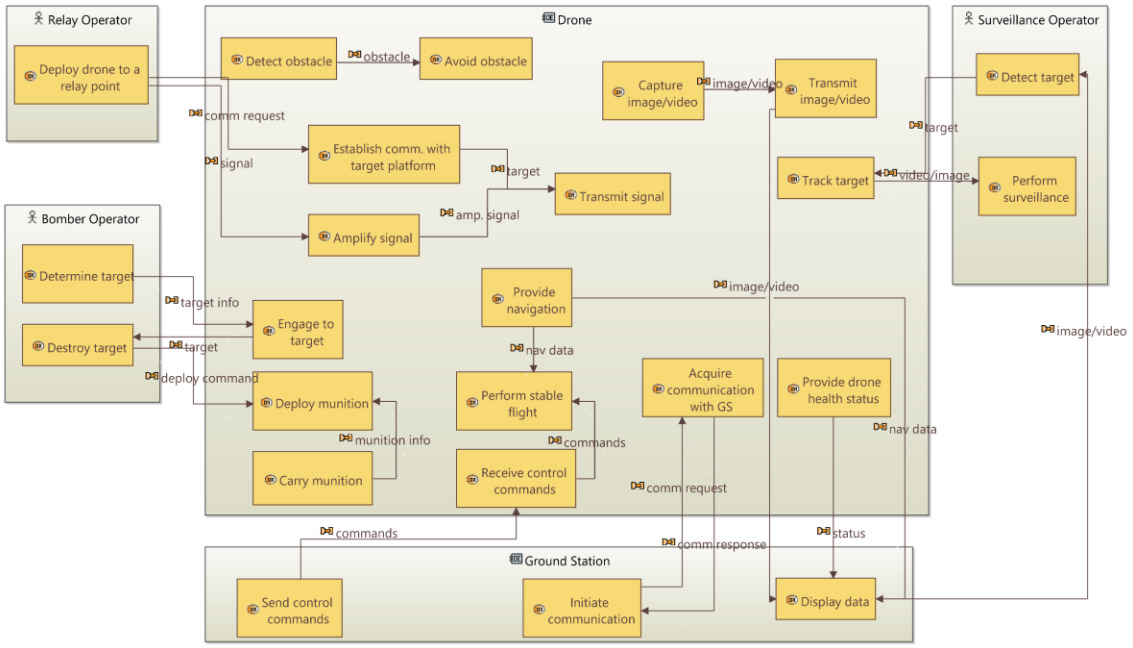

Figure 44: OAB Diagram of 150% Drone System

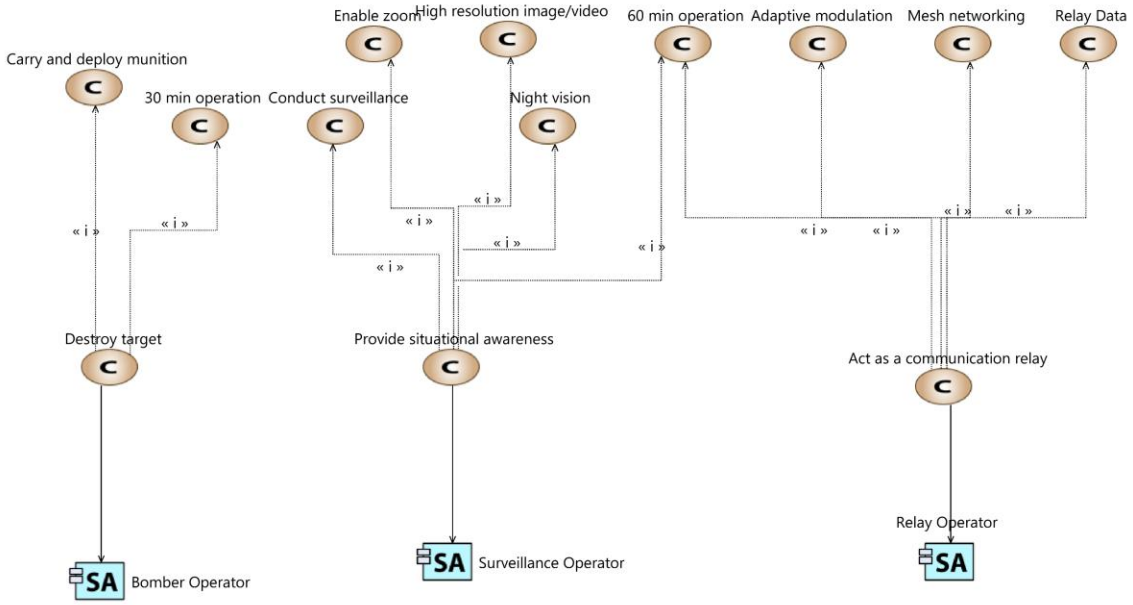

Figure 45: MCB Diagram of 150% Drone System

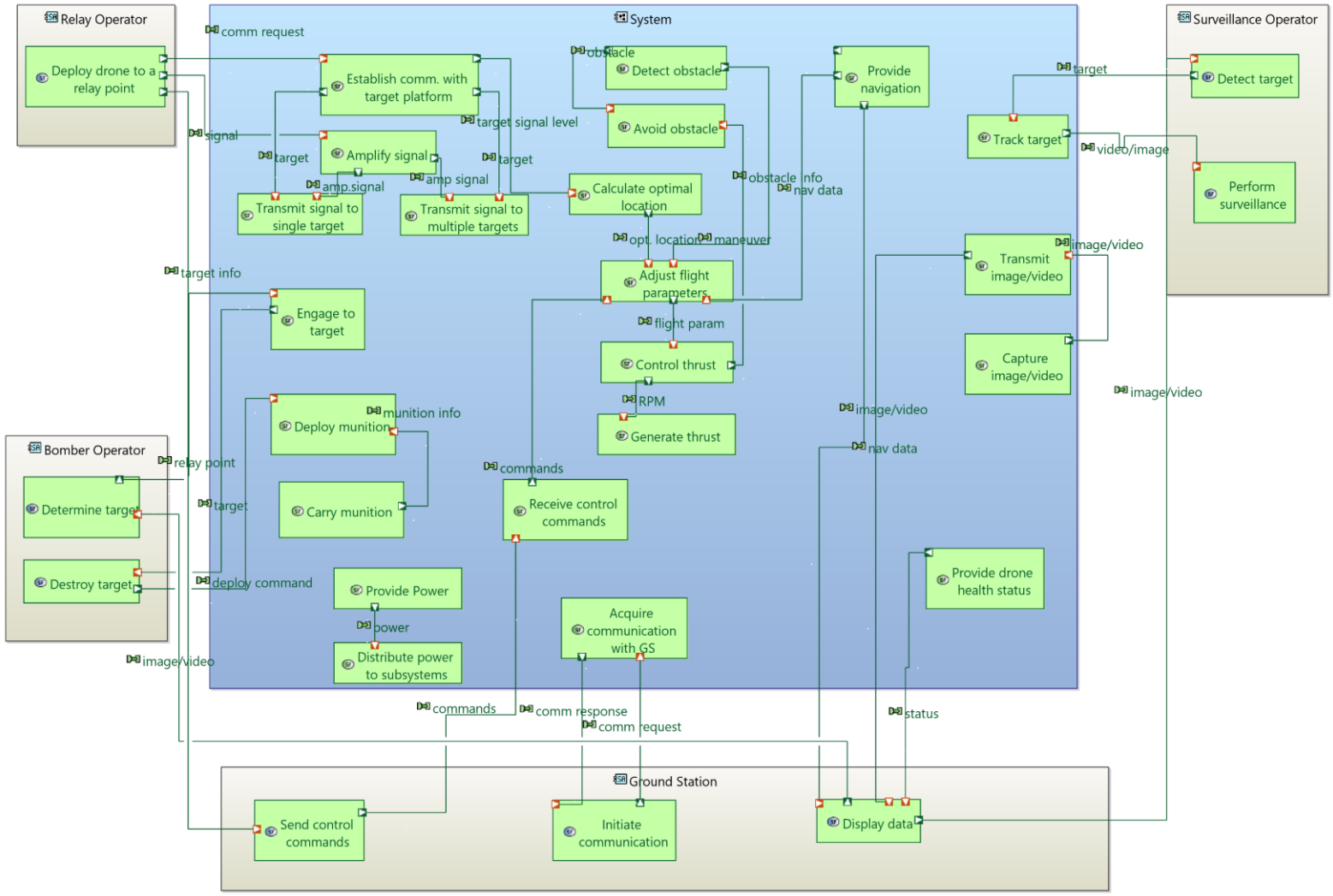

Figure 46: SAB Diagram of 150% Drone System

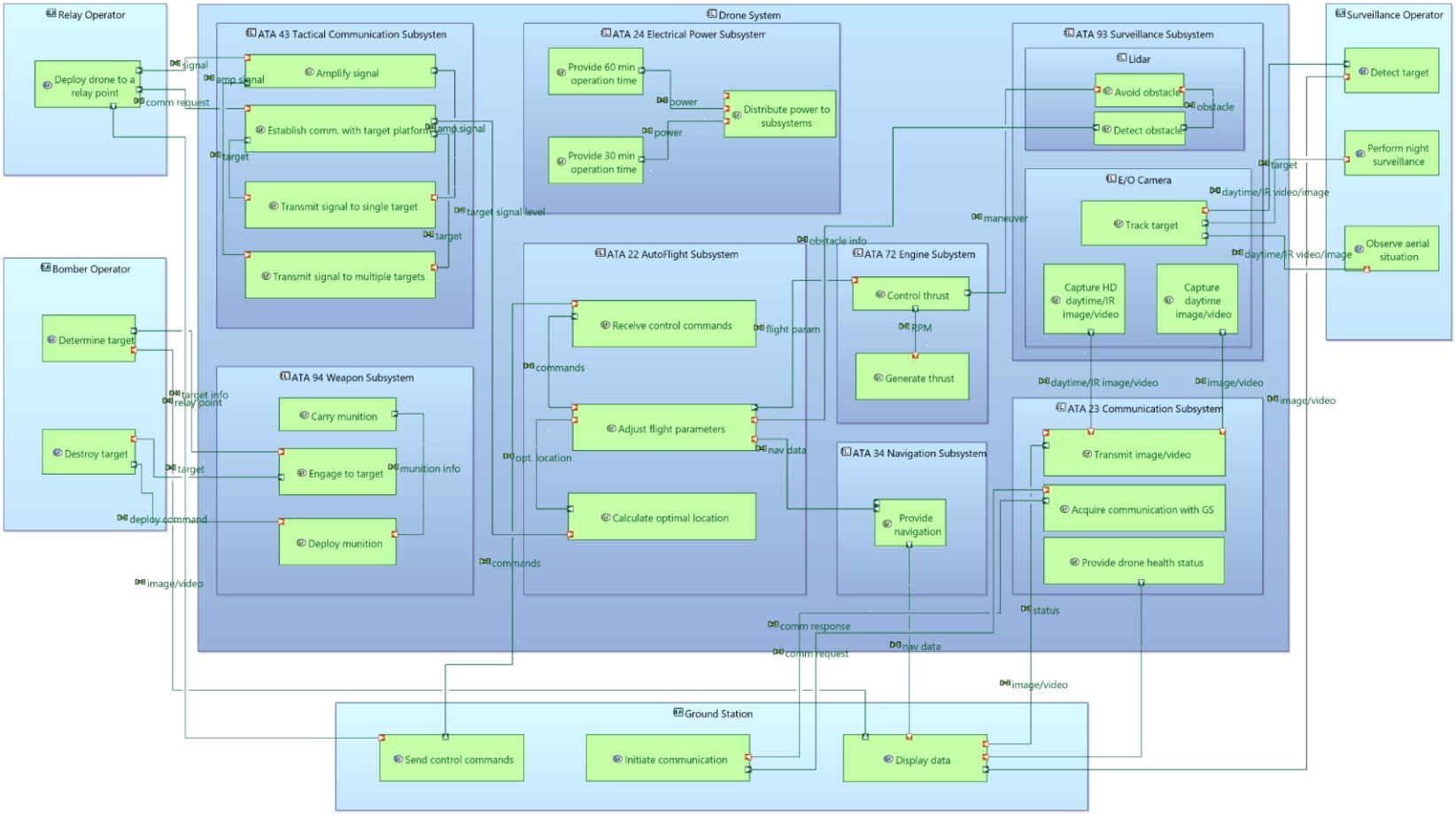

Figure 47: LAB Diagram of 150% Drone System

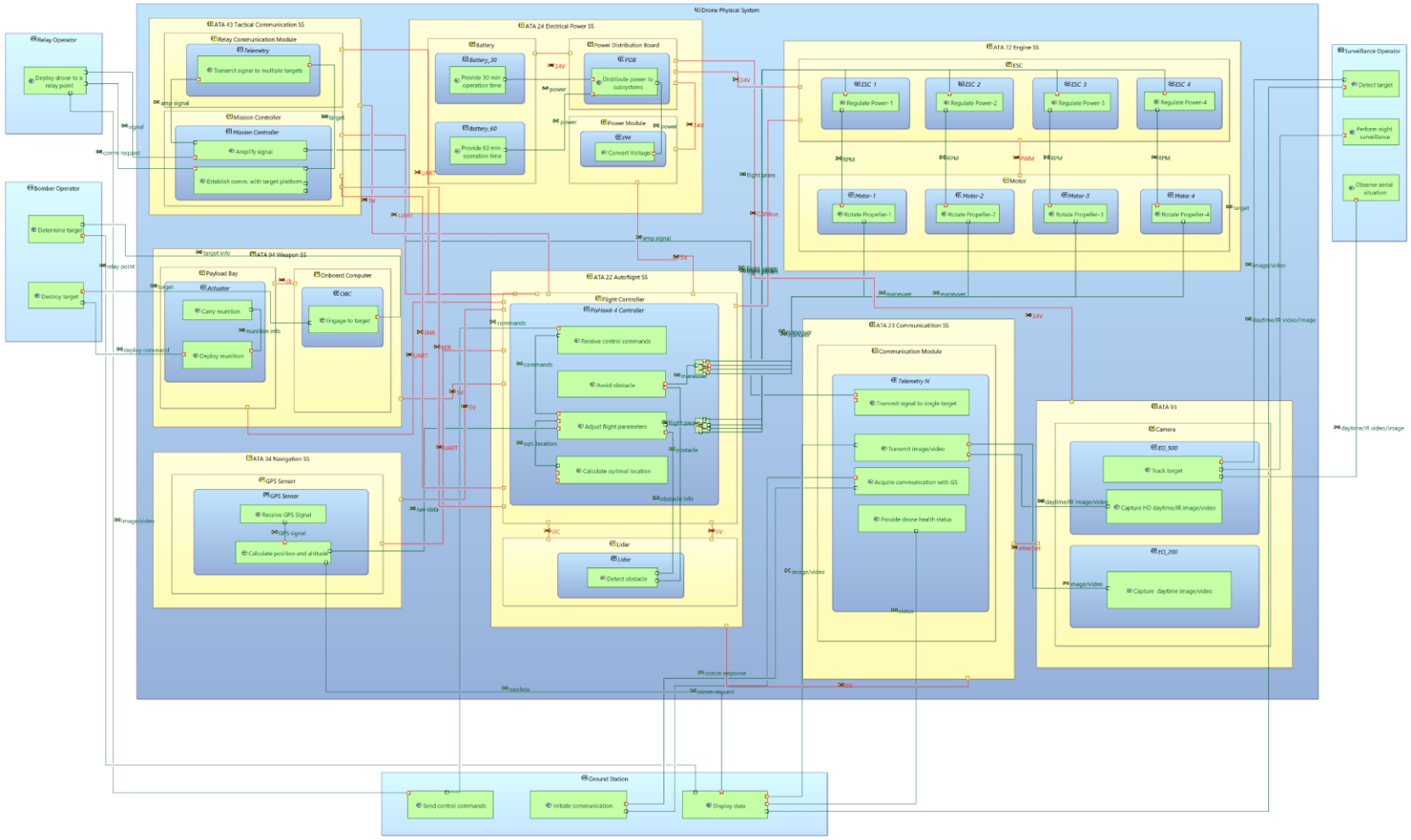

Figure 48: PAB Diagram of 150% Drone System

BOMBER DRONE

Figure 49 : Bomber Drone Feature Configuration

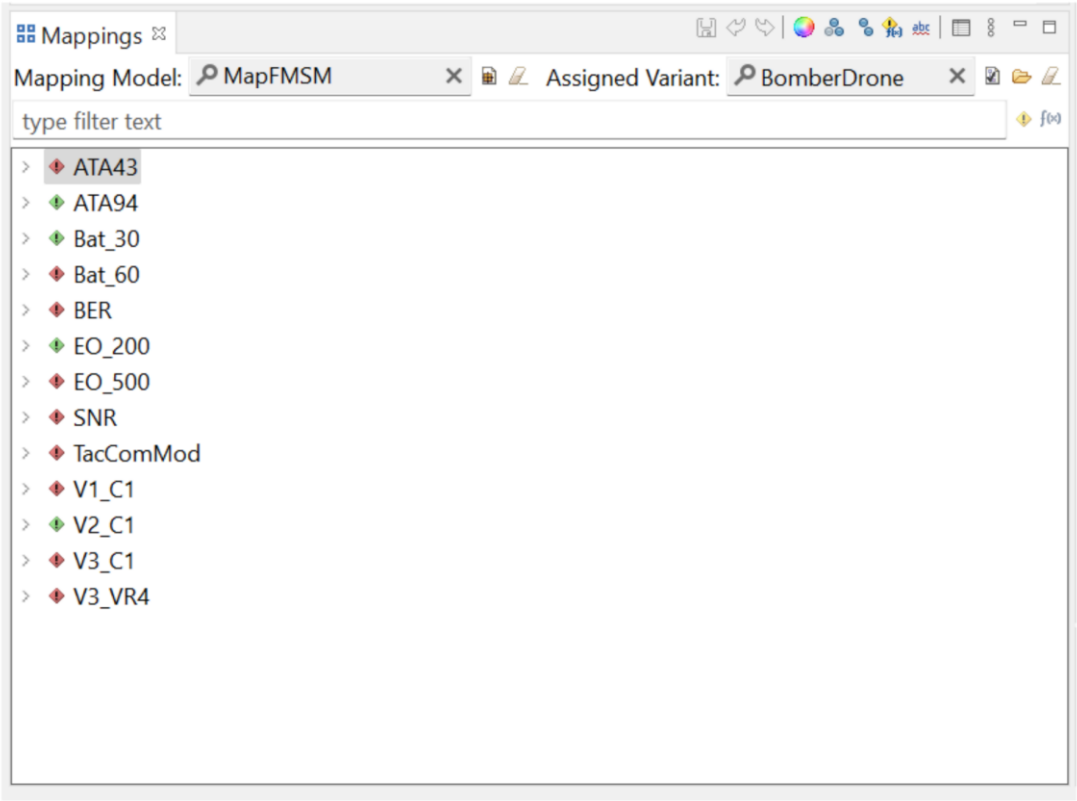

Figure 50: Bomber Drone Mapping Model

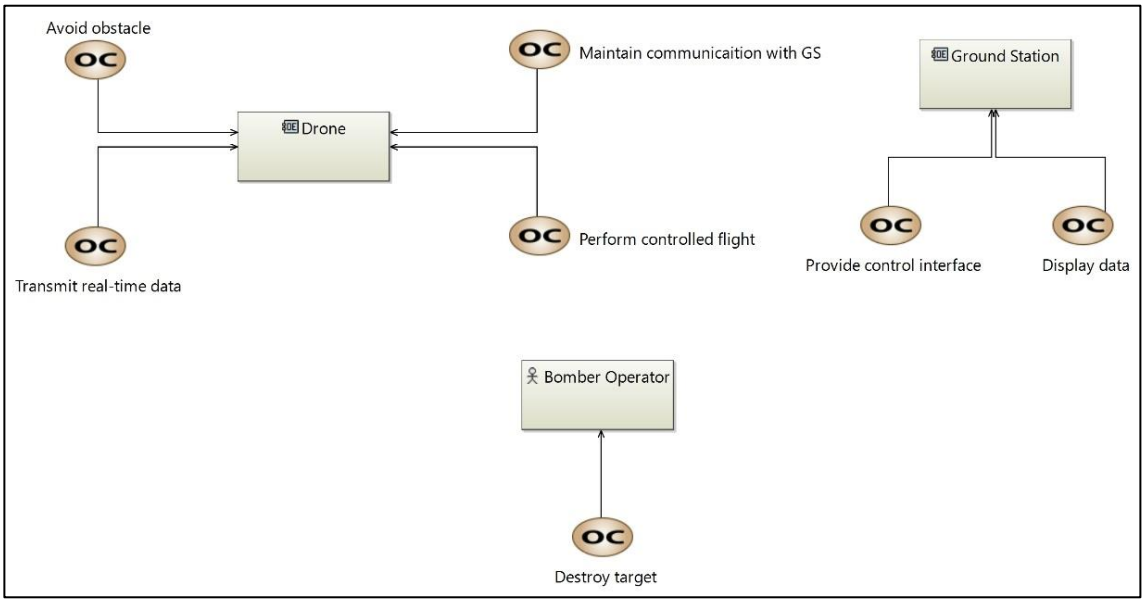

Figure 51 : Bomber Drone OCB Diagram

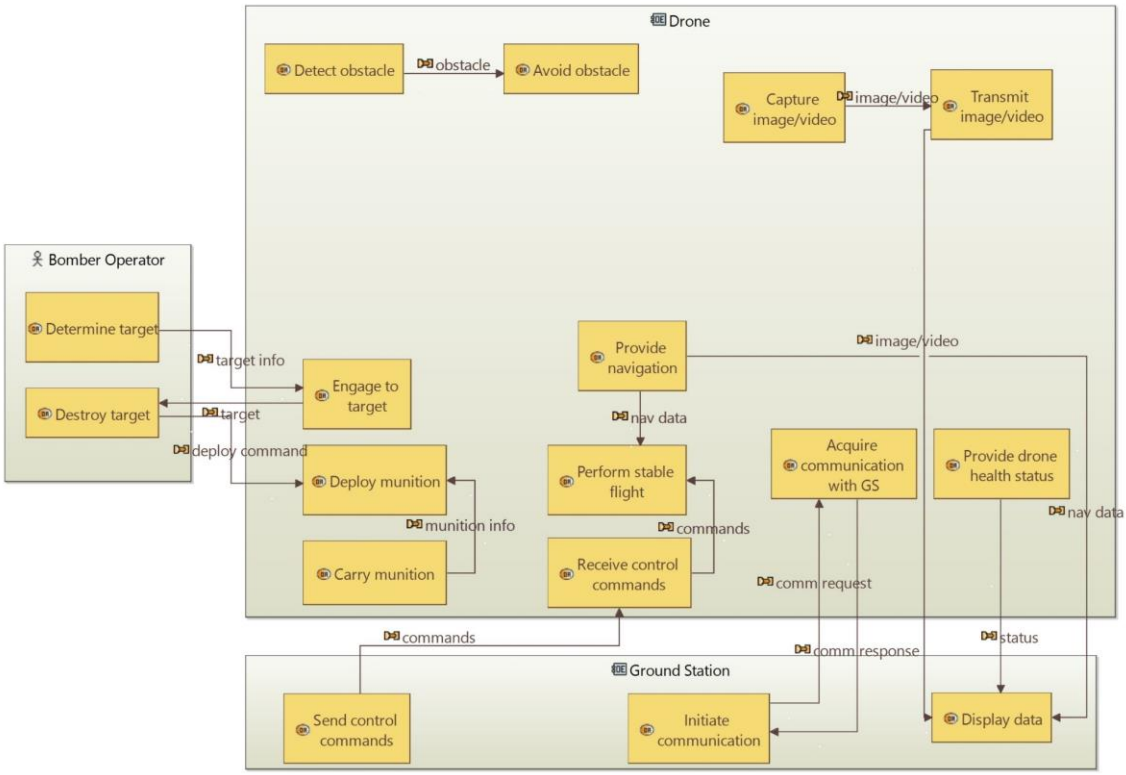

Figure 52 : Bomber Drone OAB Diagram

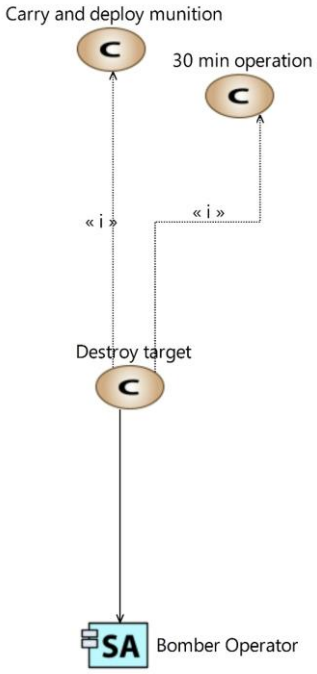

Figure 53 : Bomber Drone MCB Diagram

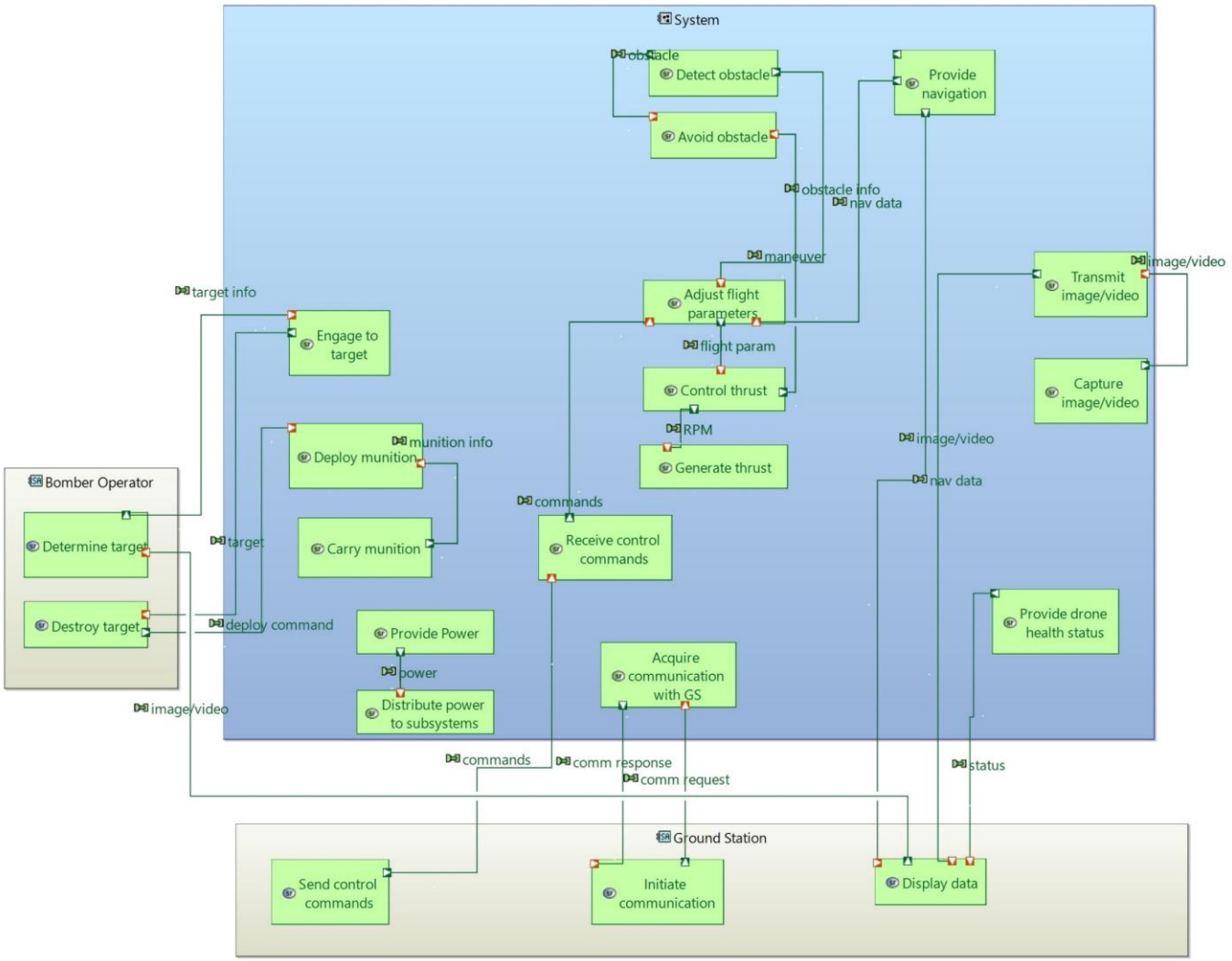

Figure 54 : Bomber Drone SAB Diagram

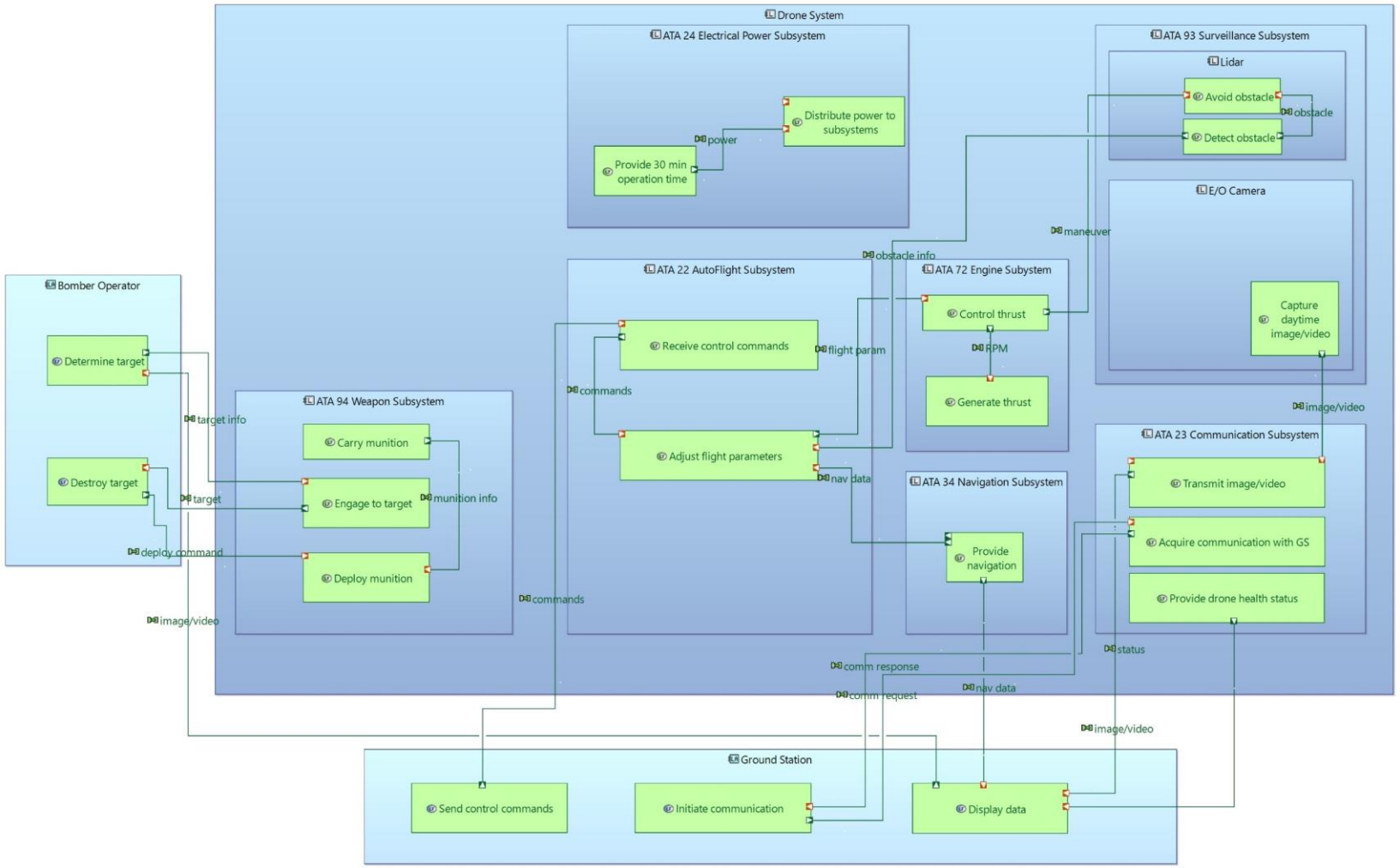

Figure 55 : Bomber Drone LAB Diagram

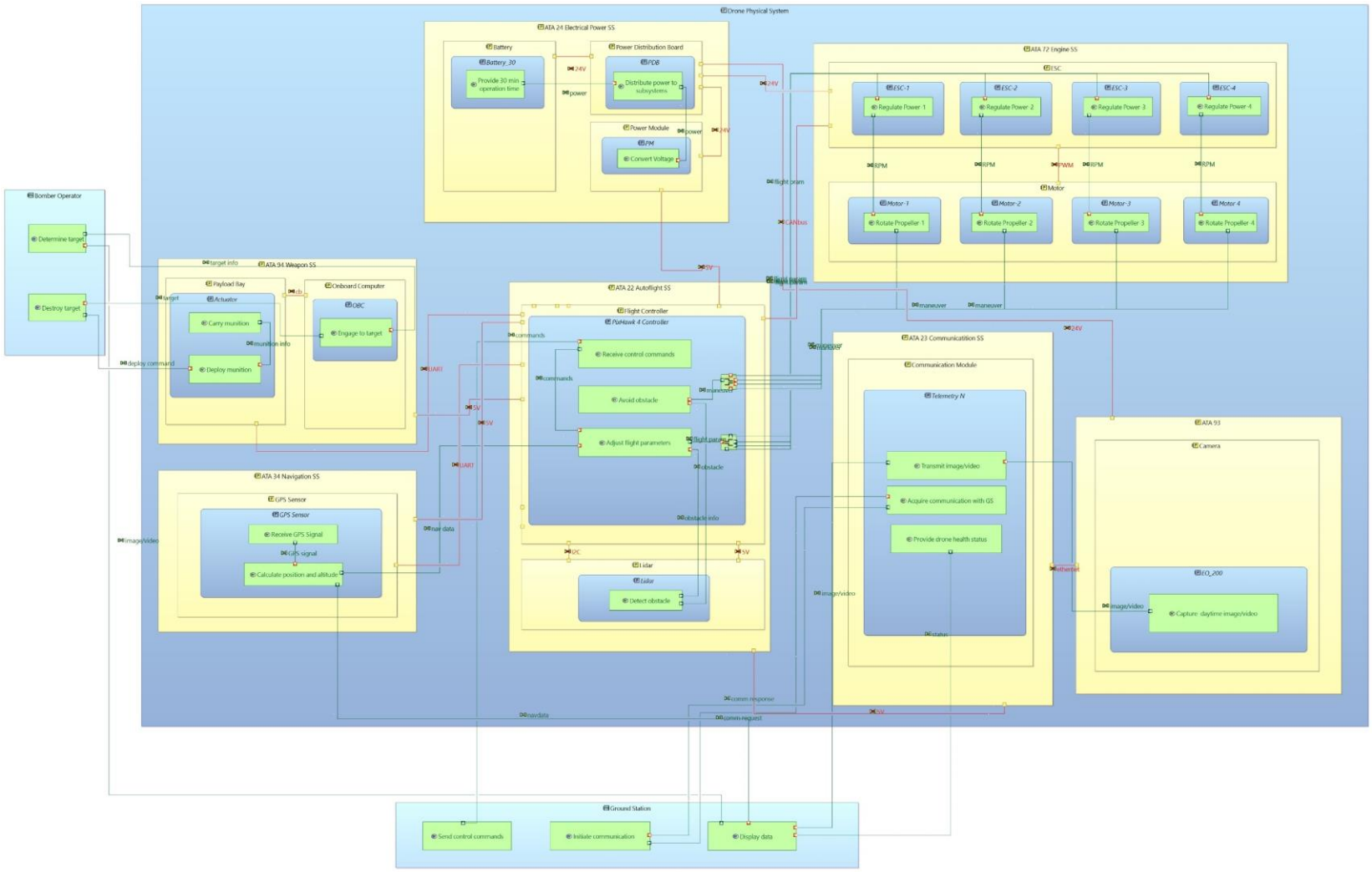

Figure 56 : Bomber Drone PAB Diagr

SURVEILLANCE DRONE

Figure 57: Surveillance Drone Feature Configuration

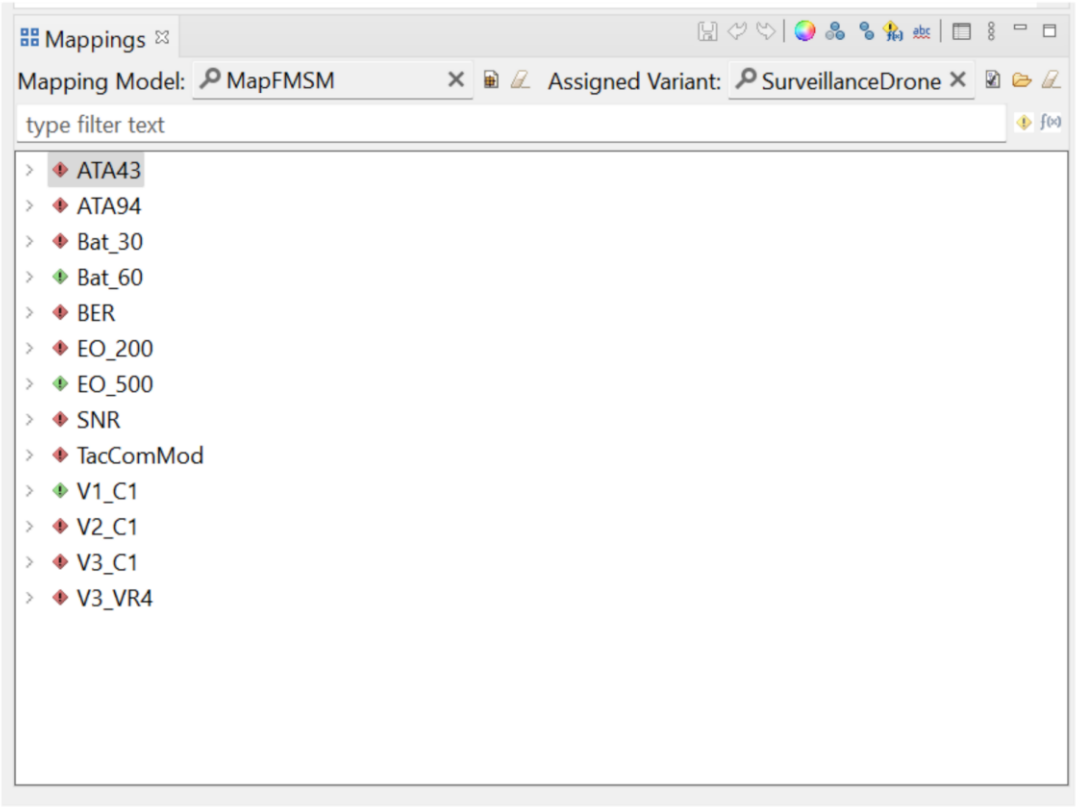

Figure 58: Surveillance Drone Mapping Model

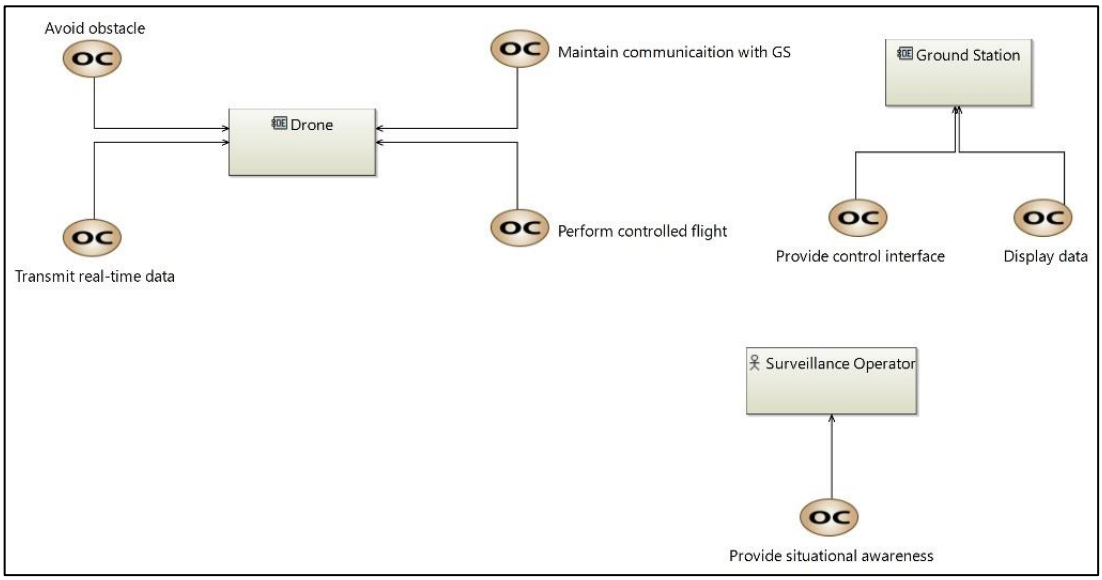

Figure 59 : Surveillance Drone OCB Diagram

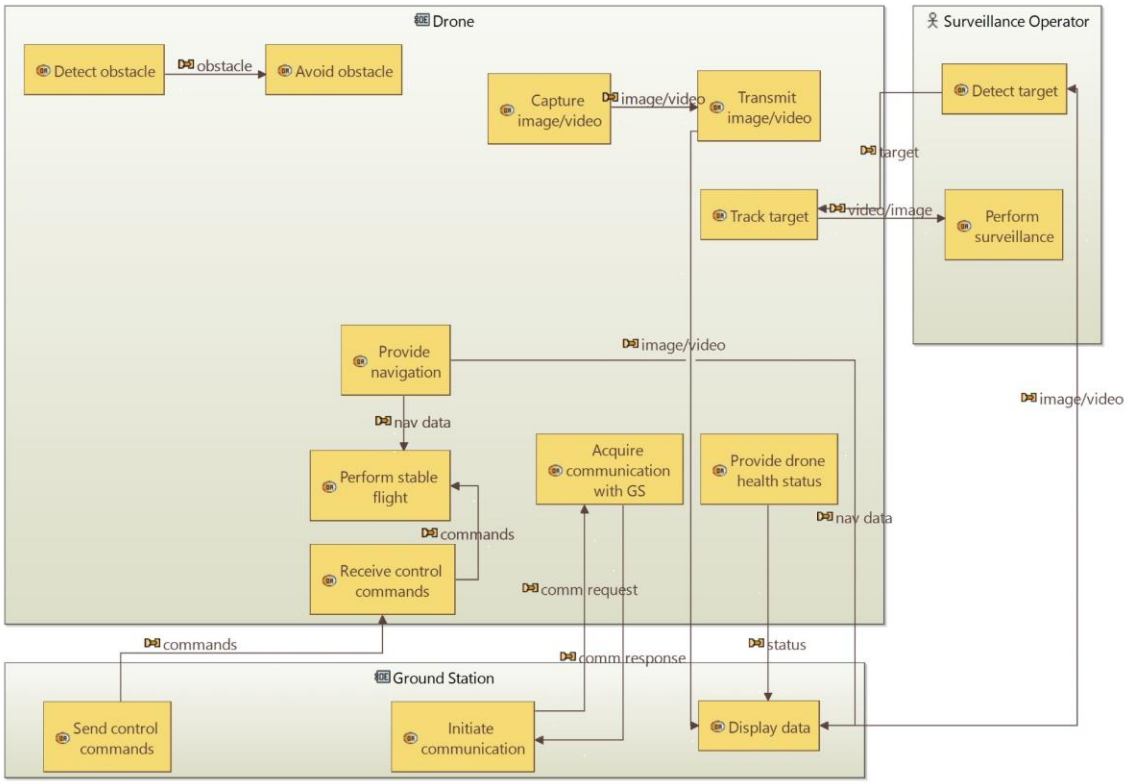

Figure 60 : Surveillance Drone OAB Diagram

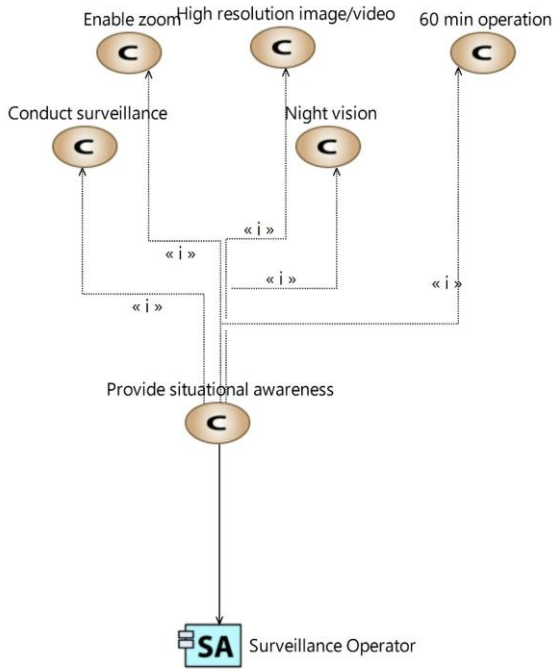

Figure 61 : Surveillance Drone MCB Diagram

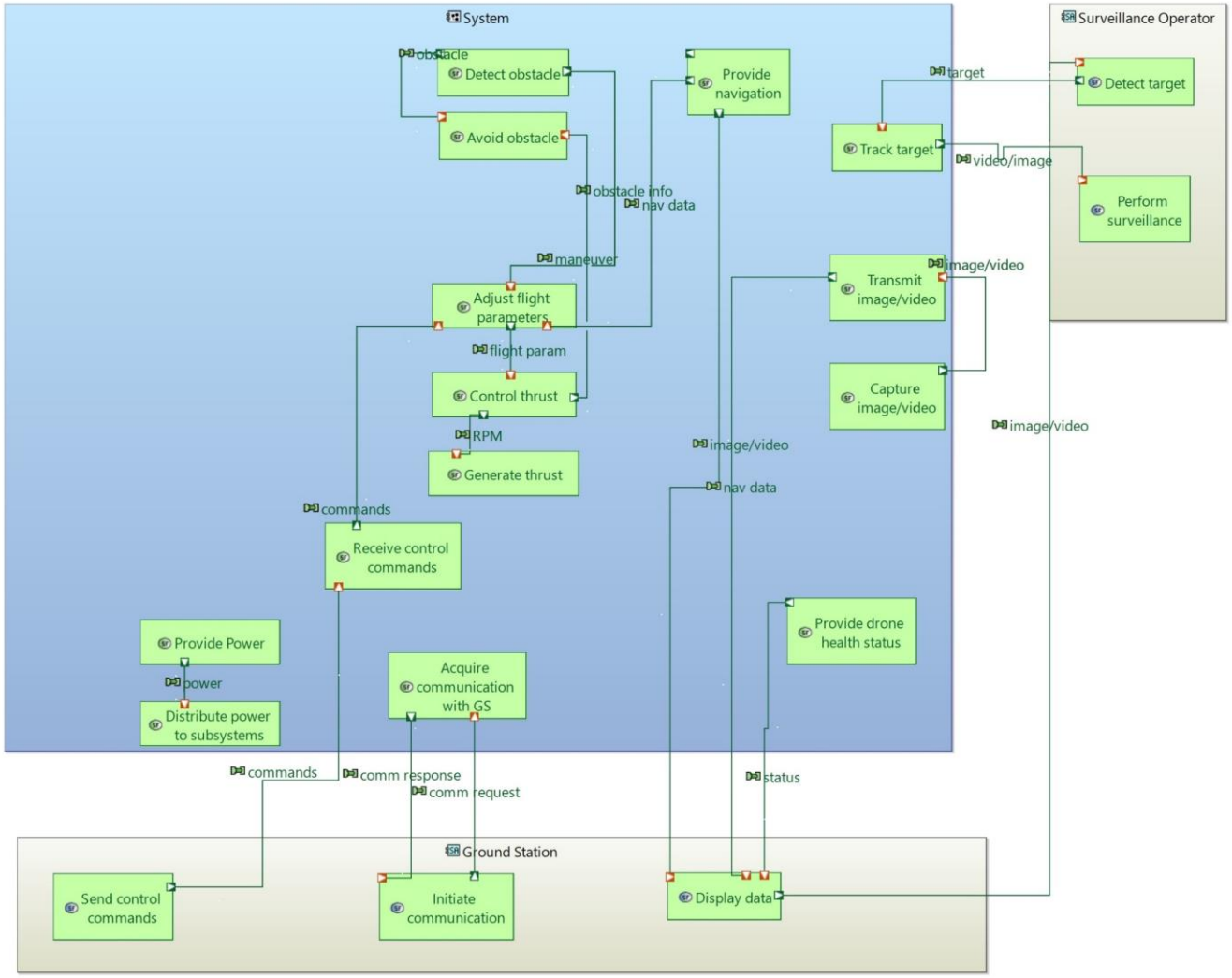

Figure 62 : Surveillance Drone SAB Diagram

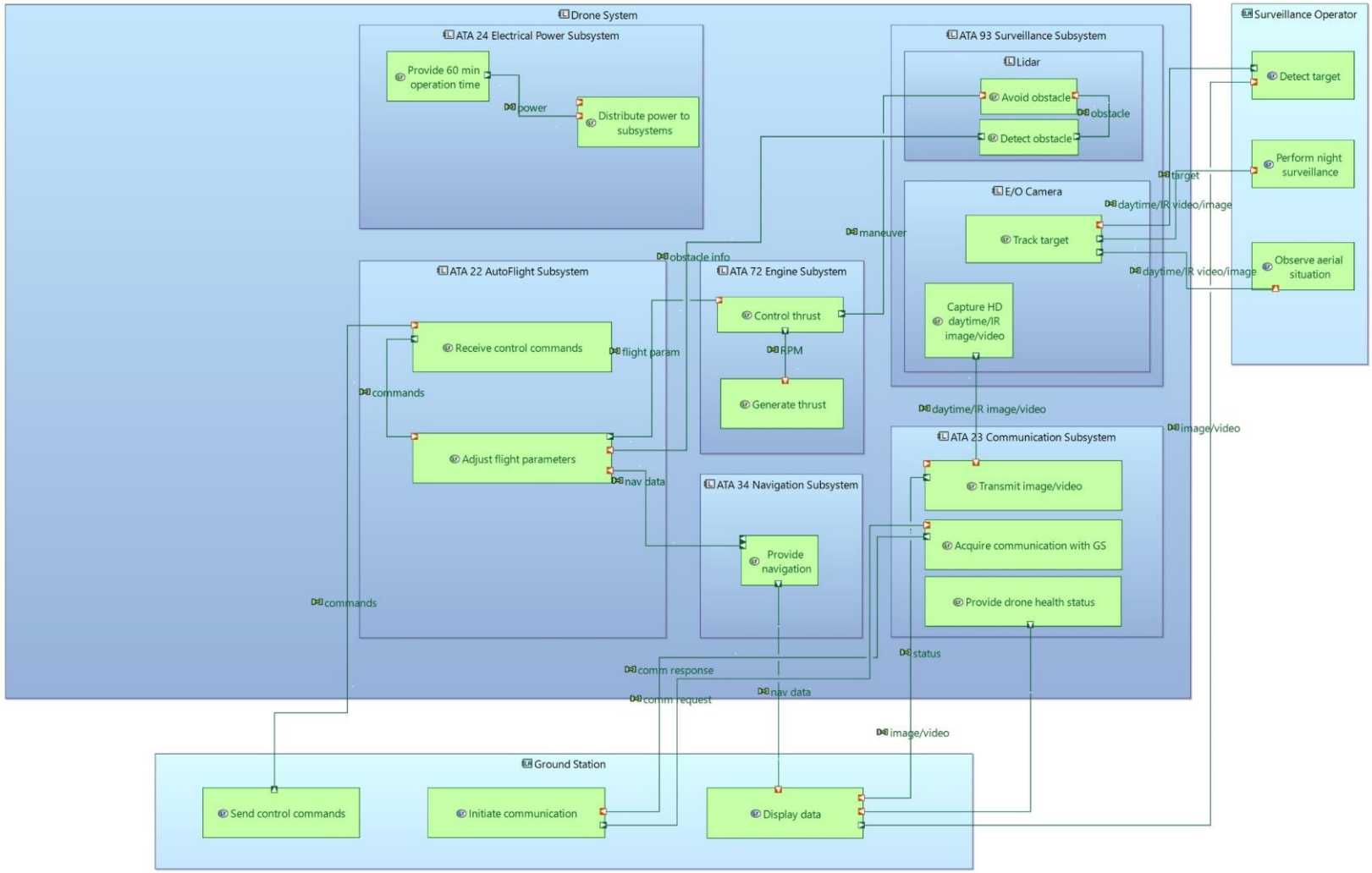

Figure 63 : Surveillance Drone LAB Diagram

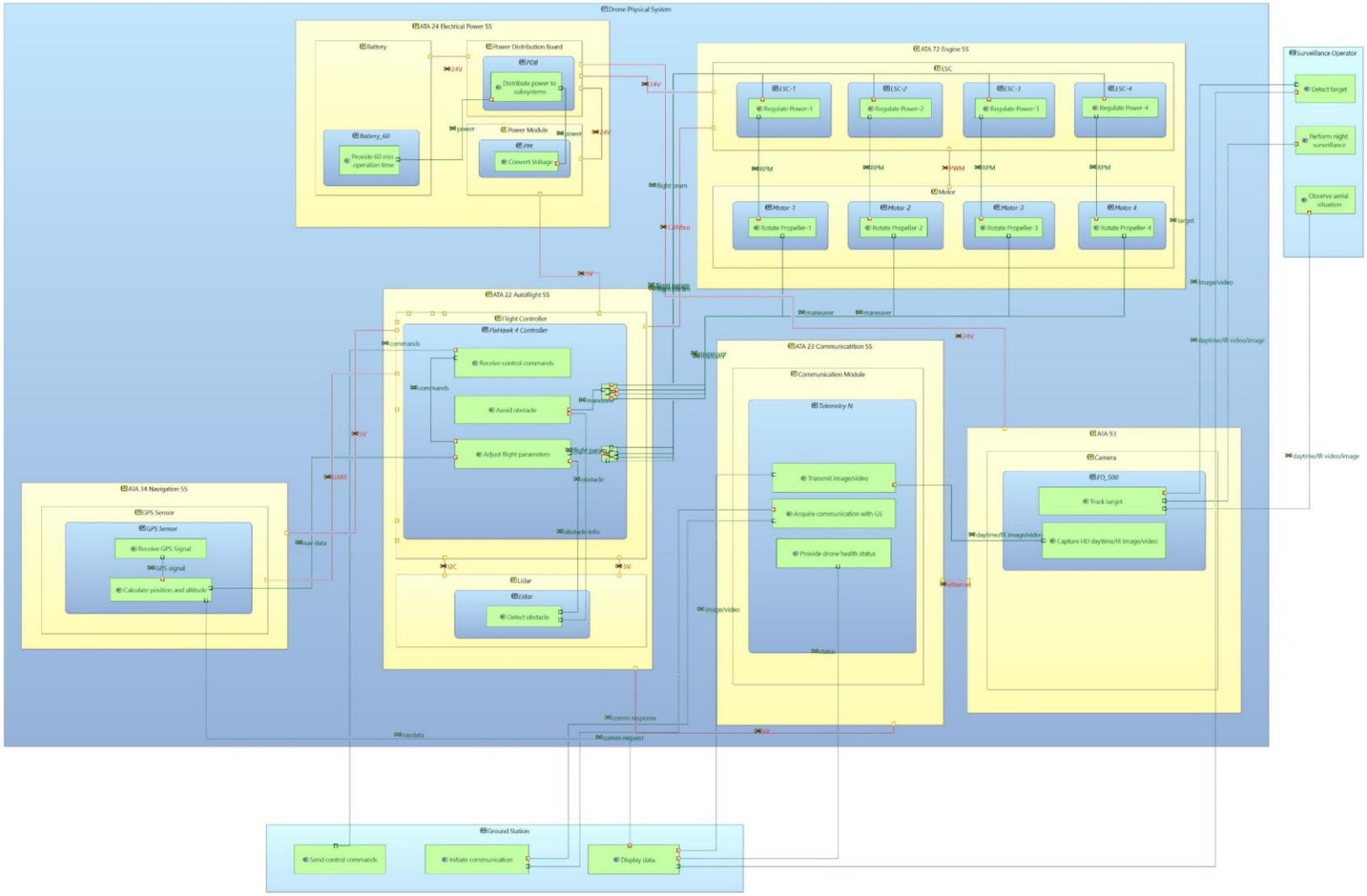

Figure 64 : Surveillance Drone PAB Diagram

RELAY DRONE 00

Figure 65 : Relay Drone 00 Feature Configuration

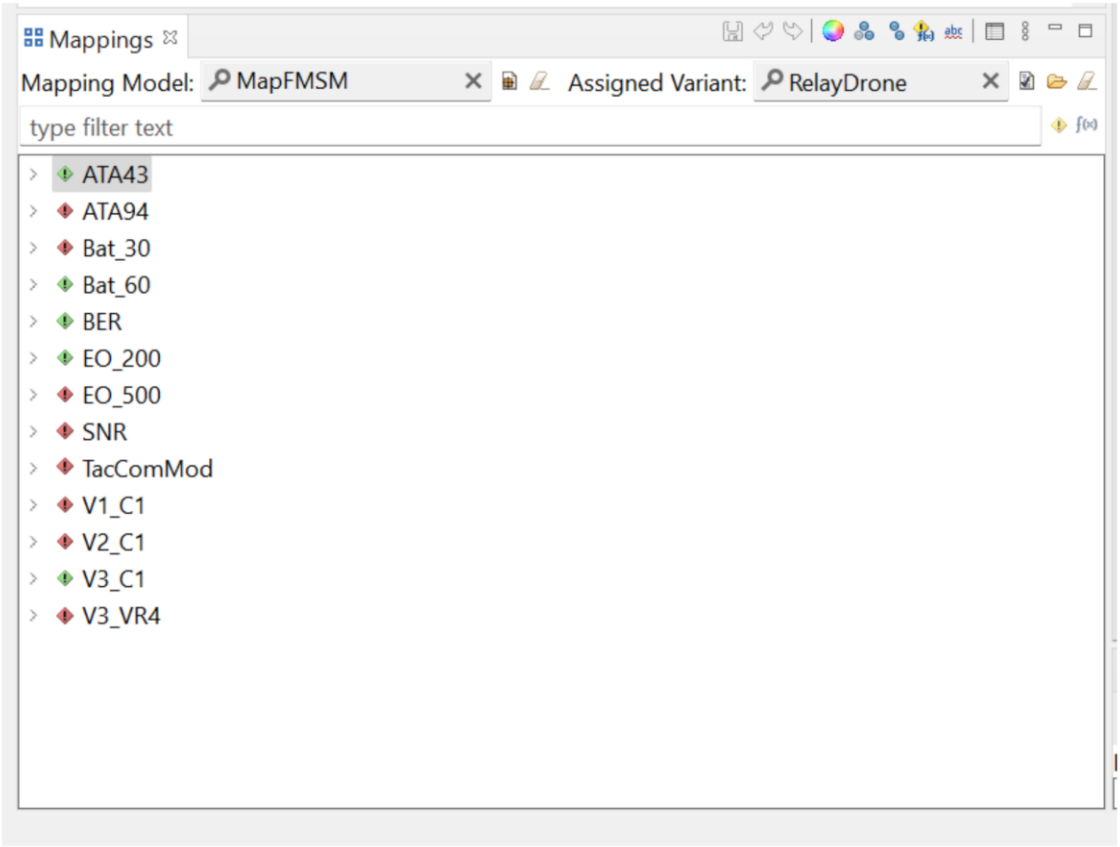

Figure 66: Relay 00 Drone Mapping Model

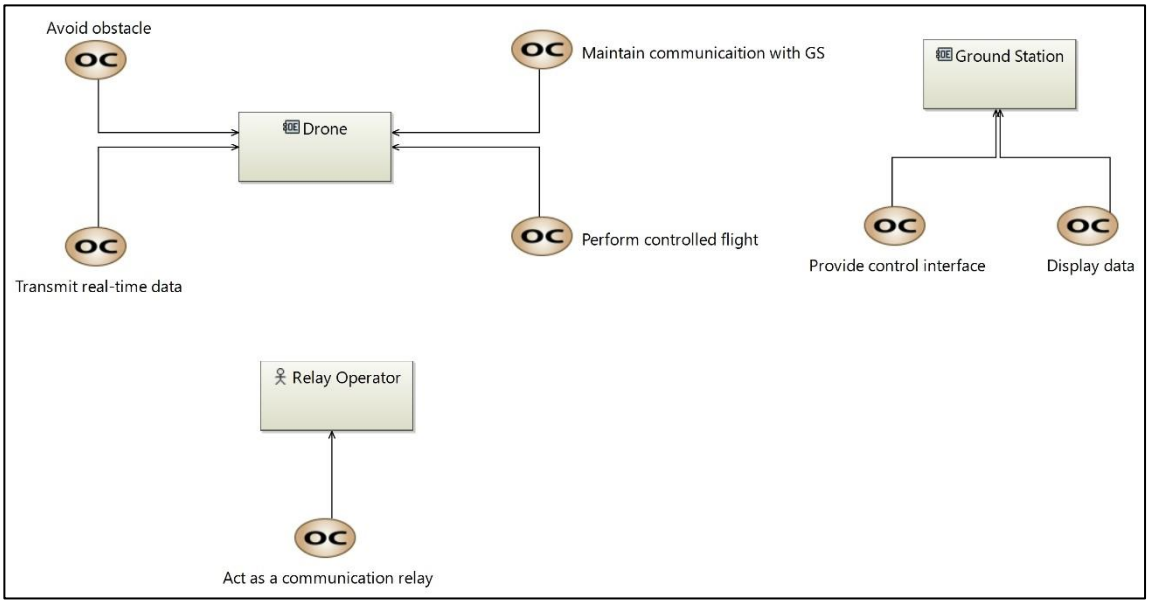

Figure 67 : Relay Drone 00 OCB Diagram

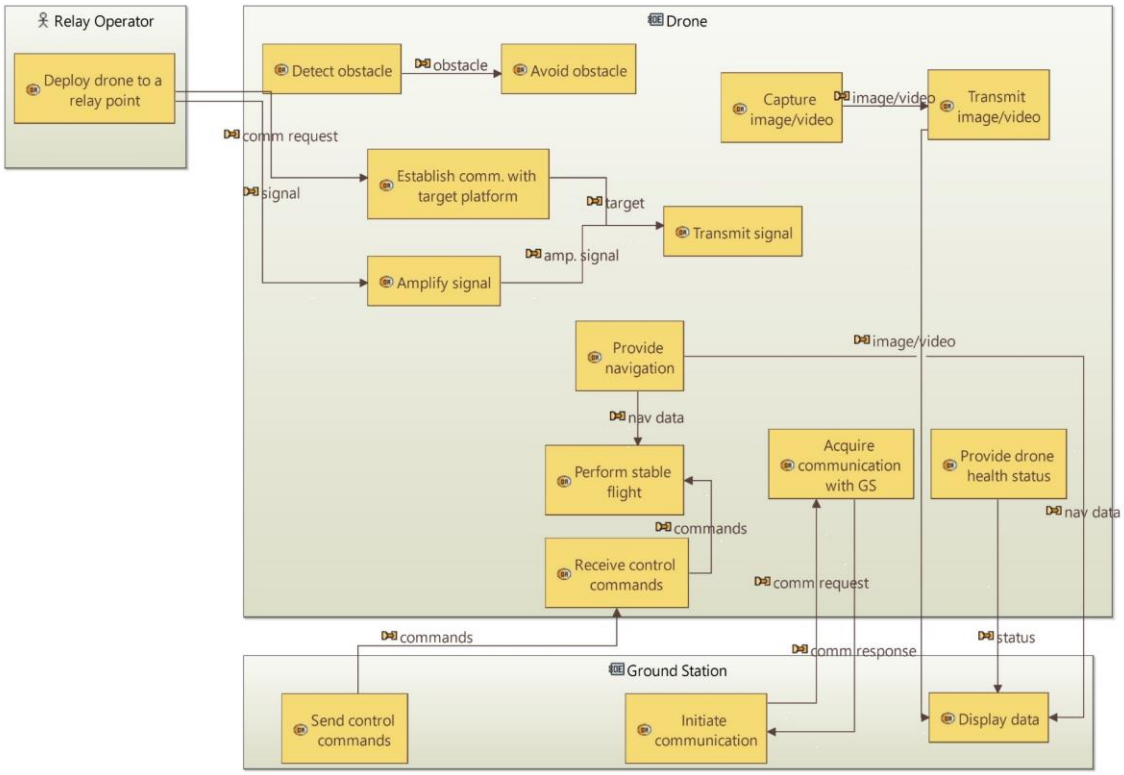

Figure 68 : Relay Drone 00 OAB Diagram

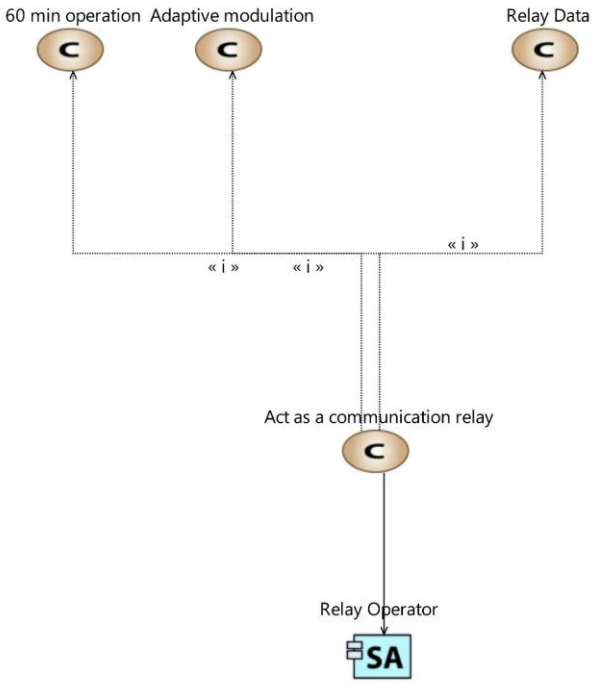

Figure 69 : Relay Drone 00 MCB Diagram

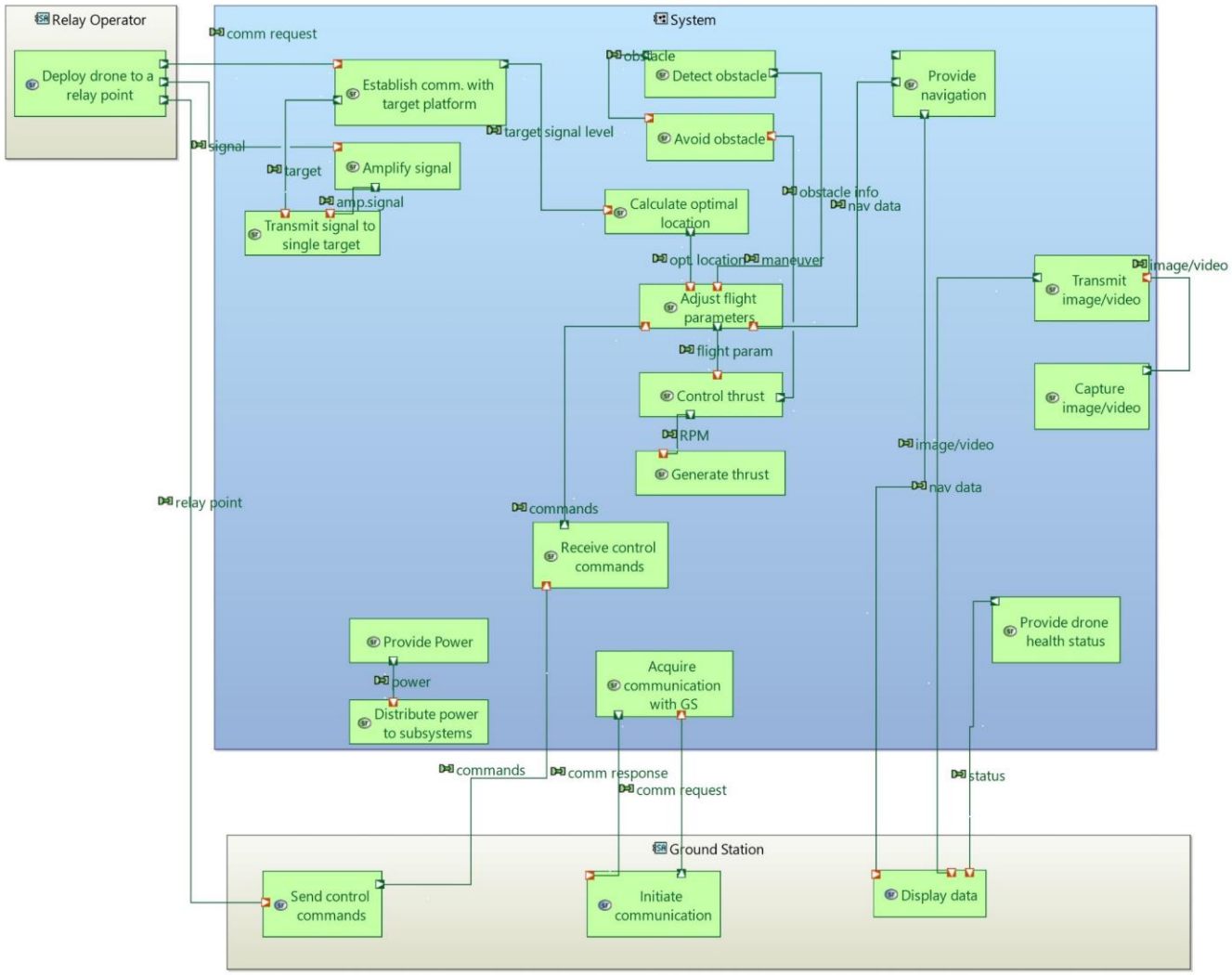

Figure 70 : Relay Drone 00 SAB Diagram

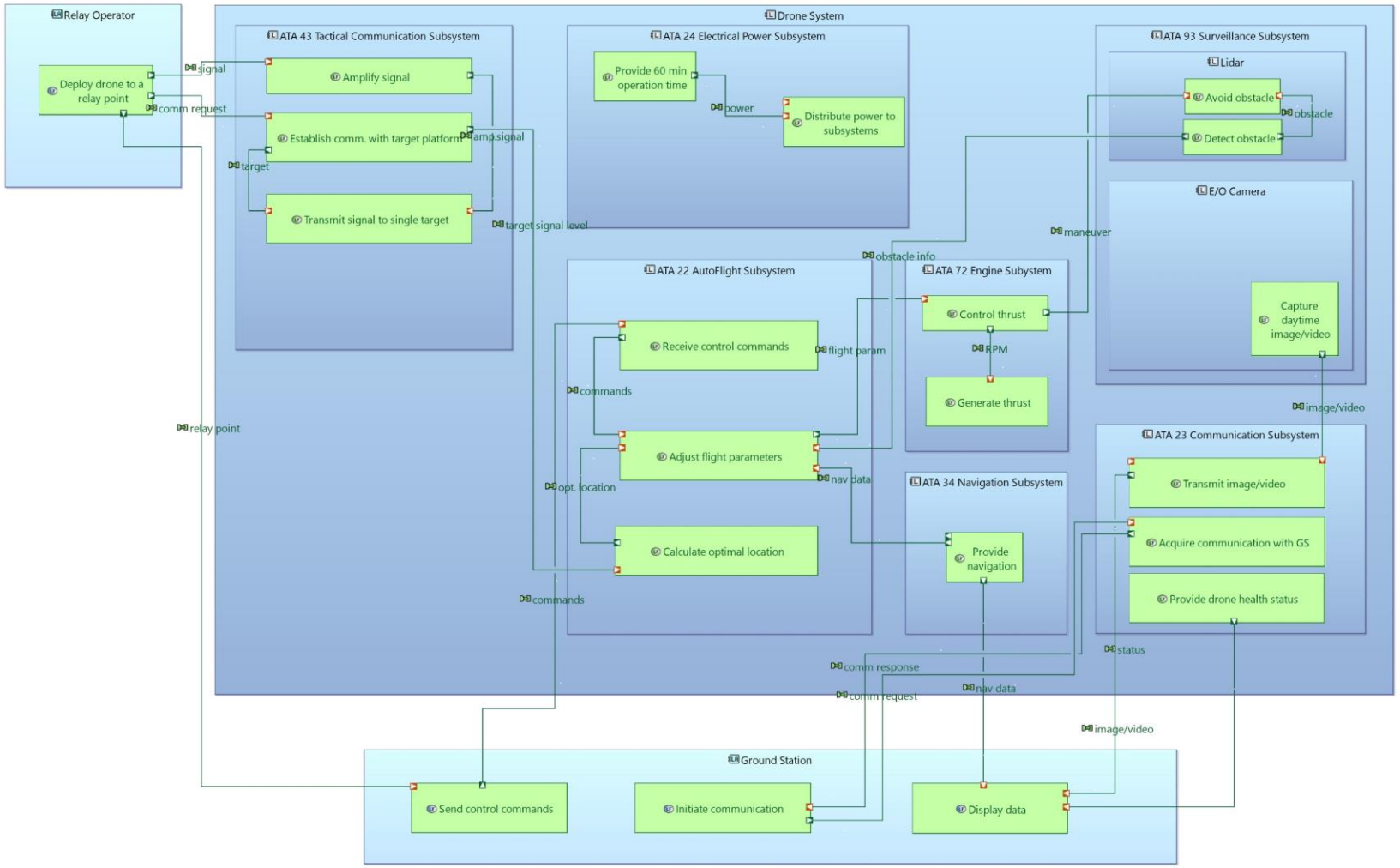

Figure 71 : Relay Drone 00 LAB Diagram

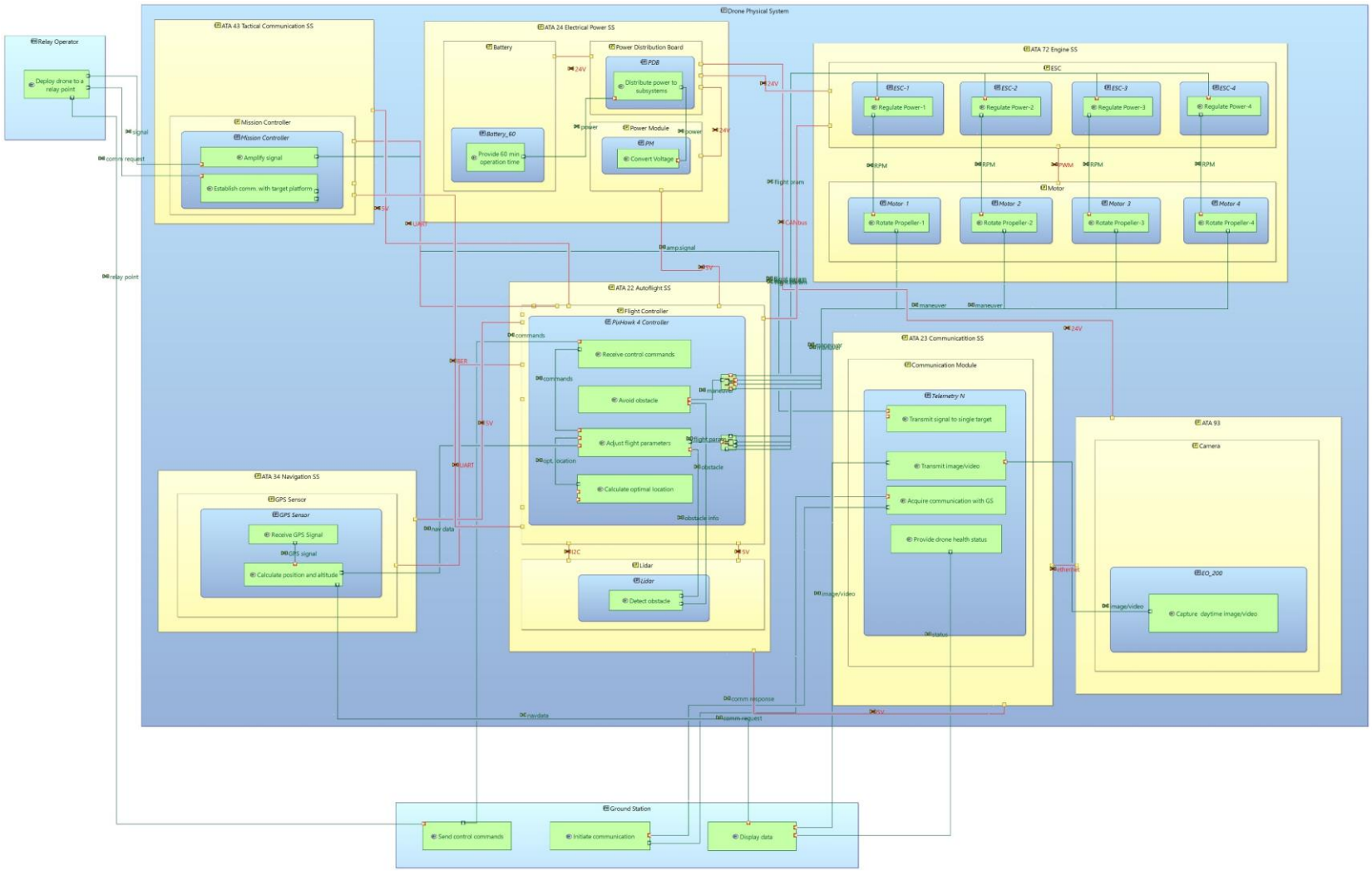

Figure 72 : Relay Drone 00 PAB Diagram

RELAY DRONE 03

Figure 73 : Relay Drone 03 Feature Configuration

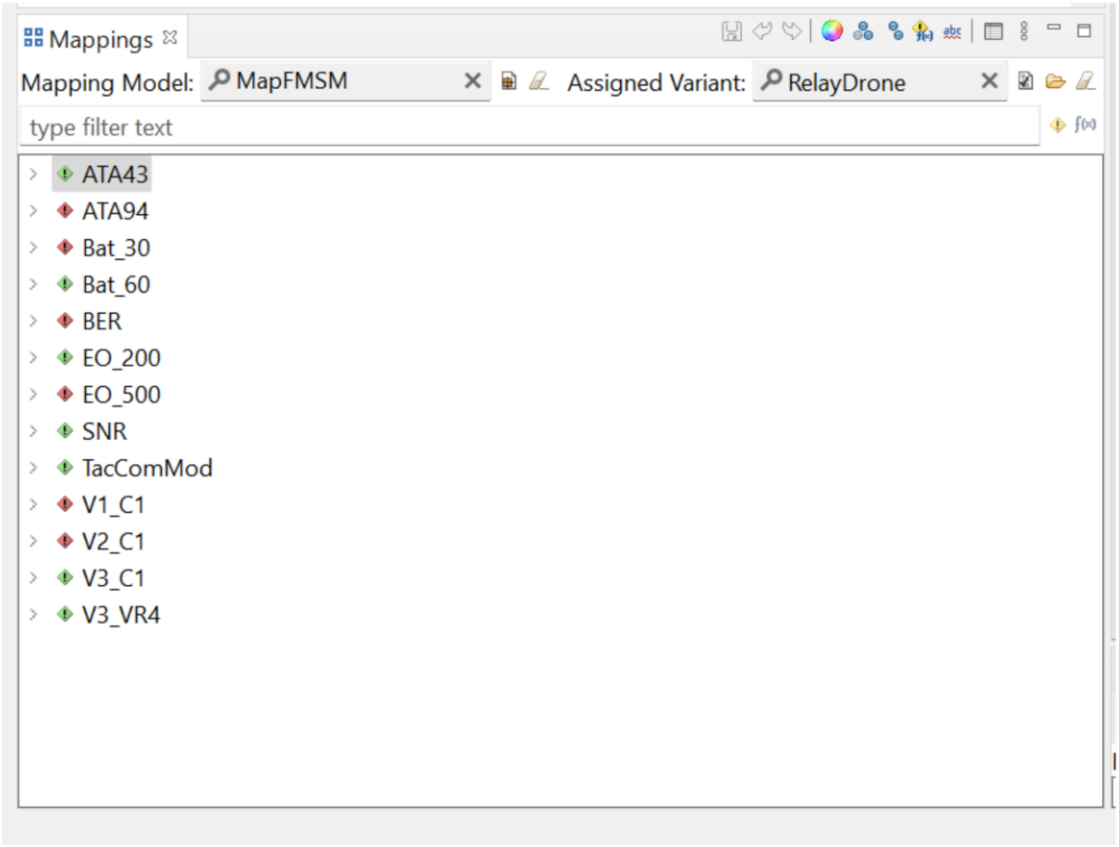

Figure 74: Relay 03 Drone Mapping Model

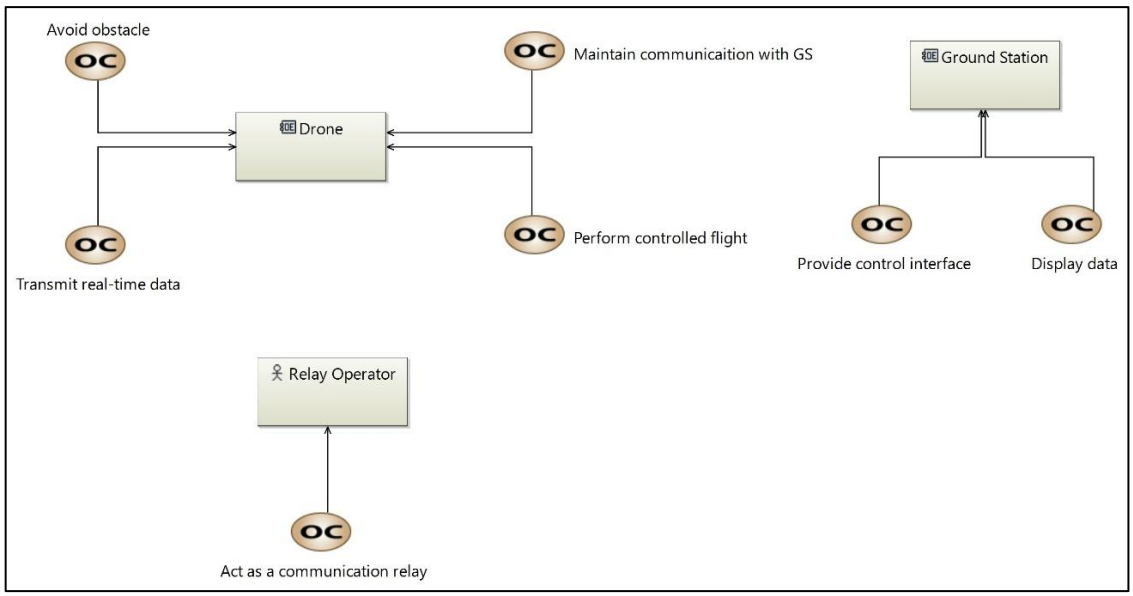

Figure 75 : Relay Drone 03 OCB Diagram

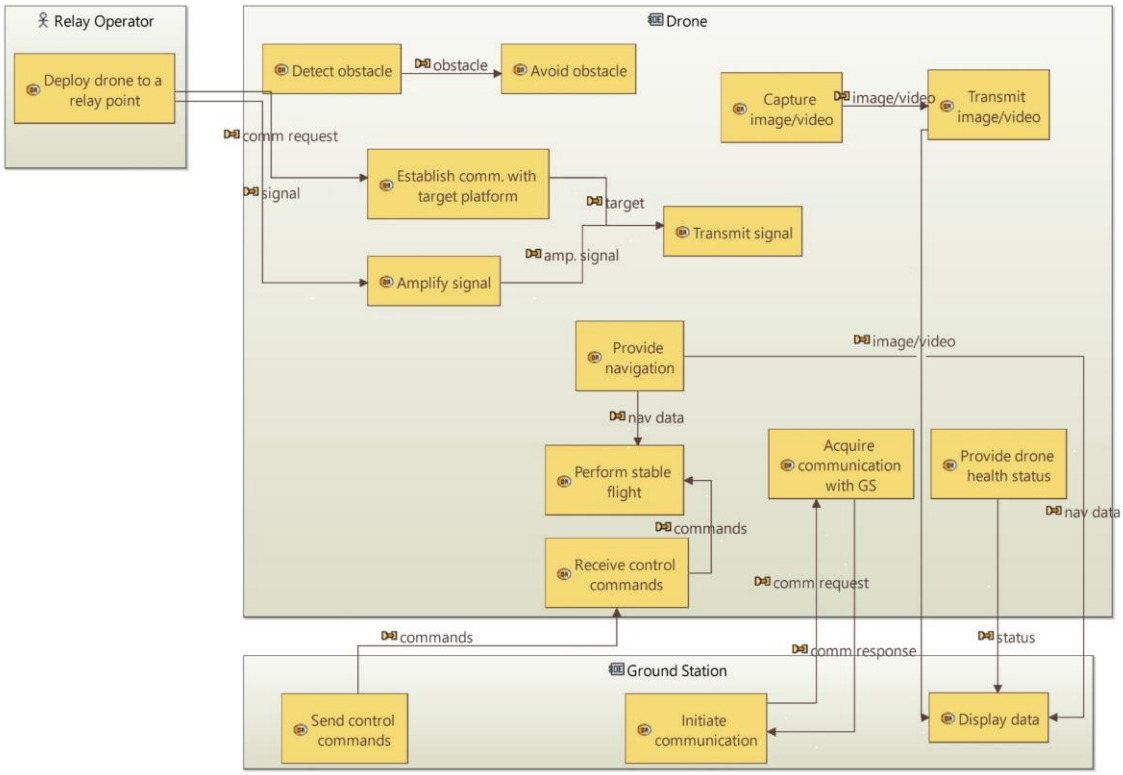

Figure 76 : Relay Drone 03 OAB Diagram

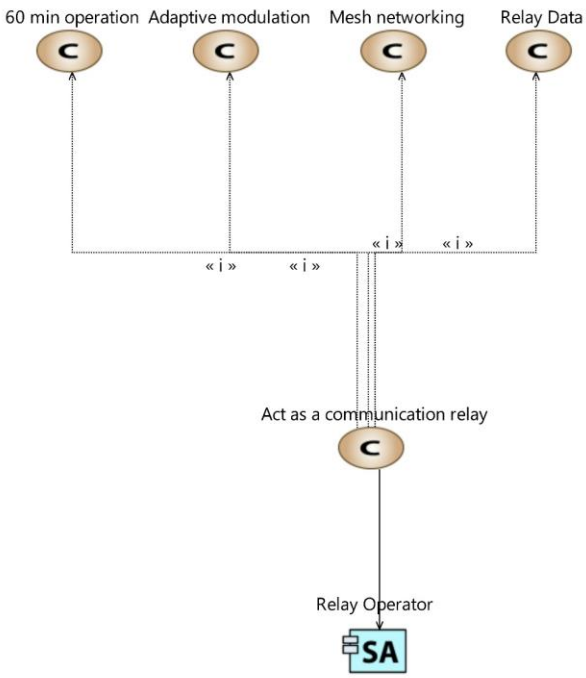

Figure 77 : Relay Drone 03 MCB Diagram

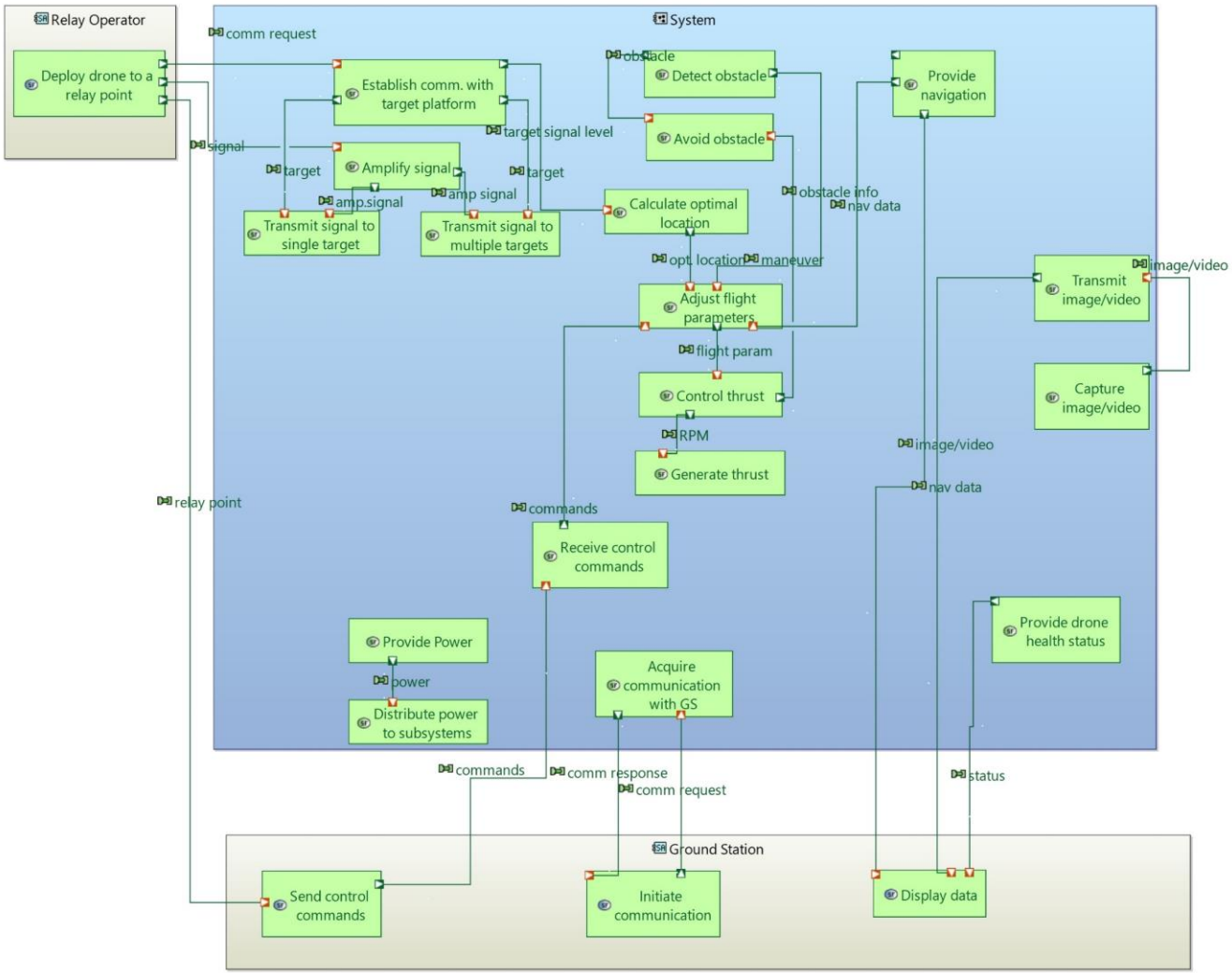

Figure 78 : Relay Drone 03 SAB Diagram

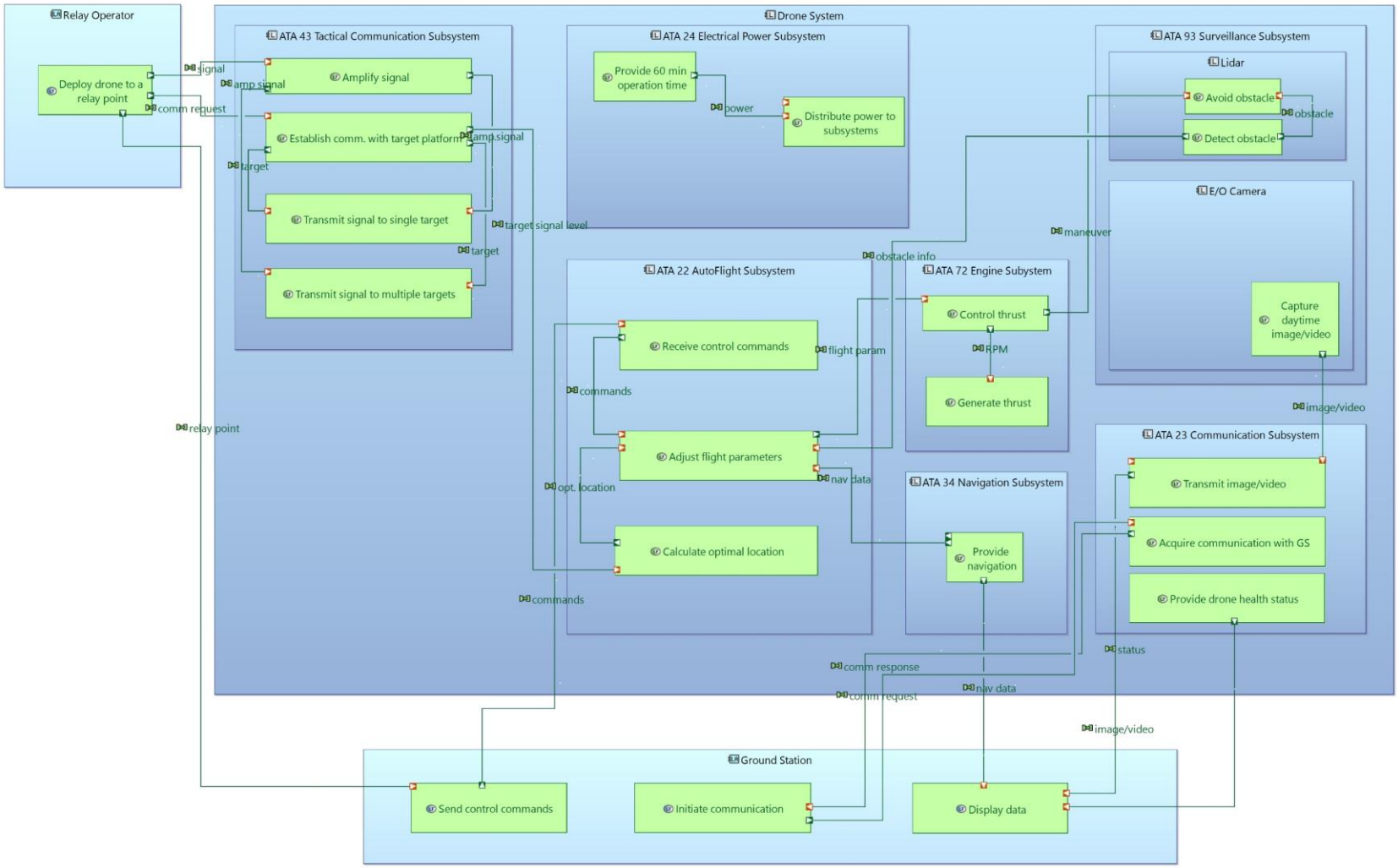

Figure 79 : Relay Drone 03 LAB Diagram

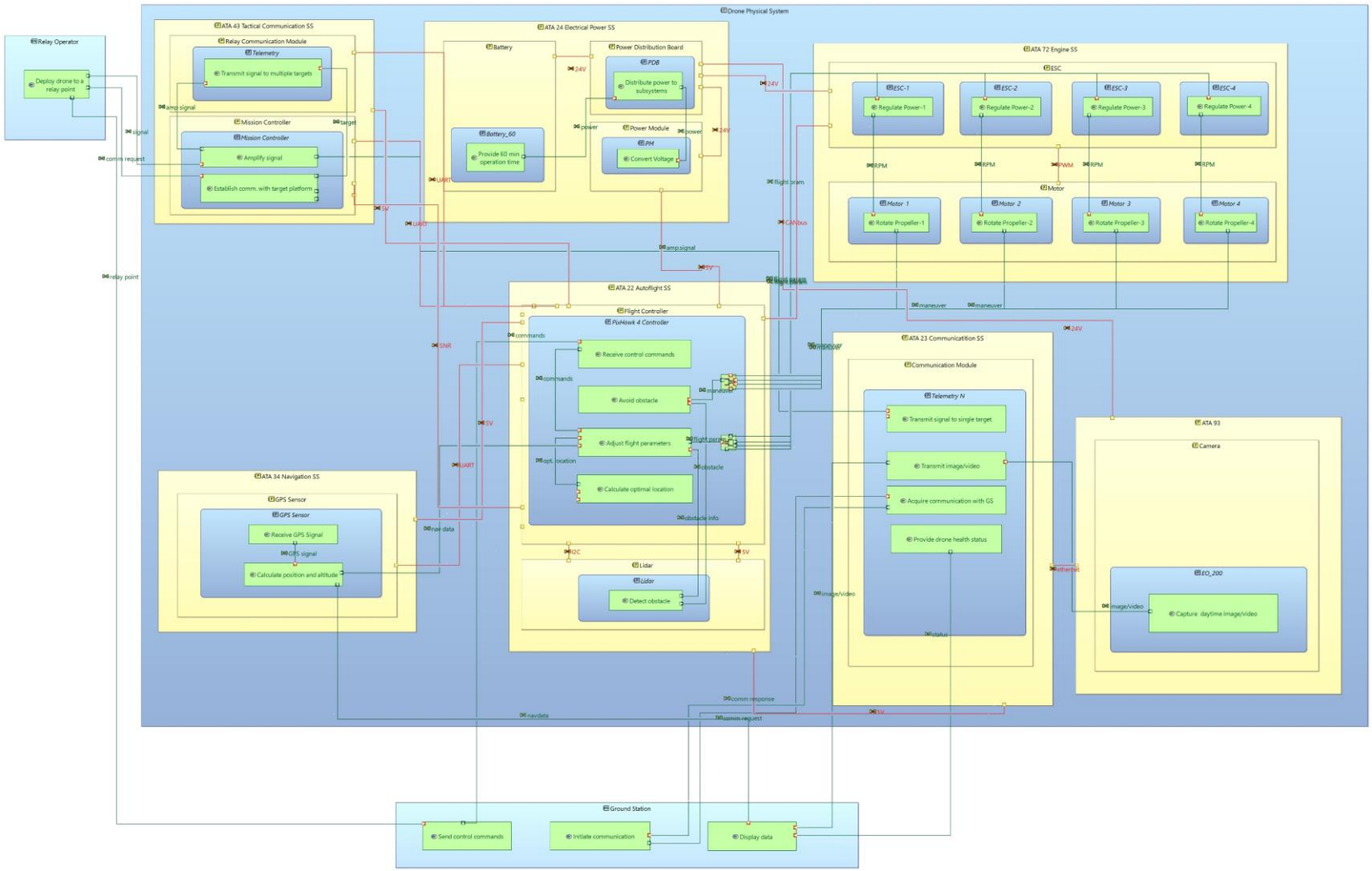

Figure 80 : Relay Drone 03 PAB Diagram