This area must be covered by the glue binding!

TU Clausthal

SYSTEMATIC MODELLING OF SAFETY CRITICAL SYSTEMS USING SYSML V2

MASTER'S THESIS

presented by

PRIYANKA ANTONY

Clausthal University of Technology
Institute for Informatics

-

[Final version, compiled on November 5, 2025 at 21:06]

Priyanka Antony: *Systematic Modelling of Safety Critical Systems Using SysML V2*

STUDENT ID
532455

ASSESSORS
First assessor: apl. Prof. Dr.-Ing. habil. Umut Durak
Second assessor: Prof. Dr. Sven Hartmann.

DATE OF SUBMISSION
November 5, 2025

[Final version, compiled on November 5, 2025 at 21:06]

EIDESSTATTLICHE VERSICHERUNG

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Alle Stellen der Arbeit, die wörtlich oder sinngemäß aus anderen Quellen übernommen wurden, wurden als solche kenntlich gemacht. Die Arbeit wurde in gleicher oder ähnlicher Form noch keiner anderen Prüfungsstelle vorgelegt.

Clausthal-Zellerfeld, den 5. November 2025

Priyanka Antony

PUBLICATION AGREEMENT

Ich erkläre mich mit der öffentlichen Bereitstellung meiner Master's Thesis in der Instituts- und/oder Universitätsbibliothek einverstanden.

Clausthal-Zellerfeld, den 5. November 2025

Priyanka Antony

ABSTRACT

Systematic modelling of safety-critical systems demands a model-centric approach such as Model-Based Systems Engineering (MBSE).A structured framework is provided MBSE for systematic development of complex systems and transition development from traditional document-based approaches to model-based approaches. Within the MBSE framework for systematically modelling safety-critical systems requires modelling languages that can effectively capture and demonstrate complex engineering constraints and emerging trends. The advanced capabilities of the System Modelling Language Version 2 (SysML V2) are a newly developing powerful tool capable of addressing these needs. SysML V2 provides higher precision, improved clarity in system representation, and systematic enhancement in designing; thus, it serves as a key enabler of advanced MBSE practices.

To emphasise the importance of domain-specific modelling, a High Altitude Platform (HAP) is used as a representative case study. The study investigates the systematic modelling of HAP using SysML V2 following the guidelines of the Magic Grid framework. The Magic Grid guidelines help improve modelling efficiency and expressiveness, traceability, and reuse across the system. By scaling SysML V2 within the Magic Grid framework, the study develops a systematic methodology that strengthens communication between stakeholders and supports scalability and reuse. This study examines the new features introduced in SysML V2 as compared to System Modelling Language Version 1 (SysML V1). The thesis focuses on the advancement of future systems engineering practices, with a particular focus on their application in the development of safety-critical systems. In general, the research highlights the potential of SysML V2 to advance MBSE practices for safety-critical systems through transparent and efficient system modelling. These results highlight its potential to guide future system engineering efforts, fostering improved traceability, reusability, and the development of robust, safety-critical architectures.

ZUSAMMENFASSUNG

Die systematische Modellierung sicherheitskritischer Systeme erfordert einen modellzentrierten Ansatz wie das Model-Based Systems Engineering MBSE. MBSE bietet einen strukturierten Rahmen für die systematische Entwicklung komplexer Systeme und unterstützt den Übergang von traditionellen, dokumentenbasierten Ansätzen hin zu modellbasierten Methoden. Innerhalb des MBSE-Rahmens ist für die Modellierung sicherheitskritischer Systeme eine Modellierungssprache notwendig, die komplexe technische Randbedingungen und aktuelle Entwicklungstrends effektiv erfassen und darstellen kann. Die erweiterte System Modelling Language Version 2 SysML V2 stellt dabei ein neu entwickeltes, leistungsfähiges Werkzeug dar, das diesen Anforderungen gerecht wird.SysML V2 bietet eine höhere Präzision, verbesserte Klarheit in der Systemdarstellung und eine systematische Unterstützung im Entwurfsprozess und fungiert somit als zentraler Enabler fortgeschrittener MBSE-Praktiken.

Um die Bedeutung domänenspezifischer Modellierung zu verdeutlichen, wird eine Hochplattform High Altitude PlatformHAP als repräsentative Fallstudie verwendet. Die Studie untersucht die systematische Modellierung einer HAP mit SysML V2 unter Anwendung der Richtlinien des Magic-Grid-Frameworks. Diese Richtlinien tragen zur Verbesserung der Modelleffizienz und Ausdruckskraft sowie zur Nachverfolgbarkeit und Wiederverwendung innerhalb des Systems bei. Durch die Integration von SysML V2 in das Magic Grid Framework wird eine systematische Methodik entwickelt, die die Kommunikation zwischen den Stakeholdern stärkt und Skalierbarkeit sowie Wiederverwendung unterstützt.

Die Arbeit analysiert die neuen Funktionen von SysML V2 im Vergleich zu SysML V1 und konzentriert sich auf die Weiterentwicklung zukünftiger System Engineering Praktiken, insbesondere im Bereich sicherheitskritischer Systeme. Insgesamt zeigt die Forschung das Potenzial von SysML V2 MBSE-Praktiken durch transparente und effiziente Systemmodellierung voranzubringen. Die Ergebnisse unterstreichen das Potenzial der Sprache, zukünftige Systementwicklungsprozesse zu leiten, die Nachverfolgbarkeit und Wiederverwendung zu verbessern und die Entwicklung robuster, sicherheitskritischer Architekturen zu fördern.

ACKNOWLEDGMENTS

I would like to express my sincere gratitude to everyone who has supported and encouraged me throughout the completion of this thesis. First and foremost, my deepest thanks go to **Prof. Dr. Umut Durak**, whose guidance, expertise, and encouragement have been invaluable to my research journey. I am also profoundly grateful to my supervisor, **Alexander Ahlbrecht**, for his practical support, insightful feedback, and continuous assistance during the development of this work. Their mentorship, patience, and constructive advice were instrumental in overcoming numerous challenges along the way. I would also like to extend my appreciation to the developers of the tools and libraries utilised in this thesis, as well as to the **OMG Systems Modelling Community**, for providing a strong foundation that made the development of my work possible. Additionally, the coding guidelines and examples referenced in this thesis were informed by the official SysML V2 libraries and example models provided in the SysML V2 GitHub release.

My heartfelt thanks go to my family and friends for their unwavering support, understanding, and encouragement. Their belief in me has been a constant source of motivation and strength throughout this process. Finally, I would like to acknowledge the use of AI-based tools during the preparation of this thesis. ChatGPT and Overleaf Assistance were used solely for language refinement, structuring non-technical explanations, and occasionally improving clarity and organisation. These tools played only a supplementary role, with all content manually reviewed and edited to ensure accuracy and authenticity.

This thesis stands as a reflection of not only my own efforts but also the collective guidance, support, and inspiration I have received from those around me. For that, I am deeply thankful to each and every one of them.

DANKSAGUNG

Ich möchte allen herzlich danken, die mich während der Anfertigung dieser Arbeit unterstützt und ermutigt haben. Zunächst gilt mein aufrichtiger Dank **Prof. Dr. Umut Durak**, dessen fachliche Anleitung, Expertise und kontinuierliche Unterstützung für den Fortschritt meiner Forschung von unschätzbarem Wert waren. Ebenso danke ich meinem Betreuer **Alexander Ahlbrecht** für seine praktische Unterstützung, seine wertvollen Rückmeldungen und seine stete Hilfsbereitschaft im Verlauf dieser Arbeit. Ihre Geduld, konstruktive Kritik und ihr Engagement haben wesentlich zum Gelingen dieser Arbeit beigetragen.

Mein besonderer Dank gilt zudem den Entwicklern der in dieser Arbeit verwendeten Tools und Bibliotheken sowie der **OMG Systems Modeling Community**, die eine solide Grundlage für die Entwicklung meiner Arbeit geschaffen haben. Die in dieser Arbeit herangezogenen Programmierleitlinien und Beispiele orientieren sich an den offiziellen SysML V2-Bibliotheken und den Beispielmodellen aus der SysML V2-GitHub-Veröffentlichung.

Mein herzlicher Dank gilt außerdem meiner Familie und meinen Freunden für ihre unerschütterliche Unterstützung, ihr Verständnis und ihren Glauben an mich. Ihre Ermutigung und Geduld waren eine ständige Quelle der Motivation und Kraft. Abschließend möchte ich den Einsatz von KI-basierten Werkzeugen bei der Erstellung dieser Arbeit erwähnen. ChatGPT und Overleaf Assistance wurden ausschließlich zur sprachlichen Verfeinerung, zur Strukturierung nicht-technischer Abschnitte sowie zur gelegentlichen Verbesserung der Verständlichkeit und Organisation verwendet. Diese Werkzeuge hatten lediglich eine unterstützende Funktion; sämtliche Inhalte wurden manuell überprüft und hinsichtlich Genauigkeit und Authentizität überarbeitet.Diese Arbeit ist nicht nur das Ergebnis meiner eigenen Bemühungen, sondern auch Ausdruck der wertvollen Unterstützung, des Wissens und der Ermutigung, die ich von vielen Seiten erfahren durfte. Dafür danke ich jedem Einzelnen von Herzen.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

MBSE	Model-Based Systems Engineering
UML	Unified Modeling Language
SysML	System Modelling Language
SysML V1	System Modelling Language Version 1
SysML V2	System Modelling Language Version 2
HAP	High Altitude Platform
MACS	High-resolution Modular Aerial Camera System
HAPSAR	High-Altitude Platform Synthetic Aperture Radar
KerML	Kernel Modelling Language
UAF	Unified Architecture Framework
API	Application Programming Interface

1 INTRODUCTION

1.1 INTRODUCTION

Safety Critical systems such as aerospace platforms, medical devices, and nuclear control systems require high levels of reliability because any mistake will cause high consequences. So, as the complexity of systems grows, it is required to step up from the document-centred approach, as it is no longer sufficient. MBSE [7] has emerged as a systematic approach to address these challenges by representing the requirements, behaviour, and architecture of the system within an integrated modelling environment. Within the MBSE domain, the System Modelling Language (SysML) [12] provides a standardised way to capture and communicate complex system designs. The recent development of SysML V2 [24] introduces a more precise syntax, stronger semantic foundations, and improved tool interoperability. Provides enhanced capabilities for modelling of systems. These features make SysML V2 suitable for projects in which essential systems are handled.

This thesis investigates systematic modelling of a safety-critical aerospace platform, the High Altitude Platform HAP Alpha, developed by the German Aerospace Centre (DLR) [11] using SysML V2. The HAP alpha is a high-altitude, long-endurance system where reliability and fault tolerance are essential. This aircraft operates in the lower stratosphere, which presents a unique set of engineering challenges due to their diverse application in communication with the ground station, remote sensing, and environmental monitoring. To structure the modelling effort, the research uses the Magic Grid framework [17], An MBSE methodology that helps to align stakeholder concerns with system views and ensures systematic coverage of requirements, behaviours, and architecture. This thesis applies SysML V2 within the Magic Grid framework; a disciplined MBSE process can improve safety assurance, design traceability, and verification throughout the entire system lifecycle. Unlike traditional document-centric systems engineering, which relies on disparate documents, informal diagrams, and spreadsheets, MBSE captures system information within a central system model. This model-based approach facilitates a more complete, consistent, and traceable system design [17].For the modelling of complex systems such as HAP, characterised by their architecture, lifecycle, and demanding operational environment, the most reliable universal application is MBSE.

To model the framework for HAP, MBSE provides consistent use of the model throughout the entire system lifecycle and at all levels of design, including the system-of-systems, system, subsystem and component [7].Effective scaling of MBSE to meet the demands of HAP development requires major advancement in core modelling languages. To achieve this demand, this research explores the next generation SysML V2, evaluating capabilities for capturing the complexity of HAP across the full lifecycle and hierarchical design levels. SysML V2 offers advanced capabilities that help to develop a structured and detailed representation of HAP systems, covering functional and non-functional requirements, system architecture, and behaviour [7]. This research looks forward to designing the interface and analysing the HAP, thereby improving the entire system and overall performance. The suggested approach using SysML V2 as a modelling language and the Magic Grid as the guidelines for how to model not only helps in the optimisation of aircraft systems like HAP but also lays the groundwork for future progress in systems engineering.

Magic Grid is introduced to provide a prescriptive modelling methodology. Magic Grid, also known as the MBSE Grid [17], is a structured framework designed to guide the application of MBSE using SysML V2. It was introduced by Morkevicius et al. [17] to provide methodological guidance, consistent abstraction levels, and traceability throughout the development lifecycle. This approach is structured around architectural viewpoints, referred to as 'domains'. There is a problem domain and a solution domain and aspects referred to as pillars, which are arranged in a matrix inspired by the Zachman Framework [17]. Each cell within this matrix defines a specific view specification of the system of interest [17], representing the intersection of a given domain and pillar. Using the Magic Grid, HAP as a case study can be used to analyse how the Magic Grid can be used in a structured way to organise the system model across abstraction levels and viewpoints [17].The Magic Grid organises system architecture development into a two-dimensional matrix. In the matrix, the vertical axis represents abstraction levels from the problem domain, which contains black box and white box, and then the solution domain, which includes design and implementation[17].The horizontal axis consists of the four pillars of SysML V2, such as requirements, structure, behaviour, and parameters [6]. Each cell in the grid consists of a specific view specification, instantiated through standard SysML V2 diagrams such as requirement, activity diagram, and internal system structure [6]. This provides a modular and traceable way to model system behaviour and architecture of the HAP.

MOTIVATION OF RESEARCH The motivation of the research is to inspect the potential of SysML V2 compared to SysML V1 in system engineering. illustrating the SysML V2 model using a particular complex system such as HAP. This research progresses in investigating how SysML V2 addresses various aspects of system design, including functional and non-functional requirements, system structure, behaviour, and performance. While modelling HAP using SysML V2, a perspective modelling methodology, the magic grid is used to get the guidelines for how to model. Magic Grid defines the methods for modelling, structure, and sequence. It enables a consistent and traceable development of system models [17]. The study focuses on the innovative features of SysML V2, such as views, viewpoints, variability, variation, and so on, to determine how effectively it can model, analyse, and verify complex systems [6]. Through this research, SysML V2, the ability to improve model reuse, streamline system integration, and minimise design errors will be evaluated, providing insights into its application in modern engineering challenges. The motivation for this research is to explore how SysML V2, with its advanced modelling capabilities along with the modelling methodology of the magic grid framework, can address the growing complexity of systems engineering in a more effective and precise manner than its predecessor, SysML V1.

RESEARCH QUESTION This research seeks to evaluate how SysML V2 innovations, such as views and viewpoint modelling, variability, individuals, timeslices, snapshots, expressions, calculations, analysis cases, verification cases, element filters, verification and validation of highly integrated systems, can improve the design compared to V1. [6] And mapping these features into the layers of abstraction of the magic grid, thus providing a well-structured framework. As Magic Grid is a framework in a matrix view, "which is designated to guide system engineers through the modelling process and help answer the questions, like how to start and structure the model, what views to build, which artefacts to deliver, and in what sequence." [17].

STRUCTURE OF THE DOCUMENT By exploring these capabilities, the aim of this research is to establish best practices for mapping SysML V2 in Magic Grid systems engineering, highlighting its ability to handle complex system behaviours and interactions more effectively than SysML V1. The study will show how SysML V2 can lead to improved design quality, reduce costly rework, and support strategic decision-making. Ultimately, it will analyse whether SysML V2 is capable of addressing the challenges faced in modern system engineering, setting new industry standards for system modelling and design. The research will proceed through several stages, starting with a literature review and initial setup in section 3, followed by research objectives in section 4, which include methodology, tooling environment, and modelling framework. Next in section 5 is the systematic modelling of HAP, defining and modelling the system following the guidelines of the Magic Grid methodology. Then section 6 includes the modelling constructs specific to SysML V2, such as reuse patterns, analysis cases, and views. Then in section 7 the evaluation of the research work, and then in section 8 conclusions and future work.

2 BACKGROUND

The ultimate goal of system engineering is to achieve the design and analysis of complex systems, which is paramount to achieving optimal performance, efficiency, and reliability. As safety-critical systems, HAP require rigors methodologies in the development phase; accuracy and consistency are highly important. This system should provide long-duration surveillance and communication services from high altitudes. And it offers a cost-effective alternative to satellites for environmental monitoring and disaster management [11]. The HAP has three payloads. This includes advanced systems such as the High-resolution Modular Aerial Camera System (MACS) and High-Altitude Platform Synthetic Aperture Radar (HAPSAR), enabling a broad spectrum of potential applications such as real-time data for applications such as environmental monitoring, disaster response, air pollution monitoring, and maritime surveillance [11]. This offers a flexible and cost-effective alternative to satellites. As HAP is an efficient and safety-critical system, it is chosen for modelling according to the MBSE standards. Using the abilities of SysML V2, the aim is to provide a clear and organised strategy that addresses the complex challenges posed by HAP frameworks [11]. The concept of MBSE has become a new approach to help with the challenges involved in designing complicated systems.

2.1 SCALING MBSE THROUGH SYSML V2

According to Bone and Cloutier [3], MBSE aims to address challenges by using models to abstract from concrete realities. The benefits of MBSE will grow as the practice matures. This involves using modelling patterns, reference architectures, and reuse libraries to improve product-family engineering and system design evolution [7].MBSE is used in HAP development to model flight dynamics, power systems, and communication payloads early in the design phase, which will reduce errors and improve integration across subsystems. System models are expected to improve thinking and answer fundamental issues like "What is the impact of a change to a certain requirement?" [7]. The models aim to increase automation for system quality checks and status reporting. The models aim to promote shared understanding between system stakeholders and improve the ability to manage complexity and risk. To fully realise the benefits of MBSE, it must be implemented throughout the life cycle of the system, from the initial stage through development, production and maintenance at every level of the system, individual systems, subsystems and components [7]. In the development of HAP, MBSE makes it possible to check system requirements and find design problems before making physical prototypes. This saves money, reduces technical risks, and prevents the schedule from being delayed. Using MBSE throughout the entire HAP lifecycle, from idea to operation, makes it easy to ensure that the systems, subsystems, and payloads are all the same. Better modelling approaches also make things more efficient and creative[20]. To deal with the complexity of HAP systems, growing MBSE requires the use of next-generation modelling languages such as SysML V2, which offers more accuracy, expressiveness, and integration throughout its lifetime.

2.2 WHY SYSML V2

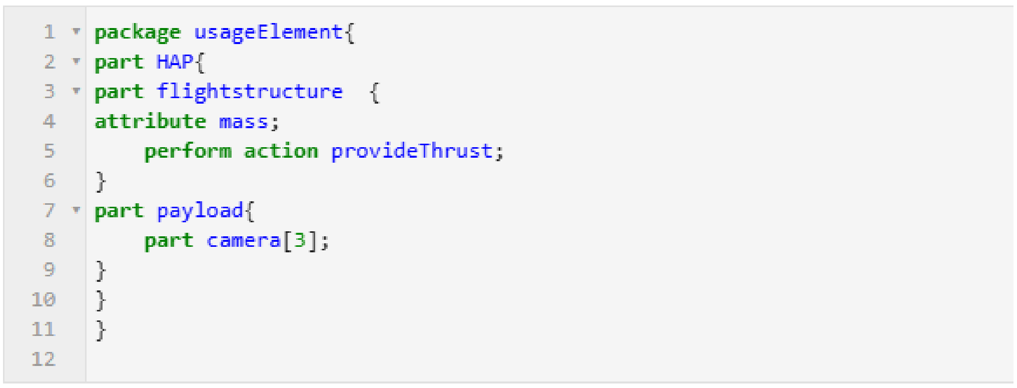
SysML V2 is an effective and flexible modelling language. It can capture complex system information and also provides a standard way to communicate and analyse it. SysML V2 is the next version of the Systems Modelling Language developed by the Object Management Group (OMG) [24]. It is designed to support MBSE and is expected to become an international standard. SysML V2 is the next-generation Systems Modelling Language, offering greater precision and expressiveness to define system requirements and behaviour. It ensures consistency across concepts, improves tool interoperability through standard interfaces, and supports general and domain-specific modelling with a user-friendly, flexible design [20]. SysML V2 for HAPs is the best fit because HAPs are complex, multi-domain systems that involve aerodynamics, communications, energy systems, and autonomous operation—all of which require precise, integrated modelling across subsystems and lifecycle stages. HAPs demand exact modelling of requirements and behaviours; SysML V2 offers greater precision than SysML V1. HAP has several multiple subsystems, such as payloads, energy power systems having solar power and battery power, flight control, communication with the ground station, and so on. The application of SysML V2 helps maintain internal consistency and traceability. SysML V2 supports large-scale modelling while remaining user-friendly for engineers. SysML V2 supports modelling throughout this entire lifecycle, enabling smoother transitions and better verification. SysML V2 introduces both graphical and textual notation to enhance the modelling experience and improve the precision of the system design [7].

While graphical notation remains essential for visualising system structure and behaviour—facilitating communication among multidisciplinary teams—the new textual notation offers a formally defined, machine-readable syntax that significantly improves model precision, maintainability, and tool integration. The textual format also supports automation, scripting, and quick edits, which are often difficult to achieve in purely graphical models. In addition, it enables efficient model versioning and seamless integration with software development tools. The dual notation system ensures that engineers can switch between visual and textual views without information loss, combining intuitive visualisation with rigors specification. [20] These capabilities make SysML V2 particularly effective for modelling complex systems such as HAP, where detailed design, traceability, and cross-domain collaboration are critical. The textual and graphical notation of SysML V2 can be represented using Figure Fig. 2.1, as shown below. The figure shows the textual notation in the first image, one in which SysML V2 modelling language elements such as 'part' and 'action performed by the part' are used, which will be explained in the following chapters in this thesis. The graphical notation presents the HAP alpha system, which helps to show the HAP system having parts such as the flight structure and payload, and the action performed here is providing thrust. Whereas the payload here is shown with a subpart as a camera, this is the basic SysML V2 diagram that is shown. Here, the advantage of using SysML V2 is that editing the code directly for corrections is comparatively faster and easier than editing the diagram using tools. SysML V2 offers significantly improved capabilities for system integration compared to its predecessor, SysML V1. Although SysML V1 was built on the Unified Modeling Language (UML), SysML V2 is based on the more modern and robust Kernel Modelling Language (KerML) [7], which provides a precise and flexible foundation for modelling complex systems.

2.2.1 *What is KerML?*

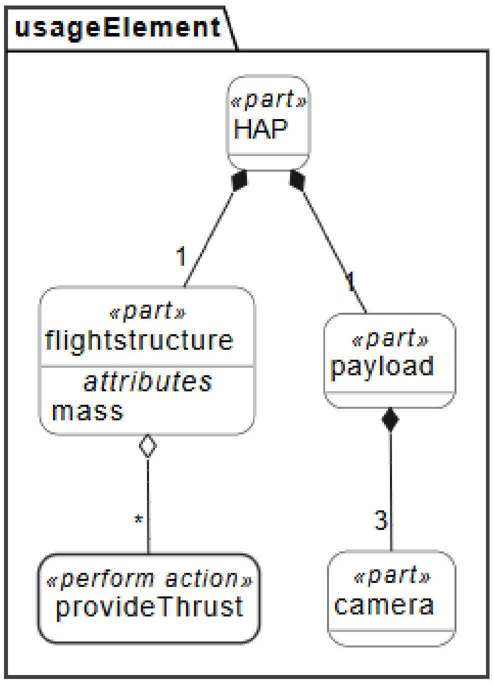
KerML is the SysML V2 kernel and is developed by the SysML V2 Submission Team [24], is known as the kernel modelling language and is the foundational language intended to define other modelling languages. The layered metamodel architecture consists of three KerML metamodel layers, and the top fourth layer is the SysML V2 layer [**24**]. Each layer has increasing semantic richness and modelling capability. The root layer is the base layer that defines the most generic syntactic elements, such as entities, relationships, annotations, and the organisation of elements into namespaces. It provides the essential structure upon which more specific modelling constructs are built [**24**].The core layer of the kernel introduces general semantic constructs grounded in classification theory. This layer defines concepts such as types, features, and classifiers, which form the basis for how elements behave and relate to each other.

At the top of the hierarchy, the kernel layer adds reusable modelling capabilities required for the modelling of complex systems. It includes constructs for classes, data types, associations, connectors, behavioural elements such as interactions and actions, functions, parameter expressions, metadata, multiplicities, and structural organisations through packages [20]. Together, these layers form the foundation for SysML V2, as this layer is expressive, flexible, reusable, and extensible for the different types of domains and systems in engineering tasks. This layer adopts a usage-orientated approach and supports both textual and graphical notation. SysML V2 is the fourth meta-layer of the architecture, where the first three layers are the KerML layers. The fourth layer contains a set of domain libraries [**24**]. SysML V1 was known as the language built on top of the UML, whereas SysML V2 is defined as a metamodel built on top of the kernel metamodel, KerML [20].The model can be shown using Figure Fig. 2.2. The layered architecture built on KerML allows for clean separation between core modelling elements



```
package usageElement{
part HAP{
part flightstructure  {
attribute mass;
    perform action provideThrust;
}
part payload{
    part camera[3];
}
}
}
```

((a)) Textual Notation



((b)) Graphical Notation

Figure 2.1: Textual and Graphical Notations of SysML V2

and domain-specific concepts, offering a more modular, scalable, and future-proof modelling environment [7].Together these features of SysML V2 make it a more expressive and powerful language for addressing the growing complexity of systems engineering[20], particularly for advanced applications such as HAP, where precise modelling and cross-domain integration are critical.

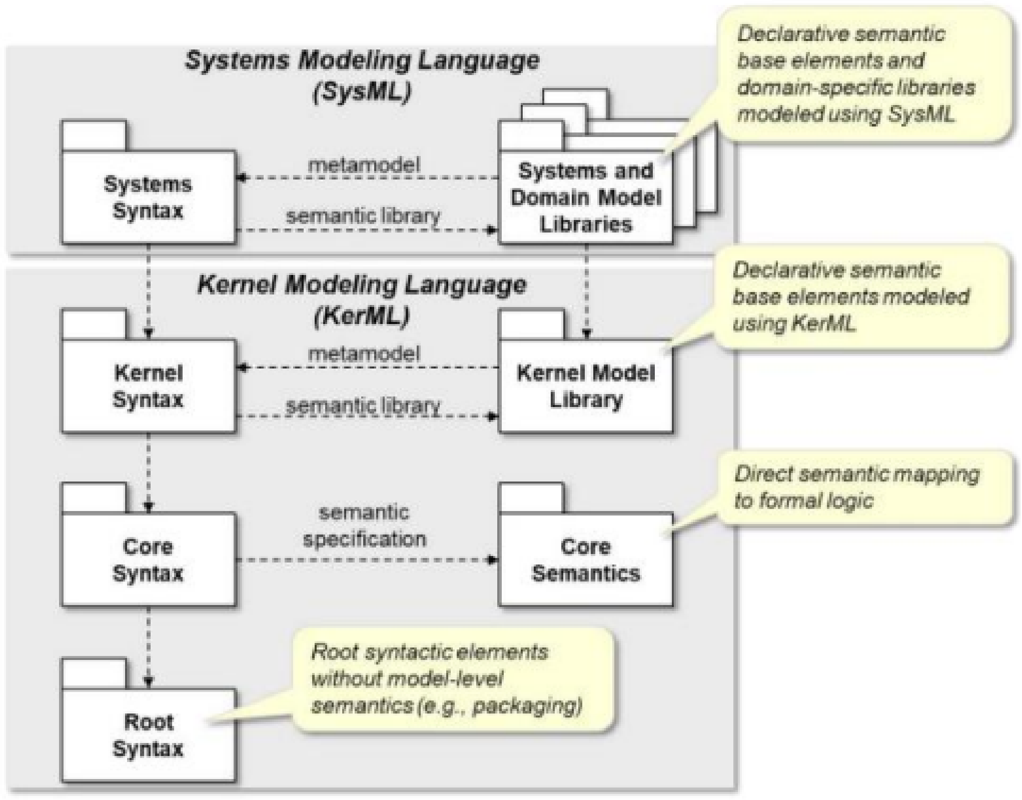

Systems Modeling Language (SysML)
Systems Syntax
metamodel
semantic library
Systems and Domain Model Libraries
Declarative semantic base elements and domain-specific libraries modeled using SysML
Kernel Modeling Language (KerML)
Kernel Syntax
metamodel
semantic library
Kernel Model Library
Declarative semantic base elements modeled using KerML
Core Syntax
semantic specification
Core Semantics
Direct semantic mapping to formal logic
Root Syntax
Root syntactic elements without model-level semantics (e.g., packaging)

Figure 2.2: Overview of KerML (taken from [20])

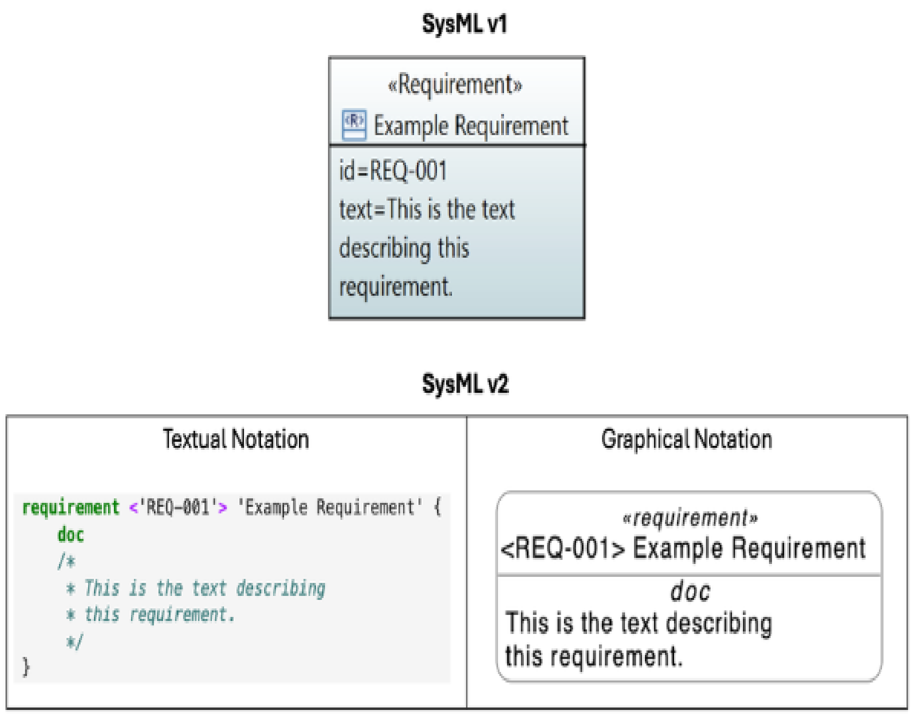
2.2.2 *Comparing SysML V2 and SysML V1*

SysML V2 is an improved version of SysML V1. Addresses many limitations of the former more effectively with modern MBSE practices. The main advanced enhancement is made for parametric modelling; modelling with textual notation and graphical notation is a major advancement. Reuse is one of the main advantages of SysML V2, as an element is defined once and reused and redefined in different scenarios. SysML V1 informally introduces the concept of definition and usage, for example, block and part property [20]. As in SysML V2, the definition and usage of elements are formal and part of the language. The application is virtually applicable to all elements in SysML V2, such as attributes, parts, ports, connections, actions, states, requirements, constraints, cases, and views. SysML V2 supports a consistent pattern of decomposition and specialisation.

SysML V2 handles variability, which was previously cumbersome or inconsistently implemented in V1 [20].SysML V2 also introduces usage and definition elements as first-class concepts, allowing a clearer separation between the specification of a system and its implementation. This distinction supports improved traceability and reusability across system models. Another key advancement is usage-focused modelling [5], which simplifies system hierarchies by allowing direct decomposition of parts, actions, and requirements, reducing redundancy and enhancing modularity. Additionally, constraint modelling is significantly refined, with better support for mathematical expressions, unit management, and consistency checking—allowing for more accurate representation of engineering rules and system behaviour [20].SysML V2 also enhances interoperability through standardised semantics and a robust Part Interconnection framework, addressing the tool-specific inconsistencies that often plagued SysML V1.

Furthermore, SysML V2 benefits from effective automation as compared to SysML V1, along with effective reuse, and facilitates learning and using the language. The concepts in SysML V2 are designed in the metamodel. SysML V2 has the ability to decompose into parts, actions, items, etc., compared to SysML V1 [20]. The language now formally incorporates analysis cases and verification cases, streamlining validation workflows and improving support for systems engineering processes such as verification and validation [20].SysML V1 focused on graphical representation, whereas SysML V2 offers balanced integration of textual and graphical notation. This helps to significantly improve usability for complex, large-scale models. The textual syntax not only enables scripting, automation, and rapid editing but also facilitates better version control and collaboration using software development tools[5].Additional improvements include enhanced support for metadata and extensibility, enabling the use of custom annotations and domain-specific extensions without compromising model consistency. While SysML provides the language constructs for system modelling, it does not define how to systematically organise and apply those constructs across the system lifecycle.

SysML V2 is not just an evolution from SysML V1, but it fulfils the demands of modern, multidisciplinary, and collaborative engineering environments [23]. The figure Fig. 2.3 shows the requirement diagram in SysML V1 and SysML V2. In SysML V2, the requirement diagram can be implemented using textual notation, and the representation can be shown using graphical notation. These improvements make SysML V2 more than an evolution of SysML V1 but an extensive redesign aimed at addressing the demands of modern, multidisciplinary, and collaborative engineering environments. Making SysML V2 lay a stronger foundation for implementing scalable and maintainable workflow across the system development lifecycle.


SysML v1
«Requirement»
Example Requirement
id=REQ-001
text=This is the text describing this requirement.
SysML v2
Textual Notation
Graphical Notation
«requirement»
<REQ-001> Example Requirement
doc
This is the text describing this requirement.

```
requirement <'REQ-001'> 'Example Requirement' {
    doc
    /*
     * This is the text describing
     * this requirement.
     */
}
```

Figure 2.3: The requirement diagram presented in SysML V1 and SysML V2 (taken from [26])

2.3 MAGIC GRID

The Magic Grid is a prescriptive modelling framework developed to support the effective application of MBSE using the SysML language. Introduced by Morkevicius et al. [17], it addresses a significant gap in MBSE practice: The framework is organised as a two-dimensional grid structure, which systematically aligns the engineering concerns of the system with the artefacts of the modelling, as illustrated in Figure Fig. 2.4. According to Morkevicius et al.[17],The MagicGrid framework organises the modelling of the system into a two-dimensional grid, where the rows represent abstraction layers or domains—namely problem, solution and implementation. These domains are essential for complex systems, engineering as they help manage different perspectives throughout the development lifecycle.

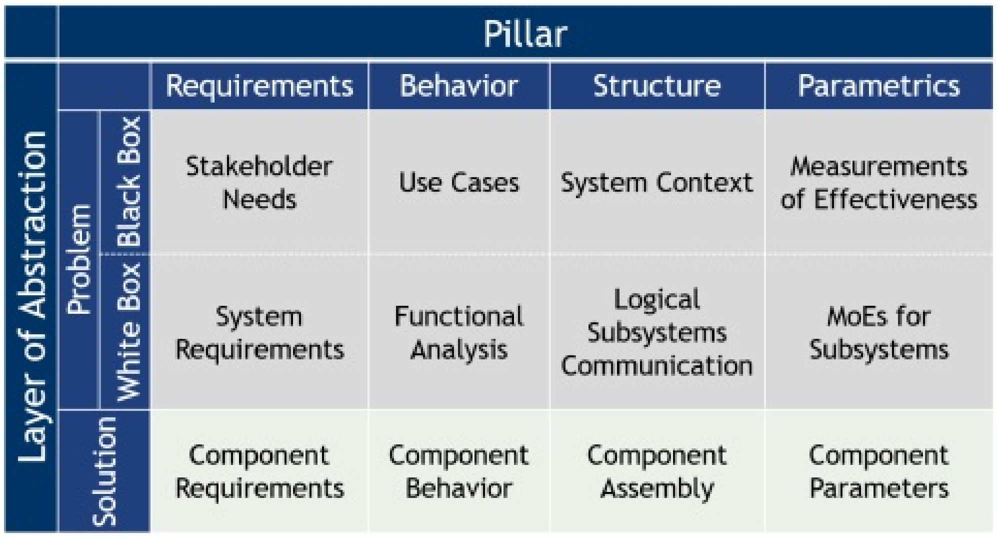
The columns correspond to key aspects of SysML models, often referred to as the four pillars of SysML: requirements, structure, behaviour, and parametrics, as initially defined by Friedenthal et al. [7]. A fifth pillar, safety and reliability, has been added more recently to address growing needs in critical system domains. Each cell in the grid—formed by the intersection of a domain known as the row and an aspect known as the column—defines a specific view, guiding which SysML diagrams and modelling elements should be used to represent relevant system information [17]. The modelling workflow prescribed by Magic Grid determines the order in which these cells are addressed. This workflow aligns with best practices in systems engineering and is in accordance with the technical processes outlined in [18], ensuring a structured and standards-compliant development approach.

Introducing the Magic Grid along with the advent of SysML V2 makes it more efficient and valuable. SysML V2 [20] introduces significant advancements in modelling semantics, notation, modularity, and tool interoperability. These improvements include improved expressiveness, precise usage relationships, improved support for reuse, and integrated views in structural and behavioural aspects. Using Magic Grid along with SysML V2 complements the process by offering a methodological foundation that guides how the language's powerful capabilities should be applied throughout system development. In practice, this means that system engineers can better manage complexity, ensure consistency, and trace system requirements from high-level concepts to low-level implementations[20].This thesis tries to illustrate this idea by applying the magic grid methodology along with the powerful SysML V2 language in HAP. HAP [11] are complex solar-powered aircraft that operate in the stratosphere for extended periods.

Their multidisciplinary nature—encompassing aerodynamics, propulsion, communication systems, energy management, and autonomous control—makes them ideal candidates for MBSE. Due to this complexity, HAP serves as a compelling example to demonstrate how modern MBSE methodologies, such as the Magic Grid framework and SysML V2 modelling language, can be applied effectively. Integration of these tools supports a structured, domain-driven modelling approach, aligning system abstractions such as the problem domain and the solution domain with key modelling aspects such as requirements, structure, behaviour and parametrics [20]. This ensures traceability, clarity, and consistency across the development lifecycle and is especially valuable for systems like HAP where cross-domain integration is critical.

2.4 HAP

Combining MBSE methodologies with the enhanced modelling capabilities of SysML V2 and the domain-driven structure of the MagicGrid framework provides a powerful, integrated approach for managing complex system development. This thesis tries to explore the practical application of these three approaches in tandem through a representative example: HAP. HAP [11] are solar-powered unmanned aerial systems designed to operate in the stratosphere—typically around 20 kilometres above sea level. Developed by institutions such as the German Aerospace Centre (DLR), HAP combine the persistent coverage capabilities of satellites with the flexibility and serviceability of conventional aircraft [11]. The DLR HAP alpha has an ultra-lightweight airframe structure design with a 27-metre wingspan and a total weight of 130 kg. The HAP is powered by solar energy and



Layer of Abstraction		Requirements	Behavior	Structure	Parametrics
Problem	Black Box	Stakeholder Needs	Use Cases	System Context	Measurements of Effectiveness
Problem	White Box	System Requirements	Functional Analysis	Logical Subsystems Communication	MoEs for Subsystems
Solution		Component Requirements	Component Behavior	Component Assembly	Component Parameters

Figure 2.4: MBSE Magic Grid Framework (taken from [17])

battery power. This made for a long-duration mission, but the initial one will be two hours long.

HAP is a great solution for reusable and cost-effective solutions for applications such as Earth observation, environmental monitoring, telecommunications, and disaster response [11]. HAP are equipped with advanced payloads such as high-resolution optical systems and synthetic aperture radar, enabling precise and versatile data collection. These platforms maintain continuous communication with ground stations for real-time data transmission, system control, and mission updates, ensuring reliable and responsive operations [11]. The image of HAP alpha is shown in Figure Fig. 2.5. Their multidisciplinary nature—spanning aerospace engineering, solar energy systems, avionics, communications, and autonomous control—makes them an ideal candidate for system-level modelling and integration. This complexity underscores the relevance of MBSE techniques and justifies the use of SysML V2 and MagicGrid to ensure clear traceability, architectural consistency, and cross-domain collaboration in the development process [11].

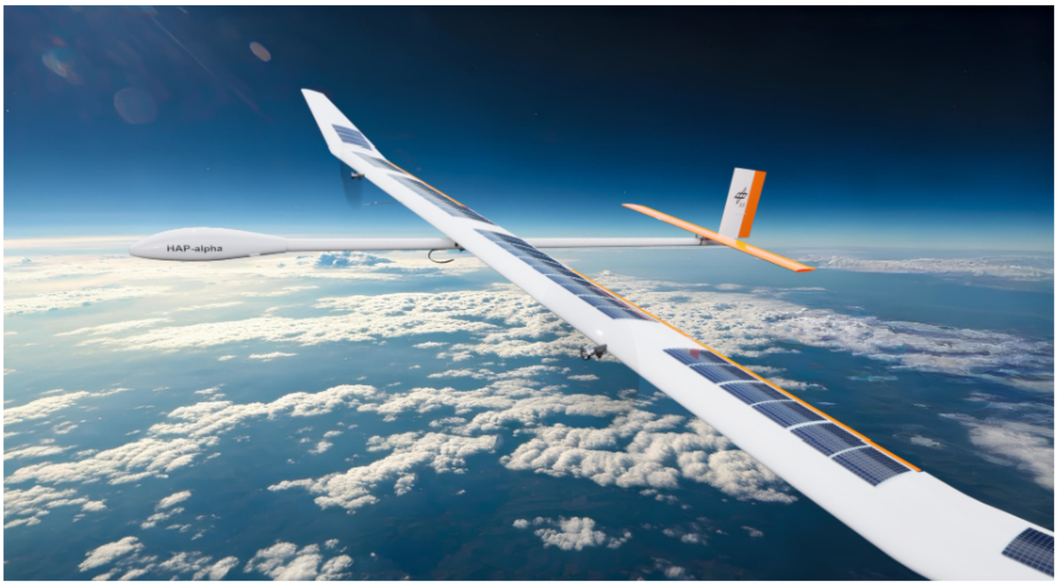


Figure 2.5: High Altitude Platform-alpha Credit: DLR (CC BY-NC-ND 3.0)

3 LITERATURE REVIEW

Since the primary goal of this paper is to highlight the possibilities of model-based development and integration of HAP systems, the literature review focuses on related methodologies and frameworks.SysML V2 emerges as a promising modelling language to support the model-based development of HAP systems, offering enhanced precision and expressiveness. The Magic Grid methodology is used as the modelling framework. This section tries to explore MBSE approaches relevant to the design, integration, and analysis of complex systems such as HAP. MBSE has become a fundamental methodology for addressing the intricacies of contemporary systems, especially in safety-critical sectors like aircraft, automotive, and healthcare. One of the main goals of MBSE is to utilise the models to check that the system meets its criteria and find problems early in the system life cycle. creating and testing the system. This greatly lowers the negative effects on programme cost, schedule, and danger of finding these problems later in the system life [7]. Sanford Friedenthal [7] explains why and how SysML V2 was made to better assist MBSE. It fixes some of the biggest problems of SysML V1, like the absence of formal semantics, the fact that tools don't work together well, and the fact that modelling complicated systems isn't very efficient. SysML V2 adds a formal metamodel, a new textual syntax, a standard API, and better support for usage-based modelling and customisable visualisations. These changes are meant to make modelling more accurate, scalable, and integrated, especially for large-scale and safety-critical systems [7].

MBSE must be applied consistently at all levels of abstraction, including system-of-systems, systems, subsystems, and components, as well as throughout the whole system lifecycle, from conceptual design to development, production, and operational support, in order to fully reap its benefits. Significant improvements in modelling capabilities are required to reach this degree of scalability, including the creation of the corresponding modelling knowledge and the growth of modelling languages, processes, and tools [7]. While modelling HAP end-to-end, multi-level MBSE enables scalable, knowledge-driven development—accelerating design, enhancing quality, and reducing lifecycle costs from systems-of-systems to individual components. Bone and Cloutier [3] present an overview of the adoption and challenges of MBSE based on responses to the OMG SysML. The study had taken a wide range of input from stakeholders, including tool vendors, practitioners, and academia, to assess the maturity of SysML and MBSE practices. The result shows that MBSE was in the early stages of adoption, and organisations experimented with SysML to support system architecture, requirements, traceability, and model-driven communication. Respondents highlighted key benefits such as improved system understanding, better integration between disciplines, and potential lifecycle cost savings.

In 2006, the Object Management Group (OMG) adopted SysML V1 [20], [12] as a standardised graphical modelling language for specifying, analysing, designing, and verifying complex systems that may encompass hardware, software, information, personnel, procedures, and facilities. SysML V1 introduced a suite of diagram types that correspond to different aspects of system modelling. These are often referred to as the "four pillars" of SysML: requirements, behaviour, structure, and parametrics. Each pillar plays a vital role in supporting the comprehensive development and analysis of systems across their lifecycle [12].SysML V1 was developed to address the particular requirements of systems engineers by extending and changing features of the unified modelling language UML [12], [22]. In contrast to UML [22], which focuses on software, SysML accommodates a wider array of systems engineering issues, encompassing the modelling of physical components, requirements traceability, and performance limitations. Friedenthal et al. [10] assert that SysML markedly enhances communication among stakeholders by offering a unified, standardised language that connects various technical specialities. The language modular diagram set facilitates scalable modelling of systems, ranging from high-level conceptual concepts to intricate technical implementations. Consequently, SysML V1 [10] has been extensively utilised in sectors such as aerospace, military, and automotive, where the management of system complexity and the assurance of design are crucial. According to Holt and Perry [13],SysML V1 models consist of two components: structural and behavioural. Both components must be clearly delineated and constantly adhered to throughout the system development process. SysML V1 has nine diagrams, five for structural modelling and four for behavioural modelling. The structural diagram consists of a block definition diagram, which shows the interaction and classification of the system, followed by an internal block diagram, which shows interaction among components. The next is a parametric diagram used for analysis and verification, then the requirement diagram for collecting and showing requirements from the stakeholder, and the package diagram. And the behaviour component consists of four diagrams: a use case diagram for functional context, a sequence diagram for modelling interactions, an activity diagram, and a state machine diagram to illustrate temporal changes. [10], [13] Diagrams collectively constitute a comprehensive modelling framework for describing, creating, analysing, and testing complex systems.

SysML requirement diagrams and parametric modelling features are particularly beneficial for guaranteeing that design decisions are both motivated by needs and informed by performance. In SysML V1, relationships are a key part of connecting different parts of a system model to make sure that it is consistent, traceable, and full. Holt and Perry [13] say that SysML has different types of relationships, such as associations, generalisations, dependencies, and containment relationships [10]. Each type of relationship is used for a different modelling reason. Association relationships show the structural connections between elements, usually by showing how blocks like system parts are linked or joined. Inheritance or specialisation is shown by generalisations. A parent-child bond is described by a generalisation. Dependencies stand for showing how one part of the model affects the other. Dependencies are used when one thing might change something else. The model is organised by containment, which shows links between whole parts or groups. It shows that one block is made up of other blocks. [13], [10] Nonetheless, despite its benefits, SysML V1 has encountered criticism for its complexity, significant learning curve, and constraints in articulating certain advanced systems engineering concepts, leading to the creation of SysML V2 to rectify these deficiencies. [13].

SYSML V2 introduces better improvements as compared to SysML V1 in the concept of modelling systems, making the process of modelling simpler, more precise, expressive, extensible, and interoperable. It is designed to be easier to learn and use, with engineering concepts inherently built into the metamodel rather than added as an extension. [9],[25] . A metamodel [9], [25] is essentially a modelling language. It states the rules, concepts, structure, and relationships of the SysML V2 language. For modelling an HAP system using SysML V2, the metamodel describes how such a model can be built. What are the elements used, such as part, part definition, port, port definition, action, item, and item definition, that are the elements in SysML V2 [9]. And how these elements interact with each other. SysML V2 enforces a consistent pattern of definition and usage, along with clearer and more uniform terminology [25]. This model supports the decomposition of parts and actions and offers greater flexibility in organising models using package filters. The SysML V2 language is more precise, featuring a textual syntax, a dedicated expression language, and formal semantics—enabling requirements to be treated as constraints. [2] ,[9], [25] In terms of expressiveness, SysML V2 introduces advanced capabilities like variant modelling, analysis cases, trade-off analysis, and representations for individuals, snapshots, and time slices [25]. Additionally, it has large libraries, making the language more extendable and reusable, facilitating improved tool integration through the MBSE ecosystem.

As HAP has a more complex and multidisciplinary nature, there is a high need for a modelling language that can support enhanced expressiveness and tool interoperability. While SysML V1 provided a foundational step towards MBSE, its limitations have become more apparent in the context of rapidly advancing technologies like HAP. As a response to these challenges, SysML V2 has been introduced to offer a more robust and scalable modelling approach. The development of the SysML V2 specification follows the standard OMG process for creating a new standard [6]. This aims to tackle key fundamental issues with the V1 language, such as enhancing accuracy, advancing integration, and promoting greater consistency and alignment with the principles underlying SysML V2. These improvements are a benefit for the HAP system. It requires high modelling precision, tool integration, and traceability across complex domains. SysML V2 enhanced accuracy and interoperability directly support the rigors design and verification needs of such advanced technologies. As mentioned by OMG Group

[20],SysML V2 will cover the full range of SysML V1 capabilities, including support for modelling structure and behaviour, parametrics, and requirements, as well as incorporating additional system modelling concepts not present in SysML V1. SysML V2 will improve upon the precision of SysML V1 by incorporating more precise semantics, enabling the execution of SysML models that include activities, state machines, and parametric solvers [20]. Additionally, it will include a logical formalism to support reasoning about SysML models. This enhanced reasoning capability will allow for more intelligent queries and model navigation, thereby enhancing support for change impact analysis and other types of analysis.

According to Friedenthal [7]SysML V2 introduces a new metamodel, enhanced precision, and a standard API to improve usability, expressiveness, and interoperability. These advancements aim to increase MBSE adoption and effectiveness while providing a smooth transition for SysML V1 users. The SysML V2 Request for Proposal (RFP) issued by OMG in 2017 [6] outlines a clear vision for addressing the limitations of SysML V1 and advancing model-based systems engineering practices. SysML V1 was based on UML, but SysML V2 is designed to be a metamodel design that is made specifically for system engineering. According to this paper, SysML V2 [6] is developed with improved precision and formal semantics to reduce ambiguity and enhance automation. This paper also states the introduction of both textual and graphical notation for more flexible model creation. It also specifies the need for a standardised API to support better interoperability between tools and systems. [6] ,[2]. This model highlights greater support for modularity, reusability, variant management, and enhanced traceability—capabilities that are especially beneficial for complex systems like HAP. These aim to align SysML V2 with the broader goals of digital engineering and system lifecycle integration [6], [2] [9]. The SysML V2 specification introduces several new features as compared to V1 [2]. It includes an integrated expression language for constraints, improved support for variability and product line engineering, and native modelling of units and quantities [9], which can be used in HAP to do several estimations of units, such as wingspan, weight of the flight, power consumption, and so on [11]. And refined modelling of interfaces, snapshots, and flows.

Modelling HAP at particular timeslices with the feature of snapshots, thereby understanding the functionality of the HAP system to exhibit in given timeslices. This SysML V2 also introduces action-based modelling for behaviour and data modelling capability [9].The use of textual notation is similar to programming language, so errors can be corrected more easily than corrections made in graphical images, thus making it easier in modelling and designing compared to V1. Furthermore, it is equipped with an integrated library of types and functions designed to enhance operational efficiency. SysML V2 is constructed upon a foundational framework known as KernelML [7], which constitutes an integral component of its standard. KernelML offers fundamental concepts, including packages, classifiers, associations, connectors, and expressions that are amenable to reuse not exclusively within the context of SysML. According to Jansen [14], in the study done with a magnifying glass, the language aligns well with guidelines for functional suitability and usability, indicating that SysML V2 is both efficient for real-world modelling tasks and easy to work with. SysML V2 is expected to be applied across a wide range of purposes and industries. This diversity could lead to the development of customised variants, making it vital to improve the language portability and maintainability.

Ahlbrecht et al. [1] investigate how SysML V2 can be applied to the model-based development of safety-critical avionics systems. They highlighted that complex, ionic engineering programmes need smooth coordination, and they need data interchange, which is hard to attain with current MBSE procedures. The study is done on notational avionics applications based on SysML V2. The study also shows how standardisation, like APIs, makes integration and interoperability. Their investigation indicates that although SysML V2 has the potential to improve avionics MBSE operations, domain-specific modifications would be required to fully exploit its functionalities in this domain [1]. According to Molnár et al. [16], this paper "offers a practical examination of the manner in which the formal, logic-based semantics of SysML V2 facilitate the automated verification of system models." The authors present an end-to-end methodology that includes the formal analysis of models using a suite of tools, including Gamma, MP-Firebird, Imandra, SAVVS, and SysMD [16], by utilising the rigors foundation provided by SysML V2 Kernel and MerML. These tools facilitate a variety of verification activities, including invariant checking, safety analysis, contract validation, defect injection, and test case generation.

The paper demonstrates the process of translating SysML V2 models into tool-specific formats, verifying them against behavioural and requirement-based constraints, and refining them based on the insights obtained through comprehensive case studies. The authors conclude that SysML V2 not only facilitates MBSE but also promotes the integration of formal verification into MBSE, providing a modelling approach for complex systems that is scalable, precise, and tool-interoperable [16]. Li et al. [15] have done a study on collaborative MBSE methodology that integrates SysML V2 modelling with Dataspaces to facilitate seamless and safe data interchange among various development partners. In this study they have suggested the technique of breaking systems and subsystems to make remote development easier, and they have introduced an architecture that includes a SysML V2 editor that is connected to a Dataspace environment. Their approach is tested with a precision engineering use case and shows both its potential and its problems in real-world collaborative situations [15].Morkevicius et al. [19] explore early efforts to align the Unified Architecture Framework Unified Architecture Framework (UAF) [19] with SysML V2. As SysML V2 has many new features and capabilities with the new version 2, UAF is a good example to explore the new features of SysML V2. This study investigates whether the SysML V2 revamped metamodel can adequately support UAF version 2, which is being reengineered on top of SysML V2. Their research shows a proof of concept highlighting how SysML V2 can serve UAF needs by mapping UAF constructs to SysML V2 elements, while also the study identifies emerging challenges during this integration [19].

According to Morkevicius [17],The examination of MBSE methodologies and enterprise architectural frameworks indicates that most are conceptual and, hence, cannot be effectively integrated with systems modelling techniques, such as SysML V1, in practice. In contrast, the MBSE Grid methodology presented in this study is entirely compatible with SysML [17]. The modelling process is explicitly delineated, the model artefacts to be generated at each phase of system specification and design are disclosed, and the management of traceability relationships, both horizontal and vertical, is elucidated, all grounded in the transparent system architecture framework. The usefulness of the MBSE Grid in conjunction with the MagicDraw toolkit [17], which facilitates SysML, is illustrated to form a current emphasis of the MBSE Grid technique, which is the creation of a system model. Using the MBSE Magic Grid method described by Morkevicius [17], this thesis aims to build on the MBSE Grid method by adapting it to the newest version of SysML V2 and examining how this updated version can be used to model safety-critical systems like HAP. And the thesis tries to model safety-critical systems in a systematic modelling and thereby analyse the new features in SysML V2 as compared to V1. And through the alignment of Magic Grid structured phases and traceability principles with SysML V2 expressive capabilities, this study will investigate how a modernised MBSE approach can be used to model complex systems in a methodical and efficient manner.

4 RESEARCH OBJECTIVE

Enhance the MBSE Magic Grid method by having it work with SysML V2 so that safety-critical HAP systems can be modelled in a more organised way. To be more precise, the study will first adapt the original MBSE magic grid framework to SysML V2 by scaling each phase and artefact, thus obtaining traceability in the system, thereby providing a system model. Secondly, a full SysML V2 model of a typical HAP system was made that shows how the Magic Grid helps to gather requirements, plan the design, analyse behaviour, ensure safety is met, and verify the work. Third, compare SysML V2 to SysML V1 by looking at how expressive the models are, how accurate the new features in V2 are, how much work goes into the model and how well they support safety-critical systems.

4.1 METHODOLOGY

For modelling the HAP using SysML V2 within the MBSE Magic Grid framework, the following tools are required. This section provides an overview of the tools used in the system development process. SysML V2 models were developed using the Gorenje Docker environment [24], offering a Jupyter-based interface for textual modelling and simulation, complemented by graphical visualisation via the SySON tool. This setup enabled graphical visualisation and supported an integrated workflow between textual and visual representations. These tools ensure a consistent, flexible, and efficient modelling process at all stages of the system development.

4.1.1 *Research approach*

The type of research is system engineering, where the approach used in this thesis is qualitative, exploratory, and model-based. The qualitative approach is based on understanding complex systems through detailed and descriptive modelling; it means detailing the structure, behaviour, and relationships in the system architecture. The method used is MBSE with SysML V2 using the Magic Grid framework, and the tools used are the Gorenje SysML V2 pilot implementation using Jupyter Lab and SySON. The focus is on developing a system architecture and a safety-critical behaviour modelling view. The expected output is the artefacts of the SysML V2 model. This modelling framework relies on a toolchain based on the Gorenje Docker environment. It includes the Jupyter Lab interface for writing and executing textual notation and thereby creating SysML V2 models [24]. In this work, Jupyter Lab is used as the primary modelling environment to develop SysML V2 models. The environment is configured with SysML V2 libraries and consists of the KerML metamodel. PlantUML is integrated within JupyterLab to render structural and behaviour diagrams from the textual model definitions. This setup enables an interactive, code-driven modelling workflow that supports both textual and visual representation of system models. As the complete development of SysML V2 has not yet been done, the pilot implementation version is the available version and is used in this thesis.

4.2 TOOLING ENVIRONMENT

4.2.1 *Why Docker Was Selected*

Docker version 4.37.1 is used in this thesis due to its ability to provide a consistent, portable, and isolated environment to execute SysML V2 modelling tools. Here, the Gorenjee SysML V2 pilot implementation using Jupyter Lab and Eclipse Syson containers is used [24].Using Docker helps ensure reproducibility, thus replicating the entire development environment across systems for consistent behaviour. The main advantage of Docker is that it provides isolation by encapsulating dependencies within the containers, and this helps to prevent conflicts with the host machine. As Docker is used in this study for the setup of SysML V2, it was comparatively simpler than manually setting up the installation process [24]. Docker simplifies deployment through the use of preconfigured images. This also provides isolation by encapsulating dependencies within the containers, thereby preventing conflicts with the host machine. As image versions are used, version control and tool stability can be ensured throughout the development process. As in this thesis, the SysML V2 Pilot implementation using Jupyter Lab and Eclipse SysON runs as different containers but is not conflicting, as they are isolated by encapsulating dependencies within their own containers [24]. So, at the same time, a greater number of different containers can run without affecting one another. This advantage makes Docker particularly suitable for model-based system engineering, where reliable toolchains and environment consistency are crucial for managing complex modelling tasks.

4.2.2 *Gorenje SysML V2 Pilot Implementation Using Jupyter Lab*

The SysML V2 pilot implementation provides an implementation focused on text-based modelling; it is known as textual notation in SysML V2, and the visualisation is PlantUML. This thesis utilises JupyterLab for text-based modelling. The SysML pilot implementation has two main environments: text-based modelling using JupyterLab or Eclipse. In this thesis, JupyterLab was selected as the modelling environment because of its interactive and user-orientated design compared to Eclipse, as the latter requires a more extensive configuration. Whereas Jupyter Lab provides a simplified and intuitive interface [24].The notebook-based IDE of JupyterLab enables seamless integration of code, explanatory text, system queries, and model outputs, making it a perfect fit for the document modelling process. Additionally, JupyterLab is compatible with version control systems and is a Python-based scripting language model with quick iteration features that are especially valuable in the development of complex systems [24]. The Goeranje SysML V2 Jupyter Docker image provides a fully integrated environment for working with the SysML V2 pilot implementation directly within JupyterLab. This setup includes the official SysML V2 kernel, associated with an API server, and the core modelling libraries, all combined in a ready-to-use package [**24**]. Thereafter, launch JupyterLab with the SysML V2 kernel already configured. This enables important features such as text modelling, running models interactively, and using commands to work with the internal Application Programming Interface (API) server. The container also comes preloaded with the SyML V2 standard library and sample notebooks, allowing immediate exploration and prototyping. It includes all necessary components for the modelling language, the kernel, the API layer, the SysML V2 libraries, and visualisation through PlantUML, enabling textual and graphical notation within the modal packages. This Docker image approach is a more reproducible, self-contained development environment that is significantly simpler and is more consistent than the traditional method of the manual installation process.

4.2.3 *SysON Graphical Visualisation Tool*

In this paper, the Eclipse-based SysON tool is used to support graphical rendering of SysML V2 models created using textual notation. SysON is deployed on the system using Docker Compose. It requires Java Runtime Environment with Eclipse Temurin and PostgreSQL 15 to connect an instance to an enterprise database [21]. SysON is a web-based open-source graphical modelling environment for authoring SysML V2 models. This tool is developed to provide support for SysML V2 by creating, editing and visualising SysML V2 models, enabling users to capture system requirements, design architectures, and analyse system behaviour [21]. As this work uses the pilot implementation of SysML V2 provided by the Gorenje Docker environment, it was observed that some of the diagrams generated through textual modelling contained dense and complex information. To overcome this complexity in some of the diagrams, Gorenje SysML V2 is still under development, and the available version is the pilot implementation. Here, SysON was incorporated as a visualisation tool capable of handling and clearly displaying such detailed model semantics. SysON is one of the few tools developed specifically for SysML V2, and it integrates directly with the modelling language. The editor includes views for general overview, interconnection view, action flow view, and state transition view. This also supports the textual format SysML V2. Here, the Gorenje SysML V2 textual notation file can be imported directly into SysON. This provides compatibility for running the files in both SysON and SysML V2 environments. SySon is an instrument for filling the gaps by providing accurate and up-to-date graphical representations for the purposes of model exploration, verification, and documentation [21]. When exploring HAP using SysML V2, some of the diagrams in this thesis, such as the activity diagram and the interconnection diagram, are visualised using SysON, as the same diagrams in SysML V2 contain more information that may be a bit confusing for the user. Such diagrams have been recreated for better visualisation using SysON.

4.3 MODELLING FRAMEWORK

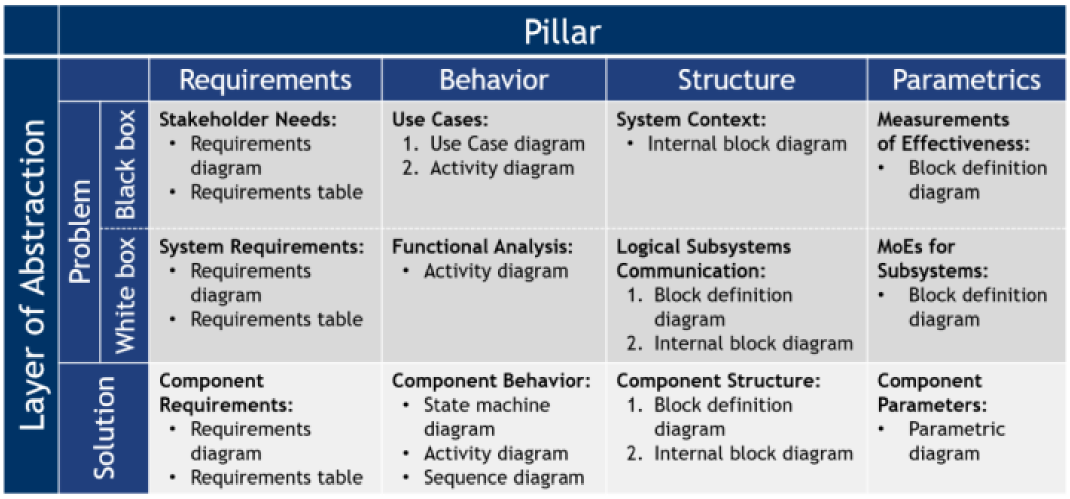
SysML V2 serves as the core modelling language for this framework, and MBSE Magic Grid is the methodology used. SysML V2 directly uses the following concept from KerML. Elements and relationships are the basic structure of a model. Dependencies of the elements, annotation as it attaches the meta-model to the model[20]. It contains namespaces that contain names, particularly packages used to organise elements in a model; specialisations of elements; and expressions that are used for calculations, case results, and so on are taken from the KerML modelling language to SysML V2.

4.3.1 *Modelling Methodology: The MBSE Magic Grid*

Regarding the modelling HAP using SysML V2, the modelling methodology used is the MBSE Magic Grid. The step-by-step modelling is in the next section. The layers of the MBSE Magic Grid are explained in detail here. The Magic Grid, as the name states, is in a grid format where the vertical axis consists of the layer of abstraction. The layer of abstraction is divided into two sections: the problem domain and the solution domain. The high-level problem domain consists of a black box and a white box. The black box layer is where the system is described. What the system does, or the functions expected from the system, is described in this section. Next, we have the white box layer, which describes behaviours that are expected from the subsystem. The next is the solution layer. In the horizontal are the four pillars of the Magic Grid: requirements, behaviour, structure, and parametrics. The MBSE Magic Grid framework is shown in the figure Fig. 4.1.

VIEW SPECIFICATIONS refers to the individual cells in the MBSE grid; each cell represents a different perspective or model within MBSE. In the grid, the view is a specific model that can be used to define a work product that presents the system architecture based on particular concerns or interests. [17]. According to Morkevicius et al. [17] mapping of the SysML in the grid cells was performed using SysML V1. The main aim of this study is to use the Magic Grid and scale the SysML V2 into the view specification and explore how the SysML V2 is effective as an emerging modelling language using the methodology of the Magic Grid.

STAKEHOLDER NEEDS In the black box row and the requirement column, the corresponding view cell is the stakeholders need. As the concept of the black



			Pillar			
Layer of Abstraction			Requirements	Behavior	Structure	Parametrics
	Problem	Black box	**Stakeholder Needs:** • Requirements diagram • Requirements table	**Use Cases:** 1. Use Case diagram 2. Activity diagram	**System Context:** • Internal block diagram	**Measurements of Effectiveness:** • Block definition diagram
		White box	**System Requirements:** • Requirements diagram • Requirements table	**Functional Analysis:** • Activity diagram	**Logical Subsystems Communication:** 1. Block definition diagram 2. Internal block diagram	**MoEs for Subsystems:** • Block definition diagram
	Solution		**Component Requirements:** • Requirements diagram • Requirements table	**Component Behavior:** • State machine diagram • Activity diagram • Sequence diagram	**Component Structure:** 1. Block definition diagram 2. Internal block diagram	**Component Parameters:** • Parametric diagram

Figure 4.1: The MBSE Grid framework (taken from [17])

box is to state what the system does, here stakeholders need to collect key user requirements, internal standards, and so on. This may be collected in different formats, such as interviews, surveys, etc. In the study by Morkevicius et al. [17], the SysML V1 requirement diagram was used to document this level. In this thesis, we try to scale the SysML V2 system requirement diagram to document the view specification in this grid cell. The stakeholders need for HAP is documented at this level.

USE CASES In the black box layer and the behaviour column, the corresponding view cell is the use case. The use case is used to state the functional needs expressed by stakeholders. In this study, the SysML V2 diagram is used to capture and document this level. Each use case describes what a primary and secondary user wants to accomplish with the system. In this level, the use case diagram and the activity diagram can be used to outline the sequence of actions or events, as well as conditions or constraints involved.

SYSTEM CONTEXT This view cell of the grid is in the black box layer and structure column. The cell in the grid is the system context; this cell is documented using a SysML V2 interconnection diagram. This cell captures the interaction of the HAP system with its surrounding environment. The main goal of this cell is to show the key interfaces the system needs to communicate with external elements, mainly the system context and its connections with external components.

MEASUREMENTS OF EFFECTIVENESS This view cell is in the black box layer and in the Parametrics column. The cell is a measurement of effectiveness; it is represented using a block diagram according to Morkevicius et al. [17]. Focuses on identifying measurable criteria that reflect system performance in relation to user expectations. In this paper, SysML V2 constructs are applied to capture and document these effectiveness measures for the HAP.

SYSTEM REQUIREMENTS This layer is the white box row and the requirement column. In this view, the specification cell in the grid system requirements diagram is documented using SysML V2. In this cell, the stakeholder needs in the black box layer are analysed, and from this, system requirements are identified and specified. The SysML V2 system requirement diagram is scalled in this cell.

FUNCTIONAL ANALYSIS This layer is the white box layer in the problem domain, and the column is behaviour. The grid cell is the functional analysis cell. This cell builds on the functional use case analysis by focusing on the internal functions of the system, which are sometimes referred to as processes [17].It identifies logical subsystems, each responsible for carrying out a specific set of functions. The result of functional analysis in this study is represented using SysML V2 action diagrams.

LOGICAL SUBSYSTEMS This step is the white box layer in the problem domain layer and the structure pillar in the column side of the magic grid. The view cell is mapped to SysML V2. In this cell, the internal subsystem, also known as the logical subsystem, mainly focuses on the logical connection of the subsystem. To represent and document this view, the interconnection diagram and internal subsystem diagram using parts, ports, and flow connections are documented using the SysML V2.

MOES FOR SUBSYSTEMS This cell comes in the white box of the problem domain and the parametric pillar. According to Morkevicius et al. [17], the emphasis is on defining measurable criteria that identify the overall performance of the system objectives. These MoEs provide a basis for validating subsystems against stakeholder requirements and system-level expectations. To represent and document this view, SysML V2 constructs are used. The effectiveness measures are linked to subsystem performance metrics directly to system requirements, enabling traceability across the Magic Grid.

COMPONENT REQUIREMENTS This step is the solution layer and the requirement in the pillar. Thus, the grid cell consists of component requirements. This cell captures the detailed and formal requirements and the design constraints for each identified component [17]. Component requirements are derived from system requirements. The view cell is scaled to SysML V2, and the view is documented using a SysML V2 requirement diagram.

COMPONENT BEHAVIOR This step is the solution layer and the behaviour in the column. Thus, the grid cell consists of component behaviour. This cell captures the detailed behaviour of each component by defining states and actions [17].

The detailed behaviour is a subject for simulation. A combination of SysML V2 state machine and action diagram is used to capture and document the detailed behaviour of this cell.

COMPONENT STRUCTURE This step is the solution layer on the row axis and the structure on the column axis of the grid. Thus, the grid cell is assigned to the component assembly. This cell captures physical connections based on physical interfaces between physical components. The physical components implement the logical subsystem created in a viewpoint that defines the problem [17].This view is documented using an internal component diagram in SysML V2.

COMPONENT PARAMETERS This step is the solution layer on the row axis of the magic grid and the parameters on the column axis. This cell in the grid is for the component parameters. As shown in the figure, the two-dimensional matrix ensures systematic coverage of the system model across various pillars and levels of abstraction, iteratively covering various viewpoints and helping to construct a system model. Investigates the scalability of the modelling language within this structured approach, using the example of HAP Alpha. The adapted Magic Grid presented below in the figure represents a scaled version aligned with SysML V2, proposed as a hypothesis and explored throughout this thesis.

5 SYSTEMATIC MODELLING

5.1 CASE STUDY

This section presents a structured, step-by-step case study of the HAP system, modelled using the Magic Grid methodology. The objective is to demonstrate the systematic application of the MBSE principles. This approach enables traceability from stakeholders' needs to the design artefacts. In this model, as described above, first the black box layer of the problem domain is explained, then the white box layer of the problem domain, and then finally the solution layer.

5.1.1 *Problem domain: black-box*

The magic grid abstraction layer first starts with defining the problem domain as a high-level perspective; here, the HAP system is viewed as a black box. In this section, the system is observed in terms of its external behaviour and interactions. In this level, the internal components or implementation details are not discussed. The abstraction level first discussed is the requirement pillar of the Magic Grid. The cell contains the stakeholders' needs. The stakeholders' needs are designed using the requirement table and the requirement diagram. From this layer, the purpose and expected results of the system can be well understood before moving to deeper levels of design and development [4].

5.1.1.1 *Requirement*

In stakeholder needs, information is collected from the stakeholders of the HAP system. The needs of stakeholders can be represented using a table and a requirement diagram [17]. For documenting the stakeholders needs, the table is documented as a spreadsheet from a Microsoft Excel file. The Stakeholder Requirements table is shown in 5.1. Each requirement is noted using a RequirementID; here the RequirementIDs are noted as SN, and then the stakeholder need is specified, following the text. The stakeholder has user needs and industry needs.

The stakeholder requirements for user needs are numbered with the requirement ID as SN1.1 for the earth observation and so on, and the text explains what the stakeholder expects the system to do in SN1.1. Here it is stated as the stakeholder need for HAP to observe earth; likewise, each stakeholder need is addressed with the corresponding requirement ID, then the user needs and the documented text of what the system is required to perform from the stakeholders' point of view. The following user needs are environmental monitoring, flood monitoring, disaster monitoring, animal tracking, maritime surveillance, and air quality, and they are numbered SN1.2 to SN1.6, and industry needs are numbered SN2.1 to SN2.8 and include photovoltaic batteries, solar power, batteries, temperature, flying altitude, wingspan, and platform length, as shown in the table.

REQUIREMENT DIAGRAM The stakeholders' needs are documented in SysML V2 with the requirement diagram, which was developed using the SysML V2 Jupyter Lab pilot implementation. The textual notation is used for the development of the requirement model. The model can be shown as below, in the textual and graphical notation. The requirement diagram in SysML V2 is developed using the textual notation that begins with the stakeholders needs package [8]. A package in SysML V2 is the namespace; it contains its members and a container for its owned members. The package can be imported and used in another package, as this helps to achieve reusability in the model[8]. A requirement definition is a special kind of constraint definition. The requirement has a humanID that is used to show the requirement in a numbering format and to group the requirements. A requirement has a document section where the information of the requirement is given as a description. Then the requirement can include specific constraints and parameters for these constraints. In the textual notation shown for the user need in Figure Fig. 5.1, the human ID for the requirement is numbered SN1, where it represents stakeholder needs, followed by the group SN1.1, and so on.

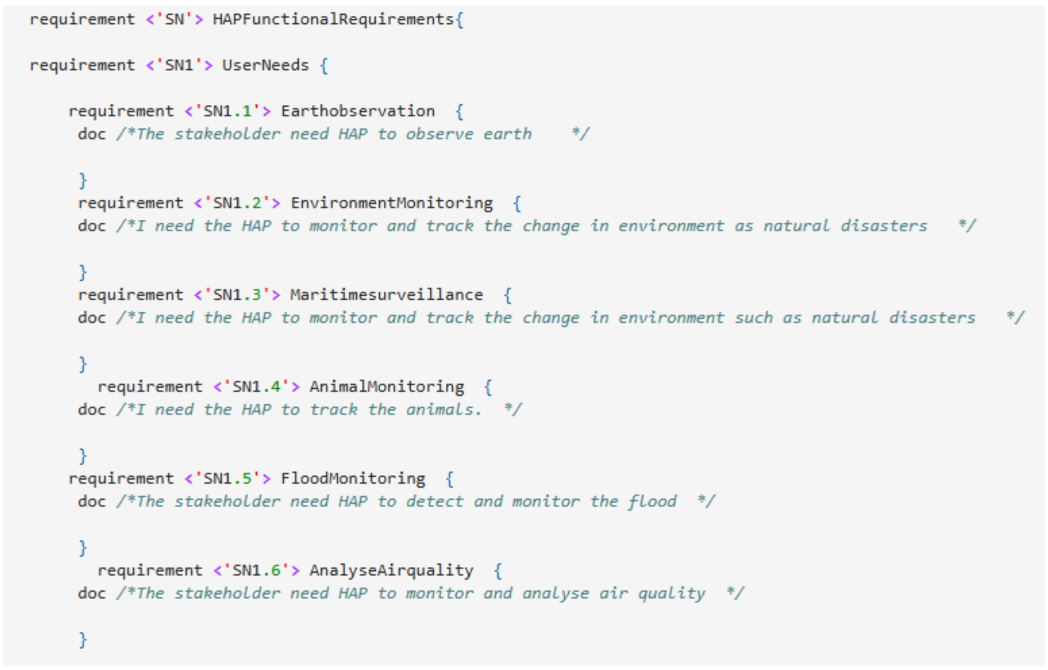
HumanID SN1.1 refers to the earth observation requirement that comes under the user needs of the stakeholders requirements, and the user need is documented as the stakeholder who needs the HAP to observe the movements of the earth. The next requirement is environmental monitoring, where natural disasters, wildfires, etc. are documented. Each requirement is documented with the requirement ID followed by the document section. The requirement in the user needs section of stakeholder needs has requirements from SN1.1 to SN1.6, and each is documented according to the stakeholder need. In the SysML V2 jupyter lab pilot implementation, the visualisation keyword is applied and the graphical notation of the diagram is attained as a result, as shown in figure Fig. 5.2, In this diagram only the user needs of the requirement diagram are shown, as the SysML V2 diagram for the stakeholder needs of the HAP system is too large and consists of a large amount of information, including the user needs and industry standards; thus, the figure Fig. 5.2 shown below only consists of a part of the requirement that are user needs. Other parts of the requirement diagram, such as industry needs, are not included in the figure. This is achieved through the visualisation keyword in the SysML V2, where parts of the system can be visualised according to each perspective. The 'viz' keyword is used in SysML V2 to show the section that is wanted to be visualised.

SN	Stakeholder needs	Text
User needs		
SN1.1	Earth observation	The stakeholder need HAP to observe earth
SN1.2	Maritime surveillance	The stakeholder need to observe the ocean and track the moving ships
SN1.3	Environment Monitoring	The stakeholder need the HAP to monitor and track the change in environment, such as natural disasters
SN1.4	Animal Monitoring	The stakeholder need the HAP to track the animal
SN1.5	Flood monitoring	The stakeholder need HAP to detect and monitor the flood
SN1.6	Analyse Air quality	The stakeholder need HAP to monitor and analyse air quality
Industry standards		
SN2.1	Light weight	The stakeholder need the HAP to be a light weight construction not exceed 130 kilogram
SN2.2	Energy-Photovoltaic	The stakeholder need solar energy as power sources for the HAP
SN2.3	Energy-battery	The stakeholder require alternative power sources as battery
SN2.4	Temperature	The stakeholder need HAP to operate at low and high temperature
SN2.5	Fly altitude	The stakeholder need HAP to fly at an altitude of 20 km in the stratosphere
SN2.6	Stationed	The stakeholder need the HAP to be stationed at an altitude that do not exceed 20 km
SN2.7	WingSpan	The stakeholder need Wing span for the HAP must not exceed 27 meter
SN2.8	PlatformLength	The stakeholder requires a body length of HAP preferably up to 11 m

Table 5.1: Stakeholder Needs for HAP System

5.1.1.2 *Behaviour:*

This cell comes under the problem domain layer and the behaviour pillar. The use case diagram and the action diagram are generated at this level. Both diagrams



```
requirement <'SN'> HAPFunctionalRequirements{

requirement <'SN1'> UserNeeds {

    requirement <'SN1.1'> Earthobservation  {
     doc /*The stakeholder need HAP to observe earth    */

     }
     requirement <'SN1.2'> EnvironmentMonitoring  {
     doc /*I need the HAP to monitor and track the change in environment as natural disasters   */

     }
     requirement <'SN1.3'> Maritimesurveillance  {
     doc /*I need the HAP to monitor and track the change in environment such as natural disasters   */

     }
       requirement <'SN1.4'> AnimalMonitoring  {
     doc /*I need the HAP to track the animals.  */

     }
    requirement <'SN1.5'> FloodMonitoring  {
     doc /*The stakeholder need HAP to detect and monitor the flood  */

     }
       requirement <'SN1.6'> AnalyseAirquality  {
     doc /*The stakeholder need HAP to monitor and analyse air quality  */

     }
```

Figure 5.1: Stakeholders Needs Textual Notation

are created using SysML V2, and for more clarity in this thesis and to use and get familiar with the SysML V2 toolkit, the SysON-SysML V2 graphical environment is used along with the SysML V2 Jupyter Lab. As traceability is one of the features of the Magic Grid framework, here the environmental monitoring requirement SN1.3 from the stakeholders' needs is taken for creating a use case diagram. In the diagram, the use cases for monitoring the environment and detecting changes are created, and the actor of the system is defined. For monitoring the environment, the actors are users and researchers, and the actor for the use case of detecting changes is the system. The diagram is generated using the textual notation in SysML V2. Here, the use case definition is used to define a required interaction between the subject and external actors [8]. An actor is a parameter of the use case representing a role played by the external person or entity towards the subject.

The objective of the use case shown for the subject here, the HAP system, is to provide a result or value to the actors of the system [8]. The figure Fig. 5.3 shows the textual notation of the use case in the SysML V2 pilot implementation; there are two use case definitions defined, 'MonitorEnvironment' and 'DetectChanges', and each objective is defined. The objective of the 'MonitorEnvironment' document is that the HAP system monitors the environmental conditions, for example, weather and temperature, and detects natural changes and disasters in the environment. In the use case 'DetectChanges', the actor interacting is the system, and the objective is to detect potential natural disasters. The system analyses sensor data in real time and records or alerts the detected changes. The figure Fig. 5.4 shows the

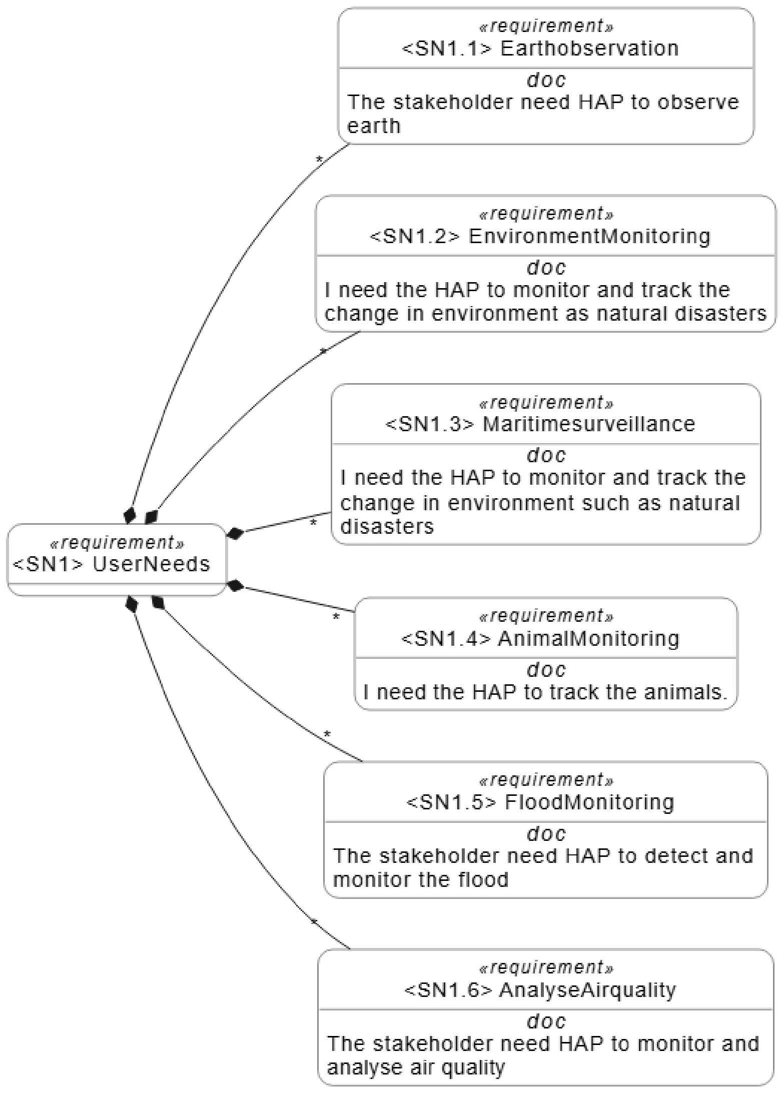
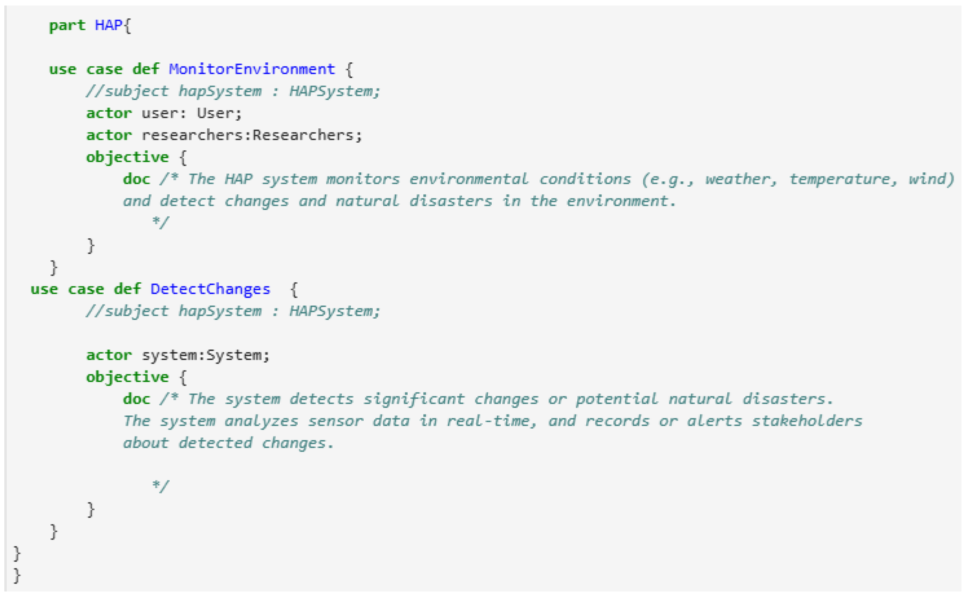

«requirement»
<SN1> UserNeeds
«requirement»
<SN1.1> Earthobservation
doc
The stakeholder need HAP to observe earth
«requirement»
<SN1.2> EnvironmentMonitoring
doc
I need the HAP to monitor and track the change in environment as natural disasters
«requirement»
<SN1.3> Maritimesurveillance
doc
I need the HAP to monitor and track the change in environment such as natural disasters
«requirement»
<SN1.4> AnimalMonitoring
doc
I need the HAP to track the animals.
«requirement»
<SN1.5> FloodMonitoring
doc
The stakeholder need HAP to detect and monitor the flood
«requirement»
<SN1.6> AnalyseAirquality
doc
The stakeholder need HAP to monitor and analyse air quality
*

Figure 5.2: Stakeholders Needs Graphical Notation

graphical representation of the use case diagram. Thus, here the use case diagram is created using textual notation and graphical notation in SysML V2 Jupyter Lab using pilot implementation. The figure Fig. 5.5 shows the use case diagram in SysON graphical representation.

ACTION DIAGRAM The action diagram is the second diagram in the behaviour pillar or the primary scenario of the use case in the problem domain layer. This is developed with the SysON SysML V2 tool. The action diagram is a scenario captured in the form of a SysML activity diagram and is used to show the flow of actions performed by the actor and the system [8]. In Figure Fig. 5.6 , the actor user and the HAP system are shown. This is derived from the use case diagram and the requirement SN1.2 Environmental Monitoring. Thus, traceability



```
    part HAP{

    use case def MonitorEnvironment {
        //subject hapSystem : HAPSystem;
        actor user: User;
        actor researchers:Researchers;
        objective {
            doc /* The HAP system monitors environmental conditions (e.g., weather, temperature, wind)
            and detect changes and natural disasters in the environment.
               */
        }
    }
  use case def DetectChanges  {
        //subject hapSystem : HAPSystem;

        actor system:System;
        objective {
            doc /* The system detects significant changes or potential natural disasters.
            The system analyzes sensor data in real-time, and records or alerts stakeholders
            about detected changes.

               */
        }
    }
}
}
```

Figure 5.3: Use Case: Textual Notation

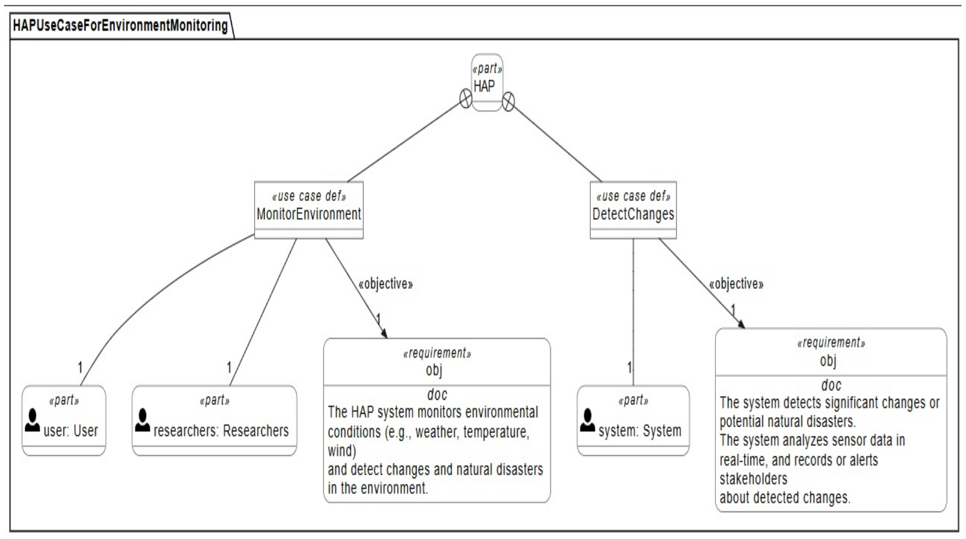
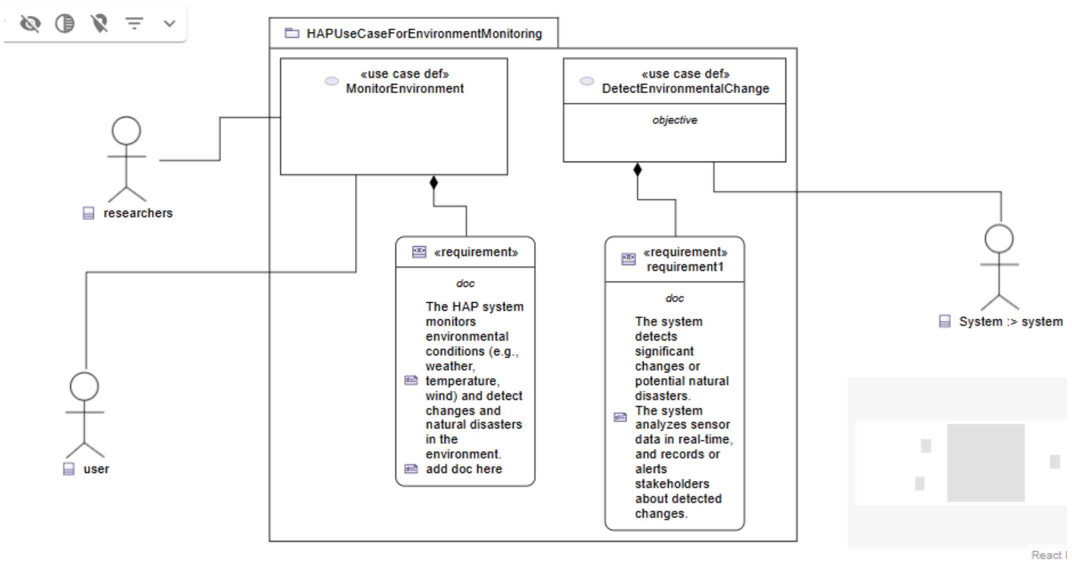

HAPUseCaseForEnvironmentMonitoring
«part»
HAP
«use case def»
MonitorEnvironment
«use case def»
DetectChanges
«objective»
«objective»
1
1
1
1
1
«part»
user: User
«part»
researchers: Researchers
«requirement»
obj
doc
The HAP system monitors environmental conditions (e.g., weather, temperature, wind) and detect changes and natural disasters in the environment.
«part»
system: System
«requirement»
obj
doc
The system detects significant changes or potential natural disasters. The system analyzes sensor data in real-time, and records or alerts stakeholders about detected changes.

Figure 5.4: Use Case: Graphical Notation

is maintained here. In the action diagram, the user that can be researchers from the ground station is requesting environmental changes. The request is sent to the HAP system, and the next action is to monitor the environment. The system monitors the environment, and the next action is deciding whether the system detected changes. If changes are detected, the next action is to record the changes, and then the changes are sent to the ground station in the next steps, and the final action is to receive the information from the user. This diagram explains the flow of action that is performed.



The HAP system monitors environmental conditions (e.g., weather, temperature, wind) and detect changes and natural disasters in the environment.
add doc here

The system detects significant changes or potential natural disasters.
The system analyzes sensor data in real-time, and records or alerts stakeholders about detected changes.

Figure 5.5: Use Case: SysON Representation

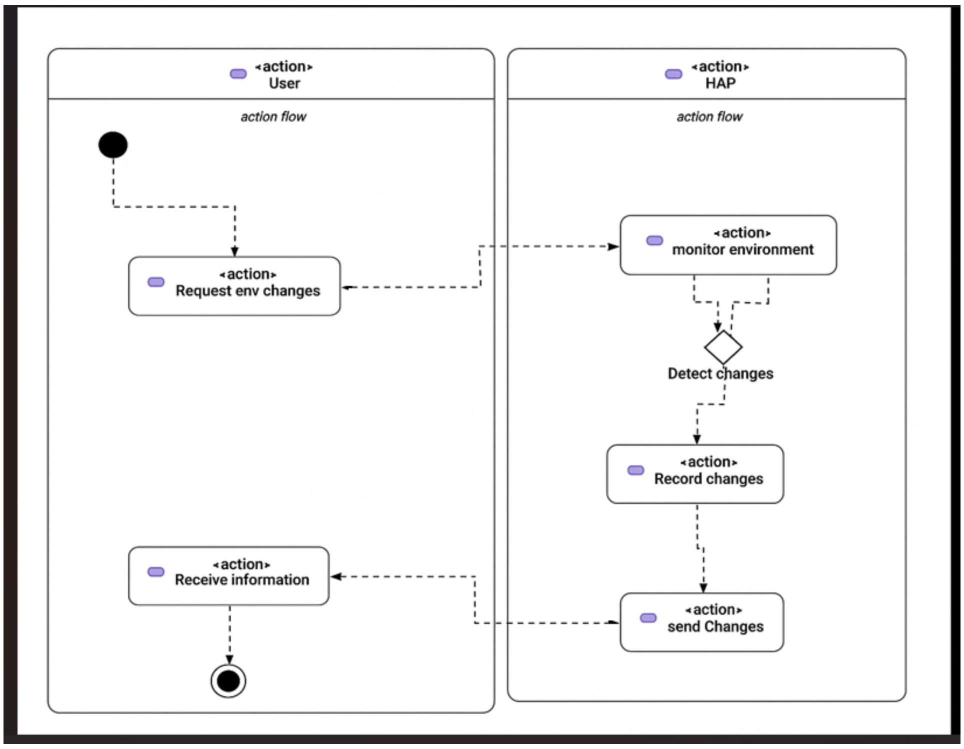


Figure 5.6: Action Diagram: Graphical Notation

5.1.1.3 *Structure*

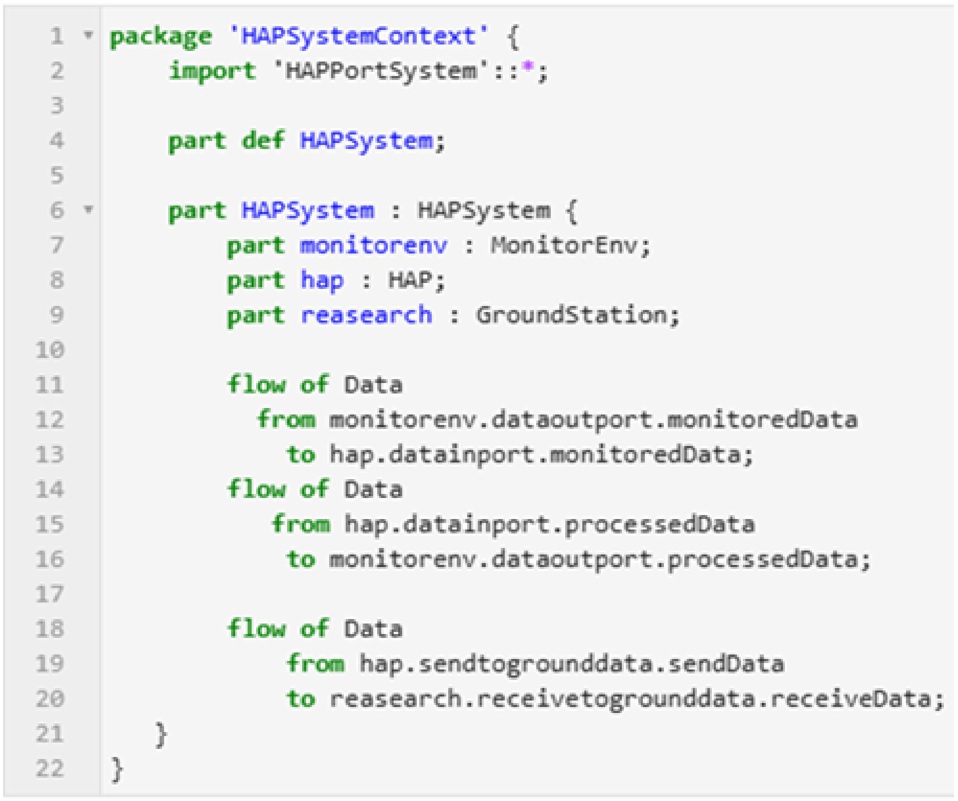
In the black box problem domain layer and the structure pillar, the system context is documented in the cell. The system context diagram is generated using SysML

V2 Jupyter Lab, using textual notation. The SysML V2 structure diagram is used to show the system context. The system context shows the system of interest and how it interacts with its environment, such as external systems, in a context identified while modelling the use case [8]. For building the system context diagram in SysML V2, the interconnection diagram was used to show the connection and the flow. From figure Fig. 5.7, the textual notation for developing the SysML V2 diagram in Jupyter Lab can be visualised. And in figure Fig. 5.8, the output generated in graphical notation is shown. Here, the HAP system is shown to interact with the part monitor environment by ports. The flow of signal from the HAP system to the part monitor data is shown; the ports are connected through the signal out port and signal in port, and the connection flow of signal is established from the signal out port of the HAP to the signal in port of the monitor system.

In SysML V2, a part is a definition of a class of systems or parts of systems, which are mutable and exist in space and time, and 'part usage' is a feature that uses the definition of a part. Here in the diagram, 'monitorenv' is a part that exists in time and space in the HAP [8], another part is the HAP system, and the ground station is also another part of the diagram. This representation shows how these parts can interact with each other. To interact, an interconnection is established that sends data and signals between the parts. For this, ports are used to establish connections. Ports are an interaction point between a part of a system and its external context, such as another part or environment [8].The flow of connection between the parts is established using the ports. From the 'monitorenv' part, the data output is established, and the flow of data is established between the 'monitorenv' part and the HAP system. From the HAP system, a flow of data is established between the HAP system and the ground station. This is the interconnection diagram in SysML V2 using the Jupyter Lab, and for more clarity, the same diagram is created using the SysON SysML V2 tool and is visualised in Figure Fig. 5.9.

5.1.1.4 *Parametrics*

This cell is the last one in the problem domain layer and parametrics pillar of the Magic Grid. The cell is known as the Measure of Effectiveness [17]. In the HAP system, MoEs are defined to evaluate the non-functional goals, such as reliability, safety, and so on, of the system. Using the SysML V2 part usage diagram, a reusable set of MoE is generated as shown in figure Fig. 5.10. This helps reusability and clarity throughout the system lifecycle. Safety is the non-functional goal defined for the HAP system in this section. The stakeholders' needs table is updated with non-functional requirements. SN3 is a safety requirement. The stakeholder requires that the HAP system incorporate a dedicated safety subsystem responsible for maintaining the operational altitude of the HAP and monitoring the flight conditions. The SN3.1 safety requirement is named the Flight Altitude Detector. The description for this is stated as, The stakeholder requires the HAP to operate at an altitude well above civilian air traffic, within the lower stratosphere, to avoid potential collisions and enhance flight safety. The flight



```
package 'HAPSystemContext' {
    import 'HAPPortSystem'::*;

    part def HAPSystem;

    part HAPSystem : HAPSystem {
        part monitorenv : MonitorEnv;
        part hap : HAP;
        part reasearch : GroundStation;

        flow of Data
          from monitorenv.dataoutport.monitoredData
            to hap.datainport.monitoredData;
        flow of Data
           from hap.datainport.processedData
            to monitorenv.dataoutport.processedData;

        flow of Data
            from hap.sendtogrounddata.sendData
            to reasearch.receivetogrounddata.receiveData;
    }
}
```

Figure 5.7: System Context: Textual Notation

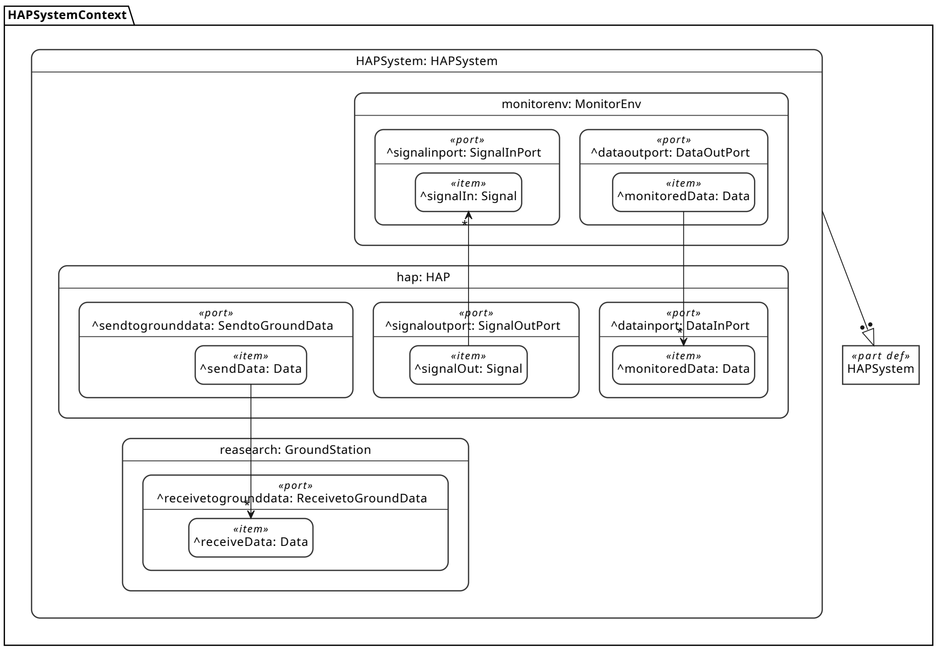


Figure 5.8: System Context: Graphical Notation

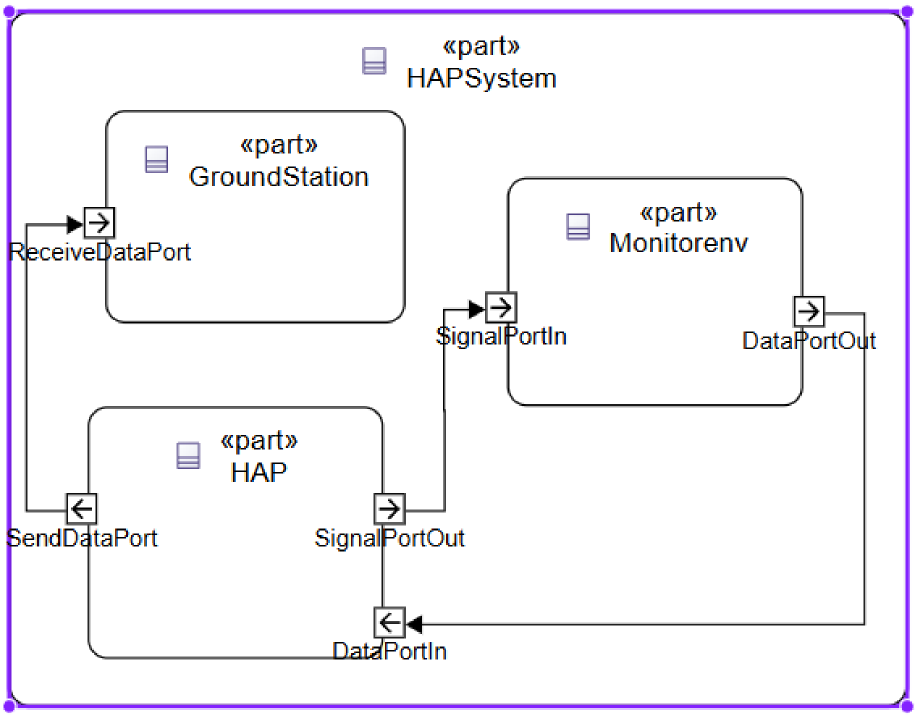


Figure 5.9: System Context: SysON

altitude parameter is set at 56,000 feet above civilian air traffic. Thus, MOEs are used to define specific performance-related requirements. The diagram for MoEs is created using SysML V2 Jupyter Lab; here the structure diagram is created using the part definition for safety, as this can be reused, and the altitude parameter is redefined to 56000 altitude, at which the HAP can safely fly above the air traffic without collision.

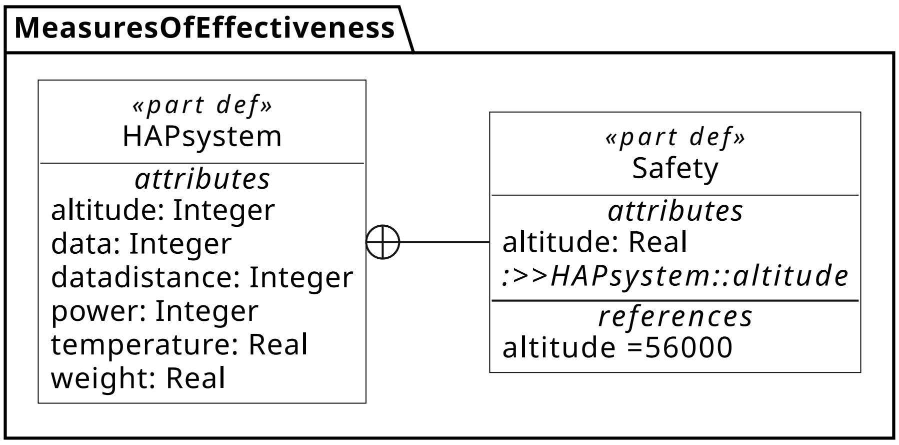


Figure 5.10: Measure of Effectiveness: Graphical Notation

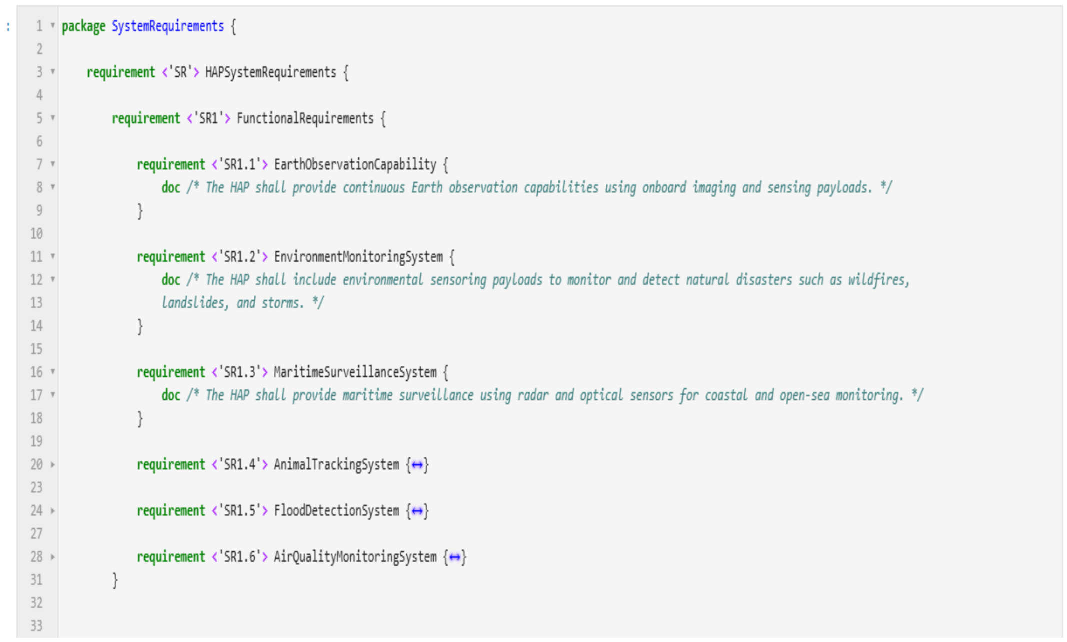
5.2 PROBLEM DOMAIN: WHITE BOX

After completion of the black box of the HAP system, the operational analysis is complete. The next step is to open the black box and transition to a white-box perspective. The white box involves explaining the internal structure and behaviour of HAP through system requirements, functional analysis, and the development of a logical architecture [17]. This phase focuses on understanding what the system is expected to do. The concept of white-box analysis of a system involves understanding and defining its internal structure, functions, and logic. Whereas black-box focuses only on the system inputs and outputs, white-box considers how the system works internally [17]. The white box approach aims to detail how the system operates to meet its requirement.

5.2.1 *System Requirement*

The system requirement cell comes in the problem domain of the white box and the requirement pillar. The system requirements are derived step by step from the needs of the stakeholders. The system requirement diagram is implemented with SysML V2 Jupyter Lab, using textual notation. As stated earlier in the black box requirement diagram, the steps of implementation are similar, with the requirement definition following the humanID for the requirement, and the requirement can include a description about the requirement in its documentation section, attributes can be defined, and the constraints of the requirement can be given. This is the structure of implementing requirements using the textual notation [8]. The figure Fig. 5.11 shows the textual notation of the system requirement. The resultant output is shown using the graphical representation in SysML V2. As the system requirement diagram is a bit large for simplicity, each group of requirements can be viewed using view and viewpoint features in SysML V2. This is a new feature in the second version of SysML compared to the first version.

The system requirement diagram in the white box analysis is shown in figure Fig. 5.12. The system requirement diagram starts with the system requirement package. The system requirement is subdivided into different system requirement groups, such as SR1 for functional requirements and SR2 for design constraints and performance, and then the SR1 and SR2 are subdivided into corresponding requirement groups. SR1 is subdivided into earth observation systems, environment monitoring systems, maritime surveillance systems, animal tracking systems, and flood detection. The requirement group is humanID as SR1.1, SR1.2, etc. Similarly, the SR2 design constraints and performance group are subdivided into lightweight structure, solar energy subsystem, battery backup subsystem, temperature operating range system, and operational altitude, and the human ID of each requirement is ordered accordingly. The system requirement diagram is derived from the stakeholders' needs in the black box. Thus, traceability is achieved here.



```
package SystemRequirements {

    requirement <'SR'> HAPSystemRequirements {

        requirement <'SR1'> FunctionalRequirements {

            requirement <'SR1.1'> EarthObservationCapability {
                doc /* The HAP shall provide continuous Earth observation capabilities using onboard imaging and sensing payloads. */
            }

            requirement <'SR1.2'> EnvironmentMonitoringSystem {
                doc /* The HAP shall include environmental sensoring payloads to monitor and detect natural disasters such as wildfires,
                landslides, and storms. */
            }

            requirement <'SR1.3'> MaritimeSurveillanceSystem {
                doc /* The HAP shall provide maritime surveillance using radar and optical sensors for coastal and open-sea monitoring. */
            }

            requirement <'SR1.4'> AnimalTrackingSystem {↔}

            requirement <'SR1.5'> FloodDetectionSystem {↔}

            requirement <'SR1.6'> AirQualityMonitoringSystem {↔}
        }


```

Figure 5.11: System Requirement Textual Notation White Box

view SysML V2, The `%viz` command is used to visualise the diagram, and the `%view` command is used to visualise different aspects of a system model through various rendering types and styles. Both of these commands can be used together to get different views of the system model, as the view command provides different rendering types of diagrams, such as tree view, interconnection view, state machine view, activity diagram, or a mix of multiple views. Style can be defined along with the view and viz command types to give specific styles to graphical output, such as colour schemes, layout orientation, showing inherited or imported elements, hiding metadata, or applying PlantUML formatting. These commands in SysML V2 help with specific visualisation effects.
Using the `%viz` command, the entire system requirement diagram can be visualised, or the `%view` command along with rendering and styles can be used to visualise the diagram in parts such as SR1 or SR2. The system requirement diagram can also be viewed in different styles; thus, SysML V2 provides different commands to visualise the system in different aspects. For viewing this diagram shown in figure Fig. 5.12, the command used is `%viz` –style LR System Requirements, and the diagram can be visualised as shown.

5.2.2 *Functional Analysis*

The action diagram is documented in the functional analysis cell; this is in the white box of the problem domain and behaviour pillar. The action diagram is implemented using the SysON SysML V2 tool. At the white box level of the MBSE Magic Grid, functional analysis focuses on decomposing and refining the

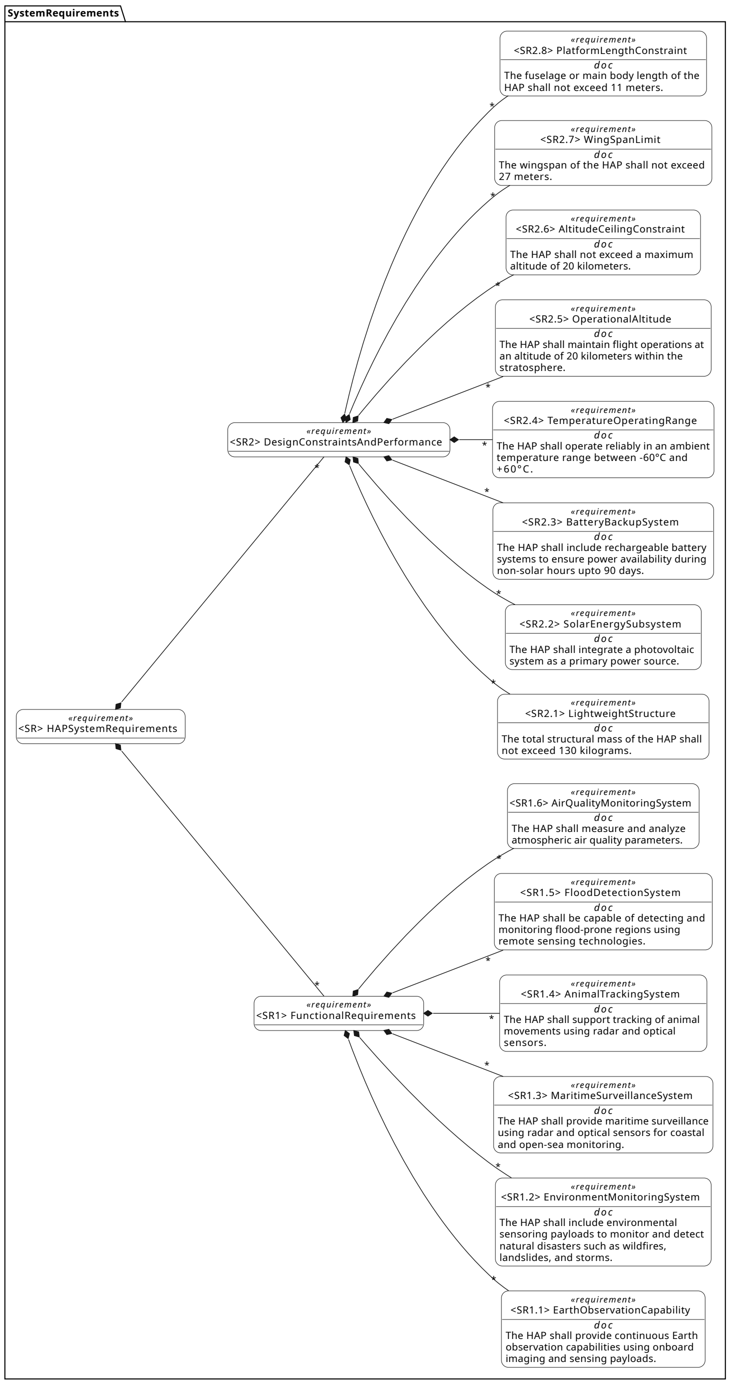

SystemRequirements
«requirement» <SR2.8> PlatformLengthConstraint
doc
The fuselage or main body length of the HAP shall not exceed 11 meters.
«requirement» <SR2.7> WingSpanLimit
doc
The wingspan of the HAP shall not exceed 27 meters.
«requirement» <SR2.6> AltitudeCeilingConstraint
doc
The HAP shall not exceed a maximum altitude of 20 kilometers.
«requirement» <SR2.5> OperationalAltitude
doc
The HAP shall maintain flight operations at an altitude of 20 kilometers within the stratosphere.
«requirement» <SR2.4> TemperatureOperatingRange
doc
The HAP shall operate reliably in an ambient temperature range between -60°C and +60°C.
«requirement» <SR2> DesignConstraintsAndPerformance
«requirement» <SR2.3> BatteryBackupSystem
doc
The HAP shall include rechargeable battery systems to ensure power availability during non-solar hours upto 90 days.
«requirement» <SR2.2> SolarEnergySubsystem
doc
The HAP shall integrate a photovoltaic system as a primary power source.
«requirement» <SR2.1> LightweightStructure
doc
The total structural mass of the HAP shall not exceed 130 kilograms.
«requirement» <SR> HAPSystemRequirements
«requirement» <SR1.6> AirQualityMonitoringSystem
doc
The HAP shall measure and analyze atmospheric air quality parameters.
«requirement» <SR1.5> FloodDetectionSystem
doc
The HAP shall be capable of detecting and monitoring flood-prone regions using remote sensing technologies.
«requirement» <SR1> FunctionalRequirements
«requirement» <SR1.4> AnimalTrackingSystem
doc
The HAP shall support tracking of animal movements using radar and optical sensors.
«requirement» <SR1.3> MaritimeSurveillanceSystem
doc
The HAP shall provide maritime surveillance using radar and optical sensors for coastal and open-sea monitoring.
«requirement» <SR1.2> EnvironmentMonitoringSystem
doc
The HAP shall include environmental sensoring payloads to monitor and detect natural disasters such as wildfires, landslides, and storms.
«requirement» <SR1.1> EarthObservationCapability
doc
The HAP shall provide continuous Earth observation capabilities using onboard imaging and sensing payloads.
*

Figure 5.12: System Requirement Diagram White Box

internal behaviour of the HAP system. Each function identified in the high-level black box functions is analysed in detail using the SysML V2 activity diagram to define the flow of control and data. For explaining the activity diagram, the requirement SR1.2 is taken from the system requirement diagram. In the SR1.2 system requirement diagram, environmental monitoring states that the HAP system shall include environmental sensing payloads to monitor and detect natural disasters such as wildfires, landslides, and storms.

Using the activity diagram, the internal flow of action is defined. The diagram is shown in the figure Fig. 5.13, The white box activity diagram is used to refine the internal behaviours of the previously defined black box function. The action request for environmental changes is submitted to the HAP system. The next action is monitoring the environment, which is done by the HAP system. The flow of action is processing the monitored data and analysing the processed data. If a natural disaster is detected, the next step is to transmit the information to the user. If a natural disaster is not detected, the flow of control goes back to monitoring environmental action. Thus, at the white box level of the grid, functional analysis focuses on decomposing and refining the internal behaviour of HAP. These internal functions align with and refine the system-level requirements, ensuring traceability across the problem and behaviour pillars of the MBSE Grid.

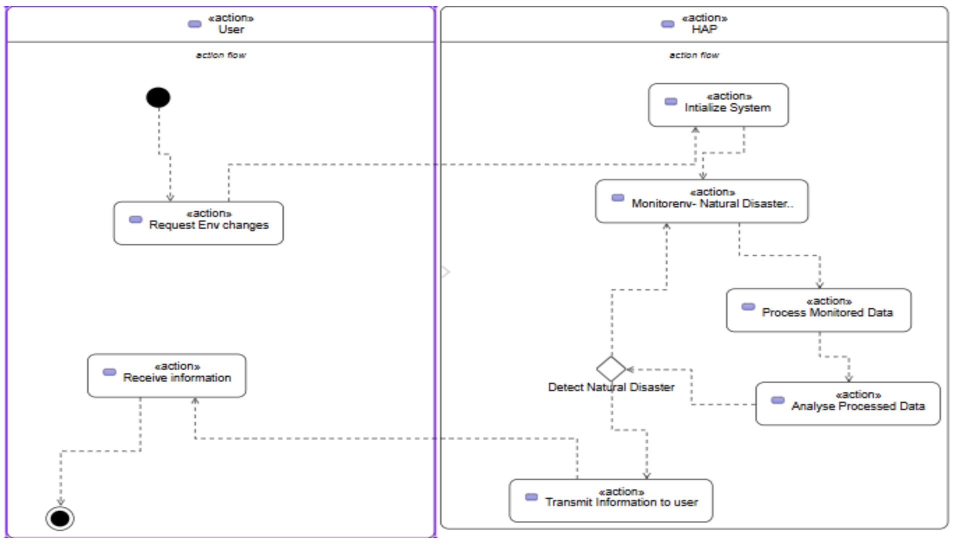
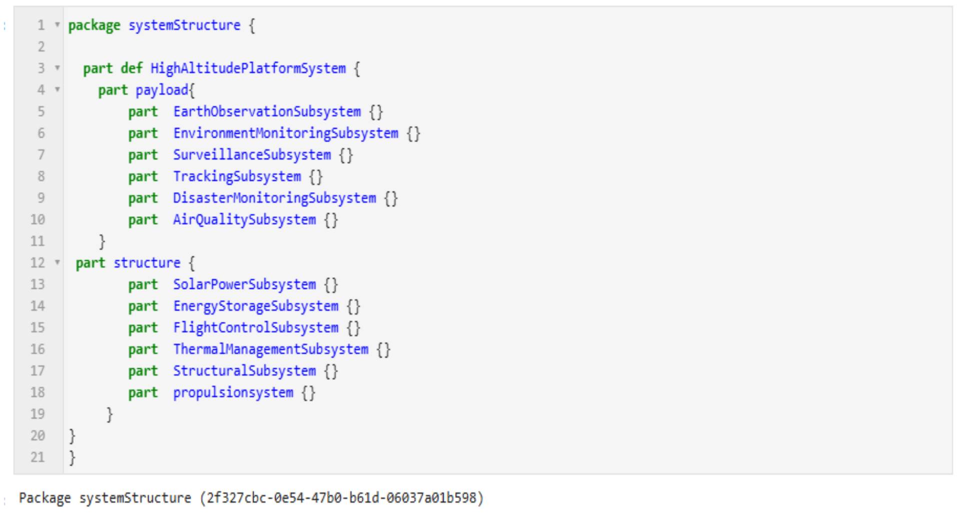

«action» User
action flow
«action» Request Env changes
«action» Receive information
«action» HAP
action flow
«action» Intialize System
«action» Monitorenv- Natural Disaster..
«action» Process Monitored Data
Detect Natural Disaster
«action» Analyse Processed Data
«action» Transmit Information to user

Figure 5.13: Activity Diagram: White Box

5.2.3 *Logical Subsystem Communications*

This cell comes in the white box of the problem domain and the structure pillar. This cell is used to document the two essential diagrams, the logical subsystem definition diagram and the internal subsystem. Both diagrams are implemented with SysML V2 Pilot Implementation within Jupyter Lab. Textual notation is used for implementing the model; SysML V2 is a text-based, iterative, and script-driven system design [8].The textual notation for the logical subsystem structure is shown in the figure Fig. 5.14. The logical subsystem of the HAP considered here is the part payload and the part structure. This is derived from the stakeholders' requirements SN1 and SN2. Thus, traceability is maintained. Through this section the logical subsystem of the system of interest is clearly defined, helping for the further stages of system development. The logical subsystem definition diagram provides a high-level view of a system internal structure by decomposing it into logical subsystems. This is similar to an interface definition diagram. This emphasises the organisation and partitioning of the system into cohesive functional units. The logical subsystem structure for the HAP system represents a modular decomposition of the overall system into functional blocks that define the architecture of the system.

This HAP system structure is divided into two principal parts: as shown in figure Fig. 5.15, the structure subsystem and the payload subsystem. The structure subsystem contains essential platform support elements, including the solar power system, energy storage subsystem, flight control board, propulsion subsystem, and so on. These components ensure the platform stability and self-sufficiency in high-altitude conditions. The payload subsystem contains the capabilities of HAP, such as monitoring air quality, responding to disasters, tracking, surveillance, environmental observation, etc. This logical organisation supports the concept of MBSE principles by simplifying complexity, enhancing modularity, and enabling better management of system requirements. The figure shown in Fig. 5.15 clearly decomposes the overall system into smaller logical parts This aligns with modular decomposition and the clear separation between the structure and payload, further divided into domain-specific subsystems, supporting functional isolation and traceability as per MBSE principles.



```
package systemStructure {

  part def HighAltitudePlatformSystem {
    part payload{
        part  EarthObservationSubsystem {}
        part  EnvironmentMonitoringSubsystem {}
        part  SurveillanceSubsystem {}
        part  TrackingSubsystem {}
        part  DisasterMonitoringSubsystem {}
        part  AirQualitySubsystem {}
    }
  part structure {
        part  SolarPowerSubsystem {}
        part  EnergyStorageSubsystem {}
        part  FlightControlSubsystem {}
        part  ThermalManagementSubsystem {}
        part  StructuralSubsystem {}
        part  propulsionsystem {}
     }
}
}
```

```
Package systemStructure (2f327cbc-0e54-47b0-b61d-06037a01b598)
```

Figure 5.14: Logical Subsystem Structure: Textual Notation: White Box

5.2.3.1 *Internal Subsystem Structure*

The internal subsystem structure diagram is implemented using the SysML V2 Jupyter Lab pilot implementation. The textual notation is shown in the figure Fig. 5.16. In the internal subsystem, the structure is derived from the logical subsystem where the HAP is divided into payload and structure.

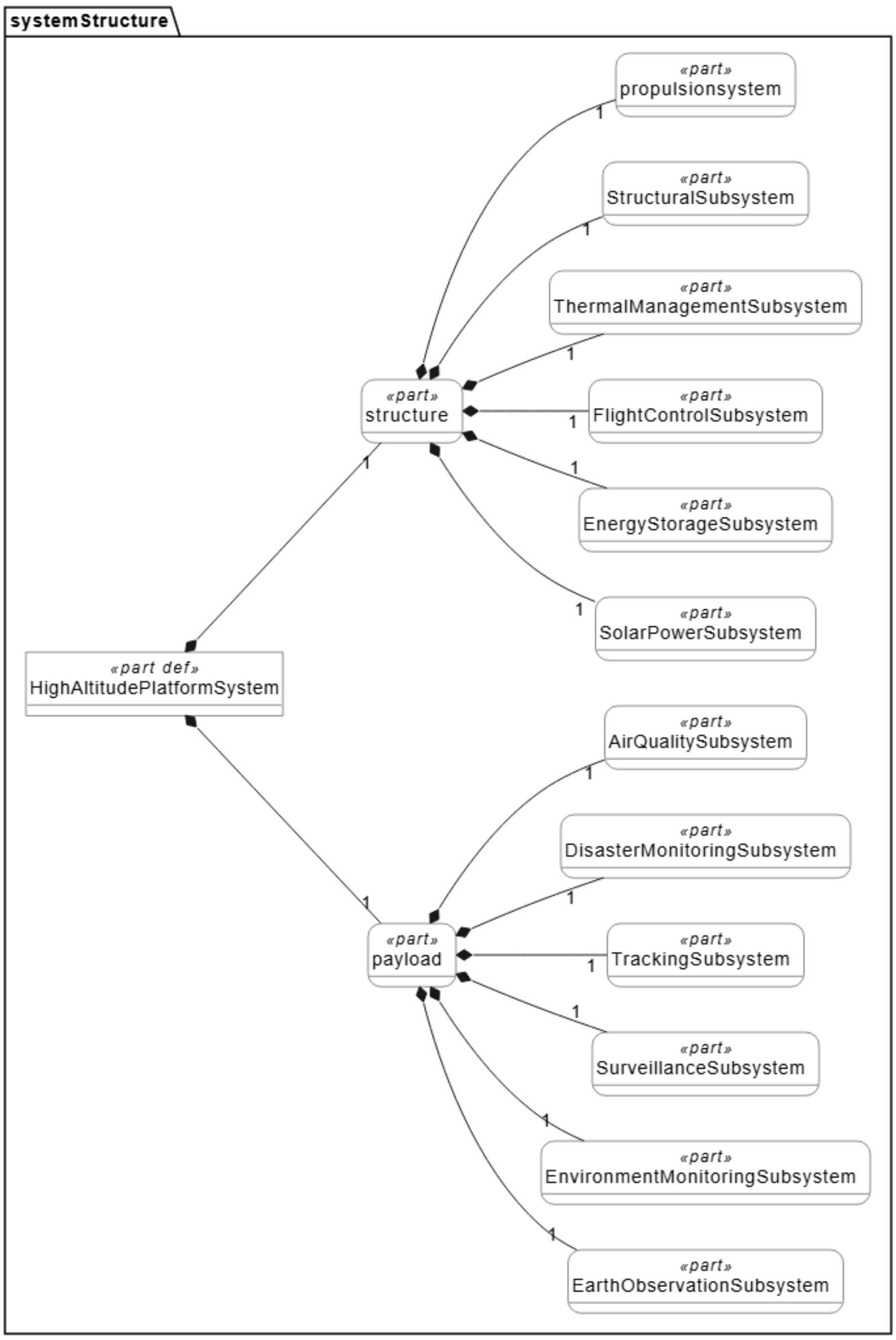

systemStructure
«part» propulsionsystem
«part» StructuralSubsystem
«part» ThermalManagementSubsystem
«part» FlightControlSubsystem
«part» EnergyStorageSubsystem
«part» SolarPowerSubsystem
«part» structure
«part def» HighAltitudePlatformSystem
«part» payload
«part» AirQualitySubsystem
«part» DisasterMonitoringSubsystem
«part» TrackingSubsystem
«part» SurveillanceSubsystem
«part» EnvironmentMonitoringSubsystem
«part» EarthObservationSubsystem
1

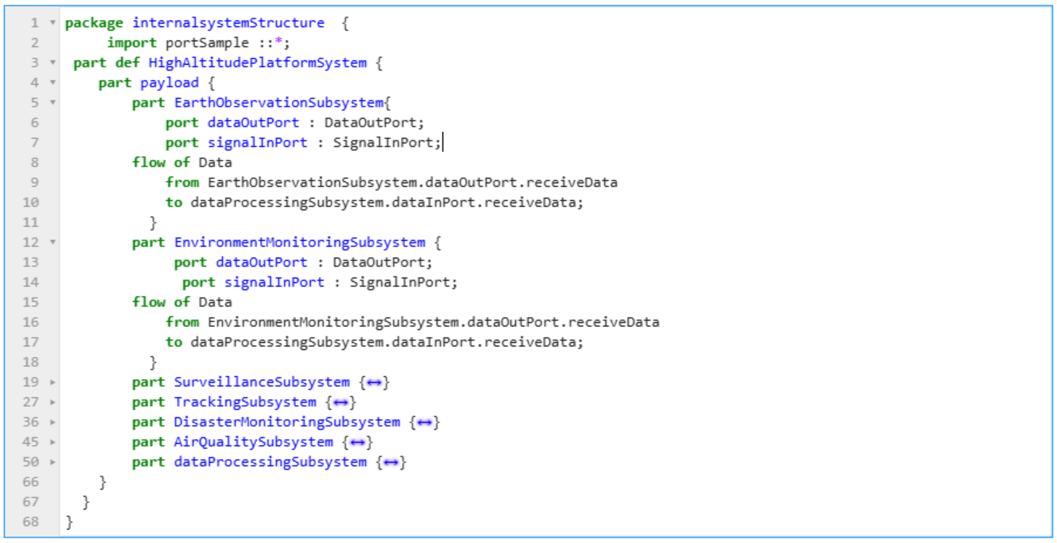
Figure 5.15: Logical Subsystem Structure: Graphical Notation: White Box

Here, the payload interconnection is taken. The payload has parts such as 'EarthObservationSubsystem', 'EnvironmentalMonitoringSubsystem' and so on, as defined in the system requirement, and sub-parts as shown in the logical subsystem. Here, the ports and interconnection between the sub-parts are addressed. The diagram of the structure of the internal system for the HAP system shown. Fig. 5.17 illustrates the internal structure of its payload subsystem. This is generated from the textual notation; the payload system is refined in this diagram for simplicity. This diagram highlights the interconnections between individual functional components through clearly defined ports, connections, and data flows.

A port definition specifies a collection of features that can be accessed through a port that serves as a connection point on a part. This allows a part definition to expose some of its internal features in a controlled and limited manner [8].Here, from the diagram, the earth observation subsystem has the signal in port and the data out port. So the connection point to this subsystem is these two ports. Ports are used to manage the interdiagram between different parts in a system. For two ports to be compatible for connection, one port must provide the features that the other port requires. This inverse relationship ensures that communication and interaction between parts occur correctly and efficiently [8]. From the diagram in figure Fig. 5.17, the earth observation subsystem has a signal in port and a data out port; the signal is compatible with the signal out port of the data processing subsystem, and the data out port is compatible with the data in port of the data processing subsystem.

A flow connection is streaming if the transfer is ongoing between the source and target. A flow connection usage is a transfer of some payload from an output of a source port to an input of a target port [8]. The connection is defined to be between the ports, but the transfer is still from an output to an input [8].Here in the HAP system, the flow connection is from the signal out port of the data processing subsystem to the signal in port of the earth observation subsystem. And the data output of the earth observation subsystem flow connection is to the data input of the data processing subsystem. Each logical subsystem—such as the Earth Observation Subsystem, Surveillance Subsystem, Environment Monitoring Subsystem, and others—is modelled with input/output ports such as dataOutPort, dataInPort, signalInPort, and signalOutPort to facilitate modular communication. The Fig. 5.16 shows the textual notation, and the generated output is shown in figure Fig. 5.17. In this diagram, the internal structure of the HAP and how the internal parts communicate through ports and the flow of signal from one part to the other are shown.

From the diagram shown in figure Fig. 5.17, the payload is taken. The payload has several subsystem structures, such as the earth observation subsystem, the environmental monitoring subsystem, the tracking subsystem, and so on. Each subsystem consists of a corresponding attribute, and each subsystem is connected to different ports, such as the input and output of the data port, then the output and the input. For example, the flow of data is from the data processing subsystem through the data in-port to the data out-port of the environmental monitoring subsystem, sending and receiving data. The diagram specifies important attributes for each subsystem, thereby capturing important system parameters. The Fig. 5.18 shows the internal system structure diagram using SysON. This diagram provides the same logical structure but another view of the diagram in the interconnection view of SysML V2. Providing different views of the same diagram, thus improving better user experience and clarity at the time of development.



```
package internalsystemStructure  {
     import portSample ::*;
 part def HighAltitudePlatformSystem {
    part payload {
        part EarthObservationSubsystem{
            port dataOutPort : DataOutPort;
            port signalInPort : SignalInPort;
        flow of Data
            from EarthObservationSubsystem.dataOutPort.receiveData
            to dataProcessingSubsystem.dataInPort.receiveData;
          }
        part EnvironmentMonitoringSubsystem {
             port dataOutPort : DataOutPort;
              port signalInPort : SignalInPort;
        flow of Data
            from EnvironmentMonitoringSubsystem.dataOutPort.receiveData
            to dataProcessingSubsystem.dataInPort.receiveData;
          }
        part SurveillanceSubsystem {↔}
        part TrackingSubsystem {↔}
        part DisasterMonitoringSubsystem {↔}
        part AirQualitySubsystem {↔}
        part dataProcessingSubsystem {↔}
    }
  }
}
```

Figure 5.16: Internal System Structure: Textual Notation: WhiteBox

5.2.4 *Measures Of Effectiveness*

In the white box layer, the system is not treated as a single unit; instead, it is treated as the internal logical structure of HAP. The measures of effectiveness are used to evaluate whether the proposed system architecture can satisfy the non-functional performance requirement defined in the black box layer. In the black box layer, the stakeholder requirement is updated with non-functional requirement SN3.1. In the white box layer of the MoEs, the safety requirements are further updated with SN3.2 Flight vertical separation, which states that the stakeholder needs the HAP to maintain a vertical separation of 16,000 feet above commercial air traffic. And SN3.3 Flight Altitude Time, which states that the stakeholder needs the HAP to sustain this altitude with stability for a 2-hour duration. MoEs are used to refine stakeholders' needs into quantifiable targets. They help ensure that requirements are verifiable and measurable. For implementing MoEs in SysML V2 Jupyter Lab, a SysML V2 part usage diagram is used to define it. The MoEs are used to define measurable outcomes.

Thus, in Figure Fig. 5.19, the HAP system is shown with attributes derived from requirements. These attributes are assigned with measurable values that give clarity and evaluate the usefulness of the system in its operational environment. Reusability and traceability are the main features of SysML V2. The 'minAltitude-Margin' is an attribute derived from the requirement SN3.2, where stakeholders need the HAP to maintain a vertical separation of 16,000 feet; thus, minAltitude-Margin is assigned with the value of 16,000 feet. As SysML V2 ensures reusability The minAltitudeMargin, which was defined as an integer, is redefined as real in

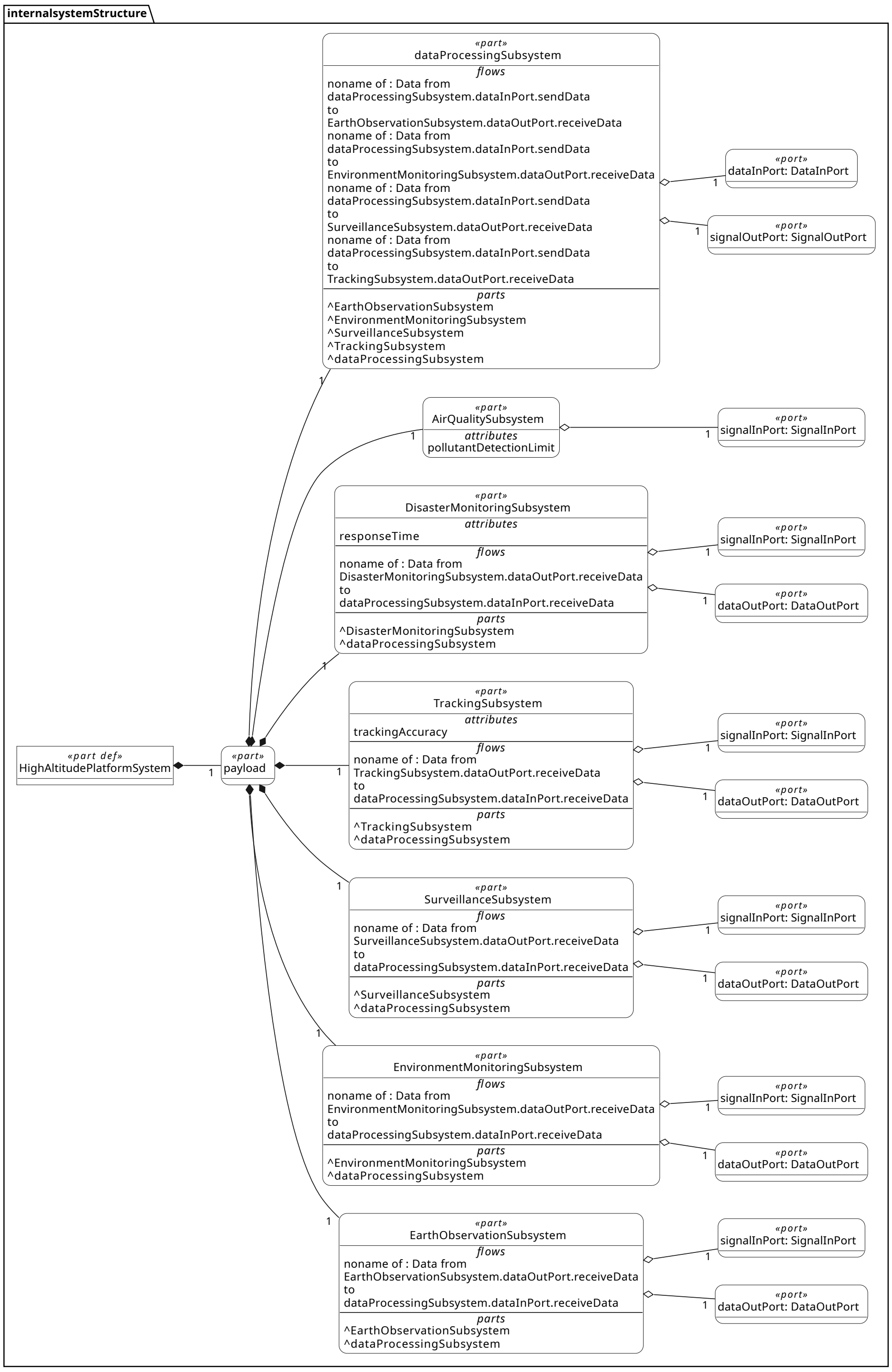

internalsystemStructure
«part»
dataProcessingSubsystem
flows
noname of : Data from dataProcessingSubsystem.dataInPort.sendData to EarthObservationSubsystem.dataOutPort.receiveData
noname of : Data from dataProcessingSubsystem.dataInPort.sendData to EnvironmentMonitoringSubsystem.dataOutPort.receiveData
noname of : Data from dataProcessingSubsystem.dataInPort.sendData to SurveillanceSubsystem.dataOutPort.receiveData
noname of : Data from dataProcessingSubsystem.dataInPort.sendData to TrackingSubsystem.dataOutPort.receiveData
parts
^EarthObservationSubsystem
^EnvironmentMonitoringSubsystem
^SurveillanceSubsystem
^TrackingSubsystem
^dataProcessingSubsystem
«port» dataInPort: DataInPort
«port» signalOutPort: SignalOutPort
«part»
AirQualitySubsystem
attributes
pollutantDetectionLimit
«port» signalInPort: SignalInPort
«part»
DisasterMonitoringSubsystem
attributes
responseTime
flows
noname of : Data from DisasterMonitoringSubsystem.dataOutPort.receiveData to dataProcessingSubsystem.dataInPort.receiveData
parts
^DisasterMonitoringSubsystem
^dataProcessingSubsystem
«port» signalInPort: SignalInPort
«port» dataOutPort: DataOutPort
«part def»
HighAltitudePlatformSystem
«part»
payload
«part»
TrackingSubsystem
attributes
trackingAccuracy
flows
noname of : Data from TrackingSubsystem.dataOutPort.receiveData to dataProcessingSubsystem.dataInPort.receiveData
parts
^TrackingSubsystem
^dataProcessingSubsystem
«port» signalInPort: SignalInPort
«port» dataOutPort: DataOutPort
«part»
SurveillanceSubsystem
flows
noname of : Data from SurveillanceSubsystem.dataOutPort.receiveData to dataProcessingSubsystem.dataInPort.receiveData
parts
^SurveillanceSubsystem
^dataProcessingSubsystem
«port» signalInPort: SignalInPort
«port» dataOutPort: DataOutPort
«part»
EnvironmentMonitoringSubsystem
flows
noname of : Data from EnvironmentMonitoringSubsystem.dataOutPort.receiveData to dataProcessingSubsystem.dataInPort.receiveData
parts
^EnvironmentMonitoringSubsystem
^dataProcessingSubsystem
«port» signalInPort: SignalInPort
«port» dataOutPort: DataOutPort
«part»
EarthObservationSubsystem
flows
noname of : Data from EarthObservationSubsystem.dataOutPort.receiveData to dataProcessingSubsystem.dataInPort.receiveData
parts
^EarthObservationSubsystem
^dataProcessingSubsystem
«port» signalInPort: SignalInPort
«port» dataOutPort: DataOutPort
1

Figure 5.17: Internal System Structure: Graphical Notation: White Box

the part definition safety, and the value given is 16000 as the HAP altitude margin to be kept for safety performance, as stated in the non-functional stakeholders' needs table. The stable time for the HAP is defined to be 2 h in the initial state. So,

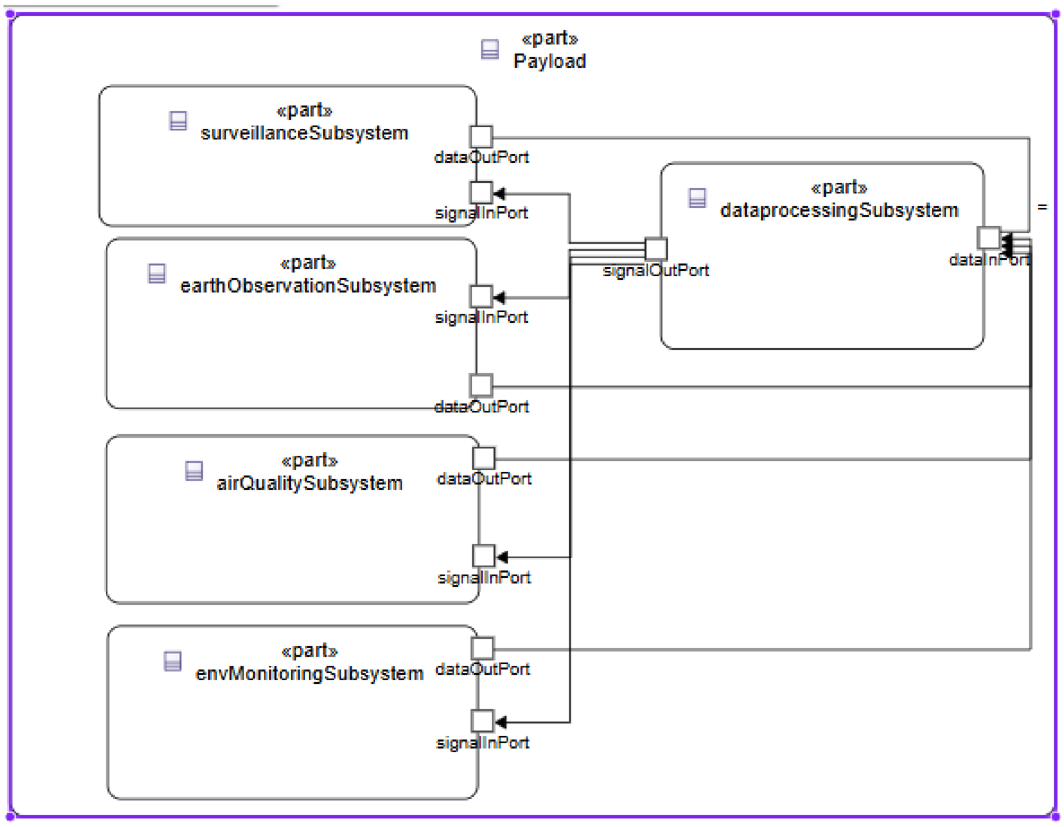

«part» Payload
«part» surveillanceSubsystem
dataOutPort
signalInPort
«part» earthObservationSubsystem
signalInPort
dataOutPort
«part» airQualitySubsystem
dataOutPort
signalInPort
«part» envMonitoringSubsystem
dataOutPort
signalInPort
«part» dataprocessingSubsystem
signalOutPort
dataInPort

Figure 5.18: Internal System Structure Diagram Using SysON

now the value assigned is two hours. These MoEs, which are the value properties, can be reused in other parts of the system by redefining the values or reusing the values. The diagram of the measure of effectiveness is shown in the figure Fig. 5.19; As safety in this model is refined from the stakeholder needs in SN3, where the flight altitude detector is stated to check whether the system maintains the required altitude to function in the task.

5.3 SOLUTION LEVEL

After defining the problem domain, the solution level can be derived. In the proposed approach, the solution architecture is composed of four key elements: component requirements, component behaviour, component structure, and component parameters. In this scenario, a component is referred to from a top-level system view down to as detailed a level as the system complexity demands [17]. The main goal of the solution architecture is to deliver a model of HAP that satisfies the requirement defined in the problem domain. This approach ensures traceability from the problem definition through to the detailed white box design, aligning structure and behaviour with intended functionality and performance expectations.

ID	Stakeholders Needs	Text
SN3	Safety	The stakeholder requires the HAP system to incorporate a dedicated safety subsystem responsible for maintaining the HAP operational altitude and monitoring flight conditions.
SN3.1	Flight Altitude Detector	The stakeholder requires the HAP to operate at an altitude well above civilian air traffic, within the lower stratosphere, to avoid potential collisions and enhance flight safety.
SN3.2	Flight Vertical Separation	The stakeholder needs the HAP to maintain a vertical separation of 16,000 feet above commercial air traffic.
SN3.3	Flight Altitude Time	The stakeholder needs the HAP to sustain this altitude with stability for 2 hours duration.

Table 5.2: HAP System Safety Requirements

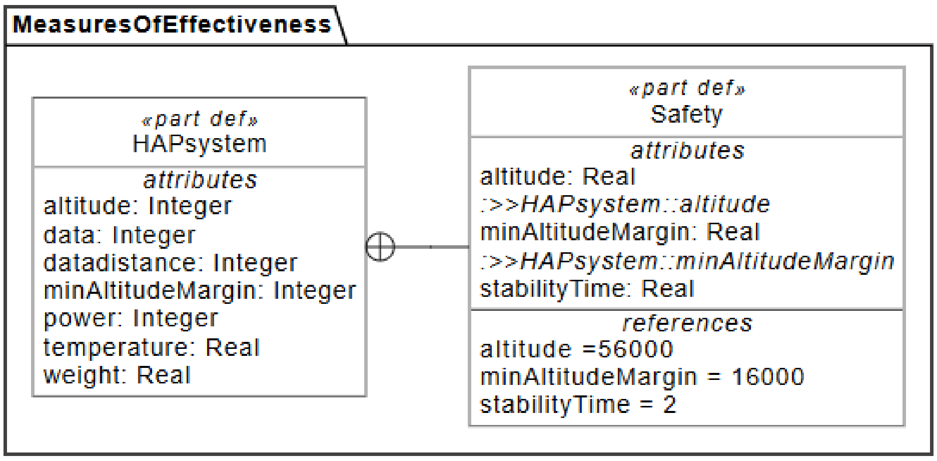


Figure 5.19: Measures Of Effectiveness: White Box

5.3.1 *Component Requirements*

The component requirement is the requirement diagram in the solution level of the abstraction layer and the requirement pillar. The component requirement diagram is built with the SysML V2 pilot implementation using Jupyter Lab. This requirement is refined from the system requirement in the white box, which is refined from the stakeholders' needs in the black box. In this diagram, the requirements are subdivided into payload subsystem CR1 and structural subsystem CR2 from the requirement diagram. As stated above, the SysML V2 textual notation is used to build this diagram. The payload subsystem is refined from the functional requirement in SR1 of the system requirement, which is refined from the user need SN1 from the stakeholders' needs. Thus, traceability is maintained in the diagram. This is one of the main advantages of using the MBSE Magic Grid. In the textual notation it is clearly stated which of all the requirements from the system requirements are satisfied.

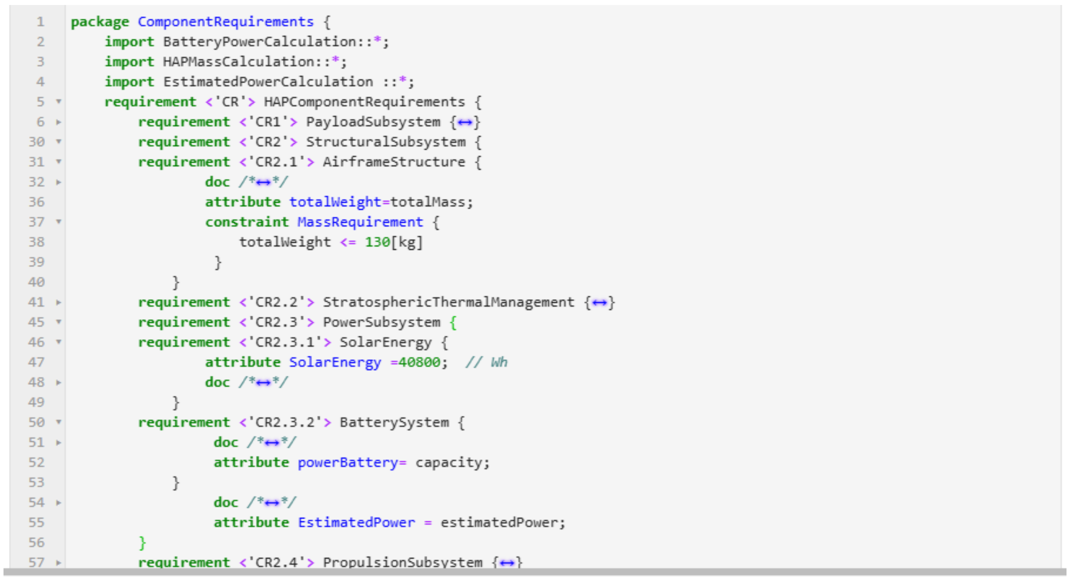
requirements in sysml v2 textual notation In SysML V2, a requirement definition is a special form of constraint definition that captures the intended conditions or capabilities of a system that must be satisfied [8]. The formal specification of requirements using attributes, features, and constraints ensures clarity and precision. Similarly to constraints, the requirements can be used to show attributes such as the total weight and actual total mass calculated, and attributes such as power battery and capacity are calculated using the evaluation of expression; thus, the requirement can be parameterised. The requirement can also be used to include a documentation comment to provide a human-readable description of the requirement that provides understanding and communication[8]. Furthermore, the requirement can be formalised by specifying required constraints, which express the necessary conditions, such as ensuring the total weight does not exceed the required weight.

payload subsystem The CR1 is the payload subsystem for the requirements of the HAP component. The CR1 is subdivided into an imaging subsystem, a radar system, a spectrometer subsystem, and a sensor. In the requirement CR1.1 for the imaging system, the requirement contains the text documentation section, where it explains what the system is intended to do in this requirement. Here, the documentation states that the HAP system will use a high-resolution optical camera, which will scan up to a 400-square-kilometre resolution, and the task of this is to record environmental changes. Thus, this requirement satisfies SR1.1 earth observation and SR1.2 environmental monitoring. This is refined from SN1.1 and SN1.2 in the stakeholder needs, which come under the user needs in SN1.1. Thus, traceability is maintained. The figure Fig. 5.20 shows the textual notation for the component requirement at the solution level. The CR1 payload subsystem is subdivided into the CR1.2 radar system, the CR1.3 spectrometer system, and the CR1.4 sensor, and each component requirement is refined from the corresponding system requirement from the white box, which is refined from the stakeholders' needs from the black box, thus maintaining traceability as shown in the figure Fig. 5.20.

structural subsystem The next CR2 is the structural subsystem, which contains the sub-requirements such as CR2.1 as the airframe structure, CR2.2 stratospheric thermal management, CR2.3 power subsystem, and CR2.4 the propulsion subsystem. The airframe structure is refined from the system requirements stated in SR2.1, the lightweight structure of HAP, SR2.7, the wingspan limit, and SR2.8 platform length constraint. Here, three requirements are satisfied in the requirement. This requirement states that the total weight of the airframe structure should not exceed more than 130 kg. The constraint for the mass requirement stated in the component requirement CR2.1 is 130 kg for the 'total weight' The attribute 'total weight' is assigned to the imported attribute 'total mass', which comes from the HAPMassCalculation package. In this package, the calc definition calculation mass calculates the actual value of the total mass for the HAP, so in this package, the actual values are entered and then assigned to the component requirement package, thereby If the 'total mass' actually calculated is 130 kg, thereafter this value is assigned to the total weight in the CR2.1, and the constraint Mass requirement is checked. The constraint MassRequirement in CR2.1 states that the total weight should be less than or equal to 130 kg for the HAP. With the help of the SysML V2 pilot implementation using Jupyter Lab using textual notation, it is easier to calculate the values of the attributes and assign and check in the requirement itself, thereby getting efficiency in the earlier stages of development. With the help of textual notation, which is similar to text-based script, it is very helpful to cheque constraints, do calculations, and express using textual notation compared to a graphical diagram. The figure Fig. 5.24 shows the calculation of the mass as explained.

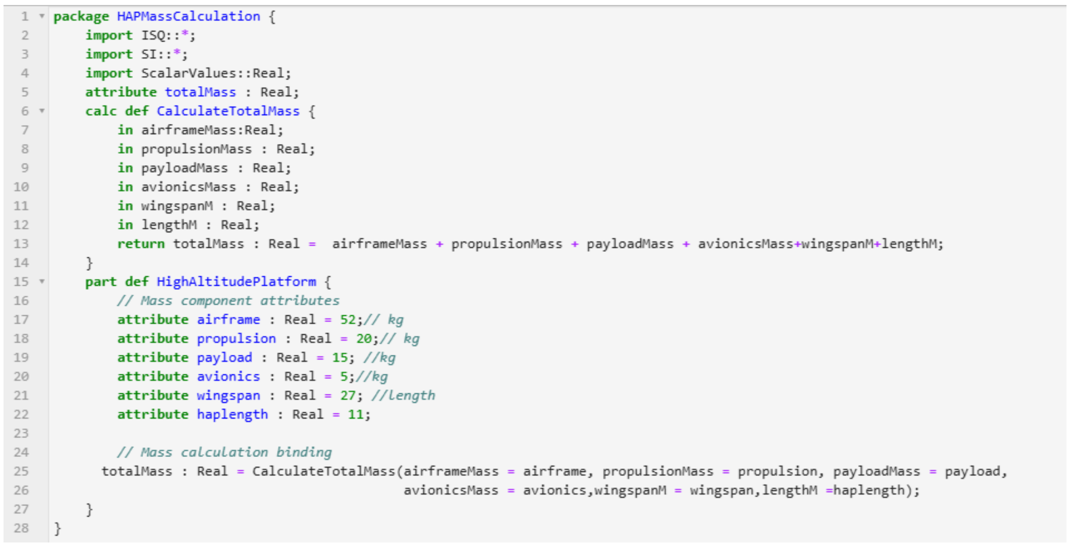
POWER SUBSYSTEM In the component requirement textual notation as shown in the figure Fig. 5.20, the power subsystem CR2.2 is subdivided into CR2.3.1 Solar energy and CR2.3.2 Battery system. The requirement is refined from the system requirements SR2.2 and SR2.3, which are the solar energy subsystem and battery backup subsystem, which are refined from the stakeholders' needs. SN1.2.2 and SN1.2.3: photovoltaic energy and battery energy. Here, the traceability of the diagram is maintained. Next, the value for the attribute battery power is calculated using the package battery power calculation, and the result is assigned to the attribute powerBattery in the CR2.3.2, and similarly Estimated power, which is the solar energy per day, is multiplied by the battery capacity to find the estimated power of the battery system. This is done with the package estimate power calculation, and the resulting value 'estimatedPower' is assigned to the CR2.3.2 battery system, thereby verifying the accuracy of the system. With the calculation of expression in textual notation in SysML V2, it helps in the verification in the earlier stages of development, and with text-based scripting, it is comparatively easier to edit and do the calculation as compared to graphical notation in SysML V1. Figure Fig. 5.22 shows the battery power calculation, and Figure Fig. 5.23 shows the estimated power system.

EVALUATION OF AN EXPRESSION In the context of SysML, V2 refers to the mechanism by which values in constraint or requirement definitions are calculated or validated. Evaluation of expression is a new feature in SysML V2 to do calculations in the textual notation, thereby giving easier practice for validation and analysis in the earlier stages of software development. This typically refers to the process of computing the result of an expression in programming or mathematical contexts. In SysML V2, the evaluation of an expression refers to computing the result of a defined calculation using provided input values [8]. The given example illustrates a package named 'BatteryPowerCalculation' as shown in the figure Fig. 5.22. The attribute values capacity, real, nightHours, and avgPowerDraw are assigned to scalarValue real from the library. In the package Battery power calculation, the cal def function required Battery capacity is used to calculate the value capacity, where nighthours is assigned to 12 hours and avgDraw is assigned 4166 watts. And the calculated value is assigned to capacity, and the capacity attribute is returned to the function call in the component requirement. For getting the result, the %eval `BatteryPowerCalculation::BatteryExample::capacity` = 49992 is used here. The command %eval is used to evaluate the expression. This is a new feature in SysML V2 as compared to V1, thereby getting a better experience in the modelling of the diagrams. The figure Fig. 5.24 shows the component requirements in graphical notation, and the visualisation keyword is used to visualise the diagram and the %viz along with the STYLE option that gives standard colour to the output generated. The STYLE option can be used to generate multiple styling options for the graphical representation.



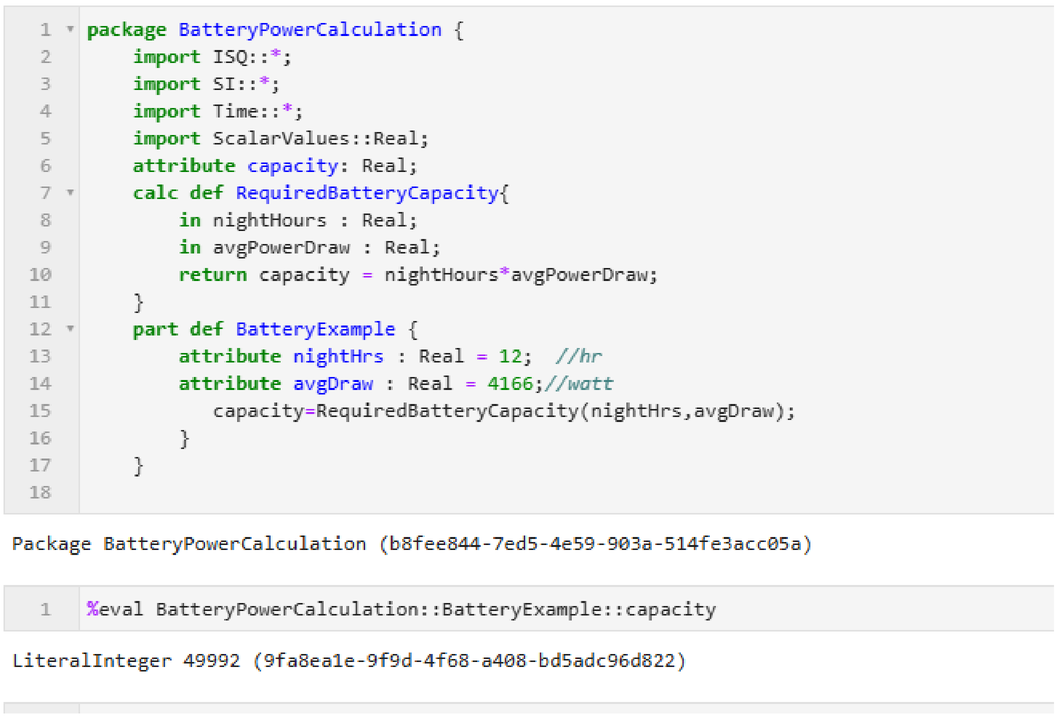
```
package ComponentRequirements {
    import BatteryPowerCalculation::*;
    import HAPMassCalculation::*;
    import EstimatedPowerCalculation ::*;
    requirement <'CR'> HAPComponentRequirements {
        requirement <'CR1'> PayloadSubsystem {↔}
        requirement <'CR2'> StructuralSubsystem {
        requirement <'CR2.1'> AirframeStructure {
                doc /*↔*/
                attribute totalWeight=totalMass;
                constraint MassRequirement {
                    totalWeight <= 130[kg]
                 }
            }
        requirement <'CR2.2'> StratosphericThermalManagement {↔}
        requirement <'CR2.3'> PowerSubsystem {
        requirement <'CR2.3.1'> SolarEnergy {
                attribute SolarEnergy =40800;  // Wh
                doc /*↔*/
            }
        requirement <'CR2.3.2'> BatterySystem {
                 doc /*↔*/
                 attribute powerBattery= capacity;
            }
                 doc /*↔*/
                 attribute EstimatedPower = estimatedPower;
        }
        requirement <'CR2.4'> PropulsionSubsystem {↔}
```

Figure 5.20: Component Requirement: Textual Notation



```
package HAPMassCalculation {
    import ISQ::*;
    import SI::*;
    import ScalarValues::Real;
    attribute totalMass : Real;
    calc def CalculateTotalMass {
        in airframeMass:Real;
        in propulsionMass : Real;
        in payloadMass : Real;
        in avionicsMass : Real;
        in wingspanM : Real;
        in lengthM : Real;
        return totalMass : Real =  airframeMass + propulsionMass + payloadMass + avionicsMass+wingspanM+lengthM;
    }
    part def HighAltitudePlatform {
        // Mass component attributes
        attribute airframe : Real = 52;// kg
        attribute propulsion : Real = 20;// kg
        attribute payload : Real = 15; //kg
        attribute avionics : Real = 5;//kg
        attribute wingspan : Real = 27; //length
        attribute haplength : Real = 11;

        // Mass calculation binding
      totalMass : Real = CalculateTotalMass(airframeMass = airframe, propulsionMass = propulsion, payloadMass = payload,
                                            avionicsMass = avionics,wingspanM = wingspan,lengthM =haplength);
    }
}
```

Figure 5.21: Mass Calculation: Textual Notation



```
package BatteryPowerCalculation {
    import ISQ::*;
    import SI::*;
    import Time::*;
    import ScalarValues::Real;
    attribute capacity: Real;
    calc def RequiredBatteryCapacity{
        in nightHours : Real;
        in avgPowerDraw : Real;
        return capacity = nightHours*avgPowerDraw;
    }
    part def BatteryExample {
        attribute nightHrs : Real = 12;  //hr
        attribute avgDraw : Real = 4166;//watt
           capacity=RequiredBatteryCapacity(nightHrs,avgDraw);
        }
    }

```

```
Package BatteryPowerCalculation (b8fee844-7ed5-4e59-903a-514fe3acc05a)
```

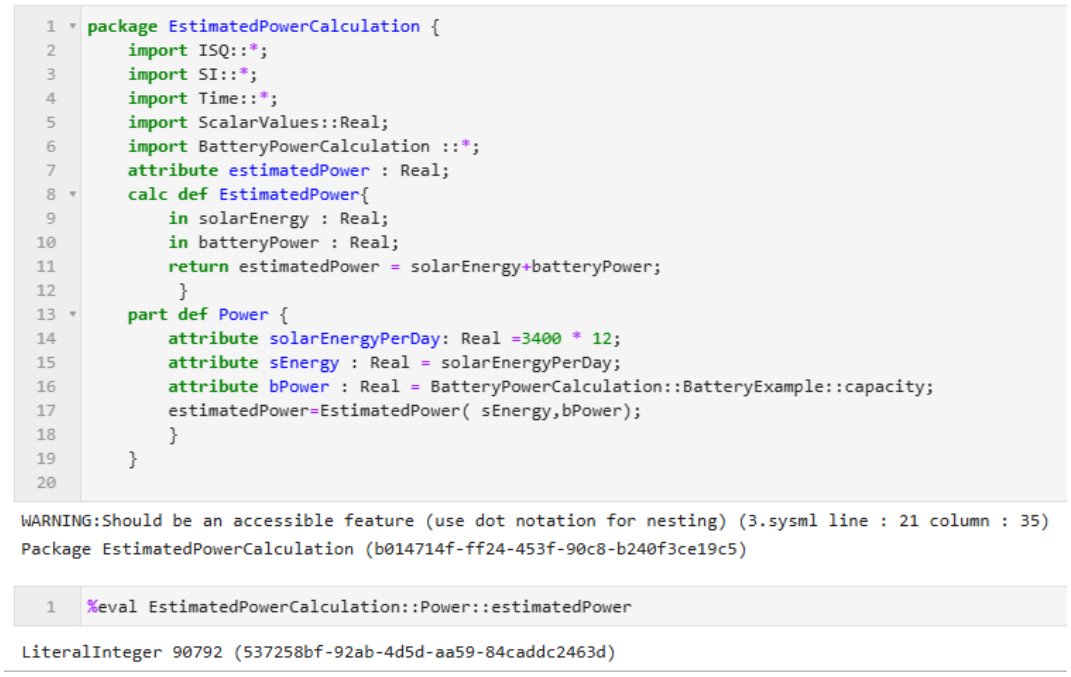
```
%eval BatteryPowerCalculation::BatteryExample::capacity
```

```
LiteralInteger 49992 (9fa8ea1e-9f9d-4f68-a408-bd5adc96d822)
```

Figure 5.22: Battery Power Calculation: Textual Notation

5.4 COMPONENT BEHAVIOR

Component behaviour refers to the way a system's components respond to input and events or changes in their environment over time. This answers one or more questions about their behaviour or physical characteristics. In component



```
package EstimatedPowerCalculation {
    import ISQ::*;
    import SI::*;
    import Time::*;
    import ScalarValues::Real;
    import BatteryPowerCalculation ::*;
    attribute estimatedPower : Real;
    calc def EstimatedPower{
        in solarEnergy : Real;
        in batteryPower : Real;
        return estimatedPower = solarEnergy+batteryPower;
         }
    part def Power {
        attribute solarEnergyPerDay: Real =3400 * 12;
        attribute sEnergy : Real = solarEnergyPerDay;
        attribute bPower : Real = BatteryPowerCalculation::BatteryExample::capacity;
        estimatedPower=EstimatedPower( sEnergy,bPower);
        }
    }

WARNING:Should be an accessible feature (use dot notation for nesting) (3.sysml line : 21 column : 35)
Package EstimatedPowerCalculation (b014714f-ff24-453f-90c8-b240f3ce19c5)
```

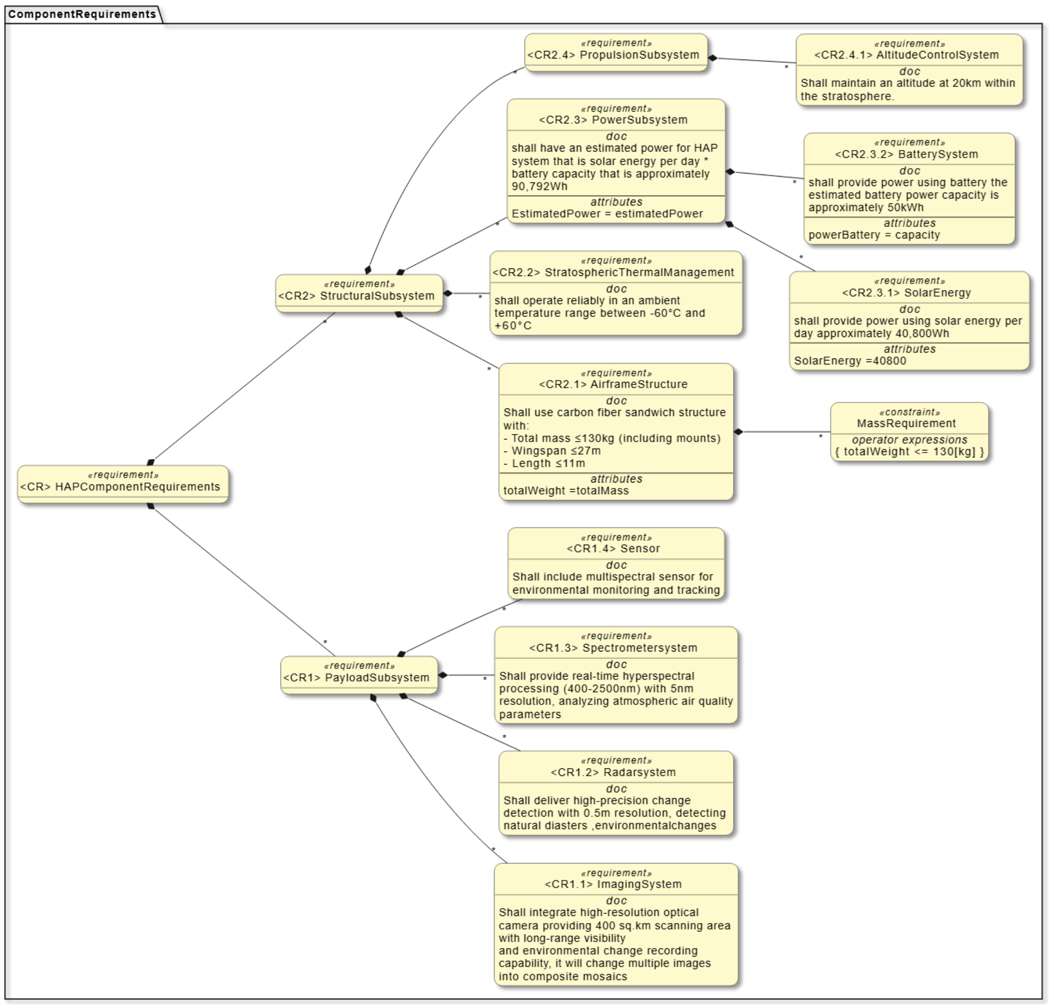
```
%eval EstimatedPowerCalculation::Power::estimatedPower

LiteralInteger 90792 (537258bf-92ab-4d5d-aa59-84caddc2463d)
```

Figure 5.23: Estimated Power Calculation: Textual Notation

SysML V2, state machine diagrams and action diagrams are used to represent this behaviour visually. Here, dynamic aspects of the component, transitions between different states, perform actions, or interact with other components. This helps developers to design, simulate, validate, and refine the components' logic before implementation.

5.4.1 *State machine*

The state machine comes in the cell of the component behaviour, which is in the solution layer of the magic grid and the behaviour pillar of the MBSE magic grid. The state machine diagram is implemented using the SysML V2 pilot implementation using Jupyter Lab. The SysML V2 textual notation is used for creating this state machine, as shown in figure Fig. 5.25. From the textual notation, attribute definition is defined for IStartSignal, IOnSignal, and IOffSignal. The attribute definition is a definition of attribute data that can be used to describe systems or parts [8].Here, state definition is defined for image system states. The state definition is the A state definition is like a state machine in UML and SysML V1. It defines a behavioural state that can be exhibited by a system [8]. In image scanning done by the HAP, the system is initially in the off state, and then the system transitions from the off state to the starting state. The states in the state machine diagram are connected by transitions that are fired on the acceptance of item transfers. Upon transition from IStartSignal to IOnSignal, the state starts action. An action starts with an entry action on an entry to the state, and next



Shall maintain an altitude at 20km within the stratosphere.

shall have an estimated power for HAP system that is solar energy per day * battery capacity that is approximately 90,792Wh

shall provide power using battery the estimated battery power capacity is approximately 50kWh

shall operate reliably in an ambient temperature range between -60°C and +60°C

shall provide power using solar energy per day approximately 40,800Wh

Shall use carbon fiber sandwich structure with:
- Total mass ≤130kg (including mounts)
- Wingspan ≤27m
- Length ≤11m

Shall include multispectral sensor for environmental monitoring and tracking

Shall provide real-time hyperspectral processing (400-2500nm) with 5nm resolution, analyzing atmospheric air quality parameters

Shall deliver high-precision change detection with 0.5m resolution, detecting natural diasters ,environmentalchanges

Shall integrate high-resolution optical camera providing 400 sq.km scanning area with long-range visibility
and environmental change recording capability, it will change multiple images into composite mosaics

Figure 5.24: Component Requirements: Graphical Notation

is to do the action while it is in it. and then an exit action is performed to exit from the state [8].The initialisation of the camera takes place, then taking action capture images is the next step, then the next step is exiting action record data, then the action is exited, and the transition accepts the IoffSignal state. Thus, using state machines for modelling supports formal analysis and simulation, allowing for early validation of system responses before actual implementation. The figure Fig. 5.26 shows the graphical notation of the state machine diagram.

5.4.2 *Activity Diagram*

The activity diagram comes as the second diagram in the component behaviour cell in the behaviour pillar and the solution layer in the layer of abstraction. This diagram helps to show the flow of action and data between different parts of the HAP system. Here, to show the action flow diagram, the requirement CR1.1

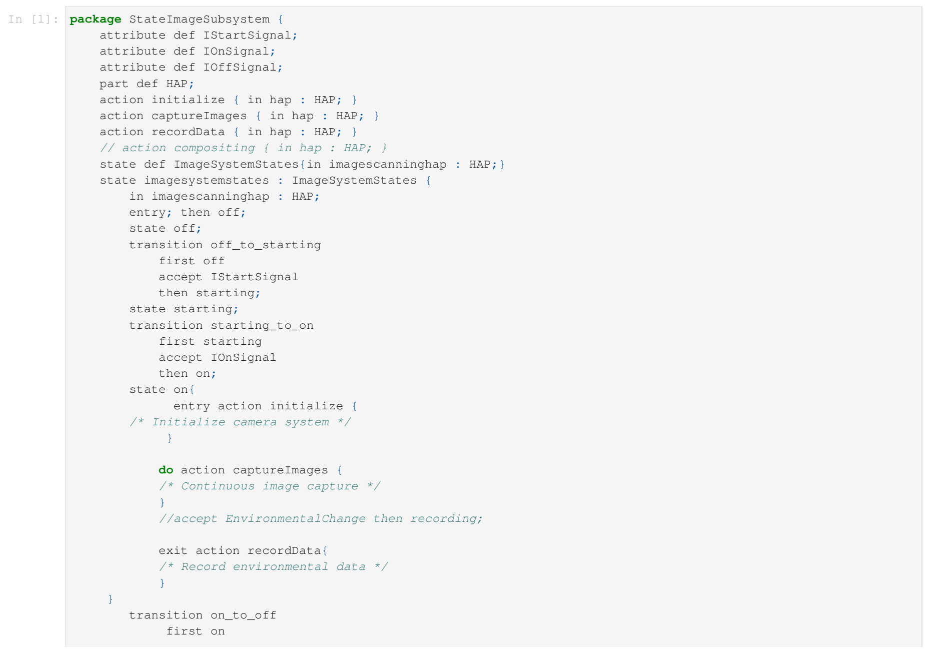

```
In [1]: package StateImageSubsystem {
            attribute def IStartSignal;
            attribute def IOnSignal;
            attribute def IOffSignal;
            part def HAP;
            action initialize { in hap : HAP; }
            action captureImages { in hap : HAP; }
            action recordData { in hap : HAP; }
            // action compositing { in hap : HAP; }
            state def ImageSystemStates{in imagescanninghap : HAP;}
            state imagesystemstates : ImageSystemStates {
                in imagescanninghap : HAP;
                entry; then off;
                state off;
                transition off_to_starting
                    first off
                    accept IStartSignal
                    then starting;
                state starting;
                transition starting_to_on
                    first starting
                    accept IOnSignal
                    then on;
                state on{
                      entry action initialize {
                /* Initialize camera system */
                     }

                    do action captureImages {
                    /* Continuous image capture */
                    }
                    //accept EnvironmentalChange then recording;

                    exit action recordData{
                    /* Record environmental data */
                    }
             }
                transition on_to_off
                     first on
```

Figure 5.25: State Machine: Textual Notation

Imaging System is taken into account from the component requirement. Here, the requirement states that the HAP system shall integrate a high-resolution optical camera providing a 400 sq. km scanning area with long-range visibility and environmental change recording capability; it will change multiple images into composite mosaics. This is the requirement stated in the CR1.1 of the component requirement. This requirement can be shown using the action diagram in this scenario. The action flow diagram is created using the SysML V2 SysON tool. The action 'user' starts with 'Requestenvdata' that action flows to HAP system 'Intialize system ' then the next action executed is 'starting system' then the action 'send signal' is sended from the HAP system to the Payload system in this the payload action 'start system' is intiated then next action 'Request for environmental data' is generated next the action is flow to the payload camera in the payload camera the action 'capture image' take place then next steps are 'high resolution optical capture' then next action 'scanning location 400 sq.km' then 'setting long range visibility' then the action 'monitor environment for natural disaster' then in the decision node the decision is taken whether the system detect changes such as natural disasters, If a natural disaster occurs, the data is recorded, then the process images take place, then the image mosaicking action takes place, then the record image information is sent to the payload system and from there to HAP and then to the user in the ground station. Thus the requirement CR1.1 is detailed using the action flow diagram, thereby proving traceability is maintained.

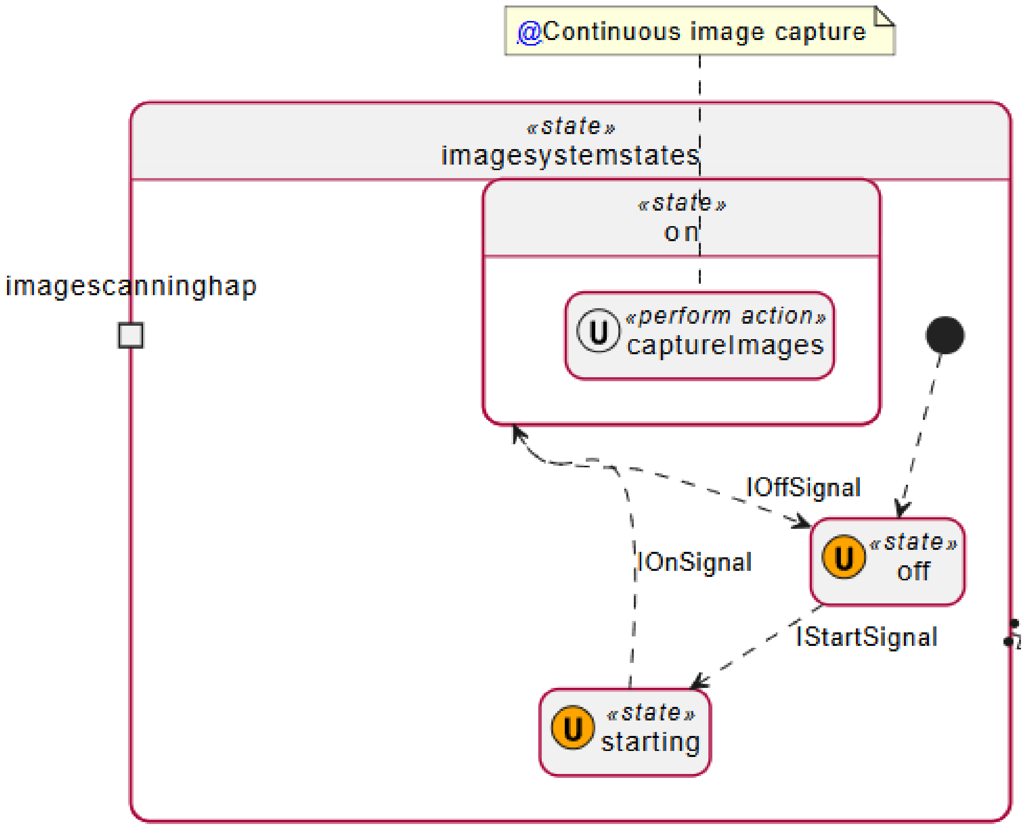


Figure 5.26: State Machine: Graphical Notation

The action flow diagram is shown in Figure Fig. 5.27; it shows the graphical notation. The representation is implemented using the SysON tool.

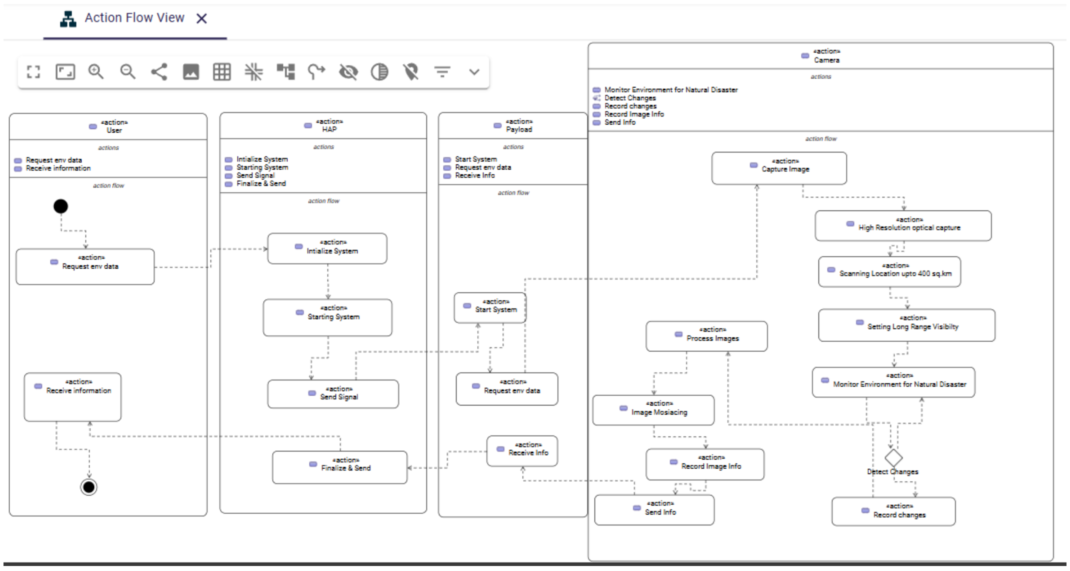


Figure 5.27: Action Flow Diagram: Graphical Notation

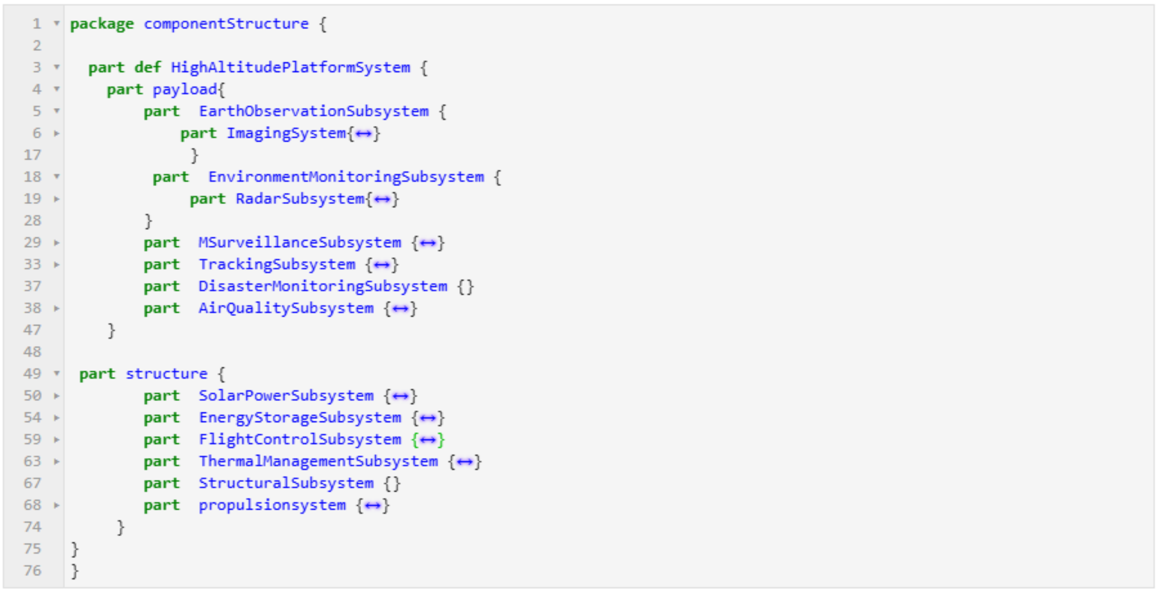
5.5 COMPONENT STRUCTURE

The component structure comes in the solution level of the layer of abstraction and the structure pillar. There are two diagrams to represent this layer system structure and the internal system structure. Here, the main focus is on modelling the HAP physical architecture, tightly aligned with the component requirements.

5.5.1 *System Component Structure*

The system component structure is the first level of diagram in the component structure cell. The code of the component system structure is shown in the figure Fig. 5.28. It is developed using textual notation in the SysML V2 pilot implementation using Jupyter Lab. A system component structure diagram is developed to show the internal parts of HAP and their functions. The hierarchy of the HAP and the components that come under each subdivision are neatly aligned, helping the developers to start the development of the model in the initial stage. The diagram is subdivided into two primary domains: the payload and the structure. This is refined from the component requirement diagram, hence maintaining traceability. The component requirements are refined from the problem domains in the black box, thereby maintaining traceability, as explained in the MBSE Magic Grid methodology. Here the payload is further subdivided into different parts that perform the task in the HAP, such as earth observation, environmental monitoring, target detection, and signal transmission.

Whereas the structure domain supports the operation of the payload by providing it HAP with airframe structure such as energy generation, propulsion, flight control subsystem, and thermal management regulation. For more clarity of the system component diagram, actions performed by most of the parts are shown in figure Fig. 5.29. From the figure, the earth observation subsystem has a subpart as an imaging system, and the action performed by this imaging system is recording and scanning. Similarly, scanning the environment. The payload subsystem has a part radar subsystem. And the required attributes, such as a signal and target position, shown in the diagram, can be used to show the action performed by each port. The graphical notation of the diagram is achieved from the textual notation. The diagram shown in figure 5.29 is created using `%viz -view TREE -style LR -style ORTHOLINE componentStructure` The command`%viz` is used to visualise the HAP. It can be viewed in the tree view that is one of the new features in SysML V2. Along with it, the diagram can be styled from left to right, enhancing readability for the model, and finally, the name of the package is given to view the diagram in SysML V2.



```
package componentStructure {

  part def HighAltitudePlatformSystem {
    part payload{
        part  EarthObservationSubsystem {
            part ImagingSystem{↔}
             }
         part  EnvironmentMonitoringSubsystem {
             part RadarSubsystem{↔}
        }
        part  MSurveillanceSubsystem {↔}
        part  TrackingSubsystem {↔}
        part  DisasterMonitoringSubsystem {}
        part  AirQualitySubsystem {↔}
    }

 part structure {
        part  SolarPowerSubsystem {↔}
        part  EnergyStorageSubsystem {↔}
        part  FlightControlSubsystem {↔}
        part  ThermalManagementSubsystem {↔}
        part  StructuralSubsystem {}
        part  propulsionsystem {↔}
     }
}
}
```

Figure 5.28: System Component Structure: Textual Notation

5.5.2 *Internal System Structure*

The second diagram in the component structure cell is the internal system structure. This diagram is implemented using textual notation in SysML V2 pilot implementation using Jupyter Lab. This diagram, as shown in Figure Fig. 5.31, explains the internal system structure. This diagram is used to show the internal composition of a system element; it includes the part, port and connections. The textual notation as shown in the figure Fig. 5.31 is used to implement the graphical representation in SysML V2. The component structure is used to define the hierarchical breakdown of a system into components and subcomponents, and the connections between the subsystem parts are established using ports [8]. One of the feature of the MBSE magic grid is to maintain the traceability, The subsystem part HAP is subdivided into the payload, as this is refined from the component requirement in CR1, and this is refined from the system requirement in SR1. Thus, traceability of the system is maintained. As the system component structure diagram is, even though bigger here, only one part of the diagram is visualised. As different parts of the system can be visualised using the command `%viz`. The payload is again subdivided into the earth observation system, and the earth observation system contains subsystem parts such as EODataProcessor, a multispectral camera, and a high-resolution camera. Each subpart is connected using the ports, and through the ports, a flow of connection is established. The ports used are the signal-in port and signal-out port. And the image in the port and the image out of the port. The subsystem receives the signal from the EO data processor, and it receives image input from other subsystems. After connecting each subsystem with relevant ports, the flow of connection is shown. The flow of

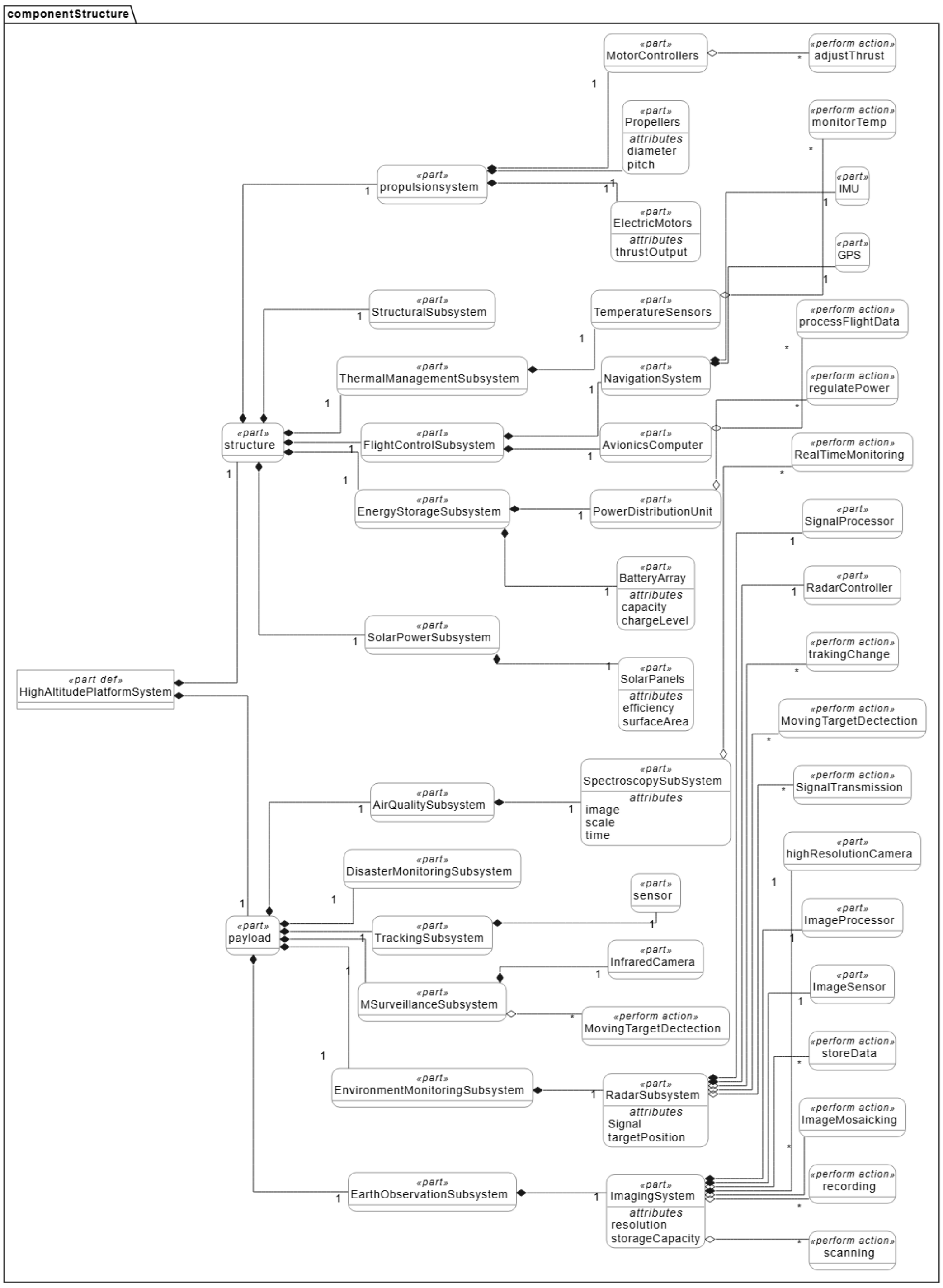
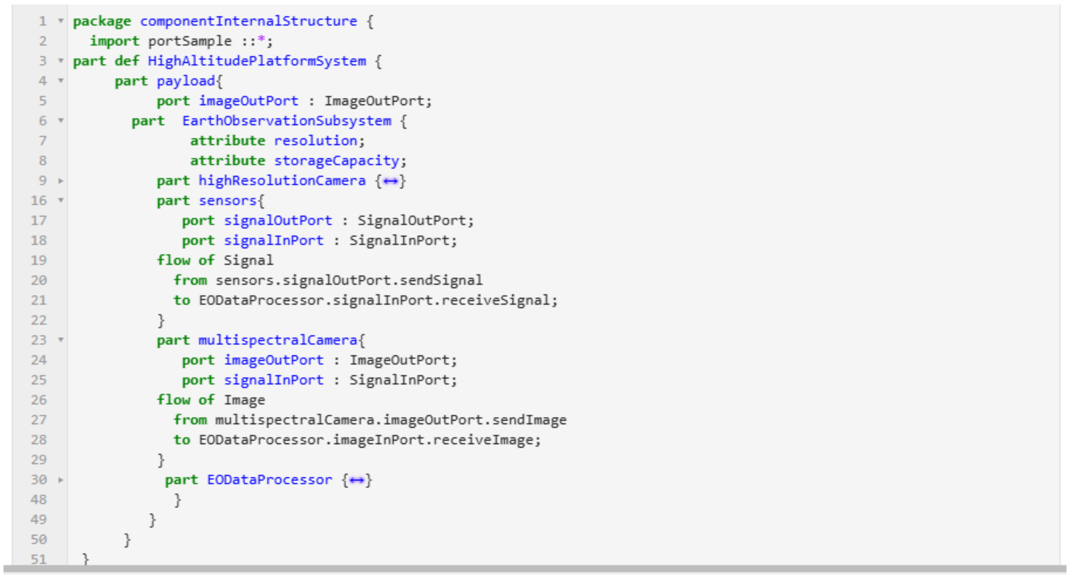

componentStructure
«part» MotorControllers
«perform action» adjustThrust
«part» Propellers attributes diameter pitch
«perform action» monitorTemp
«part» propulsionsystem
«part» IMU
«part» ElectricMotors attributes thrustOutput
«part» GPS
«part» StructuralSubsystem
«part» TemperatureSensors
«perform action» processFlightData
«part» ThermalManagementSubsystem
«part» NavigationSystem
«perform action» regulatePower
«part» structure
«part» FlightControlSubsystem
«part» AvionicsComputer
«perform action» RealTimeMonitoring
«part» EnergyStorageSubsystem
«part» PowerDistributionUnit
«part» SignalProcessor
«part» BatteryArray attributes capacity chargeLevel
«part» RadarController
«part» SolarPowerSubsystem
«perform action» trakingChange
«part» SolarPanels attributes efficiency surfaceArea
«part def» HighAltitudePlatformSystem
«perform action» MovingTargetDectection
«part» SpectroscopySubSystem attributes image scale time
«part» AirQualitySubsystem
«perform action» SignalTransmission
«part» highResolutionCamera
«part» DisasterMonitoringSubsystem
«part» sensor
«part» ImageProcessor
«part» payload
«part» TrackingSubsystem
«part» InfraredCamera
«part» ImageSensor
«part» MSurveillanceSubsystem
«perform action» MovingTargetDectection
«perform action» storeData
«part» EnvironmentMonitoringSubsystem
«part» RadarSubsystem attributes Signal targetPosition
«perform action» ImageMosaicking
«perform action» recording
«part» EarthObservationSubsystem
«part» ImagingSystem attributes resolution storageCapacity
«perform action» scanning

Figure 5.29: System Component Structure: Graphical Notation

connection is from the EOData processor to other subsystems and vice versa, and this can be illustrated in the given diagram.



```
package componentInternalStructure {
  import portSample ::*;
part def HighAltitudePlatformSystem {
     part payload{
          port imageOutPort : ImageOutPort;
       part  EarthObservationSubsystem {
              attribute resolution;
              attribute storageCapacity;
          part highResolutionCamera {↔}
          part sensors{
             port signalOutPort : SignalOutPort;
             port signalInPort : SignalInPort;
          flow of Signal
            from sensors.signalOutPort.sendSignal
            to EODataProcessor.signalInPort.receiveSignal;
          }
          part multispectralCamera{
             port imageOutPort : ImageOutPort;
             port signalInPort : SignalInPort;
          flow of Image
            from multispectralCamera.imageOutPort.sendImage
            to EODataProcessor.imageInPort.receiveImage;
          }
           part EODataProcessor {↔}
            }
         }
      }
 }
```

Figure 5.30: Internal System Structure: Textual Notation

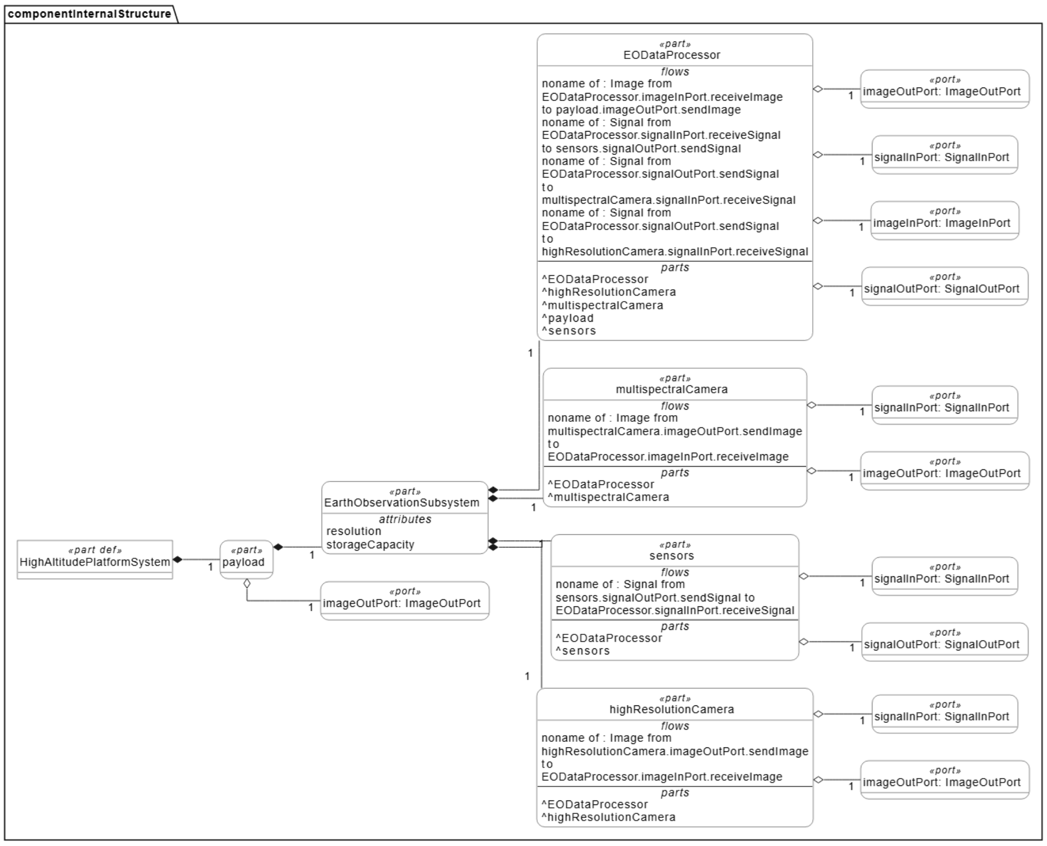

componentInternalStructure
«part def» HighAltitudePlatformSystem
«part» payload
«part» EarthObservationSubsystem
attributes
resolution
storageCapacity
«port» imageOutPort: ImageOutPort
«part» EODataProcessor
flows
noname of : Image from EODataProcessor.imageInPort.receiveImage to payload.imageOutPort.sendImage
noname of : Signal from EODataProcessor.signalInPort.receiveSignal to sensors.signalOutPort.sendSignal
noname of : Signal from EODataProcessor.signalOutPort.sendSignal to multispectralCamera.signalInPort.receiveSignal
noname of : Signal from EODataProcessor.signalOutPort.sendSignal to highResolutionCamera.signalInPort.receiveSignal
parts
^EODataProcessor
^highResolutionCamera
^multispectralCamera
^payload
^sensors
«port» imageOutPort: ImageOutPort
«port» signalInPort: SignalInPort
«port» imageInPort: ImageInPort
«port» signalOutPort: SignalOutPort
«part» multispectralCamera
flows
noname of : Image from multispectralCamera.imageOutPort.sendImage to EODataProcessor.imageInPort.receiveImage
parts
^EODataProcessor
^multispectralCamera
«port» signalInPort: SignalInPort
«port» imageOutPort: ImageOutPort
«part» sensors
flows
noname of : Signal from sensors.signalOutPort.sendSignal to EODataProcessor.signalInPort.receiveSignal
parts
^EODataProcessor
^sensors
«port» signalInPort: SignalInPort
«port» signalOutPort: SignalOutPort
«part» highResolutionCamera
flows
noname of : Image from highResolutionCamera.imageOutPort.sendImage to EODataProcessor.imageInPort.receiveImage
parts
^EODataProcessor
^highResolutionCamera
«port» signalInPort: SignalInPort
«port» imageOutPort: ImageOutPort
1

Figure 5.31: Internal System Structure: Graphical Notation

5.6 COMPONENT PARAMETERS

This cell contains the measure of effectiveness in the solution layer of the layer of abstraction and the parameters in the pillar. This is developed using the

SysML V2 pilot implementation using Jupyter Lab. In the solution layer, The measure of effectiveness is used to evaluate whether the HAP can satisfy the non-functional performance requirement defined in the white box. Here, safe operation at high altitude is taken into account, as explained above. To maintain traceability across the layers, the stakeholders' needs for the non-functional requirement SN3.1 are refined in the stakeholders needs table. To maintain reusability in SysML V2, the definition of parts is used, as it can be reused throughout the system, and the defined attributes can be redefined and used, giving it the opportunity to change the type of data as required. From the stakeholders needs table as shown in Table 5.3, the SN3 states that the HAP system should incorporate a dedicated safety subsystem responsible for maintaining the HAP operational altitude and monitoring flight conditions. Thus avoiding collisions and enhancing flight safety. A subcomponent, the Flight Altitude Controller, is expected to record the 'HAPaltitude' and ensure that the designated altitude is consistently maintained. This requirement, SN3.1, from the white-box level is refined with an altitude checker, where the actual altitude is recorded and cross-checked. The HAP must also maintain a vertical separation of at least 16,000 feet above commercial aircraft, according to the stakeholders' needs in SN3.2. This is refined by the Flight Vertical Separation checker in SN3.3, which uses an altitude adjuster capable of dynamically giving alerts if the separation is not maintained. Additionally, the 'flight altitude time' requirement in stakeholders needs specifies that the HAP should be capable of maintaining this high altitude stably for durations of at least two hours. And this requirement is refined at the solution level by adding a flight altitude time checker; it checks whether the time is maintained as required. The measure of effectiveness is shown using a SysML V2 diagram, as shown in the Figure Fig. 5.32. It is developed using textual notation, and the diagram is represented in graphical notation.

ID	**Name**	**Description**
SN3	Safety	The stakeholder requires the HAP system to incorporate dedicated safety subsystems responsible for maintaining the HAP operational altitude and monitoring flight conditions.
SN3.1	Flight Altitude Detector	The stakeholder requires the HAP to operate at an altitude well above civilian air traffic, within the lower stratosphere, to avoid potential collisions and enhance flight safety.
SN3.2	Flight Altitude Controller	The stakeholder requires the HAP to record the flying altitude and verify that the altitude is maintained.
SN3.3	Flight Vertical Separation	The stakeholder needs the HAP to maintain a vertical separation of 16,000 feet above commercial air traffic.
SN3.4	Flight Vertical Separation Checker	The stakeholder requires the HAP to maintain an altitude adjuster that dynamically triggers alerts if a violation is predicted.
SN3.5	Flight Altitude Time	The stakeholder needs the HAP to sustain this altitude with stability for a duration of 2 hours.
SN3.6	Flight Altitude Time Checker	The stakeholder requires the HAP to record the flying time and verify that the duration is maintained.

Table 5.3: Updated Stakeholder Needs Table

```
%viz --view TREE --style LR --style ORTHOLINE MeasuresOfEffectiveness
```

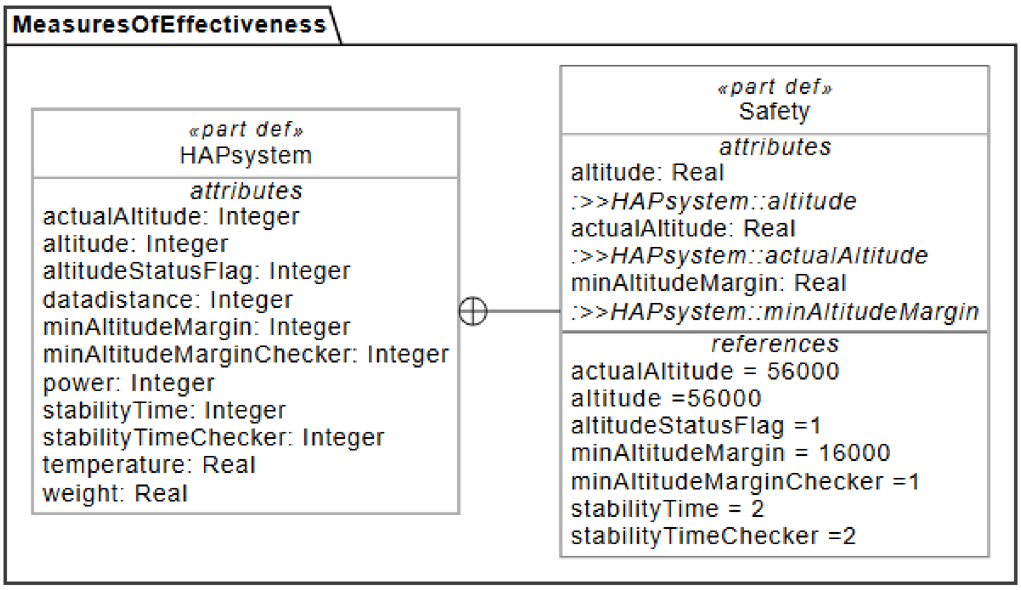


Figure 5.32: Measure Of Effectiveness: Graphical Notation

6 MODELLING CONSTRUCT SPECIFIC TO SYSML V2

6.1 SYSML V2 REUSE PATTERNS

6.1.1 *Definition and Usage*

The following modelling constructs are specific to SysML V2. These modelling constructs are new concepts in version V2 compared to version 1. It is described below: Part definition and part usage are new aspects of SysML V2 compared to V1 [20]. A definition element specifies the general concept of an element, such as a part, action, or requirement. In the SysML V2 modelling language, it is known as 'part definition', 'requirement definition', 'item definition' and so on. It serves as a reusable element. For example, in HAP, the part definition camera can be defined, and the camera can be reused and reassigned in different contexts of the modelling. Whereas the usage element is the usage or application of the element in the particular context; for example, if the camera is defined as 'part camera', then it is a part of HAP in that particular context, but if it is 'partdef camera', then the camera can be inherited or reused in different contexts across the modelling language. Likewise, an attribute defined with 'attribute definition' can be reused and redefined with other data types and values across the modelling language. For example, in the figure Fig. 5.32 for MoEs, the attribute 'altitude' is defined as type integer in HAP Systems, and then in the part definition for safety, the attribute is redefined to 'real' and assigned a value of 56,000 feet. This approach provides modularity, reusability, and clarity.

6.1.2 *Variability*

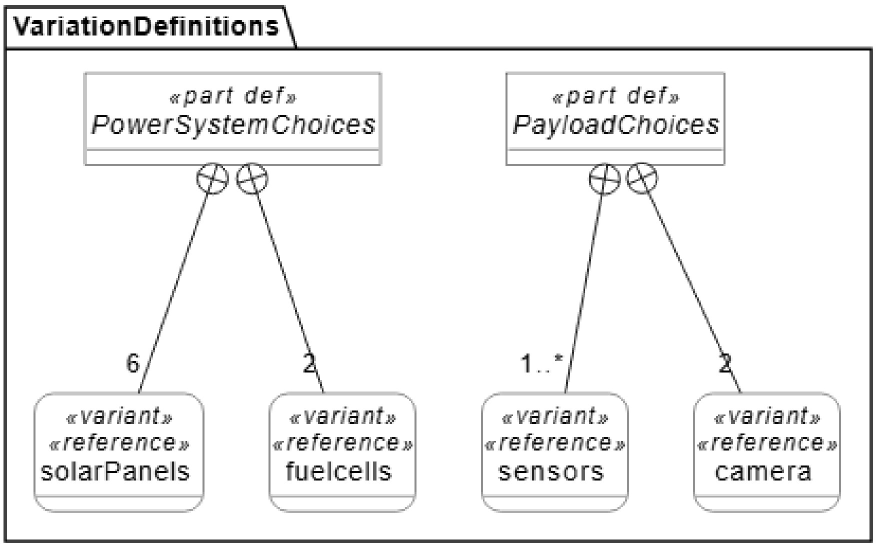
Variability modelling is a new aspect of SysML V2 to as compared to SysML V1. Variability defines the variation points that can vary, and it can be applied to all definition and usage elements in V2. A variant represents a particular choice at a variation point. The variation point gives different choices at each variation definition. Thus the system can be configured by making choices at each variation point [20].In variability modelling here, the variation points are defined within a model where a choice can be made to select a specific variant [20]. In the HAP system, the variation configuration can be defined for different parts, such as the earth observation subsystem and the environment observation subsystem, and the variant defined here is payload choices. From Figure Fig. 6.1, the definition of variation is used to make choices for the power system. The variants are solar panels and fuel cells, while the payload choices are sensors and cameras. This choice can be made by different parts of the HAP system, such as the payload system or the HAP energy power system. In variation, a configuration can be made in which the reusability in configuring the system is made. The system is configured into long-duration HAP, where the variant that is chosen is payload choice, which is a camera, and the power system cell is fuel cells. And the next configuration is short-duration HAP, where the payload choice is sensors and the power system is solar panels. These choices in the variation configuration help to implement the system with different options, thereby providing reusability.

6.1.3 *Individuals and Snapshots*

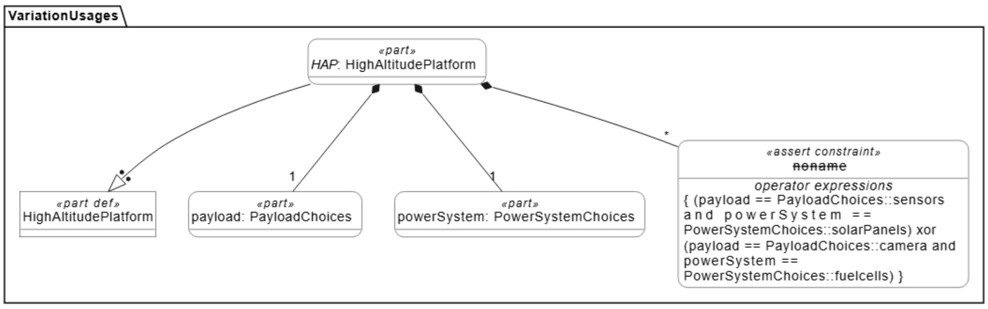
In SysML V2, an individual definition is a specialised occurrence definition to represent the single instance of a system. This defines how the system evolves over time. It shows occurrence as snapshots that describe their state at a particular moment in time. Attributes can be constrained to hold specific values within a given snapshot. 'Individual usage' is used to refer to an individual during some portion of its lifetime. Thus, attribute values can be changed or redefined in various snapshots. The modelling of individuals with specific identities: Here in HAP, individuals and snapshots can be used to represent the system in two different timeslots [8]. From Figure Fig. 6.2, the individual part payload has two snapshots in time slices to and t1. The time slice to is for the high-resolution camera, and the time slice t1 is for the infrared camera, and the value of the attribute mass is 20 in time slice to, and the attribute mass is assigned with a different value in the time slice t1. Thus, as mentioned above, the values of attributes can change in different timeslices, but the same attribute is redefined, thereby enhancing reusability across the system.

6.2 analysis case

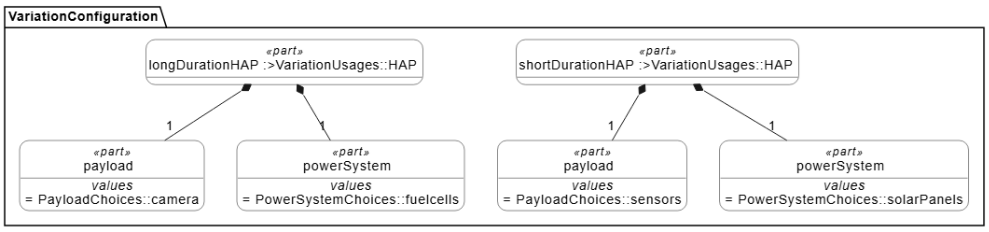
An analysis case definition defines the computation of the result of analysing some subject and meeting an objective. The modelling of requirements is a special type of constraint that a subject must satisfy to be a valid solution. The modelling of cases in SysML V2 involves defining cases that outline the steps needed to achieve a goal related to a subject. In analysis cases, it is used for examining the subject; in verification cases, it is used to check if the subject meets a requirement. [20] From Figure Fig. 6.3, the textual notation is used to implement the power analysis in the SysML V2 pilot implementation using Jupyter Lab. Here the analysis definition is power sum verification for the HAP. Through this analysis, the system tries to check if the requirement stated in the CR2.3 is true or false. Thus, the objective of this analysis is to determine whether the subject HAP can satisfy the energy requirement of the power subsystem. The analysis result is declared as a return. Here the requirement states that power for the HAP must be equal to 90792 watt-hours. So the observed solar energy and the battery power are calculated, and the total is analysed with the required value. If the expected value of 90792 watt-hours


VariationDefinitions
«part def» PowerSystemChoices
«part def» PayloadChoices
6
2
1..*
2
«variant» «reference» solarPanels
«variant» «reference» fuelcells
«variant» «reference» sensors
«variant» «reference» camera

((a)) Variation Definition


VariationUsages
«part» HAP: HighAltitudePlatform
«part def» HighAltitudePlatform
1
«part» payload: PayloadChoices
1
«part» powerSystem: PowerSystemChoices
*
«assert constraint» ~~noname~~
operator expressions
{ (payload == PayloadChoices::sensors and powerSystem == PowerSystemChoices::solarPanels) xor (payload == PayloadChoices::camera and powerSystem == PowerSystemChoices::fuelcells) }

((b)) Variation Usage


VariationConfiguration
«part» longDurationHAP :>VariationUsages::HAP
«part» shortDurationHAP :>VariationUsages::HAP
1
1
1
1
«part» payload values = PayloadChoices::camera
«part» powerSystem values = PowerSystemChoices::fuelcells
«part» payload values = PayloadChoices::sensors
«part» powerSystem values = PowerSystemChoices::solarPanels

((c)) Variation Configuration

Figure 6.1: Variation model of HAP configurations showing configuration and definition views.

is equal to the calculated total, then the result is true, or the result is false. Thus, the objective of the analysis case used here is to check the correctness of the state system state in relation to its power requirements.

evaluation of power analysis result The expression `%eval` represents the command used to evaluate the result. The `%eval` command in the SysML V2

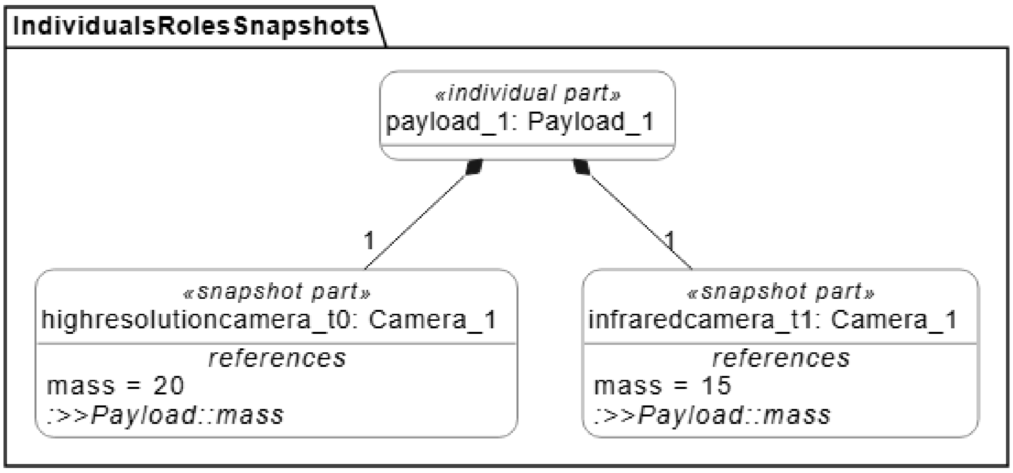
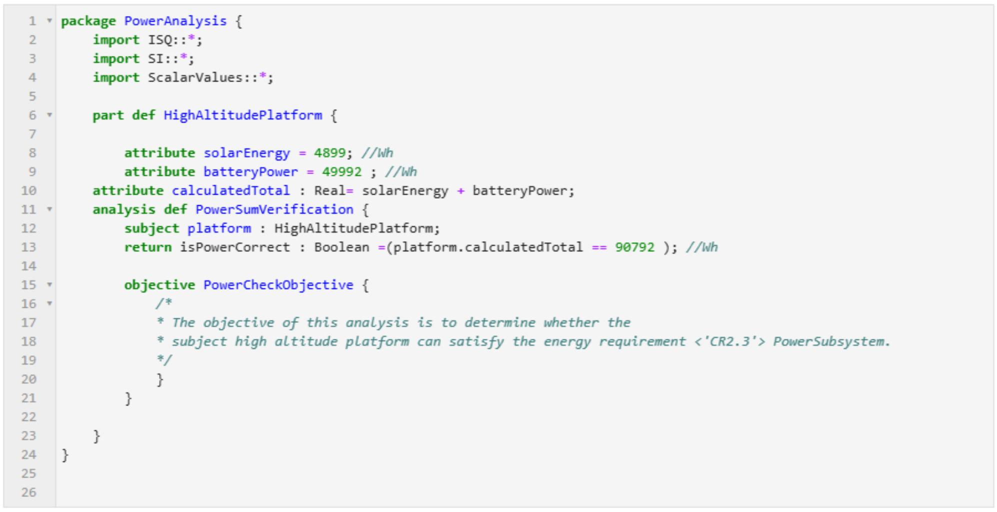


Figure 6.2: Individuals and Snapshots: Graphical Notation

pilot implementation using Jupyter Lab is to dynamically evaluate an expression defined within the model. As explained, *isPowerCorrect* is a boolean value; if the assigned or calculated value for solar energy and battery power is equal to the expected value, then the result is true; otherwise, the value is false. For further verification, the second step is that the calculated total value is displayed using the `%eval` command for clarity. The `%eval` keyword indicates that the expression is evaluated dynamically, producing a result that supports the verification of requirements within the model.

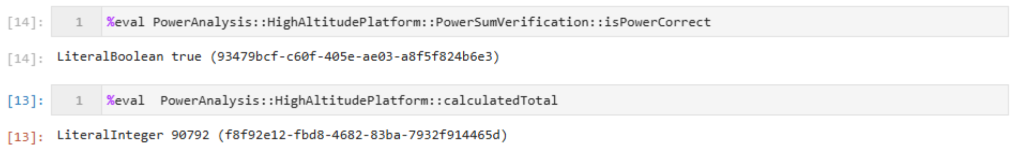
6.3 VIEW AND VIEWPOINTS

Views are the queries on the model that are used to display the results in a way that meets the needs of those viewpoints. For example, in the HAP, different views such as TREE, TABLEVIEW, etc. can be visualised. A view definition is used to filter the elements to be included in the specified view. The SysML library package models the SysML abstract syntax, which can be used to filter on specific kinds of model elements [8]. A viewpoint is a requirement to present information from a model in a view that addresses certain stakeholder concerns[8]. To illustrate the view and viewpoint, SysML V2 textual notation is used in the SysML V2 pilot implementation, Jupyter Lab. Here in the Figure Fig. 6.4, a package for system structure is shown which is developed using the textual notation. In part HAP is subdivided into part payload and part structure, and each part is subdivided into subsystem parts such as earth observation subsystem, environmental subsystem, surveillance subsystem, and so on. And here an attribute 'isMandatory' is assigned to boolean in the metadata def of safety. Metadata is defined using a metadata definition. A specific type of metadata can be applied in the textual notation as an annotation to one or more model elements. That is, annotation, in simple words, is used to tag certain elements as a group [8]. Then metadata safety is assigned



```
package PowerAnalysis {
    import ISQ::*;
    import SI::*;
    import ScalarValues::*;

    part def HighAltitudePlatform {

        attribute solarEnergy = 4899; //Wh
        attribute batteryPower = 49992 ; //Wh
    attribute calculatedTotal : Real= solarEnergy + batteryPower;
    analysis def PowerSumVerification {
        subject platform : HighAltitudePlatform;
        return isPowerCorrect : Boolean =(platform.calculatedTotal == 90792 ); //Wh

        objective PowerCheckObjective {
            /*
            * The objective of this analysis is to determine whether the
            * subject high altitude platform can satisfy the energy requirement <'CR2.3'> PowerSubsystem.
            */
            }
        }

    }
}
```

((a)) Power Analysis: Textual notation



```
[14]:  1  %eval PowerAnalysis::HighAltitudePlatform::PowerSumVerification::isPowerCorrect
[14]: LiteralBoolean true (93479bcf-c60f-405e-ae03-a8f5f824b6e3)

[13]:  1  %eval  PowerAnalysis::HighAltitudePlatform::calculatedTotal
[13]: LiteralInteger 90792 (f8f92e12-fbd8-4682-83ba-7932f914465d)
```

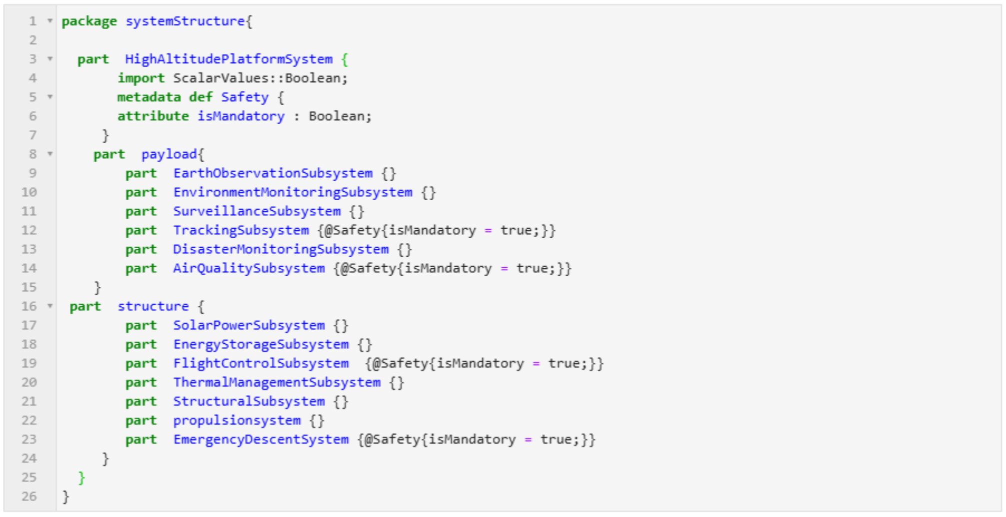
((b)) Result Analysis

Figure 6.3: Textual notation for analysis cases.

to four subsystems in the structure and payload part. In the output shown in the diagram Fig. 6.4(b), the graphical notation of the system structure and the command given: `%vizsystemStructure::HighAltitudePlatformSystem::payload` Here the command states to only show the payload part in the system structure. Thus from the output shown in the graphical notation, the tracking subsystem and the air quality system are assigned the safety metadata.

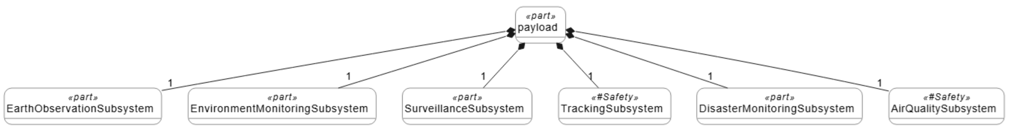
In the package 'viewpointusage', the viewpoint 'HAPPerspective' addresses the concern 'system breakdown', which emphasises the importance of understanding how the system decomposes into subsystems and components. This viewpoint includes a constraint that states that the system structure view shall show the hierarchical part decomposition of a HAP system[8].In the view definition, a filter is applied as 'SysML::PartUsage' to focus the view on relevant model elements. Then 'HAPsystemstructureview' is instantiated, which exposes actual system elements and parts of the HAP system and renders them in a selected format, such as a tree diagram.

In figure Fig. 6.4(c), to view the safety features, the 'safetyview' is defined to focus on the subsystem that is assigned with safety metadata, as shown in Figure 6.4a. The view is created to expose the 'HighAltitudePlatformSystem' keyword to filter elements annotated with safety. As the 'HighAltitudePlatformSystem' is subdivided into payload and structure, which is further decomposed into thirteen subsystems. When the filter element is annotated with safety, then four subsystems that are assigned with safety 'ismandatory' true are displayed in the output. making the structure and relationships of safety-relevant subsystems—such as the TrackingSubsystem, AirQualitySubsystem, FlightControlSubsystem, and EmergencyDescentSystem. This kind of targeted visualisation helps system engineers isolate and analyse safety features in complex architectures.

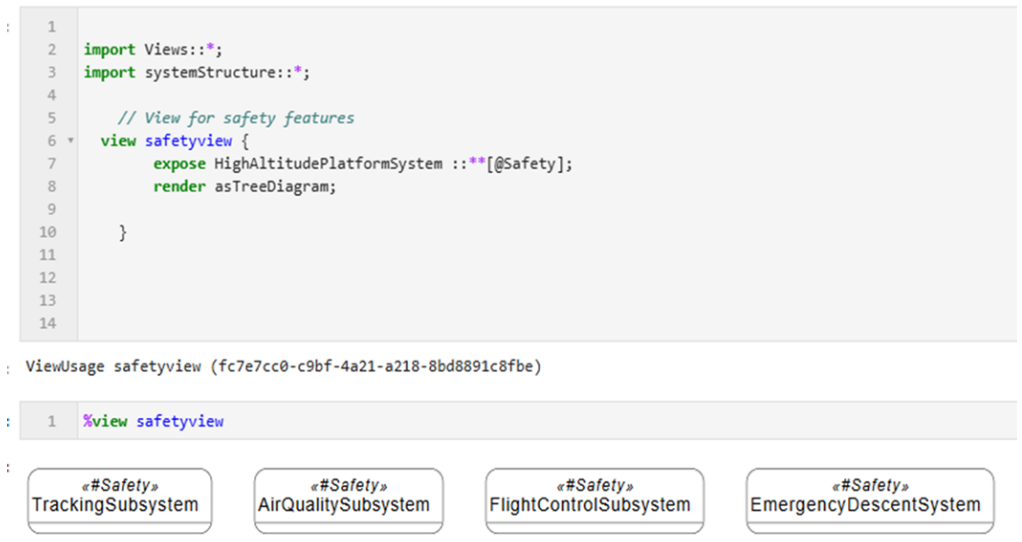


```
package systemStructure{

  part  HighAltitudePlatformSystem {
       import ScalarValues::Boolean;
       metadata def Safety {
       attribute isMandatory : Boolean;
     }
    part  payload{
        part  EarthObservationSubsystem {}
        part  EnvironmentMonitoringSubsystem {}
        part  SurveillanceSubsystem {}
        part  TrackingSubsystem {@Safety{isMandatory = true;}}
        part  DisasterMonitoringSubsystem {}
        part  AirQualitySubsystem {@Safety{isMandatory = true;}}
    }
 part  structure {
        part  SolarPowerSubsystem {}
        part  EnergyStorageSubsystem {}
        part  FlightControlSubsystem  {@Safety{isMandatory = true;}}
        part  ThermalManagementSubsystem {}
        part  StructuralSubsystem {}
        part  propulsionsystem {}
        part  EmergencyDescentSystem {@Safety{isMandatory = true;}}
     }
  }
}
```

((a)) System Structure: Textual notation



((b)) System Structure Graphical Notation



```

import Views::*;
import systemStructure::*;

   // View for safety features
  view safetyview {
        expose HighAltitudePlatformSystem ::**[@Safety];
        render asTreeDiagram;

     }
```

```
ViewUsage safetyview (fc7e7cc0-c9bf-4a21-a218-8bd8891c8fbe)
```

```
%view safetyview
```

((c)) ViewUsage

Figure 6.4: Textual Notation for View and Viewpoint

7 EVALUATION

From the case study done on SysML V2, it is clear that the new version addresses the complexities inherent in the development of HAP systems. And the scaling of SysML V2 in the MBSE Magic Grid is one of the research questions achieved. As compared to SysML V1, new features, such as views and viewpoints modelling, variability, individuals and snapshots, expressions, calculations, and analysis cases, are achieved in SysML V2. These features support clearer system representation, enhanced traceability, and improved validation compared to SysML V1. These capabilities of SysML V2 are proved through the case study done in this thesis. Through the research done in this study, the main advantage observed in SysML V2 is the use of textual notation. In this thesis, the textual notation is scripted with the SysML V2 pilot implementation in Jupyter Lab. Through the use of textual notation, implementing modelling is far easier as compared to modelling graphical figures, as the editing of the model can be far more easily achieved as compared to graphical notation. From the analysis of the thesis, the use of textual notation is one of the main achievements in SysML V2 as compared to SysML V1.

The application of SysML V2 in the MBSE Magic Grid framework provides a systematic approach to display and visualise the graphical notation of SysML V2 in a step-by-step approach in which the system advances layer by layer, providing more insight into the HAP system. By using SysML V2 graphical notation and scaling it in the Magic Grid, the development team and stakeholders are helped to visualise the system incrementally, aligning with the abstraction levels defined in the Magic Grid. The modelling begins with the black box, where the external interfaces and behaviours are defined, and then the system increments to the white box views, where SysML V2 images are used to show the logical subsystem identification, internal structures, and behaviour modelling. In the solution-level layer, a deeper insight into HAP is given. This is one of the main achievements of SysML V2 scaling in Magic Grid; as with SysML V2, more accurate modelling is possible, as SysML V2 has stricter syntax and semantics. With SysML V2, improved traceability is achieved through strong reuse of definitions and usage. Efficient reuse and consistency allow a model element defined once to be reused consistently across different cells of the grid. And with the help of SysML V2 diagrams, refining the diagrams in each layer of abstraction is achieved. Thereby providing clear visualisation of system evolution with SysML V2 improved graphical and textual notation that helps in the step-by-step refinement process of the Magic Grid. This modelling ensures traceability and consistency across all development phases, enabling better communication between stakeholders.

This study, as stated, focused mainly on the innovative features of SysML V2 and how effectively it can model, analyse, and verify complex systems such as HAP.

This thesis also focused on developing different models using SysML V2 graphical notation using the SysON tool. Developing action diagrams and interconnection diagrams with the SysON tool was more presentable compared to developing diagrams with textual notation using SysML V2 pilot implementation in Jupyter Lab. Modelling with textual notation in the SysML V2 pilot implementation using Jupyter Lab is more time-saving and easier, as editing can be done easily in the scripted text and is understandable. However, modelling with textual notation in the Jupyter Lab environment was notably more efficient. As it allowed for easier editing and a faster development phase as compared to drawing and aligning diagrams with different tools. Even though there is enough improvement to be made in SysML V2, for now it is in the pilot implementation phase. The improvements should be focused on unwanted and crowded diagrams in the graphical notation of the interconnection diagrams and action diagrams; this should be taken as an area of advancement in the future.

While discussing the process of modelling HAP using SysML V2 within the MBSE Magic Grid, it provides benefits and limitations of MBSE approaches. The Magic Grid provides a structured method to decompose and organise system elements. From the case study done on the HAP, it is clear that the systematic modelling, traceability and refinement of the model in each layer are achieved. And it is demonstrated in the systematic modelling section of this thesis, where each layer from the black box, white box and solution level is refined, and traceability of each cell is explained. Moreover, SysML V2 diagrams are found to be well aligned with the magic grid framework, allowing a structured visual representation across various cells. Thus SysML V2 diagrams are well scaled with each cell of the grid. Whereas the naming of the diagrams in certain levels, such as structure diagrams, are named as internal block diagrams in SysML V1, the corresponding naming or reference on the diagram naming or representation of the diagram in the parametric and structure cell was not correctly available, as SysML V2 is in the pilot implementation phase and it lacks complete standardisation in certain aspects of the diagram representation and tooling support.

8 CONCLUSION

The research explores the application of SysML V2 for modelling HAP and highlighting its improvement over SysML V1. It demonstrates the advanced capabilities of SysML V2 in the modelling of HAP. The modelling is done using SysML V2 textual notation using Jupyter Lab. Through this study, the research tries to answer research questions, such as evaluating the improvements achieved in SysML V2 compared to SysML V1. Through this study, the new features of SysML V2 are explored, which are highly useful in the HAP modelling, as the new features, such as definition and usage elements in SysML V2, provide reuse, traceability and a more reliable modelling environment. The features, such as variability, individual snapshots, analysis cases, expressions, and calculations, provide a new modelling perspective that enhances flexibility and supports dynamic system behaviour and representation.

Taking HAP as an example for a case study, this thesis tries to show how SysML V2 supports system integration across the layer of abstraction in the Magic Grid framework. The systematic development of HAP through each cell of the magic grid helps maintain traceability and consistency. Scaling SysML V2 into the Magic Grid framework and systematically modelling the system using the Jupyter Lab pilot implementation and SysON tool further highlights the flexibility of SysML V2 in supporting different modelling workflows. SysML V2 modelling is achieved through textual notation that makes modelling much easier as compared with the traditional method of designing models, which requires more time and difficulties in correction. This work establishes SysML V2 as a leading methodology for complex system design, which could have an impact on future engineering practices by providing fresh approaches to problems in multidisciplinary and safety-critical projects. With advanced features of SysML V2 and the integration of the technology in a magic grid methodology, it provides a systematic modelling approach, which will be a new benchmark in system engineering. This approach has improvement in the area of scalability, reuse, and collaboration. This method can be adopted as a reference for similar applications in other complex engineering domains. thereby prompting the wider adoption of MBSE practices aligned with emerging standards

FUTURE WORK The systematic modelling of the HAP system using SysML V2 is considered a robust and forward-looking methodology for MBSE. Through this work, the foundation for understanding the potential of SysML V2 is laid, but there are still many areas that require further exploration and development. Especially there should be a lot of improvement in the graphical notation, for example, action flow diagrams and interconnection diagrams. In most graphical representations, unwanted information is included that causes the representation to be cumbersome, large, and unclear. As SysML V2 is in the development stages, this will be improved in the future release. So in further research, work can be done on the usability of SysML V2 for complex system design and promote its wider adoption in engineering practice. Additionally, integration of the SysML V2 model with simulation and verification would strengthen model validation and improve the accuracy of the system. Here, the SysON tool is used as the SysML V2 tool. As the tool should provide much more user-friendly options for a better user experience, this is a major area of enhancement. There should be more standardised practice and well-defined modelling templates and better guidelines for the magic grid framework, thereby promoting consistency and reusability in future MBSE projects.

BIBLIOGRAPHY

[1] A. Ahlbrecht, B. Lukić, W. Zaeske, and U. Durak. “Exploring SysML v2 for Model-Based Engineering of Safety-Critical Avionics Systems”. In: *2024 AIAA DATC/IEEE 43rd Digital Avionics Systems Conference (DASC)*. IEEE. 2024, pp. 1–8.

[2] M. Bajaj, S. Friedenthal, and E. Seidewitz. “Systems modeling language (SysML v2) support for digital engineering”. In: *Insight* 25.1 (2022), pp. 19–24.

[3] M. Bone and R. Cloutier. “The current state of model based systems engineering: Results from the omg™ sysml request for information 2009”. In: *Proceedings of the 8th conference on systems engineering research*. 2010.

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

[5] J. Fisher, J. Estefan, and X. Hou. “Usage-Focused Modeling in SysML v2”. In: *Proceedings of the International Conference on Model-Driven Engineering and Software Development (MODELSWARD)*. SciTePress, 2021, pp. 123–134. DOI: 10.5220/0010316201230134.

[6] S. Friedenthal. “Requirements for the next generation systems modeling language (SysML® v2)”. In: *Insight* 21.1 (2018), pp. 21–25.

[7] S. Friedenthal. “Future Directions for MBSE with SysML v2.” In: *MODELSWARD*. 2023, pp. 5–9.

[8] S. Friedenthal. *SysML V2 Pilot Implementation: Introduction to the SysML v2 Language Textual Notation*. https://github.com/Systems-Modeling/SysML-v2-Release/blob/master/doc/IntrototheSysMLv2Language-TextualNotation.pdf. Accessed: May 28, 2025. 2023.

[9] S. Friedenthal. *SysML v2 Basics*. https://www.omgwiki.org/MBSE/lib/exe/fetch.php?media=mbse:sysml_v2_transition:sysml_v2_basics-incose_iw-sfriedenthal-2024-01-28.pdf. Presented at INCOSE International Workshop, SysML v1 to SysML v2 Transition Information Session, January 28, 2024. Accessed: May 28, 2025. 2024.

[10] S. Friedenthal, A. Moore, and R. Steiner. *A practical guide to SysML: the systems modeling language*. Morgan Kaufmann, 2014.

[11] German Aerospace Center (DLR). *High Altitude Platform (HAP) Project*. https://www.dlr.de/en/hap. 2025.

[12] M. Hause et al. “The SysML modelling language”. In: *Fifteenth European systems engineering conference*. Vol. 9. 2006, pp. 1–12.

[13] J. Holt and S. Perry. *SysML for Systems Engineering: A Model-Based Approach*. 2nd. London, UK: Institution of Engineering and Technology (IET), 2013.

[14] N. Jansen, J. Pfeiffer, B. Rumpe, D. Schmalzing, and A. Wortmann. "The Language of SysML v2 under the Magnifying Glass." In: *J. Object Technol.* 21.3 (2022), pp. 3–1.

[15] Z. Li, F. Faheem, and S. Husung. "Collaborative model-based systems engineering using dataspaces and SysML v2". In: *Systems* 12.1 (2024), p. 18.

[16] V. Molnár, B. Graics, A. Vörös, S. Tonetta, L. Cristoforetti, G. Kimberly, P. Dyer, K. Giammarco, M. Koethe, J. Hester, et al. "Towards the Formal Verification of SysML v2 Models". In: *Proceedings of the ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems*. 2024, pp. 1086–1095.

[17] A. Morkevicius, A. Aleksandraviciene, D. Mazeika, L. Bisikirskiene, and Z. Strolia. "MBSE grid: a simplified SysML-based approach for modeling complex systems". In: *INCOSE International Symposium*. Vol. 27. 1. Wiley Online Library. 2017, pp. 136–150.

[18] A. Morkevicius, A. Aleksandraviciene, and Z. Strolia. "System verification and validation approach using the magicgrid framework". In: *Insight* 26.1 (2023), pp. 51–59.

[19] A. Morkevicius and G. Krisciuniene. "Towards UAF Implementation in SysML V2". In: *INCOSE International Symposium*. Vol. 34. 1. Wiley Online Library. 2024, pp. 2110–2123.

[20] O. M. G. (OMG). *OMG Systems Modeling Language (SysML) v2 Specification*. https://www.omg.org/spec/SysML/2.0/. 2023.

[21] Obeo and C. List. *SysON Overview*. https://doc.mbse-syson.org/syson/main/user-manual/what-is.html.

[22] Object Management Group (OMG). *OMG Unified Modeling LanguageTM (OMG UML), Superstructure*. Tech. rep. formal/2011-08-06. Available at: https://www.omg.org/spec/UML/2.4.1/. Object Management Group (OMG), 2011.

[23] Office of the Under Secretary of Defense for Research and Engineering (OUSD(R&E)). *SysML v1 to SysML v2 Model Conversion Approach*. Technical Report. Office of the Under Secretary of Defense for Research and Engineering, 2024. URL: https://www.cto.mil/wp-content/uploads/2024/04/SysML-Approach-Report-March2024.pdf.

[24] E. Seidewitz and M. Bajaj. *Systems-Modeling/SysML-v2-Release: The Latest Incremental Release of SysML V2*. GitHub repository, Systems Modeling Organization. 2024. URL: https://github.com/Systems-Modelling/SysML-v2-Release.

[25] SysML v1 to SysML v2 Transition Community. *SysML v1 to SysML v2 Transition Plan Template*. OMG Wiki MBSE Working Group. Last updated: 2024-01-22, Accessed: 2024-06-06. 2024. URL: https://www.omgwiki.org/MBSE/doku.v1_to_sysml_v2_transition_guidance.

[26] S. Team. *Transitioning from SysML v1 to SysML v2: Textual Notation*. `https://resources.sysgit.io/transitioning-from-sysml-v1-to-sysml-v2-textual-notation/`. 2024.