**T.R.**
**GEBZE TECHNICAL UNIVERSITY**
**GRADUATE SCHOOL**

# IMPLEMENTING PARTIAL MBSE IN GAS TURBINE ENGINE ARCHITECTURE DEVELOPMENT AND ITS REPORTED ADVANTAGES AND DISADVANTAGES

**TUĞÇE COŞKUNTUNA DAŞDEMİR**

**A THESIS OF MASTER OF SCIENCE**
**DEPARTMENT OF ENERGY TECHNOLOGIES**
**APPLIED PROPULSION SYSTEM DESIGN & ENGINEERING**
**FOR AEROSPACE TECHNOLOGIES PROGRAM**

ADVISOR: PROF. DR. MEHMET ALİ ARSLAN

**JUNE 2024**

**T.R.**
**GEBZE TECHNICAL UNIVERSITY**
**GRADUATE SCHOOL**

# IMPLEMENTING PARTIAL MBSE IN GAS TURBINE ENGINE ARCHITECTURE DEVELOPMENT AND ITS REPORTED ADVANTAGES AND DISADVANTAGES

**TUĞÇE COŞKUNTUNA DAŞDEMİR**

**A THESIS OF MASTER OF SCIENCE**
**DEPARTMENT OF ENERGY TECHNOLOGIES**
**APPLIED PROPULSION SYSTEM DESIGN & ENGINEERING FOR AEROSPACE TECHNOLOGIES PROGRAM**

ADVISOR: PROF. DR. MEHMET ALİ ARSLAN

**JUNE 2024**

**T.C.**
**GEBZE TEKNİK ÜNİVERSİTESİ**
**LİSANSÜSTÜ EĞİTİM ENSTİTÜSÜ**

# GAZ TÜRBİNLİ MOTOR MİMARİSİ GELİŞTİRMEDE MBSE`NİN KISMİ UYGULANMASI AVANTAJ VE DEZAVANTAJLARI

**TUĞÇE COŞKUNTUNA DAŞDEMİR**

**YÜKSEK LİSANS TEZİ**
**ENERJİ TEKNOLOJİLERİ ANABİLİM DALI**
**HAVACILIK VE UZAY TEKNOLOJİLERİNDE UYGULAMALI İTKİ SİSTEMİ TASARIM MÜHENDİSLİĞİ PROGRAMI**

DANIŞMAN: PROF. DR. MEHMET ALİ ARSLAN

**HAZİRAN 2024**

**MASTER of SCIENCE JURY APPROVAL FORM**

A thesis submitted by Tuğçe COŞKUNTUNA DAŞDEMİR, defended on 10/07/2024 before the jury formed with the 08/07/2024 date and 2024 /35 numbered decision of the GTU Graduate Administration Board, has been accepted as a MASTER of SCIENCE thesis in the Department of Energy Technologies, Applied Propulsion System Design & Engineering For Aerospace Technologies Program.

**JURY**

MEMBER
(THESIS ADVISOR) : Prof.Dr. Mehmet Ali ARSLAN

MEMBER : Dr. Mahmut AKSİT

MEMBER : Prof.Dr. Gokhan BAYAR

**APPROVAL**

Gebze Technical University Graduate Administration Board

…..…/…..…/..…… date and ……/…….. numbered decision.

SIGNATURE/SEAL

*To my little sunshine Zubeyde and all the stray animals …*

# ABSTRACT

Gas turbine engines, as one of the primary components of modern aircraft engines, function by drawing in air to generate propulsive force, which is then expelled forward. These engines are critically important in the aviation industry, aimed at enhancing aircraft performance and improving fuel efficiency. Among gas turbine engines, turbofan engines play a particularly significant role in achieving these goals.

The perspective of systems engineering is essential for integrating and optimizing complex components like aircraft engines to enhance performance and ensure reliability in the aviation industry. This approach emphasizes the integrated design, development, implementation, and management of complex systems throughout their lifecycle and aircraft engines are no exception as they have to be modeled and analyzed through the lens of systems engineering, taking account of interactions between components to optimize performance.

Systems engineering encompasses various methodologies for analyzing, designing, and managing complex systems. Among these methodologies, two prominent approaches are Document-Based Systems Engineering (DBSE) and Model Based Systems Engineering (MBSE). Model Based Systems Engineering represents a more innovative approach, where information such as system requirements, designs, and analyses are represented through models that play a central role in the system development process.

This study aims to investigate the application of Partial MBSE in developing gas turbine engine architectures, focusing on evaluating its advantages and disadvantages. By examining real-world case studies and industry practices, this research aims to provide insights into the practical implications of adopting MBSE in engine design. It is expected that the findings of this study will contribute to the existing knowledge base on MBSE and its applications in aerospace engineering, offering valuable recommendations to companies seeking to enhance their engine design processes.

Despite the primary emphasis of this study being on the design of turbofan engines, proprietary constraints set by the company prevent the use of the ongoing turbofan engine project in this research. Therefore, the TJ90 turbojet engine project has been chosen as a pilot project for the purposes of this study.



**Keywords: Systems Engineering, Model Based Systems Engineering (MBSE), Systems Modeling Language (SysML), Aviation Industry, Gas Turbine Engine, Turbofan Engine, Turbojet Engine**

# ÖZET

Modern uçak motorlarının ana bileşenlerinden biri olan gaz türbini motorları, itme kuvveti oluşturmak için havayı çekerek çalışır ve bu kuvvet daha sonra ileri doğru atılır. Bu motorlar, uçak performansını ve yakıt verimliliğini artırmayı amaçlayan havacılık endüstrisinde kritik öneme sahiptir. Gaz türbinli motorlar arasında turbofan motorlar bu hedeflere ulaşmada özellikle önemli bir rol oynamaktadır.

Sistem mühendisliği perspektifi, havacılık endüstrisinde performansı artırmak ve güvenilirliği sağlamak amacıyla uçak motorları gibi karmaşık bileşenlerin entegre ve optimize edilmesi için gereklidir. Bu yaklaşım, karmaşık sistemlerin yaşam döngüleri boyunca entegre tasarımını, geliştirilmesini, uygulanmasını ve yönetimini vurgular. Uçak motorları gibi karmaşık sistemler, performansı optimize etmek için bileşenler arasındaki etkileşimler dikkate alınarak sistem mühendisliği merceğinden modellenir ve analiz edilir.

Sistem mühendisliği, karmaşık sistemleri analiz etmek, tasarlamak ve yönetmek için çeşitli metodolojileri kapsar. Bu metodolojiler arasında öne çıkan iki yaklaşım Doküman Tabanlı Sistem Mühendisliği (DTSM) ve Model Tabanlı Sistem Mühendisliği (MTSM)'dir. Model Tabanlı Sistem Mühendisliği, sistem gereksinimleri, tasarımlar ve analizler gibi bilgilerin, sistem geliştirme sürecinde merkezi rol oynayan modeller aracılığıyla temsil edildiği daha yenilikçi bir yaklaşımı temsil eder.

Bu çalışma, Kısmi MBSE'nin gaz türbinli motor mimarilerinin geliştirilmesinde uygulanmasını araştırmayı, avantaj ve dezavantajlarını değerlendirmeye odaklanmayı amaçlamaktadır. Gerçek dünyadaki örnek olay incelemelerini ve endüstri uygulamalarını inceleyerek bu araştırma, motor tasarımında MBSE'yi benimsemenin pratik sonuçlarına dair içgörü sağlamayı amaçlamaktadır. Bu çalışmanın bulgularının, MBSE'ye ve havacılık ve uzay mühendisliğindeki uygulamalarına ilişkin mevcut bilgi tabanına katkıda bulunarak, motor tasarım süreçlerini geliştirmek isteyen şirketlere değerli öneriler sunması bekleniyor.

Bu çalışmanın ana vurgusu turbofan motorların tasarımı olmasına rağmen, şirket tarafından belirlenen özel kısıtlamalar, devam eden turbofan motor projesinin bu araştırmada kullanılmasını engellemektedir. Bu nedenle bu çalışmanın amaçları doğrultusunda pilot proje olarak TJ90 turbojet motoru projesi seçilmiştir.

**Anahtar Kelimeler: Sistem Mühendisliği, Model Tabanlı Sistem Mühendisliği (MTSM), Sistem Modelleme Dili (SysML), Havacılık Endüstrisi, Gaz Türbinli Motorlar, Turbofan Motor, Turbojet Motor**

# ACKNOWLEDGEMENTS

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

First and foremost, I am profoundly grateful to my beloved husband, Nurettin Ali Daşdemir, whose love, support, and understanding have been invaluable during this journey. I also extend my heartfelt thanks to my dear family, Hülya and Adnan Coşkuntuna and my siblings, whose sacrifices and unwavering belief in me has always been a continuous source of strength and motivation. Their constant encouragement and support have been crucial throughout this process. A special thanks to my mother, for being a wonderful mother and an extraordinary source of inspiration.

I would also like to express my sincere gratitude to my dear colleagues, Ömer Ecevitoğlu and Yücel Beki, for their constructive feedback, suggestions, invaluable contributions, and the time they have spent, all of which have significantly enhanced the quality of this thesis. I am also grateful to İrem Karakuş, a wonderful colleague and classmate, for her friendship and support.

I also owe a debt of gratitude to Andrew Smith for his encouragement, patience, understanding and invaluable friendship.

A special thanks to my cats, whose presence provided much-needed emotional support and comfort throughout this process.

I would also like to thank my advisor, Prof. Dr. Mehmet Ali Arslan, for his role in this journey.

Finally, I would like to acknowledge TUSAS Engine Industries Inc. (TEI) for offering this master's program and providing the projects discussed in this study. This research would not have been possible without the company’s invaluable assistance.

Thank you all for your support and encouragement.

# TABLE OF CONTENTS

# LIST OF SYMBOLS AND ABBREVIATIONS

| | | |
|---|---|---|
| act | : | Activity Diagram |
| bdd | : | Block Definition Diagram |
| BOM | : | Bill of Material |
| CAD | : | Computer Aided Design |
| CASE | : | Computer Aided Software Engineering |
| ConOps | : | Concept of Operations |
| DBSE | : | Document-Based Systems Engineering |
| Hz | : | Hertz |
| ibd | : | Internal Block Diagram |
| IEEE | : | Institute of Electrical and Electronics Engineers |
| INCOSE | : | International Council on Systems Engineering |
| ISO | : | International Organization for Standardization |
| MBD | : | Model Based Design |
| MBE | : | Model Based Engineering |
| MBSE | : | Model Based Systems Engineering |
| MoE | : | Measures of Effectiveness |
| MVP | : | Minimum Viable Product |
| GE | : | General Electric |
| GTÜ | : | Gebze Teknik Üniversitesi |
| OMG | : | Object Management Group |
| PBS | : | Product Breakdown Structure |
| PD170 | : | TEI – Piston Engine Project |
| PDM | : | Product Data Management |
| PLM | : | Product Lifecycle Management |
| ROI | : | Return on Invesment |
| RPM | : | Revolutions per Minute |
| SE | : | Systems Engineering |
| SEMP | : | Systems Engineering Management Plan |
| SoI | : | System of Interest |
| SysML | : | Systems Modeling Language |
| SWFTS | : | The Submarine Warfare Federated Tactical Systems |
| TAI | : | Turkish Aerospace Industries |
| TAA | : | Turkish Aeronautical Association |
| TAFF | : | Turkish Armed Forces Foundation |
| TEI | : | TUSAS Engine Industries Inc. |
| TJ90 | : | TEI – Turbojet Engine Project |
| TS1400 | : | TEI – Turboshaft Engine Project |
| UAV | : | Unmanned Aerial Vehicle |
| uc | : | Use Case Diagram |
| UML | : | Unified Modeling Language |

# LIST OF FIGURES

# 1. INTRODUCTION

Gas turbine engines are a type of reaction engine that generates thrust by expelling a high-speed jet of fluid, typically air mixed with fuel. They are widely used in aircraft propulsion due to their high efficiency and power. Gas turbine engines function based on Newton's third law of motion: for every action, there is an equal and opposite reaction.

## 1.1. Types of Gas Turbine Engines

### 1.1.1. Turbojet Engines

**Operation:** An axial or centrifugal compressor draws air into the engine and compresses it. High-temperature, high-pressure gases are produced in the combustion chamber by igniting a mixture of compressed air and fuel. Thrust is produced when these gasses, which are expanding quickly, are released via a turbine and nozzle.
**Uses:** Turbojets are primarily used in high-speed military aircraft and early commercial jet airliners.

### 1.1.2. Turbofan Engines

**Operation:** Similar to turbojets, but with an additional fan at the front. The fan generates additional thrust by bypassing a portion of the air around the engine core. This makes turbofans more efficient and quieter than turbojets.
**Uses:** Turbofans are the most common type of jet engine used in modern commercial airliners and many military aircraft.

### 1.1.3. Turboprop Engines

**Operation:** These engines use a gas turbine to drive a propeller. The turbine extracts energy from the exhaust gases to turn the propeller, which generates the majority of the thrust.

**Uses:** Turboprops are used in smaller commuter aircraft and cargo planes where high efficiency at lower speeds is desired.

### 1.1.4. Turboshaft Engines

**Operation:** Similar to turboprops, but designed to drive a shaft rather than a propeller. The shaft power is used to drive other components, such as helicopter rotors or electrical generators.
**Uses:** Turboshaft engines are primarily used in helicopters and some types of industrial machinery.

### 1.1.5. Ramjet and Scramjet Engines

**Operation:** Ramjets and scramjets do not have compressors or turbines. Instead, they rely on the high-speed forward motion of the engine to compress incoming air. Fuel is injected and ignited in the combustion chamber, and the resulting high-speed exhaust gases produce thrust. Scramjets are an advanced type of ramjet that operate at hypersonic speeds.
**Uses:** Ramjets are used in some types of missiles and high-speed aircraft, while scramjets are experimental and aimed at future hypersonic flight applications.

The aerospace industry is a cornerstone of modern transportation, with turbofan engines playing a pivotal role in powering commercial and military aircraft.

Turbofan engines are a type of gas turbine engine used primarily in aircraft propulsion. They are a crucial component of aerospace engineering, as they provide the thrust necessary to propel an aircraft through the air. Turbofan engines are known for their high efficiency, reliability, and ability to produce large amounts of thrust.

The importance of turbofan engines in aerospace engineering cannot be overstated. They are the primary means of propulsion for most commercial and military aircraft, including passenger jets, cargo planes, and fighter jets. Without turbofan engines, modern air travel as we know it would not be possible.

Traditionally, the design and development of turbofan engines have relied on a combination of theoretical analysis, empirical testing, and physical prototyping. Engineers would use mathematical models and computer simulations to predict the

performance of different engine designs, and then build physical prototypes to test these predictions in real-world conditions. This process was time-consuming, expensive, and often resulted in suboptimal designs.

In recent years, however, there has been a shift towards using Model Based Systems Engineering (MBSE) in the design and development of complicated and/or complex machines like gas turbine engines. MBSE is a methodology that uses models to represent the system being designed, allowing for a more comprehensive understanding of the system's behavior and interactions. The approach has several advantages over traditional methods.

Overall, using MBSE in the design and development of turbofan engines has the potential to revolutionize the aerospace industry, leading to more efficient, reliable, and cost-effective development strategies.

This thesis aims to explore the implementation of Partial MBSE in gas turbine engine architecture development, especially for turbofan engines, focusing on assessing its advantages and disadvantages. By examining real-world case studies and industry practices, this research seeks to provide insights into the practical implications of adopting MBSE in gas turbine engine design. The findings of this study are expected to contribute to the existing body of knowledge on MBSE and its applications in aerospace engineering, offering valuable recommendations for companies seeking to enhance their powerplant design processes.

Although the primary focus of this study is on the design of turbofan engines, due to proprietary constraints imposed by the company, the ongoing turbofan engine project cannot be utilized in this research. Instead, the TJ90 turbojet engine project has been selected as a pilot project for the purposes of this study.

## 1.2. TUSAS Engine Industries Inc. (TEI)

Founded in 1985 as a joint venture between Turkish Aerospace Industries Inc., General Electric (GE), Turkish Armed Forces Foundation (TAFF), and Turkish Aeronautical Association (TAA), TUSAS Engine Industries Inc. (TEI) is an incorporated corporation. With the high-quality goods and services it provides to the aviation sector, TEI has grown into a global design hub and manufacturer. The company offers a wide

range of services such as engine design, manufacturing, assembly, testing and maintenance and repair.

TEI makes significant contributions to Turkey's defense industry with its domestic and national engine design projects. For example, projects such as the domestic helicopter engine (TS1400), the piston aircraft engine (TEI-PD170) and the unmanned aerial vehicle engine TJ90 are important steps towards increasing the country's independence and technology capacity.

TEI carries out intensive R&D studies to constantly develop new technologies and improve existing technologies. While these studies increase the company's global competitiveness, they also contribute to the advancement of the aviation industry.

TEI continues to contribute to the country's achievement of its strategic goals with its critical role in Turkey's defense and aerospace industry and its domestic production capacity.

## 1.3. TJ90 The Turbojet Engine

TJ90 turbojet engine represents a significant advancement in the development of small-scale, high-performance propulsion systems. Engineered by TUSAŞ Engine Industries (TEI), a leading Turkish aerospace manufacturer, the TJ90 is designed to provide reliable and efficient thrust for a variety of unmanned aerial vehicles (UAVs) and target drones. This engine underscores Turkey's growing expertise in aerospace technology, particularly in the field of jet propulsion for UAV applications.


**Figure 1.1:** TEI - TJ90 Turbojet Engine [1]

With a weight of approximately 5.5 kilograms and a maximum rotational speed of 85,000 revolutions per minute (RPM), the TJ90 is optimized for high-speed applications. The engine operates on standard aviation fuels, ensuring compatibility and ease of maintenance. Its efficient design allows for prolonged operational endurance and reliability, crucial for both military and civilian UAV missions.

The TJ90 turbojet engine plays a pivotal role in enhancing the performance and capabilities of various UAV platforms. Its compact size and high thrust-to-weight ratio make it ideal for target drones like the Şimşek, which are used to simulate enemy aircraft and missiles for air defense training. By providing realistic training scenarios, these UAVs enable defense forces to improve their readiness and response strategies.

Moreover, the TJ90 turbojet engine contributes to the broader strategic objectives of Turkey's defense industry by reducing dependency on foreign propulsion systems. The development of indigenous technologies such as the TJ90 reflects Turkey's commitment to achieving self-sufficiency in critical defense capabilities. This engine not only supports national security initiatives but also positions Turkey as a competitive player in the global aerospace market, with potential export opportunities.

The TJ90 turbojet engine is a testament to the innovative engineering and strategic vision of TUSAŞ Engine Industries (TEI) and Turkey's aerospace sector. Its development highlights the nation's ability to produce advanced propulsion systems that meet the demanding requirements of modern UAV applications. As a reliable and efficient engine, the TJ90 will continue to play a crucial role in the advancement of unmanned aerial systems, contributing to enhanced defense capabilities and technological independence.

## 1.4. Şimşek The Platform

Şimşek Unmanned Aerial Vehicle (UAV) platform represents a significant advancement in aerospace engineering and military technology. Developed by Turkish Aerospace Industries (TAI), Şimşek is a high-speed target drone designed to simulate enemy aircraft and missiles for the purpose of testing and training air defense systems. This UAV platform demonstrates Turkey's growing capabilities in the field of unmanned systems, providing versatile and reliable solutions for both military and civilian applications.

Şimşek is engineered to operate under various challenging conditions, offering exceptional performance in terms of speed, maneuverability, and operational endurance. It is capable of reaching speeds up to 400 knots and can fly at altitudes ranging from 1000 feet to 25,000 feet, making it suitable for a wide range of mission profiles. The UAV is equipped with an advanced navigation system, ensuring precise control and accuracy during flight missions.


**Figure 1.2:** TAI - Şimşek [2]

The platform is designed to be launched from ground-based catapults and can be recovered via parachute, enhancing its operational flexibility and ease of deployment. Additionally, Şimşek can be configured with various payloads, including radar and infrared augmentation devices, to simulate different types of threats and provide comprehensive training scenarios for air defense units.

The Şimşek UAV serves as a critical tool for the Turkish Armed Forces and allied military units, offering a cost-effective and efficient means of training and evaluating air defense systems. Its ability to mimic enemy targets allows defense forces to practice and refine their tactics, techniques, and procedures in a controlled environment, ultimately improving readiness and operational effectiveness.

Moreover, the development and deployment of Şimşek underscore Turkey's commitment to advancing its indigenous defense capabilities. The UAV platform not only enhances national security but also positions Turkey as a competitive player in the global defense industry, with potential export opportunities to allied nations seeking advanced training solutions.

The Şimşek UAV platform is a testament to the innovative engineering and strategic foresight of Turkish Aerospace Industries. By providing a high-performance, versatile,

and reliable target drone, Şimşek plays a pivotal role in enhancing air defense training and operational readiness. Its continued development and deployment will likely contribute significantly to the advancement of unmanned aerial systems and defense capabilities on a global scale [2].

# 2. PURPOSE OF THE STUDY

The integration of Model Based Systems Engineering (MBSE) into the aerospace industry has garnered significant attention in recent years, reflecting a paradigm shift in engineering methodologies. MBSE offers a holistic approach to system design, utilizing models to represent the complex interactions and behaviors of systems, thereby enhancing understanding, collaboration, and decision-making processes.

In the context of engine design, the application of MBSE holds particular promise due to the intricate nature of propulsion systems and the need for efficient, reliable, and environmentally sustainable solutions.

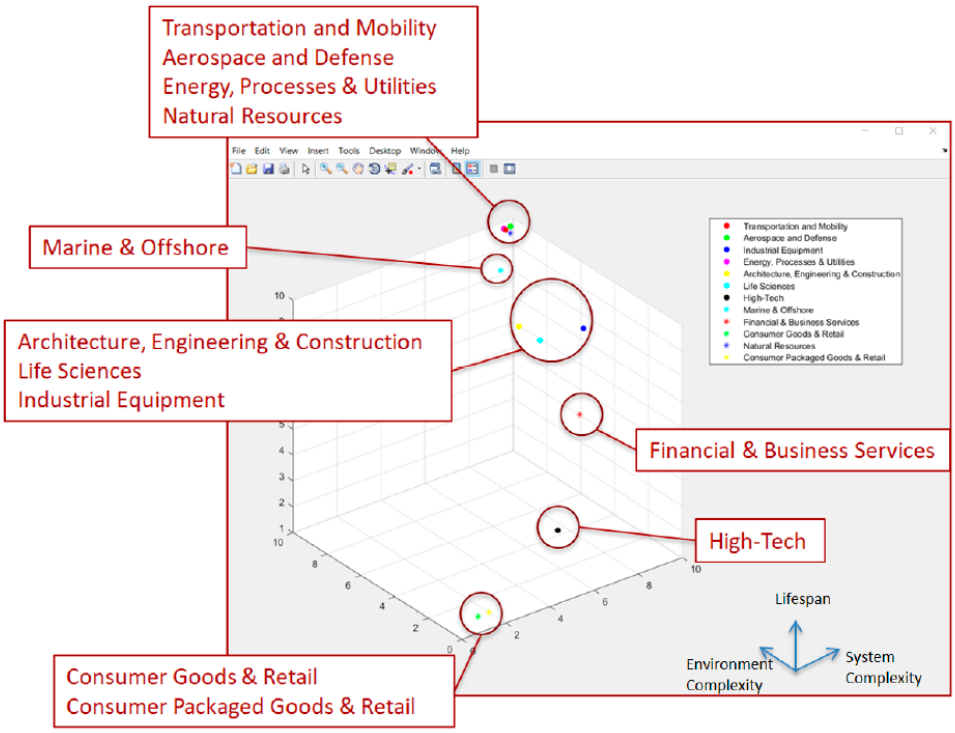



**Figure 2.1:** The potential calculation based on industry classification according to lifetime, environment, and system complexity [3]

Previous research has highlighted the advantages of MBSE in various aerospace engineering domains, including aircraft design, avionics systems, and spacecraft development.

Studies such as that by Kaitlin Henderson and Alejandro Salado (2019) [4] has demonstrated the effectiveness of MBSE in improving design integration, reducing development time, and enhancing system performance. However, while these studies provide valuable insights into the broader applications of MBSE, there remains a notable gap in the literature regarding its specific implementation in aerospace engineering.

A limited number of studies have explored the use of MBSE in engine design, with most focusing on conceptual modeling, requirements management, and system architecture definition. For instance, the work of Bayramoglu (2022) [5]demonstrated the industrial adoption of MBSE especially in the defense industry. Similarly, the study by Havemana, Bonnemaa and van den Bergb (2014) [6] emphasized the importance of making design decisions early in the design process with late design changes costing negative effects in product lifecycles. MBSE in facilitating collaboration between design teams and ensuring consistency across multiple engine subsystems. It is mentioned that increasing complexity of products and production systems in the book of Mehr and Lüder [7]. By facilitating the evaluation and efficient creation of model-based development environments, Schöberl (2019) [8] enables enterprises to use new digital possibilities and adjust to the demands of increasingly complex products.

Despite these contributions, there is still a lack of comprehensive research addressing the full spectrum of MBSE applications in gas turbine engine design. Specifically, there is a need for studies that delve deeper into the integration of MBSE across the entire engine lifecycle, from conceptual design to manufacturing and maintenance. Additionally, there is a dearth of empirical evidence examining the practical challenges and benefits of implementing MBSE in engine development projects, particularly in the context of real-world aerospace engineering organizations.

Therefore, this thesis aims to fill this gap in the literature by conducting a thorough investigation into the use of MBSE in gas turbine engine architecture development. By examining existing literature, industry best practices, and case studies, this research seeks to identify the key advantages and limitations of MBSE in this specific domain. Moreover, this study aims to provide practical insights and recommendations for aerospace engineering practitioners seeking to leverage MBSE to enhance their engine design processes.

In summary, while previous research has laid the groundwork for understanding the broader applications of MBSE in aerospace engineering, there is still much to be explored regarding its implementation in gas turbine engine design. This thesis seeks to address this gap by providing a comprehensive analysis of MBSE in engine architecture development, thereby contributing to the advancement of both academic knowledge and industrial practice in this field.

# 3. SYSTEMS ENGINEERING

The field of engineering covers the processes of designing, developing, implementing and managing multiple parts and interrelated systems throughout the product's existence. These systems consist of interrelated components often originating from different disciplines. Systems engineering involves bringing these components together, understanding their interactions, optimizing them, and integrating them to meet specified requirements.

For instance, complex systems such as the design and production of an aircraft rely on principles of systems engineering. In this process, various components (e.g., engines, wings, fuselage, systems) are integrated, and their interactions are optimized. Systems engineers analyze the relationships between these components to ensure the safety, performance, and efficiency of the aircraft, while managing the development process.

In this section, we aim to introduce systems engineering, delving into the concept of a system, the systems engineering paradigm, and methodologies for process adaptation.

## 3.1. Definition of System

There exist system definitions that share similarities and serve as the foundational basis for existing studies. Systems exhibit a diverse range of complexity, spanning from rudimentary mechanical apparatuses to intricate socio-technical systems. Their delineation is marked by attributes including delineated boundaries, inputs, outputs, functions, and the interconnectedness among constituent components.

In accordance with the standards established by the International Council on Systems Engineering (INCOSE) and the International Organization for Standardization (ISO) 15288, a system is defined as "a combination of interacting elements organized to achieve one or more stated purposes"[9], [10].

"A set or arrangement of elements and processes that are related and whose behavior satisfies customer/operational needs and provides for life cycle sustainment of the products."[11]

"A system is a construct or collection of different elements that together produce results not obtainable by the elements alone. The elements, or parts, can include people, hardware, software, facilities, policies, and documents; that is, all things required to produce systems-level results. The results include system-level qualities, properties, characteristics, functions, behavior, and performance. The value added by the system as a whole, beyond that contributed independently by the parts, is primarily created by the relationship among the parts; that is, how they are interconnected." [9]

"A combination of interacting elements organized to achieve one or more stated purposes."[10]

"The combination of elements that function together to produce the capability to meet a need. The elements include all hardware, software, equipment, facilities, personnel, processes, and procedures needed for this purpose. The end product (which performs operational functions) and enabling products (which provide life-cycle support services to the operational end products) that make up a system." [12]

## 3.2. System Life Cycle

Systems engineering is centered around the early analysis and elicitation of customer needs and desired functionality during the development phase. This involves documenting requirements, followed by design synthesis and system validation, all while considering the entirety of the problem and the system lifecycle. It necessitates a comprehensive understanding of all stakeholders involved. According to Oliver ,[13] The systems engineering can be exam under the following categories,

- Technical processes in systems engineering
- Managerial processes in systems engineering

In various applications, numerous models are utilized within the industry, all with the common objective of delineating the relationships among the aforementioned stages while integrating feedback. Two of these models that serve as examples are the Waterfall model and the V model [14].

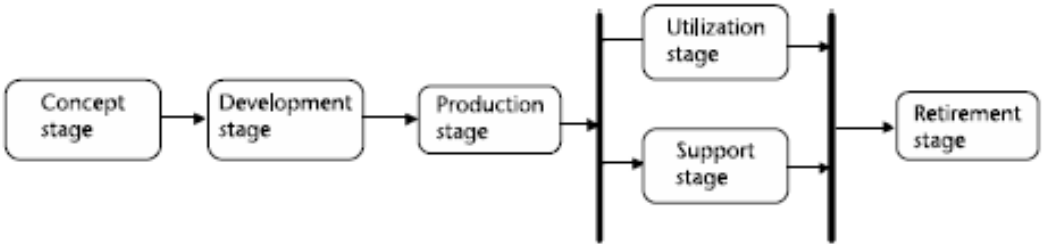



**Figure 3.1:** System Life Cycle (V Model) [13]

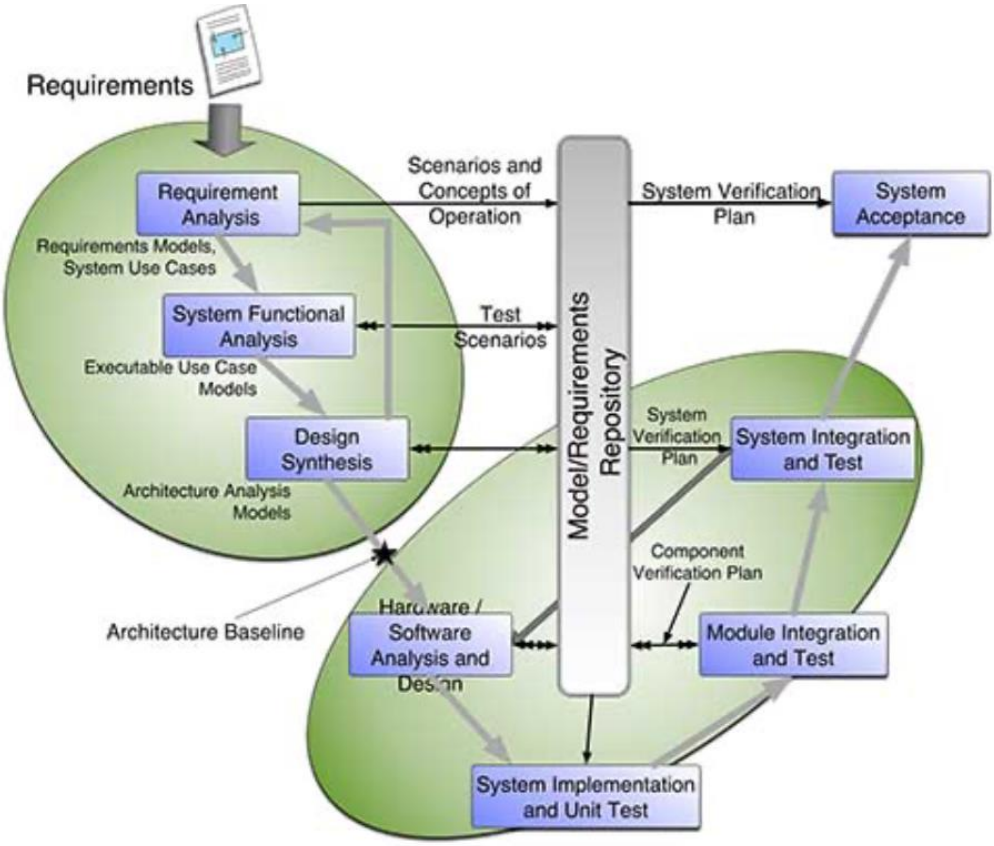



**Figure 3.2:** SE Lifecycle Modeling [14]

The inaugural edition of ISO 15288:2002 [10] delineates the system life cycle from the conceptualization stage to the retirement stage, illustrated in Figure 3.1 and outlined below.

**The Concept Stage** is when it is determined that a new system is required. Analysis, cost estimates, testing, searches, and feasibility assessments are some of the activities. After evaluating several options, outputs such as concept of operations, risk registers, stakeholder requirements, feasibility assessments, and preliminary design outlines are produced.

**The Development Stage** is where specific system requirements are defined and design solutions are developed in order to produce a workable product. Interfaces must be specified, designed, fabricated, integrated, and tested as needed. Requirements for production, training, and support facilities are determined. The system design, production schedules, operating manuals, training guides, maintenance protocols, cost projections, updated risk registers, and descriptions of the enabling processes for later phases are examples of outputs.

**The Production Stage** commences upon assent for system production. The system could be either matchless or mass-produced, and enhancements or redesigns may occur during this phase. The primary thruput is the manufactured item, accompanied by outputs pertaining to quality control and item transfer.

**The Utilization Stage** is following setup of the system and switching to operational use, the utilization stage begins. The system operates at designated sites, potentially evolving through different configurations. This stage concludes upon system decommissioning, with outputs including the deployed system, performance monitoring, exit conditions and cost analyses.

**The Support Stage** encompasses upkeep and supplies for transportation, including operational support system processes. Outcomes involve trained maintenance personnel, product maintenance, and logistical support. The utilization and support phases might coincide with the phase of manufacturing, in the manner indicated by ISO 15288.

**The Retirement Stage** involves the system's pulling out along with any related maintenance and technical assistance. Key thruputs pertain to disjoining activities, such as disposal, refurbishment, or recycling in compliance with relevant regulations. Waste removal and staff reallocation are significant considerations [15].

## 3.3. What is Systems Engineering?

According to Simon Ramo[16]; the roots of systems engineering can be traced back to ancient times, perhaps even as far as the construction of the pyramids. Various large-scale endeavors throughout history have inherently employed aspects of the systems approach. Notable technological advancements of the 19th and early 20th centuries, such as the railroad transportation system, electric power generation and distribution,

and the telephone system, exemplify early examples of major systems. The development of these systems laid the groundwork for modern communication design techniques. Furthermore, advancements like radar and the atomic bomb during World War II required systematic engineering approaches and expanded the boundaries of applied physics. Operations research techniques, initially developed during World War II to optimize system performance, have become invaluable tools for systems engineers. Despite these early applications, systems engineering did not gain widespread recognition as a distinct engineering discipline. It wasn't until the post-war era, with the development of ground-to-ground, ground-to-air, and air-to-air projectile systems, that modern systems engineering received significant attention. These systems encompassed various technologies, including communications, radar, controls, aerodynamics, structures, and propulsion, further solidifying the importance of systems engineering in addressing complex challenges.

The origin of the term Systems Engineering can be traced back to the 1940s and it is generally accepted that the term was coined by Bell Telephone Laboratories in the USA. Since then, the term has come to be used for complex structures in which the overall behavior of the parts that make up the system differs from the behavior of the whole [17].

The primary objective of the field of systems engineering is to handle diversity. In other words, taking into account intricate systems with multiple relationships between their constituent pieces, both internal and external. Thus, managing complexity entails managing several components that communicate in various ways, whether they are software-, mechanical-, or electrical-intensive [15].

The aerospace and defense sectors have widely embraced systems engineering as a method to address complex and mission critical challenges. These solutions typically encompass hardware, software, data, personnel, and infrastructure. The acknowledged benefits of systems engineering in risk mitigation, imperspicuity management, and improving output and quality has led to its adoption in various other industries, including automotive and telecommunications etc.

In contemporary times, systems engineering is widely recognized as a fundamental aspect of major space and military development endeavors. An illustrative instance of

a latest expansive space system is the development of the tracking and data relay satellite system (TDRSS) undertaken for NASA.

The definition of systems engineering is included in different sources and standards as follows:

"An interdisciplinary approach and means to enable the realization of successful systems. It encapsulates the entirety of a system's lifecycle, commencing from conception through disposal, and necessitates the amalgamation of diverse engineering disciplines, management protocols, and technical methodologies to ascertain that the system aligns with stakeholder needs and specifications." [9], [10]

"System engineering is a robust approach to the design, creation, and operation of systems. In simple terms, the approach consists of identification and quantification of system goals, creation of alternative system design concepts, performance of design trades, selection and implementation of the best design, verification that the design is properly built and integrated, and post-implementation assessment of how well the system meets (or met) the goals."[12]

Three commonly used definitions of systems engineering are as follows. The common theme of all of them is provided by the best-known technical standards.

- "A logical sequence of activities and decisions that transforms an operational need into a description of system performance parameters and a preferred system configuration." (MIL-STD-499A, Engineering Management, 1 May 1974. Now cancelled.) [18]
- "An interdisciplinary approach that encompasses the entire technical effort and evolves into and verifies an integrated and life cycle balanced set of system people, products, and process solutions that satisfy customer needs." (EIA Standard IS-632, Systems Engineering, December 1994.) [19]
- "An interdisciplinary, collaborative approach that derives, evolves, and verifies a life-cycle balanced system solution which satisfies customer expectations and meets public acceptability." (IEEE P1220, Standard for Application and Management of the Systems Engineering Process, [Final Draft], 26 September 1994.) [20]

In essence, systems engineering can be described as an interdisciplinary engineering management process, which is focused on developing and validating an integrated,

life-cycle balanced set of solutions for systems that effectively meet the requirements of the customer.

The systems engineering process is delineated into three stages for examination:

- Systems Engineering Management
- Systems Engineering Process
- Systems Engineering Considerations

The engineering exertion is viewed as a process that spans the entire system life cycle. This perspective contrasts with the 67 case studies that lacked metrics and primarily focused on applying a method to specifications engineering based on MBSE. In his thesis "Systems Engineering Return on Investment," Honour illustrated this point with the graphic in Figure 3.3, This displays the variation in the overall effort and the systems engineering effort. It's crucial to remember that the total systems engineering effort covers the entire development life cycle. The shaded areas in the graphic represent the requirements effort, which NASA refers to as the definition phase [21].

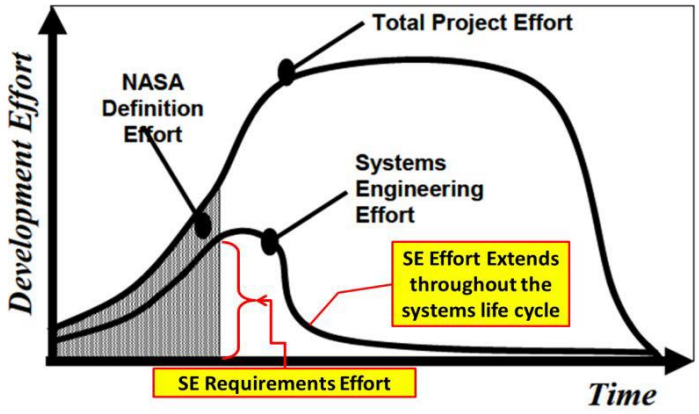



**Figure 3.3:** Comparison of SE Effort to Total Effort [22]

### 3.3.1. Systems Engineering Management

Systems engineering management involves the integration of three primary activities,

- Development phasing, which oversees the design process, establishes baselines to coordinate design efforts, and serves as a crucial connection between technical and acquisition management.

- The systems engineering process, which offers a structured framework for addressing design challenges and tracking requirements flow throughout the design phase.
- Life cycle integration, which engages customers in the design process and ensures the viability of the system developed over its entire lifespan.

Each of these activities plays a vital role in effective development management. Development phasing serves to control the design effort, establish design baselines, and facilitate key events for assessing design viability, thereby influencing acquisition management milestones. At the core of the systems engineering process, aims to transform needs into foundation configurations, structures, and standards through a structured yet adaptable approach. This iterative process ensures solutions are developed with strict control and traceability to satisfy client requirements. Furthermore, life cycle incorporation is imperative to sustain the viability of the design choice during the course of the system's existence. It encompasses planning for product and process development and integrates various functional considerations into the design and engineering process. This perspective optimizes product cycle time, design and renewal needs [23].

### 3.3.2. Systems Engineering Process

The process of systems engineering is a comprehensive, iterative, and recursive problem-solving approach, applied sequentially across all stages of development. Its primary objectives are to translate needs and requirements into detailed system descriptions, provide decision makers with relevant information, and guide subsequent development stages.

Outlined in Figure 3.4, the core activities of systems engineering include Requirements Analysis, Functional Analysis and Allocation, and Design Synthesis. The methods and resources referred to as Systems Evaluation and Control are used to support these operations. Controls in systems engineering are used to keep an eye on requirements and selections, manage connections, uphold technical norms, track costs and schedules, minimize risks, evaluate technical performance, and verify that criteria are met.

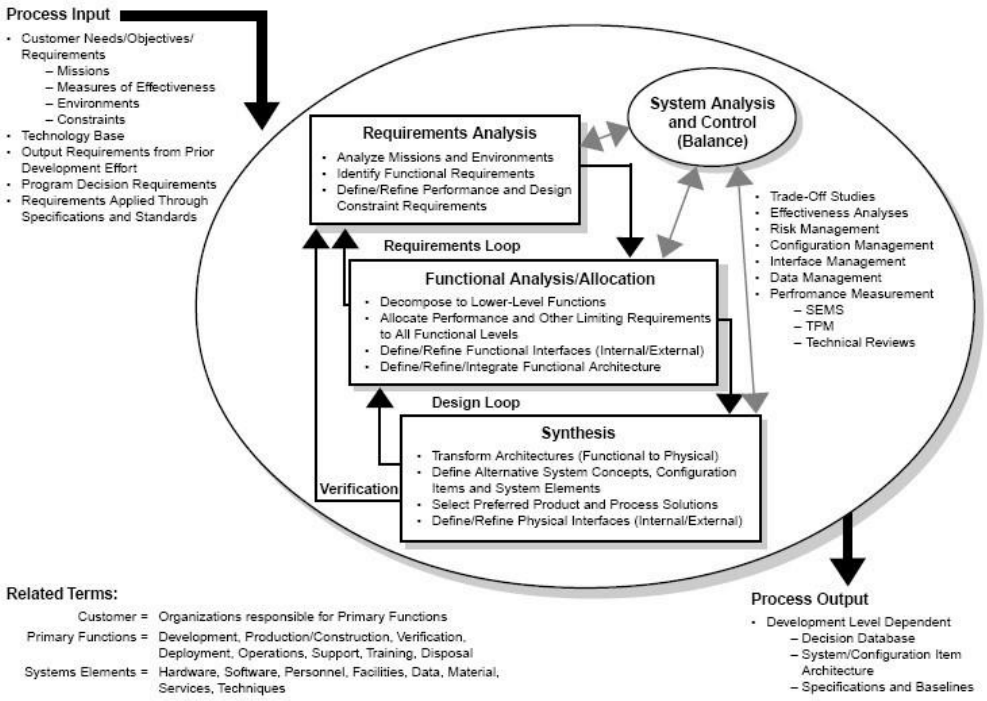



**Figure 3.4:** The Systems Engineering Process [23]

Throughout the systems engineering process, architectures are developed to enhance system understanding. The term "architecture" is utilized in different engineering contexts, serving as a general framework for subsystem integration and as a detailed description of specific system aspects. In the realm of Systems Engineering Management, three universally applicable architectures are recognized: functional, physical, and system architectures.

The Functional Architecture organizes and structures allocated functional and performance requirements, while the Physical Architecture illustrates the system breakdown into subsystems and components. Lastly, the System Architecture delineates all necessary products and processes to support the system across its lifecycle, including creation, manufacturing, implementation, management, assistance, elimination, instruction, and confirmation [23].

### 3.3.3. Systems Engineering Considerations

Systems engineering serves as a standardized and systematic management process aimed at developing system results consistently, even in environments characterized

by change and uncertainty. It facilitates concurrent product and process development and fosters effective communication through a common framework.

The role of systems engineers encompasses various responsibilities:

- Crafting holistic system design solutions that strike a balance between cost, schedule, performance, and risk.
- Generating and monitoring technical data is essential for deciding.
- Ensuring that technical solutions align with client specifications.
- Designing systems that are economically viable and sustainable throughout their lifecycle.
- Assessing and managing internal and external interface compatibility using an open systems approach.
- Establishing baselines and implementing configuration control measures.
- Providing proper focus and structure for system and major sub-system level design Integrated Product Teams (IPTs) [23].

## 3.4. Tailoring The Process

Systems engineering is implemented throughout all stages of acquisition and support for systems of various scales, including both new developments and product enhancements, as well as for single or multiple procurements. It is imperative that the process be customized to accommodate diverse needs and requirements. Factors to consider in tailoring include the size and complexity of the system, the level of detail in system description, operational situations and missions, limitations and specifications, technological underpinnings, primary risk considerations, and best practices for organizations and capabilities.

The foundational principles of systems engineering, which have stood the test of time, must be preserved to ensure continuity and control. In the design of complex systems, a comprehensive understanding of system objectives ought to come before creating component effectiveness specifications, which in turn precede detailed component descriptions. While certain aspects of the structure might be mandated by constraints or interfaces, the design process generally commences with an analysis of requirements and the determination of system functionalities before considering

physical alternatives. It is essential to maintain configuration control and effectively manage risks [23].

Careful tailoring of the systems engineering process is necessary to mitigate the introduction of unforeseen risks and uncertainties. In the absence of systems engineering's regulation, coordination, and traceability, an unpredictable environment develops, often resulting in unforeseen challenges that significantly impact cost and schedule. Systems engineering promotes taking advantage of tools and methodologies to better understand and handle the complexities inherent in systems. System architecture, modelling and simulation, statistical evaluation, mathematical optimization, system behavior, systems evaluation, reliability engineering, and deciding approaches are a few examples of these kinds of tools.

Since behaviors and interactions among system components are unpredictable, taking a multidisciplinary approach to system engineering entails inherent complexity. One of the primary objectives of systems engineering is to define and characterize these systems and subsystems, as well as the interactions among them, bridging the gap between informal user requirements, operational needs, marketing considerations, and technical specifications.

# 4. CONTRASTING THE DOCUMENT-BASED AND MODEL BASED APPROACH

Systems engineering encompasses various methodologies for analyzing, designing, and managing complex systems. Among these methodologies, two prominent approaches are Model Based Systems Engineering (MBSE) and Document-Based Systems Engineering (DBSE).

Model Based Systems Engineering emphasizes the use of mathematical and graphical models throughout the design and development process of systems. This approach highlights the clear definition of system requirements, functions, behaviors, and relationships. Model Based Systems Engineering involves steps such as establishing system architecture, analyzing design alternatives, and evaluating system performance. This approach provides an effective tool for managing the complexity of systems and facilitates effective communication across different disciplines.

On the other hand, Document-Based Systems Engineering emphasizes maintaining system requirements, design decisions, and other critical information in document form. This approach prioritizes the use of written documents and emphasizes the documentation and tracking requirements throughout every stage of the system development process. Document-Based Systems Engineering ensures the traceability and verifiability of system requirements and design decisions. This approach may provide convenience, especially for teams familiar with traditional methods, and underscores the importance of the documentation process.

Both approaches have their own advantages and disadvantages, and the choice of method may vary depending on the project requirements, team expertise, and project complexity. Model Based Systems Engineering may be more suitable for the design and analysis of complex systems, while Document Based Systems Engineering may be more suitable for smaller and simpler projects.

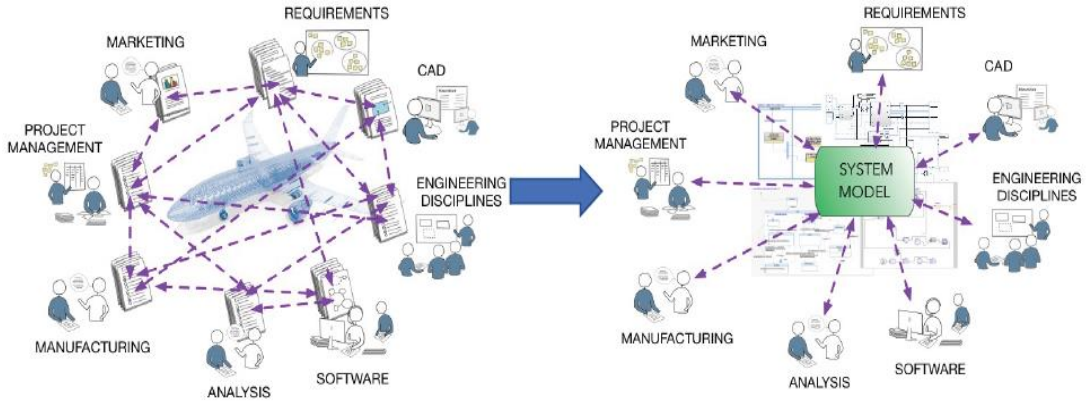



**Figure 4.1:** DBSE vs MBSE

## 4.1. Document-Based Systems Engineering Approach

Large-scale projects traditionally employ a document-centric approach in systems engineering for effective management. This method involves creating textual specifications and design documents, either physical or electronic, shared among stakeholders like customers, users, developers, and testers. These documents convey system requirements and design specifics through textual descriptions, graphical representations from drawing tools, and tabular data or plots derived from analysis models or databases. The focus is on managing and validating documentation to ensure accuracy, completeness, and consistency, thereby verifying that the developed system aligns with specified requirements.

Specifications are typically structured hierarchically in a specification tree, encompassing the system, its subsystems, and hardware/software components. A Systems Engineering Management Plan (SEMP) outlines how the engineering process is executed within the project, fostering collaboration across engineering disciplines to produce necessary documentation aligned with the specification tree. Planning involves estimating time and resources required for document generation, with progress evaluated based on document completion status.

Concept of operations documents define how the system supports its mission, while functional analysis breaks down system functions and assigns them to components using tools like functional flow diagrams and schematic block diagrams. Engineering

disciplines conduct trade studies and analyses to optimize designs and allocate performance requirements. Various analysis models assess system aspects such as performance, reliability, safety, and mass properties.

The traceability of specifications is maintained via the document-centric method, which links requirements across various standard levels. These requirements, together with verification techniques, are extracted by tools for managing requirements and stored in databases to guarantee traceability across system components or subsystems. But even with all of its rigor, this approach is not without its limitations. Due to information being scattered over several papers, evaluating the consistency, completeness, and linkages between requirements, design components, engineering evaluations, and test data can be difficult. This difficulty makes it more difficult to do traceability and modification effect evaluations, as well as to comprehend certain system components [24].

Moreover, it may be difficult to maintain or reuse system specifications and design data for changing or alternative system designs if specifications, system-level design, and specific lower-level designs are not coordinated. It can be troublesome and may not fairly represent the quality of system specifications and designs to base systems engineering operations just on documentation status. Throughout the implementation, evaluation, or post-delivery phases, these constraints may result in inefficiencies that have an impact on budget, schedule, and possible quality problems [24].

## 4.2. Model Based Systems Engineering Approach

A model based methodology has long been established as standard method in various engineering disciplines such as electrical and mechanical design. Mechanical engineering, for example, transitioned from conventional drawing methods to increasingly sophisticated two-dimensional and three-dimensional computer-aided design (CAD) tools starting in the 1980s. Similarly, electrical engineering shifted from manual circuit design to automated schematic capture and circuit analysis within a comparable timeframe. In the realm of software engineering, computer-aided software engineering (CASE) gained popularity during the 1980s, utilizing graphical representations to abstractly express software beyond programming languages. The

adoption of modeling for software development has been on the rise, particularly with the introduction of the Unified Modeling Language (UML) in the 1990s.

The model based approach is now gaining traction in systems engineering as well. The formalization of Model Based Systems Engineering (MBSE) began in 1993 with the introduction of mathematical formalism by Wayne Wymore. The increasing capabilities of computer processing, storage, and network technologies, coupled with the emphasis on systems engineering standards, have presented an opportunity to significantly advance the practice of MBSE. It is anticipated that MBSE will follow a trajectory similar to that of other engineering disciplines, gradually becoming standard practice and fully integrated into a broader model based engineering framework.

In essence, Model Based Systems Engineering (MBSE) entails the systematic use of modeling to support various systems engineering activities, including requirements analysis, draw up, evaluation, confirmation, and endorsement. Unlike the document-based approach described earlier, where specifications and design details are expressed primarily through textual documents, MBSE emphasizes the use of models to represent system components, behaviors, and interactions. The outcome of MBSE activities is a cohesive system model, integrated into the engineering baseline, with an emphasis on refining and evolving the model using model based methods and tools. This approach aims to enhance the quality of system specifications and designs, facilitate the recycling of system items, and enhance communication among the group of developers [24].

Examples of MBSE applications include the evolution of intricate aerospace systems (spacecraft or aircraft), where intricate interactions between various subsystems need to be accurately modeled and analyzed. Additionally, MBSE can be applied in the automotive industry for designing advanced driver assistance systems (ADAS) or in the healthcare sector for developing medical devices with complex functionality and safety requirements.

# 5. MODEL BASED SYSTEMS ENGINEERING

## 5.1. What is a model?

Models serve various critical functions within systems engineering, with their definitions spanning multiple interpretations. A model can be conceived as:

- A conceptualization of reality crafted to address specific inquiries concerning the tangible world.
- A mimicry, similarity or portrayal of a genuine method or framework.
- A theoretical, calculated, or tangible instrument employed to guide deciding.

What constitutes a model?

- It encapsulates a streamlined rendition of a notion, occurrence, correlation, configuration, or system.
- It manifests as a graphical, mathematical, or tangible depiction.
- It abstracts reality by omitting extraneous components.

The purposes of employing a model encompass:

- Facilitating comprehension.
- Assisting in decision-making processes and exploring hypothetical scenarios.
- Providing explanations, oversight, and prognostic capabilities [12].

### 5.1.1. Modeling

In the initial stages of a systems engineer's engagement with a difficult problem, visual descriptions are indispensable for conveying the functional and data requirements of a system. Various graphical representations serve this purpose, including Functional Flow Block Diagrams (FFBD), Model Based Designs, Data Flow Diagrams (DFD), $N^2$ charts, IDEF0 diagrams, Use Case diagrams, Sequence diagrams, Block diagrams, Signal-flow graphs, and USL function maps. These representations establish connections among the different subsystems or parts of a system via interfaces, data, or functions, chosen according to industry-specific needs. For example, $N^2$ charts may be utilized to highlight system interfaces. During the design phase, structural and

behavioral models are crafted to delineate the functional relationships among subsystems or components effectively.

Once the system requirements are comprehended, the responsibility shifts to the systems engineer, alongside other engineers, to refine and select the optimal technology for the task at hand. Commencing with a commerce research, systems engineering advocates for employing biased choices to discern most suitable selection. Determination matrices, such as the Pugh method (alternatively, Quality Function Deployment), serve as viable tools for this purpose, considering all relevant criteria. The outcomes of the trade study subsequently inform the design process, influencing the graphical symbols of the system, albeit deprived of altering the underlying needs. In the systems engineering manner, this stage corresponds to an iterative step undertaken until a feasible solution is reached. Techniques like System behavior (feedback control), statistical assessment, analysis of reliability, and optimization techniques are often employed to populate decision matrices.

The practice of modeling has been integral to problem-solving and mathematical endeavors throughout history. From a Systems Engineering standpoint, early models, such as the primitive forms of Gantt Charts devised by Karol Adamiecki in 1896 (formalized by Gant before World War I), represent seminal attempts at visual modeling techniques, laying the groundwork for modern methodologies [25].

Modeling serves as a solution to the core challenges in Systems Engineering: complexity, comprehension deficits, and communication obstacles.

Complexity refers to the failure to identify or manage the intricacies inherent within a System.

- Essential complexity: This inherent complexity remains unalterable within the System, yet modeling enables pinpointing and control measures, such as minimizing dependencies and interactions.
- Accidental complexity: Arising from human factors, this complexity can be mitigated through modeling insights, streamlining developmental approaches to minimize its impact.
- Lack of comprehension: Understanding the initial system needs is pivotal for systems engineers. Continual comprehension throughout the System's Life

Cycle demands modeling as a tool for understanding and identifying deficiencies.

- Communication barriers: Effective communication relies on a shared language across various interfaces, including interpersonal and inter-organizational interactions. Establishing a common lexicon is crucial, though a challenging endeavor explored further in subsequent chapters [17].

### 5.1.2. Two Dimensions of the Model

Modeling is an indispensable tool in Systems Engineering, offering insights and understanding crucial to navigating the complexities of systems development.

Modeling finds its place at every stage of the system's lifecycle, its utilization contingent upon the specific objectives and requirements of the project. The efficacy of modeling efforts hinges upon their capacity to augment the value of ongoing work, serving as a conduit for enhanced comprehension and analysis.

In any modeling endeavor, it is imperative to address the system's behavioral and structural aspects:

Structural dimensions: Concerns the foundational elements and their interrelationships within the system, elucidating the fundamental components, their associations, and functionalities.

Behavioral dimensions: Encompasses the dynamic behaviors exhibited by system components, comprising their interactions, sequencing, conditional responses, and temporal considerations.

A comprehensive understanding of the system necessitates a holistic exploration of both the "what" and "how" dimensions, transcending static representations to encompass the dynamic interplay of stakeholders, processes, and functionalities [17].

## 5.2. Model Based Systems Engineering

This part offers a comprehensive examination of MBSE, delineating its rationale, elucidating fundamental modeling concepts, and outlining the primary components essential for its implementation.

It discusses the advantages of MBSE over conventional document-centric approaches in systems engineering, highlighting the various uses of modelling and the critical function that perspectives and points of view play in enabling effective MBSE practices. Furthermore, it explores the critical components of MBSE, including the modeling language, methodology, and tools, alongside an analysis of alternative options available for each element. Additionally, the section provides an overview of SysML, the primary language utilized in MBSE methodologies discussed in the text.

MBSE embodies a systematic approach that integrates modeling throughout the systems engineering process, aiding in specification, analysis, design, and validation activities. Its primary objective is to generate coherent system models, thereby enhancing specification precision, design efficiency, and communication within development teams.

In essence, MBSE serves as a formalized method for leveraging modeling to support many features of system development, encompassing requirements, strategy, analysis, and validation, from initial conceptualization through subsequent development stages and life cycle phases.

### 5.2.1. Why Do We Model?

The conventional paradigm in systems engineering predominantly relies on a document-centric approach. Throughout the system's lifecycle, an array of documents is produced by diverse stakeholders to encapsulate the decisions and outcomes of engineering endeavors. Given the iterative nature of engineering activities, adjustments to the system's design are inevitable in response to identified flaws. Moreover, the involvement of stakeholders with varying concerns and expertise necessitates the utilization of diverse document formats to convey identical information, resulting in redundancy and inconsistency. Consequently, the effective management of these disparate documents becomes imperative to uphold the integrity, completeness, and validity of the information. However, this document-centric approach proves to be fraught with inefficiencies, consuming considerable time and resources, and is susceptible to errors, such as overlooking updates to system component names across relevant documents.

In contrast, Model Based Systems Engineering (MBSE) presents a viable alternative by formalizing modelling used to assist with system needs, engineering, evaluation, validation, and testing tasks. Unlike the conventional approach, where models serve as auxiliary tools, MBSE prioritizes the creation of an integrated, coherent system model as the central artifact. MBSE has garnered widespread acceptance across various industries due to its ability to address the shortcomings inherent in the document-centric approach. Companies embracing MBSE report notable enhancements in quality, productivity, and knowledge dissemination. Through the automation of tasks like document generation and error reduction, MBSE augments design integrity, traceability, and reusability. Furthermore, the establishment of metrics grounded in modeling facilitates the evaluation of design quality and progression. Nonetheless, ensuring the efficacy of such metrics necessitates the establishment of stringent quality criteria aligned with the model's intended purpose [15].

### 5.2.2. How We Model?

In order to conduct modeling effectively and efficiently, it is imperative to exercise control and management over the modeling process. While it may be simple to generate numerous Views of a System, producing meaningful Views that collectively form the Model presents a distinct challenge.

Successful modeling necessitates a clear understanding of both the 'what' and the 'how' of View creation. To determine the appropriate Views to construct (and the methodology for doing so), the establishment of templates, termed Viewpoints, is essential. These Viewpoints are formulated based on domain-specific language or Ontology and, in conjunction, constitute the framework.

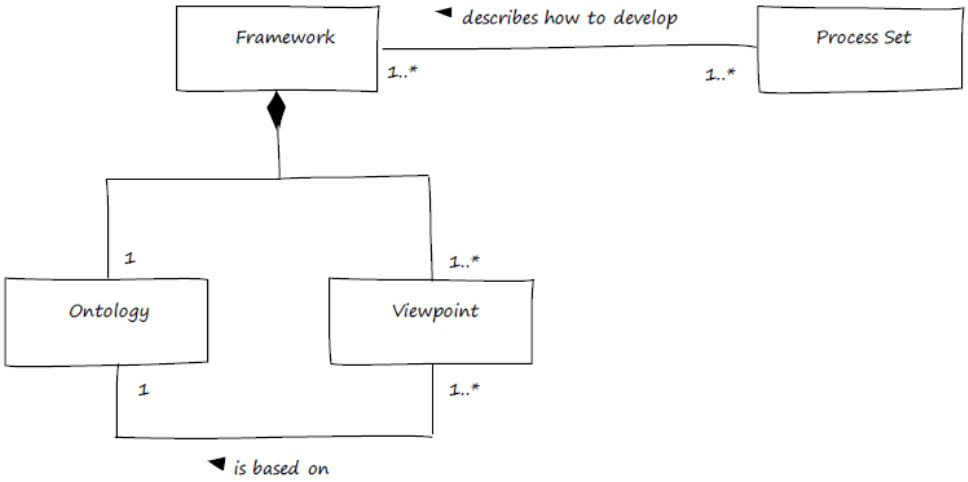



**Figure 5.1:** A sample modeling template [17]

The sample modeling template delineates several terms as follows:

**Framework**, defined as a structured collection comprising Viewpoints and an accompanying Ontology, which prescribes the Views permissible within a Model, representing the architectural essence of a System.

**Viewpoint,** constitutes the structured blueprint delineating the arrangement and substance of a View. Its construction is contingent upon employing concepts sourced exclusively from the Ontology, akin to a procedural guideline for crafting a View.

**Ontology**, embodies a comprehensive compendium of term definitions (termed as Ontology Elements) and the interconnections among them (referred to as Ontology Relationships), pertaining to a specific domain or organizational context. This framework establishes the conceptual framework and the linguistic apparatus utilized for denominating said concepts [17].

### 5.2.3. What is MBSE?

The MBSE approach introduces a fundamental shift from the conventional document-based methodology commonly practiced in engineering. While both approaches entail similar life cycle activities outlined in the INCOSE Systems Engineering Handbook, their primary differentiation lies in the artifacts they produce throughout these activities.

In the document-based approach, systems engineers manually generate various items such as concept of operations (ConOps) documents, required details, interface

definition documents (IDDs), and architecture description documents (ADDs), which are typically disparate text documents, spreadsheets, diagrams, and presentations stored in separate repositories.

However, the document-based approach proves to be cumbersome and costly to maintain. A significant portion of the total life cycle cost is incurred in managing this disjointed set of artifacts, leading to inconsistencies and obsolescence when changes are made. For instance, when a system architect refactors a block in the system hierarchy and decides to rename it, she must manually update every document and repository where the block is referenced, leading to potential errors and inefficiencies.

In contrast, MBSE offers a streamlined solution to these challenges. The main product of life cycle activities in MBSE is a combined, identical, and logical system model that is made with a specialized systems modelling tool. The same technology is used to automatically produce all additional secondary artefacts from the system model. The system model acts as the hub for design choices, encapsulating each choice as a relationship or model element.

Diagrams and automatically generated text artefacts in the MBSE technique are perspectives, not representations of the underlying system model. This is an important distinction that serves as the foundation for the higher return on investment (ROI) that MBSE offers compared to the conventional method. When changes are made to the model, they are instantly propagated to all associated diagrams and text artifacts, ensuring consistency, and reducing the risk of errors.

Ultimately, MBSE promises increased quality and affordability by preventing defects rather than fixing them later in the development process. At the heart of this approach lies the system model, the primary artifact that captures design decisions and drives the generation of all other artifacts, ensuring continuous consistency throughout the development lifecycle.

The diagram illustrates the three primary components of the approach: the System, the Approach, and Visualization. Three key points need to be clarified:

- The notation, such as SysML, is fundamentally distinct from the approach itself.

- The objective is to develop a Model that abstracts the System. This Model comprises various Views, which must be based on well-defined Viewpoints integral to our approach.
- The notation we utilize offers different types of Diagrams to visualize the Views that constitute our Model.
- The Ontology is critical to all activities within MBSE.

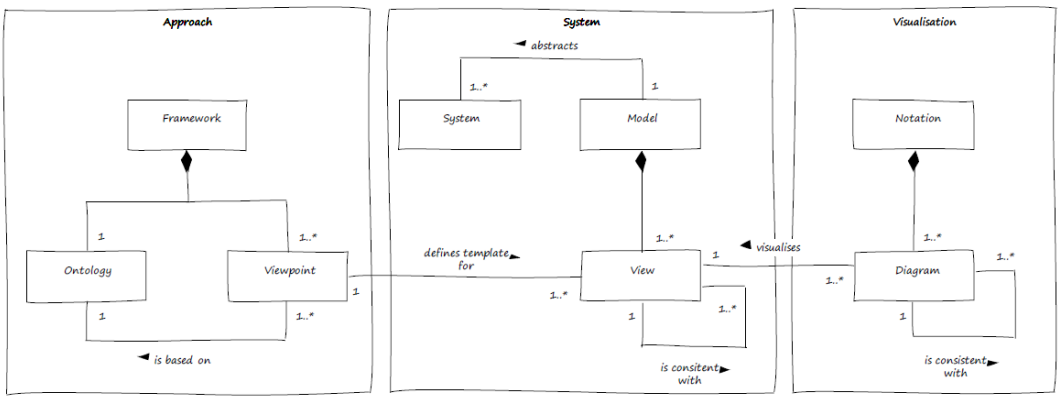



**Figure 5.2:** Approach, System and Visualisation [17]

## 5.2.4. Implementing MBSE

John Holt explains implementing MBSE with MBSE Mantra in "Don`T Panic MBSE" [17]. MBSE Mantra has main elements: People, Process and Tools.

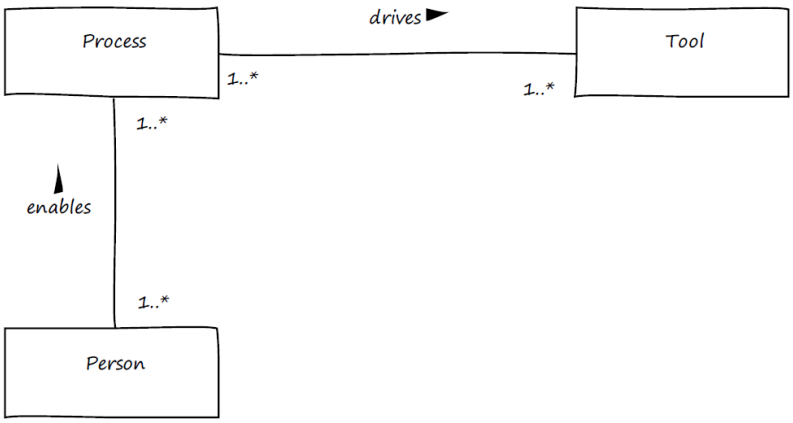



**Figure 5.3:** MBSE Mantra [17]

**Person (People)**, with the rights skills to enable the overall Process.

**Process**, the overall approach, including the Ontology, Frameworks and Process Sets.

**Tools**, sharp Tools that make our life easier and that are driven by the overall Process.

The people (Person) must enable the approach (Process) and the approach (Process) must drive the Tool.

**Jki?>**

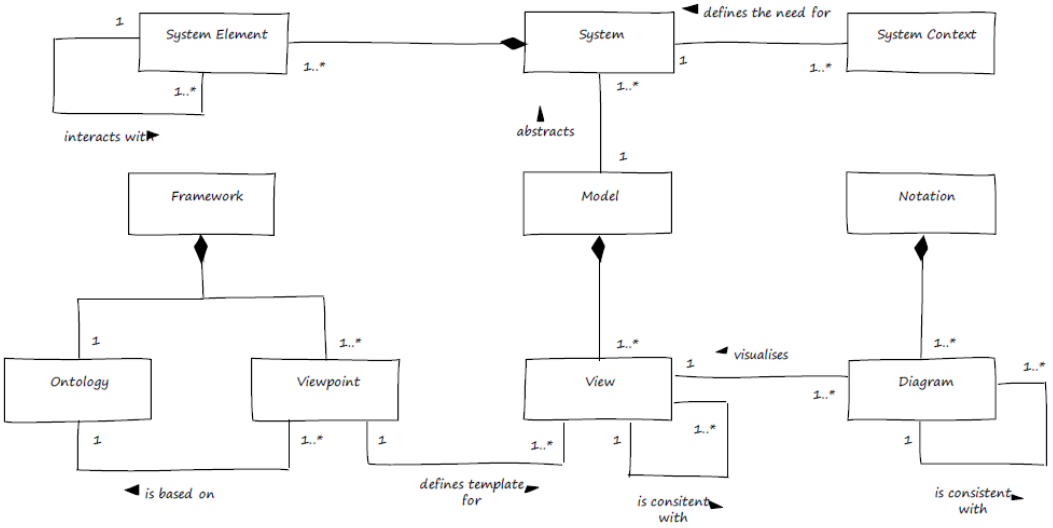



**Figure 5.4:** This diagram shows the main MBSE concept [17].

The importance of a common language in the form of an Ontology has been discussed, alongside the introduction and definition of key MBSE (Model Based Systems Engineering) terms. By integrating all these terms, a high-level MBSE Ontology is created. This Ontology is modeled in SysML. The diagram (Figure 5.4) illustrates the primary MBSE concepts that must be considered when implementing MBSE within an organization.

## 5.3. Modeling Languages, Methods and Tools for MBSE

The three foundational components of MBSE encapsulate the essential requirements for engineering teams to effectively implement the approach. These components consist of a modeling tool capable of executing tasks outlined in a specified methodology, facilitating system design, and consolidating the resultant design into a cohesive, centralized model represented in a standardized modeling language. In the

subsequent sections, it goes into depth around the basic elements and provides a brief review of the alternatives available for each [26].

### 5.3.1. Modeling Languages

When a model is constructed, a form of communication is essentially engaged in using a specialized language known as a modeling language. This language, typically semi-formal in nature, dictates the permissible elements to be included in the model, the acceptable relationships among these elements, and, particularly in graphical modeling languages, the approved notations for representing these elements and relationships on diagrams.

In the realm of Model Based Systems Engineering (MBSE), practitioners commonly utilize the Systems Modeling Language (SysML) to develop models capturing several parts of a system, containing its assembly, actions, requirements, and constraints. While this study primarily focuses on SysML, it's important to note that SysML is just one among many modeling languages available. Engineers and analysts working in different design domains, such as systems-of-systems, software, hardware, performance, and business processes, may opt for alternative modeling languages that better suit their specific system design requirements.

These alternative languages encompass both graphical and text-based modeling languages, such as UML, UPDM, BPMN, MARTE, SoaML, IDEFx, Verilog, and Modelica. However, regardless of the specific modeling language employed, the fundamental principle remains consistent: each language serves as a standardized medium of communication, providing unambiguous meaning to the elements and relationships within the model. The ability to construct and interpret well-defined models lies at the core of the MBSE methodology.

#### 5.3.1.1. SysML

The SysML, as defined by the Object Management Group (OMG), is characterized as a versatile graphical modeling language utilized for delineating, examining, designing, and validating intricate systems encompassing information, hardware, personnel, procedures, software, and facilities. The language employs graphical diagrams to

depict various aspects of systems, including requirements, behavior, structure, and constraints pertaining to system properties [27].

SysML emerged through a collaborative effort between INCOSE and OMG, departing from the UML language tailored for software systems. This initiative aimed to formulate a modeling language tailored for systems engineering endeavors, furnishing enhanced meaning, and introducing novel parts and diagrams are necessary for delineating a system's design intricacies and supporting the requirements of systems engineering activities. In terms of facilitating communication via models, SysML, the MBSE language, incorporates specific Viewpoint and View elements aligned with the ISO-42010 standard, thereby enabling the visualization of SysML models. Within SysML, a perspective is defined as a pattern outlining the principles and regulations governing the production of artifacts designed to present customized representations of the information encapsulated within a SysML model [28].

As previously noted, while SysML remains the predominant language, the adoption of MBSE does not inherently necessitate the use of SysML. Various other modeling languages cater to different design domains, including mechanical, software, and electronics, offering better alignment with activities throughout the system's life cycle. Furthermore, other languages for model-based systems engineering than SysML present compelling features and properties worth considering.

SysML serves as a highly versatile graphical modelling language, allowing users to efficiently visualize and convey several facets of a system's needs, behavior, framework, and parameters. Through SysML, nine distinct types of diagrams are defined, each tailored to convey specific details about different facets of a system's design, thereby facilitating comprehensive communication of system design information [26].

**SysML Purpose and Key Features,** A flexible graphical modelling language called SysML was created to facilitate the analysis, specification, design, validation, and verification of complex systems that include hardware, software, data, people, processes, buildings, and other components of both natural and man-made systems. It aims to aid in specifying and architecting systems while facilitating the specification of components for subsequent design using domain-specific languages like UML for software, VHDL for electrical, and three-dimensional geometric modeling for

mechanical design. SysML's primary goal is to enable the application of model based systems engineering (MBSE) by creating a unified and coherent model of the system. It achieves this by illustrating different facets of entities, parts, and systems, such as: [27]

- Classification, connectivity, and structural composition.
- Behaviour based on flow, messages, and states.
- Limitations on the attributes of performance and physical aspects.
- Distributions among conduct, framework, and limitations.
- The specifications and how they relate to test cases, design aspects, and other needs.

SysML diagrams are an important tool used to facilitate the study, creation, and exchange of intricate systems. These illustrations show a system's behavioral and structural traits. while also enabling the management of requirements and analysis of system performance. While structure diagrams illustrate a system's components and the relationships between them, behavior diagrams define system behaviors and how they change based on conditions. Requirements diagrams identify the functional and performance requirements of the system, while diagrams used for analysis and design evaluate system performance and reliability. SysML diagrams also serve as an effective communication tool among project stakeholders, making it easier to share information about complex systems and enhance understanding. Therefore, SysML diagrams play a significant role in model based systems engineering processes, assisting system engineers in decision-making and communication.

The SysML specification includes Figure 5.5, which offers a comprehensive a summary of the linkages and classifications among SysML diagrams. However, understanding this figure requires familiarity with its elements, particularly the lines with hollow, triangular arrowheads, denoting generalizations. These generalizations signify "is a type of" in the direction of the arrowhead. Activity diagrams, sequence diagrams, state machine diagrams, and use case diagrams fall under behavior diagrams, while block definition diagrams, internal block diagrams, and package diagrams are classified as structure diagrams. Parametric diagrams are a subset of internal block diagrams, making them a type of structure diagram. Additionally, requirements diagrams form a distinct category, yet they serve as a valuable

complement to the SysML diagram family, each diagram type serving specific purposes within the modeling framework [26]

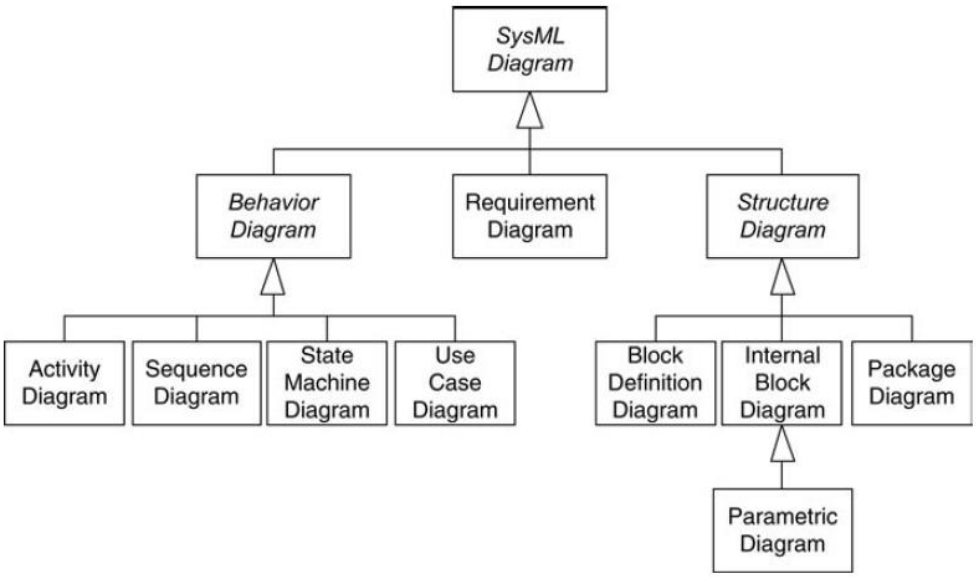



**Figure 5.5:** SysML Diagram Taxonomy [26]

The modeling process in SysML involves two main aspects: structural modeling and behavioral modeling.

Structural modeling, this aspect depicts the "what" of the system, identifying and defining system components, their assets, behaviors, and associations. The block definition diagram, one of the four structural diagrams in SysML, is commonly used for illustrating structural modeling concepts. It comprises blocks representing system entities and relationships linking them. Blocks should be named using nouns or noun phrases, and relationships should form meaningful sentences when associated with blocks.

Behavioral modeling, this aspect illustrates the "how" of the system, focusing on system behavior at different levels and within elements and operations. State machine diagrams, one of the four behavioral diagrams in SysML, are frequently used for behavioral modeling. These diagrams describe the behavior of system entities over time, with states representing different conditions, transitions indicating possible state changes, and events governing transition occurrences. State machine diagrams are particularly useful for modeling systems with stateful behavior, such as Blu-ray

players, where states like playing or fast-forwarding correspond to different operational behaviors.

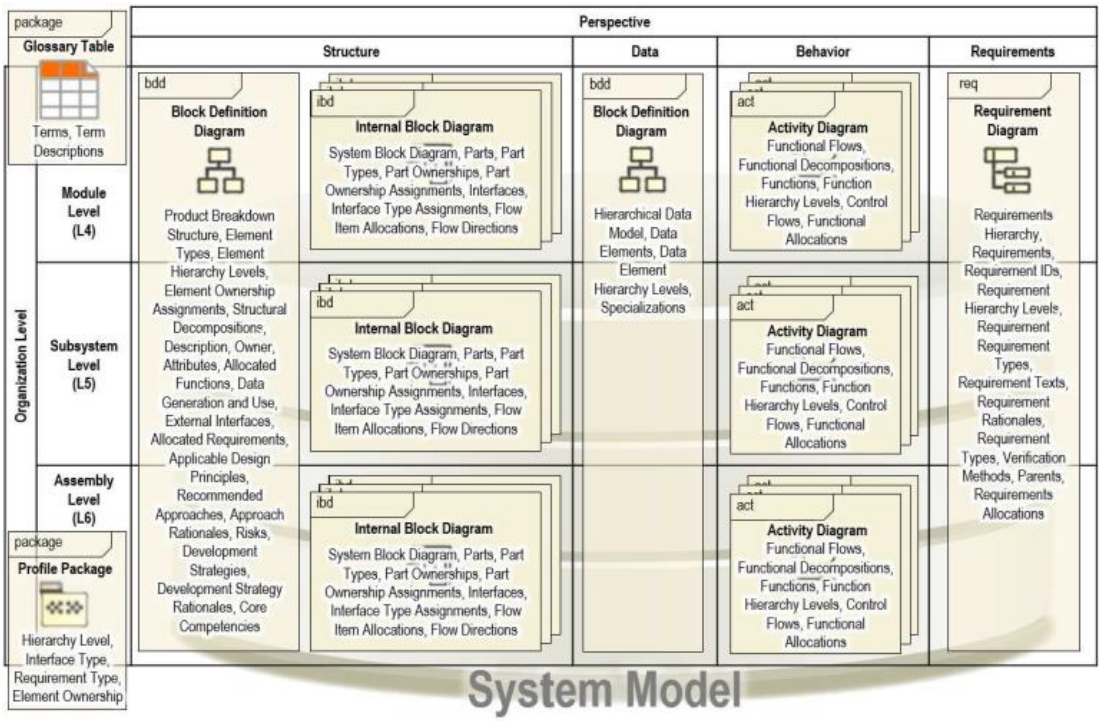



**Figure 5.6:** Information model for the sysml based MBSE approach showing individual diagram types, the information content captured each type of diagram, and the types of diagrams generated in each architecture organization level and perspective [29].

Every SysML diagram has a distinct function in communicating various facets of the behavior and architecture of a system:

**Block Definition Diagram (BDD):** Shows the relationships between elements that are frequently used to represent system and categorization trees, such as blocks and value kinds.

**Internal Block Diagram (IBD):** outlines a single block's internal structure and the connections between its internal components and connections.

**Use Case Diagram:** Represents the use cases a system performs, and the actors involved, offering a black-box view of system services.

**Activity Diagram:** Specifies behavior focusing on control flow and input-output transformations through actions, often used for analysis and expressing desired system behavior.

**Sequence Diagram:** Specifies behavior emphasizing interactions between block parts via operation calls and asynchronous signals, serving as a detailed design tool and specifying test cases.

**State Machine Diagram:** Specifies behavior highlighting block states and possible transitions in response to events, providing a precise specification for the development stage.

**Parametric Diagram:** Expresses constraints bound to system properties, assisting with trade research and engineering analysis of physical structures.

**Package Diagram:** Displays the organization of a model in a package containment hierarchy, showing interconnections among model parts included in packages.

**Requirements Diagram:** Shows the links between text-based requirements and other model elements, such as confinement and derivation [26].

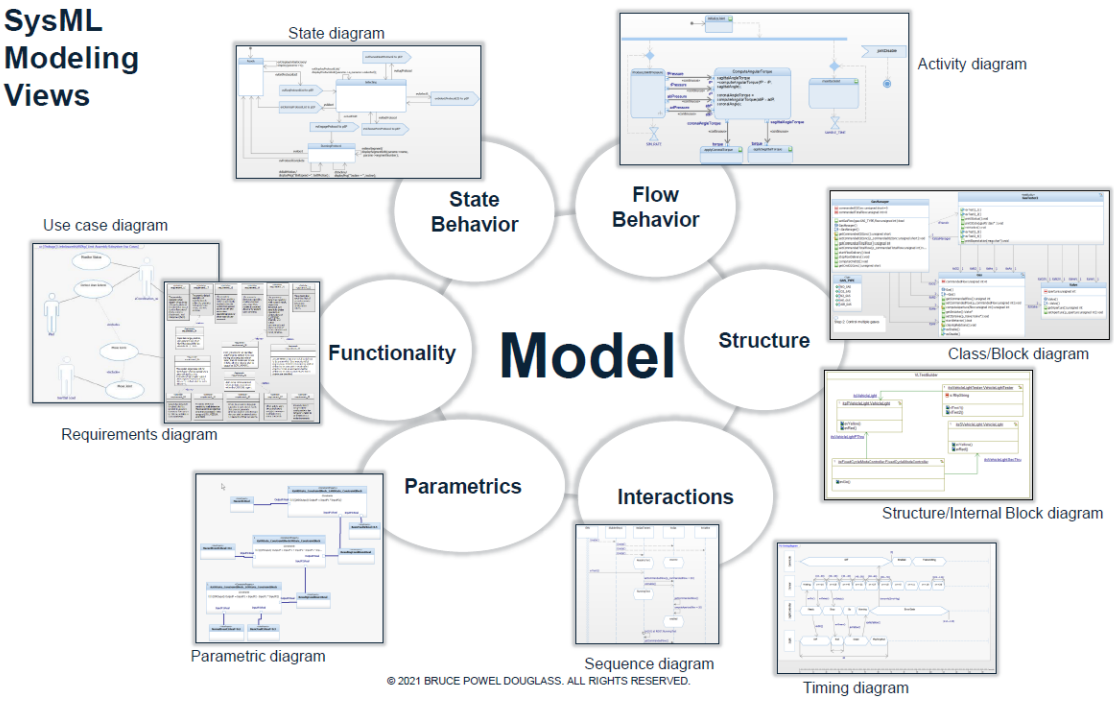



**Figure 5.7:** SysML Modeling Views [30]

### 5.3.2. Modeling Methods

A method is a collection of connected actions, procedures, conventions, representations, and results used to carry out one or more processes, usually accompanied by a set of tools. In the context of model-based systems engineering (MBSE), a method is a documented set of design tasks that typically encompass all or part of the systems engineering process and result in a system model as one of its primary artifacts [25].

Acquiring proficiency in a modeling language marks just the initial step in the MBSE journey. While a modeling language defines a set of rules that distinguish between well-formed and ill-formed models, it does not specify when and how to use the language to create a model; it does not prescribe a specific modeling approach. In contrast, a specified sequence of design tasks known as a modeling technique acts as a roadmap, ensuring team members create the system model consistently and work toward the same end goal. In the absence of such direction, each team member's contributions to the system model will differ significantly in terms of depth, scope, and accuracy.

According to the definition provided by the Object Management Group (OMG), a method is defined as "a collection of related activities, techniques, conventions, representations, and products that implement one or more processes and are generally supported by a suite of tools." This description is consistent with what is discussed above. Furthermore, the OMG concurs with the description of an MBSE methodology provided by Estefan et al.[31] as "the collection of related processes, methods, and tools used to support the discipline of systems engineering in a'model-based' or'model-driven' context." In light of this, an MBSE approach is deemed successful if it "implements all or part of the systems engineering process and produces a system model as one of its primary artifacts."

Employing an appropriate methodology in MBSE is akin to following a recipe in cooking essential for achieving desired outcomes. However, the methodological aspect remains a significant challenge in MBSE, with many applications lacking a formalized process or utilizing custom processes. Consequently, the establishment of standardized methods emerges as a key objective within MBSE working groups [24].

Every MBSE project commences with a plan, rooted in its purpose. The initial step involves addressing fundamental questions: Why is modeling undertaken? Specifically, what outcomes are anticipated? Is the model intended solely as the authoritative record of design decisions, or will it generate text artifacts for review and approval? Will it facilitate requirements traceability and impact analysis? Is it meant for conducting trade studies or integrating with simulation tools? Will it serve as input for downstream development teams or contain integration and acceptance test cases? These answers delineate the project's modeling purpose. Subsequently, a new set of inquiries arises: What aspects of the external environment and system merit modeling? How deeply should internal structures and behaviors be decomposed? Determining these factors defines the scope of the system model. The scope establishes the project's goal and signifies model completeness, aligning with the outlined purpose. Various modeling methods exist, each tailored to specific needs and objectives. While this book focuses on SysML proficiency, it briefly introduces renowned modeling methods to guide readers further.

These modeling methods cover various stages of the systems engineering life cycle. Not all steps outlined may be applicable to the project. It's essential to customize any chosen method to align the project’s specific requirements. These methods serve as valuable starting points.

- INCOSE Object-Oriented Systems Engineering Method (OOSEM)[24]
- Weilkiens System Modeling (SYSMOD)[32]
- Method: IBM Telelogic Harmony-SE[33]
- Method: Magic Grid[34]

The Magic Grid framework, akin to a Zachman style matrix, serves as a structured guide for engineers during the modeling process. It addresses critical questions such as "How should the model be organized?", "What is the modeling workflow?", "What model artifacts should be produced at each step of this workflow?", and "How are these artifacts connected to each other?”

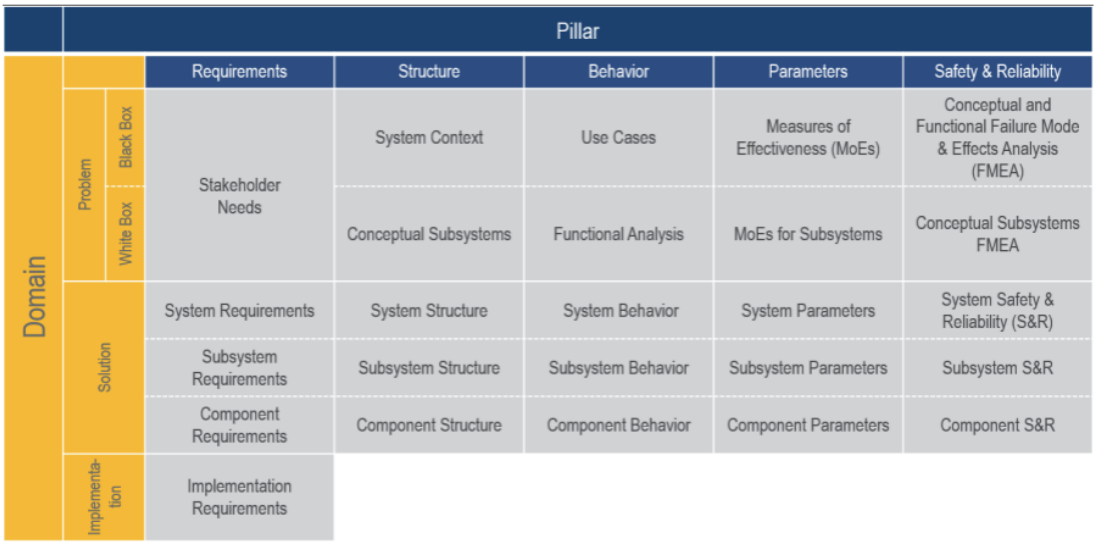



**Figure 5.8:** MagicGrid Method

The framework delineates the Problem, Solution, and Implementation domains for the development of the system of interest (SoI). Each domain is represented as a distinct row in the MagicGrid framework. The Problem domain row bifurcates to signify the necessity of defining the SoI from both black-box and white-box perspectives. The Solution domain row contains multiple inner rows to denote the various levels of detail at which the solution architecture can be specified. Conversely, the Implementation domain row remains unsplit and is less prominently highlighted, indicating that aspects beyond the implementation requirements specification fall outside the purview of Model Based Systems Engineering (MBSE) and are thus not integral to the matrix.

Each domain definition encompasses different aspects of the SoI, aligning with the four pillars of SysML: Requirements, Structure, Behavior, and Parameters. A cell at the intersection of a row and column represents a view of the system model, potentially comprising one or more presentation artifacts such as diagrams, matrices, maps, or tables. While not explicitly depicted in the framework, it is possible to provide more than one solution architecture for a single problem. In such instances, a trade-off analysis is conducted to select the optimal solution for SoI implementation.

MagicGrid specifically addresses the technical ISO/IEC/IEEE 15288 processes. To initiate the Business or Mission Analysis process, it is considered input to the Problem domain of MagicGrid and is typically delineated using other technologies, such as the OMG Unified Architecture Framework (UAF), prior to employing the MagicGrid framework for system development. The Problem domain encompasses the entire

Stakeholder Needs and Requirements Definition process, mapping subprocesses accordingly: Stakeholder Needs Definition subprocess to the Stakeholder Needs cell, Operational Concept and Other Life Cycle Concepts Development subprocess to the System Context, Use Cases, and Measures of Effectiveness (MoEs) cells collectively, and Transformation of Stakeholder Needs into Stakeholder Requirements subprocess to the Conceptual Subsystems, Functional Analysis, and MoEs for Subsystems cells collectively. The System Requirements Definition process is encapsulated within the System Requirements cell, with potential extension to the Subsystem Requirements and Component Requirements cells. Following the completion of system requirements, the Architecture Definition process ensues, encompassing all system-level and subsystem-level cells of the Solution domain, excluding the System Requirements and Subsystem Requirements cells. The Architecture Definition process also partially covers the component-level cells, which are also addressed by the Design Definition process, thereby delineating the convergence of architecture and design, or the intersection of MBSE and Model Based Design (MBD) and encompassing activities of the Assess architecture candidates subprocess. Once these activities are concluded, the Design Definition process commences.

Ultimately, the Implementation process translates requirements, architecture, and design into actionable steps that facilitate the creation of system elements in accordance with the practices of the chosen implementation technology, leveraging relevant technical specialties or disciplines. This process is encapsulated within the Implementation domain of MagicGrid.

The table below presents a summary of the methodologies utilized in Model Based Systems Engineering (MBSE):

| *OOSEM* | *Process* |
|---|---|
| The Object-Oriented Systems Engineering Method (OOSEM) is a top-down, scenario driven approach that leverages object-oriented concepts and other modeling techniques to help architect systems.<br>**Language:** SysML<br>**Views:**<br>• **Structural:** BDD and IBD diagrams<br>• **Behavioral:** activity diagrams<br>• **Other:** requirement diagrams | 1. Analyze stakeholder needs<br>2. Define system requirements<br>3. Define logical architecture<br>4. Optimize and evaluate alternatives<br>5. Synthesize physical architecture<br>6. Integrate and verify the system<br>7. Manage requirements traceability |

| *SysMod* | *Process* |
|---|---|
| SysMod is a service-oriented approach where engineers first identify the system services. It is a pragmatic approach to model a system from analysis to design.<br>**Language:** SysML<br>**Views:**<br>• **Structural:** BDD and IBD diagrams<br>• **Behavioral:** sequence and state machine diagrams<br>• **Other:** use cases diagrams | 1. Determine requirements<br>2. Model the system context<br>3. Model use cases<br>4. Model domain knowledge<br>5. Create glossary<br>6. Realize use cases. |

| *Harmony* | *Process* |
|---|---|
| Service request driven approach including the case of passing off information to software engineering. An Agile version of Harmony developed by Bruce Powell Douglass is referenced in Chapter 12.<br>**Language:** SysML and UML<br>**Views:**<br>• **Structural:** BDD and IBD diagrams<br>• **Behavioral:** Sequence and state machine diagrams<br>• **Other:** Use cases diagrams, activity diagrams | 1. Requirements capture<br>2. Definition of the system use cases<br>3. System functional analysis<br>4. System architectural design<br>5. Subsystem architectural design |

| *OPM* | *Process* |
|---|---|
| The Object Process Methodology by Dov Dori [12] OPM can be used to formally specify the function, structure, and behavior of artificial and natural systems in a large variety of domains.<br>**Language:** OPM<br>**Views:**<br>• **Structural:** Object-Process Diagram<br>• **Behavioral:** Object-Process Diagram | 1. Identify the system function (process) – top-level OPD<br>2. Identifying the system's beneficiaries—added as object element to the OPD<br>3. Identify the relation between the process and the object as a transformation of the object or the state of the object<br>4. Proceed iteratively to refine the model by modeling the subprocesses for the process and the constituent objects for the object |

**Figure 5.9:** Summary of Some of MBSE Methodologies[15]

## 5.3.3. Modeling Tool

The third pillar of MBSE is proficiency with a modeling tool. Modeling tools are a specialized class of tools designed and implemented to comply with the rules of one or more modeling languages. These tools enable the construction of well-formed models in those languages. Unlike diagramming tools like Visio, SmartDraw, or others, where diagrams are created as shapes on a page, modeling tools allow the

creation of a model as a set of elements and relationships between them. When an element on a diagram within a modeling tool is modified, the underlying element in the model is actually being modified. The tool then instantaneously updates all other diagrams displaying that same element. This capability is unique to modeling tools. A modeling language standard like SysML is vendor neutral, it should be mentioned. One vendor's application of that language definition is a particular modeling tool. Numerous non-profit consortiums and commercial tool suppliers have produced modeling tools for different modeling languages. The price, functionality, and adherence to linguistic standards of these programs differ. The organization's MBSE adoption process should include the selection of the optimal tool based on the particular requirements of the project and the available budget. Some SysML modeling tools are listed. The following are commercial-grade modeling tools:

- Agilian (by Visual Paradigm)
- Artisan Studio (by Atego)
- Enterprise Architect (by Sparx Systems)
- Cameo Systems Modeler (by No Magic)
- Rhapsody (by IBM Rational)
- UModel (by Altova)

The following are free modeling tools, offered under an Eclipse Public License (EPL) or General Public License (GPL):

- Modelio (by Modeliosoft)
- Papyrus (by Atos Origin)

Tools from IBM and NoMagic have a long history of using SysML to achieve the whole MBSE strategy. These sophisticated tools uphold the OMG SysML standard. They can, however, provide strong functionality for integration with requirement management systems (like DOORS) or model simulation and validation because of this very reality and their support for relevant standards. Plugins for modeling toolkits and architecture frameworks (DoDAF2, MODAF), for instance, are available for NoMagic's MagicDraw. A new tool for systems engineering called Cameo Systems Modeler is derived from MagicDraw and has been carefully chosen and enhanced [26].

# 6. IMPLEMENTATION

For the modeling language to be useful, it needs to be paired with a methodology. Modeling Solution is a productive infrastructure for implementing model-driven development that consists of modeling language(s), modeling tool, and methodology. In this study, SysML is used as the language, MagicGrid is used as the methodology and Cameo is used as the tool for a turbojet engine project.

Partial MBSE applications were carried out for the TJ90 turbojet engine project, which was selected as the pilot project. From an MVP (Minimum Viable Product) perspective, MBSE applications were first implemented for the parts of a project that were most difficult.

The steps applied in this section are explained respectively according to the MagicGrid method [34].

## 6.1. Problem Domain

The goal of the problem domain specification is to assess stakeholder needs and improve them using SysML model elements to get a clear and consistent definition of what functions the SoI should perform in the established environment.

As seen in the figure below, problem area analysis is carried out in two stages. In the first stage, SOI is considered a black box. The main focus is on how it interacts with the defined environment without gaining any knowledge about its internal structure and behavior (i.e. operational analysis of the SoI is performed). In the second stage, the black box is opened and the SoI is analyzed from the white box perspective; this helps to understand the expected behavior and conceptual structure of the SoI (i.e., functional analysis of the SoI is performed). It is also important to note that the problem definition is not related to the logical or physical architecture of the SOI. As it can be seen in the figure below, both phases of problem domain definition consist of determining the requirements, structure, behavior and parameters of the SoI. The only difference is in perspective.

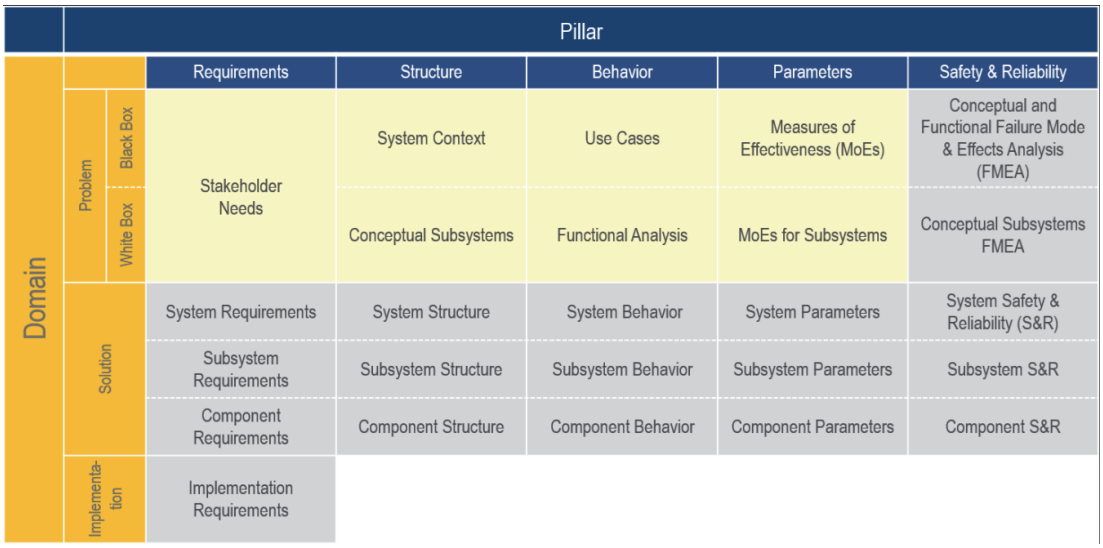



**Figure 6.1:** MagicGrid Framework – Problem Domain [34]

The model which is established for TJ90 project and its main titles are shown:

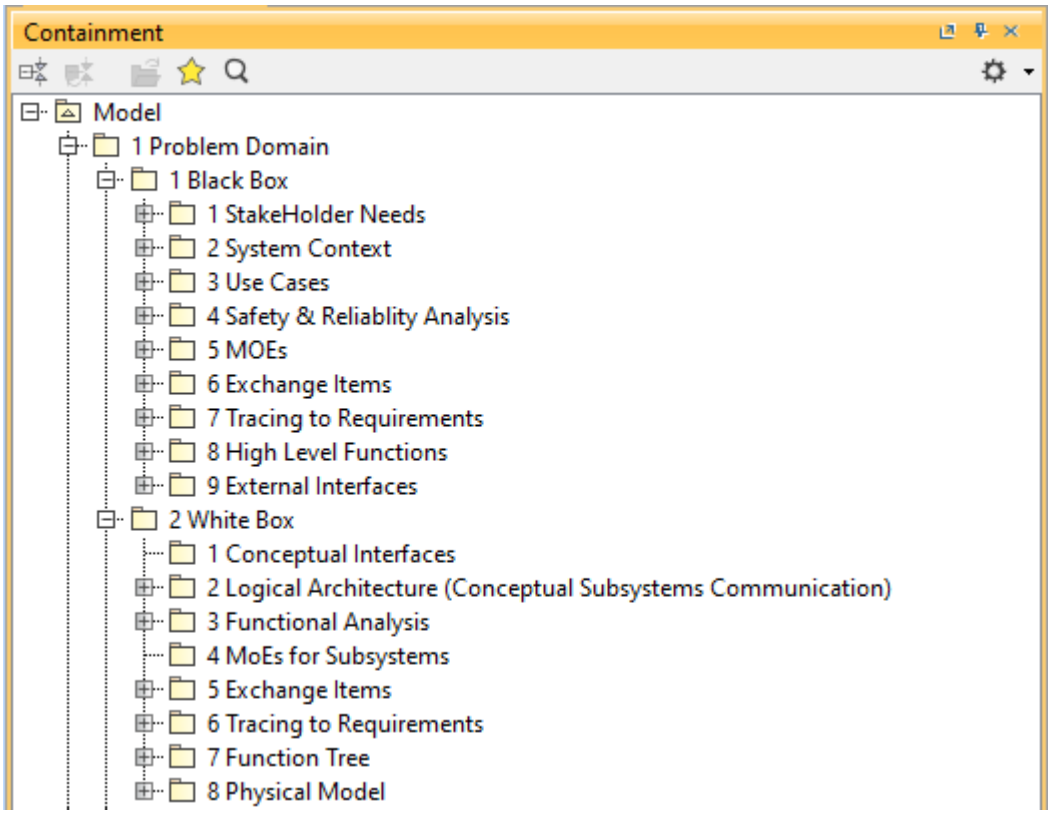



**Figure 6.2:** TJ90 main titles in model

Sequentially, “Stakeholder Needs” contains captured needs and requirements, “System Context” contains definitions of contexts in IBD and BDD’s, “Use Cases” contain high level functions of the SOI as in use case and activity diagrams for the

defied contexts, “Safety & Reliability Analysis” coInterfaces” containodes analysis for the captured and derived functions, “MOEs” contain the measures of effectiveness of the SOI with all the value types and units, “Exchange Items” contain all the interface details discovered during the modeling activities, “Tracing to Requirements” and “High Level Functions” and “External Interfaces” contain reporting tables and diagrams stating the summary of the system’s definitions.

Similarly, in the White Box folder tree the “Conceptual Interfaces” contain external interfaces that are flowed down to subsystems, “Logical Architecture” contain the initial subsystems definition, “Functional Analysis” contains break down of SOI’s external behaviours to the initial subsystems, “MoE’s for Subsystems” contain allocation of the system’s expected performance parameters to the subsystems, “Exchange Items” contain initial interactions between subsystems, “Tracing to Requirements” and “Function Tree” and “Physical Model” contain reporting tables and diagrams stating the summary of the subsystems definitions.

### 6.1.1. Black-Box Perspective

The initial stage in determining what the SoI will accomplish to meet the expectations of the stakeholders is to define the problem domain and its definition from a black-box perspective. The purpose of this phase is to understand how the SoI interacts with its surroundings in order to address existing challenges. The examination does not look into the underlying structure or operations of the SoI because it is considered a "black box." The analysis is performed to determine the key inputs and outputs, black-box functions, and measurable properties of the SoI under various system circumstances.

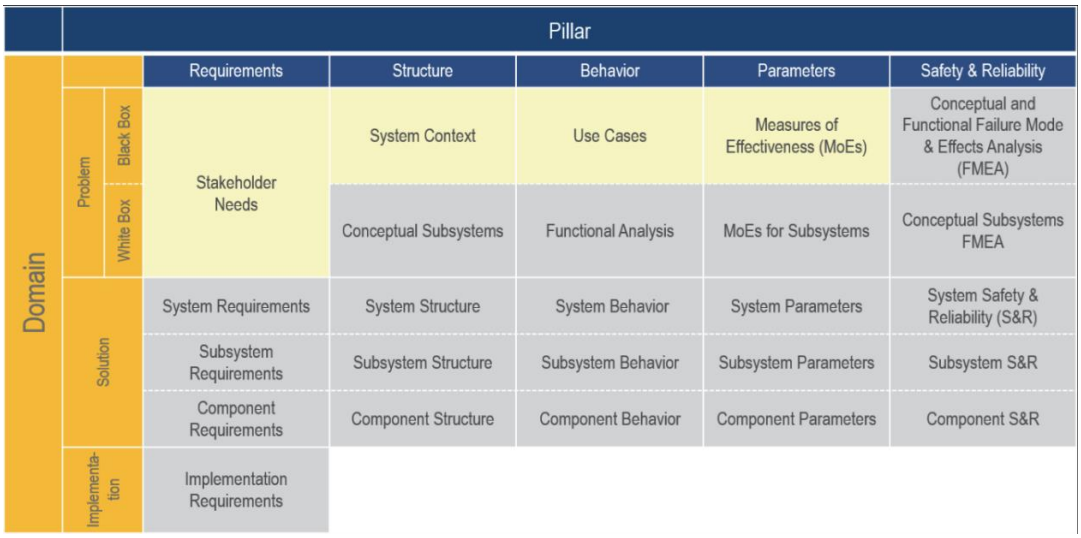



**Figure 6.3:** MagicGrid Framework – Black Box [34]

### 6.1.1.1. Stakeholder Needs

The cell within the problem domain model functions as a repository for diverse information sourced from stakeholders of the System of Interest (SoI). This encompassing dataset comprises primary user requirements, governmental regulations pertinent to the system, industry standards, policies, procedures, internal guidelines, and other relevant inputs. Although this information remains in its raw form, it necessitates meticulous analysis and refinement across other cells within the problem domain model. Subsequently, traceability relationships are established to delineate the specific artifacts within the problem domain model that refine each stakeholder need. These refined requirements form the foundational elements for articulating system requirements.

Within the modeling tool, individual stakeholder needs are encapsulated as SysML requirements, each possessing a unique identification number, name, and textual specification. A comprehensive compilation of stakeholder needs is showcased within a SysML requirement table or diagram, facilitating hierarchical organization based on various criteria, such as source. Stakeholder needs are typically categorized into two main types: functional and non-functional, streamlining subsequent analysis and evaluation processes.

| # | Name | Text |
|---|---|---|
| 1 | SN1 Thrust/SFC | |
| 2 | SN1.3 Thrust-3 | The engine shall provide a minimum thrust of N at an airspeed of Mach , at an altitude of meters. |
| 3 | SN1.1 Thrust-1 | With the engine at sea level at rest (SLS) ISA 0, shall provide a minimum thrust of N. |
| 4 | SN1.2 Thrust-2 | The engine shall provide a minimum thrust of N at an airspeed of Mach , at an altitude of 0 meters. |
| 5 | SN1.4 Thrust-4 | The engine shall provide a maximum thrust of N at idle speed at sea level standstill (SLS). |
| 6 | SN1.5 SFC | Engine specific fuel consumption (excluding fuel to be used for lubrication) shall be equal to less than g/(min*N) g/(kN*h)) at maximum operating RPM in SLS. |
| 7 | SN2 Air Inlet | |
| 8 | SN2.1 Air Speed Interface | The motor inlet (inlet boundary plane) airspeed shall be less than mach. |
| 9 | SN2.2 Pressure Loss Interface | Air intake pressure loss shall be less than % and equal. |
| 10 | SN3 Operating Envelopes | |
| 11 | SN3.3 Engine Operating Envelope | The engine shall be operable within the graphed altitude-mach range. |
| 12 | SN3.1 Atmospheric Conditions | The engine shall be operable within the graphed altitude-temperature |
| 13 | SN3.2 Engine Start Envelope | The engine shall be operable within the graphed altitude range. |
| 14 | SN4 Acceleration/Deceleration | |
| 15 | SN4.2 Deceleration Time-1 | The engine shall reach max RPM to idle RPM in less than seconds, per the provided chart. |
| 16 | SN4.1 Acceleration Time | The engine shall reach max RPM from idle RPM in less than seconds, per the provided chart. |
| 17 | SN4.3 Deceleration Time-2 | The engine shall reach from RPM to idle RPM in less than seconds, per the provided chart. |
| 18 | SN5 RPM | |
| 19 | SN5.2 Max Operating RPM | The engine shall not exceed the maximum operating RPM more than than RPM. |
| 20 | SN5.4 RPM Oscillation-1 | The amount of oscillation during engine operation shall be less than RPM. |
| 21 | SN5.3 Rotation Direction | The motor rotation direction shall be counterclockwise when viewed from the rear to the front. |
| 22 | SN5.1 RPM Oscillation-2 | Upon reaching MAX RPM, the engine RPM shall seize to oscillation in equal to or less than seconds. |
| 23 | SN6 Safety and Reliability | |
| 24 | SN6.1 Protection From Dust and Debi | The engine shall have means of protection from dust and debris from the incoming air. |
| 25 | SN6.2 Fireproof | The engine's fuel and lubrication systems shall be fireproof per TBD. |
| 26 | SN6.3 High Energy Debris | The engine shall contain any high energy debris at all operating speeds. |
| 27 | SN6.4 Contact by Hand | The engines mounting and designated handling locations shall be safe to contact per TBD protection equipment. |
| 28 | SN6.5 Single to Hazardous Condition | The engine shall not have any hazardous conditions from a single point failure. |

Filter is not applied. **28** rows are displayed in the table.

**Figure 6.4:** TJ90 Turbojet Engine Requirements Set

In this example functional requirements are categorized as “P” performance, “I” interface, “U” utilization. Non-functional requirements are categorized as “R” restrictions.

A well-structured model offers enhanced readability, comprehension, and manageability, thus advocating for the organization of the model into packages. Packages within a SysML model operate akin to folders in a file system: just as folders are employed to arrange files (including additional folders), packages serve to structure diagrams and elements (including other packages). As per the MagicGrid framework design, model artifacts capturing stakeholder needs are ideally organized within packages, as illustrated in the subsequent figure. Here, the top-level package signifies the domain, the intermediate-level package signifies the perspective, and the lower-level package signifies the cell.

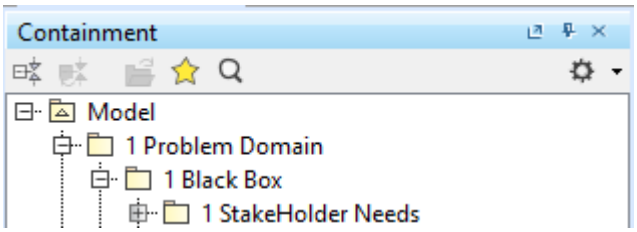



**Figure 6.5:** Stakeholder Needs in the containment tree.

In the Black Box folder tree the "Stakeholder Needs" is the location where initial staholder requirements are imported and the requirements are categorized respectively.

Multiple approaches exist for capturing stakeholder needs within the SysML model. This can be achieved directly within the Model Browser or through presentation views, such as requirement diagrams or tables, or other diagrams capable of displaying requirements, such as block definition diagrams (BDDs). In this study, the requirement table method is opted, deemed the most efficient means of integrating stakeholder needs into the model from external sources like Microsoft Excel spreadsheets or IBM Rational DOORS.

Stakeholder needs can be incorporated into the model through various means, including:

- Capturing information directly within the model.
- Copying information from a Microsoft Excel spreadsheet.
- Synchronizing information from a Microsoft Excel spreadsheet.
- Importing information from a ReqIF file.
- Synchronizing information from IBM Rational DOORS

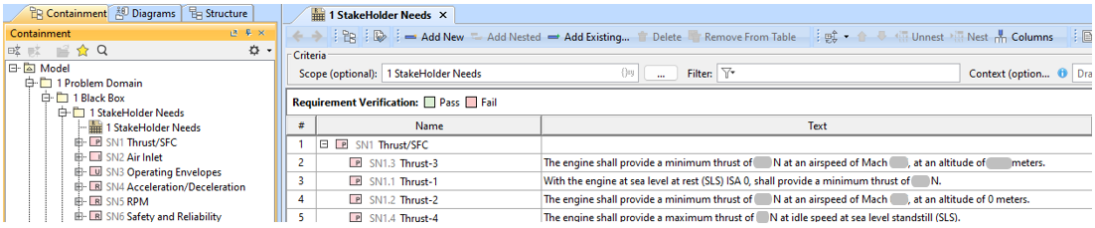



**Figure 6.6:** Stakeholder Needs

### Synchronizing information from IBM Rational DOORS

In scenarios where stakeholder needs are collected within an IBM Rational DOORS project, like TJ90 project, the integration of these needs into the SysML model can be achieved through data synchronization between DOORS and the modeling tool. In this study, we presume that DOORS serves solely for stakeholder needs capture, with subsequent analysis conducted within the SysML model. Thus, it becomes imperative to establish one-way synchronization between DOORS and the modeling tool. In this unidirectional synchronization setup, items are transferred from DOORS to the modeling tool, while no items are reciprocally transferred from the modeling tool to DOORS.

### Grouping Stakeholder Needs

Once stakeholder needs are incorporated into the model, they can be organized into groups based on their nature. These groups may encompass various categories such as system-related government regulations, user needs, industry standards, or business requirements. To facilitate grouping, additional elements of the SysML requirement type are created, each representing a distinct group of stakeholder needs, commonly referred to as grouping requirements. These grouping requirements typically lack textual content, comprising solely a unique identification number and name.

However, if stakeholder needs are synchronized from external requirements management software, such as IBM Rational DOORS, any modifications made in the SysML model will be overwritten during routine synchronization. Consequently, grouping, numbering, and categorization of stakeholder needs should be conducted within the requirements management tool.

### Numbering Stakeholder Needs

SysML requirements utilized for capturing stakeholder needs may possess hierarchical numbering. It's crucial to differentiate between the hierarchical number, which should remain unaltered, and the unique identifier. Stakeholder needs are typically numbered using natural numbers, ranging from 1 to infinity. To distinguish stakeholder needs from system or implementation requirements, it's advisable to assign them special prefixes (e.g., SN- where S denotes stakeholder and N signifies needs).

Similarly, if stakeholder needs are synchronized from external requirements management software, any alterations made within the SysML model will be

overridden during routine synchronization. Consequently, numbering, grouping, and categorization of stakeholder needs should be executed within the requirements management tool.

**Categorizing Stakeholder Needs**

Stakeholder needs can be classified as either functional or non-functional. Functional stakeholder needs delineate the expected behaviour of the System of Interest (SoI) and are typically refined through use cases and use case scenarios. Conversely, non-functional stakeholder needs identify quantifiable characteristics of the SoI and are refined through measures of effectiveness. Categorization aids in further analysis, particularly when handling extensive datasets. This step may be omitted if deemed unnecessary for the specific case.

While non-functional stakeholder needs can be directly captured as SysML requirements, functional stakeholder needs should be converted into SysML functional requirements. Function requirements, alongside interface, performance, and design requirements, inherit all characteristics of the parent requirement type, as they represent sub-types of requirements.

Stakeholder needs help us figure out the system context. When it comes to functional needs, we break it down further into use cases and use case scenarios, to analyze and improve. Non-functional quantifiable stakeholder needs are refined by measures of effectiveness.

### 6.1.1.2. System Context

The system context, also known as the operating environment, provides an external perspective of the system, delineating all entities external to the system but interacting with it. Multiple system contexts can be defined for a single System of Interest (SoI). Apart from the system itself, envisaged as a black box at present, the assemblage of elements within a particular system context may encompass external systems, whether natural or artificial, as well as users, be they human or organizational, engaged in data, matter, energy, or resource exchanges with the SoI.

This cell of the model yields:

- Definitions of system contexts

- Participants within each system context, including the SoI, anticipated system users, and other external systems
- Interactions among participants within each system context
- Items exchanged during these interactions

A system context diagram portrays the SoI centrally, devoid of internal structural details, surrounded by its interacting entities, such as anticipated users and external systems. Within the SysML model, system contexts are represented as elements of the SysML block type, with all entities also captured as SysML blocks. As per SysML conventions, upon utilization of the entity block within the system context block, a part property, typed by the former, is generated for the latter.

A SysML internal block diagram (IBD) proves instrumental in depicting entities within a specific system context and their interactions. Each entity is depicted as a part property of the associated system context block. Interactions within the system context are illustrated through connectors, facilitating one or more item flows between these part properties. These items, including data, matter, or energy, can be stored within the model as signals, blocks, or value types, contingent upon their characteristics, collectively referred to as exchange items. It is advisable to utilize blocks instead of actors (stick man notation) even for human-nature entities, such as users of the SoI, as actors cannot be incorporated within the system context within the SysML model. Furthermore, system context diagrams may incorporate diverse visual elements, enhancing presentation for stakeholders, including those not adept at interpreting models.

In accordance with the MagicGrid framework, system contexts and their associated SysML internal block diagrams, along with the corresponding elements they depict, are ideally housed within a distinct package nested within the Black Box package.

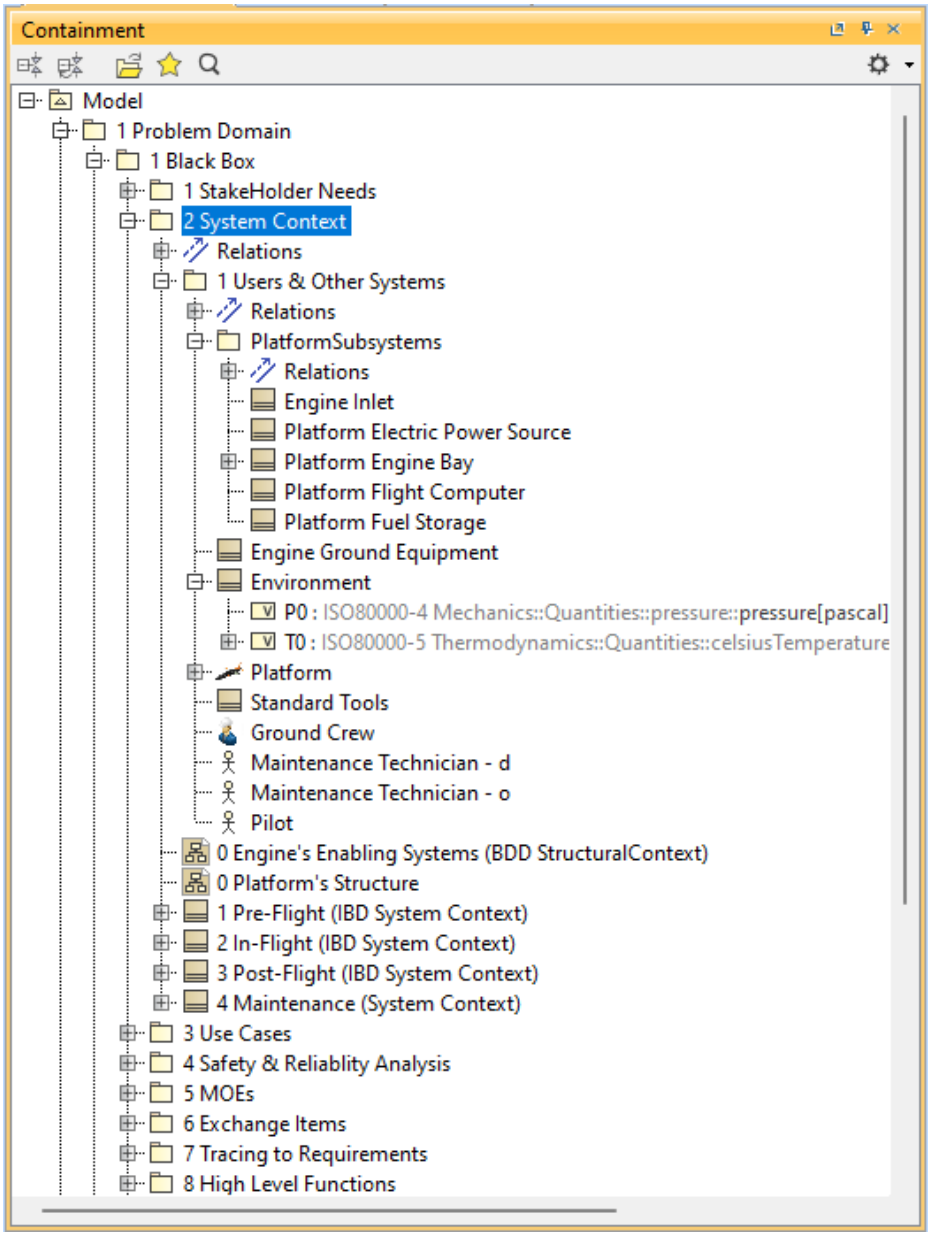



**Figure 6.7:** System Context in the containment tree

The file tree under “System Context” displays the items utilized in IBD and BDDs, which are the system’s end users and the interacted external subsystems.

A BDD of the upper system was created to represent the subsystems of the platform with which the engine interacts.

Subsystems that interface the engine with the platform have been defined. These have been created under block definition diagrams.

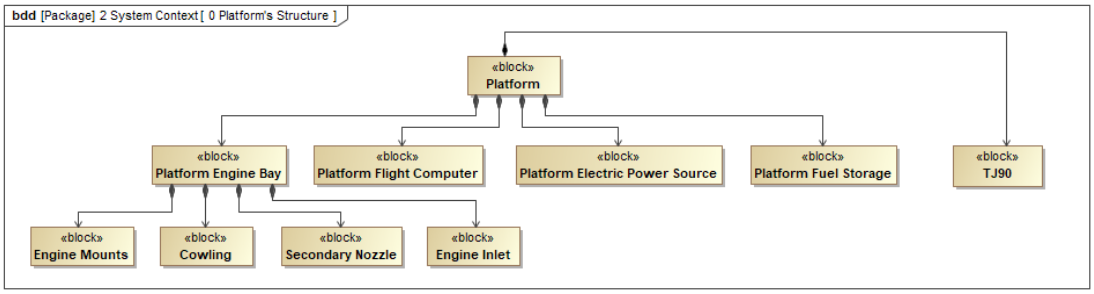



**Figure 6.8:** bdd[Package]2 System Context

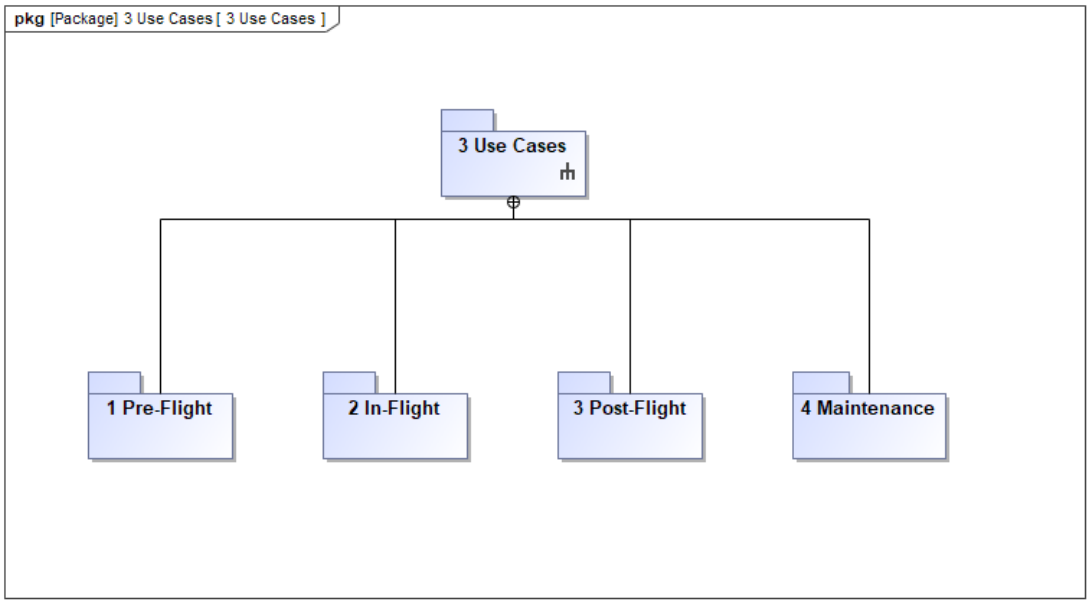



**Figure 6.9:** pkg[package] 3 Use Cases

Four different system contexts of the engine are designated as:

1 *Pre-Flight*

2 *In-Flight*

3 *Post-Flight*

4 *Maintenance*

A separate internal block diagram (IBD) has been created for each of them.

The figure below (Figure 6.10) defines the flowed interface between the end users and the platform within the Pre-Flight context, which is within the operational context.

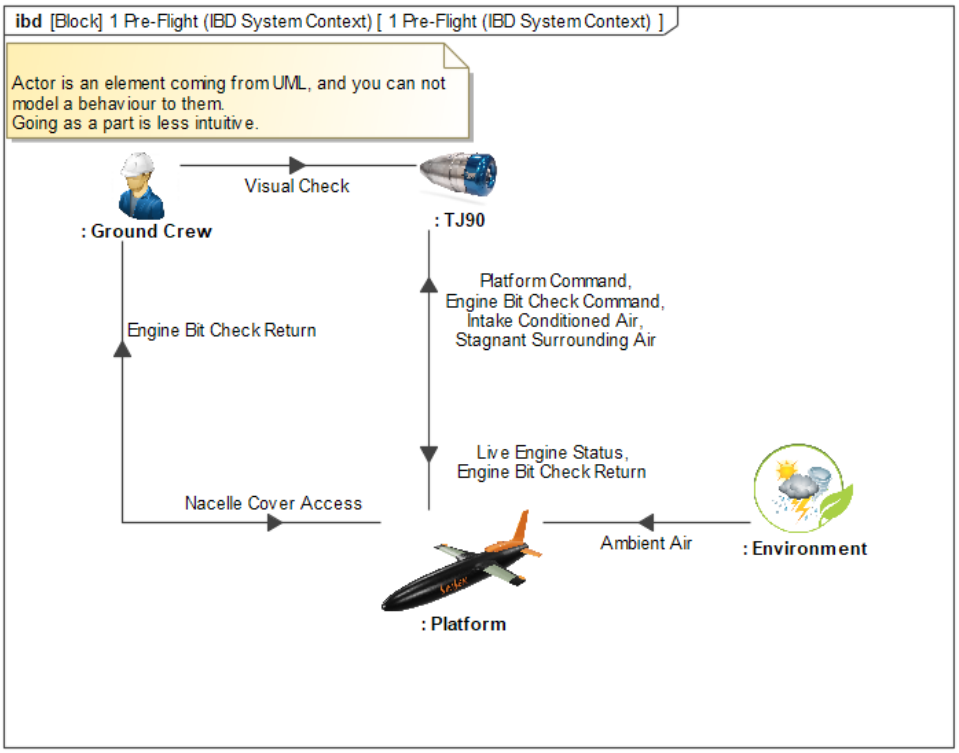



**Figure 6.10:** ibd[Block] 1 Pre-Flight (IBD System Context

Figure 6.11 defines the flowed interfaces between platform and the end user within the In-Flight context, which also shows the system in operation.

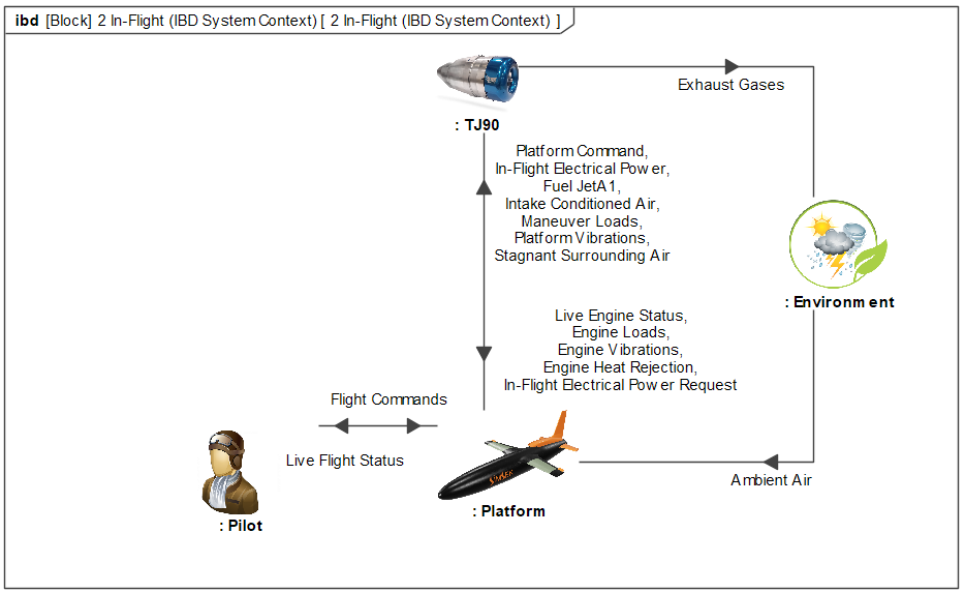



**Figure 6.11:** ibd[Block] 2 In-Flight (IBD System Context

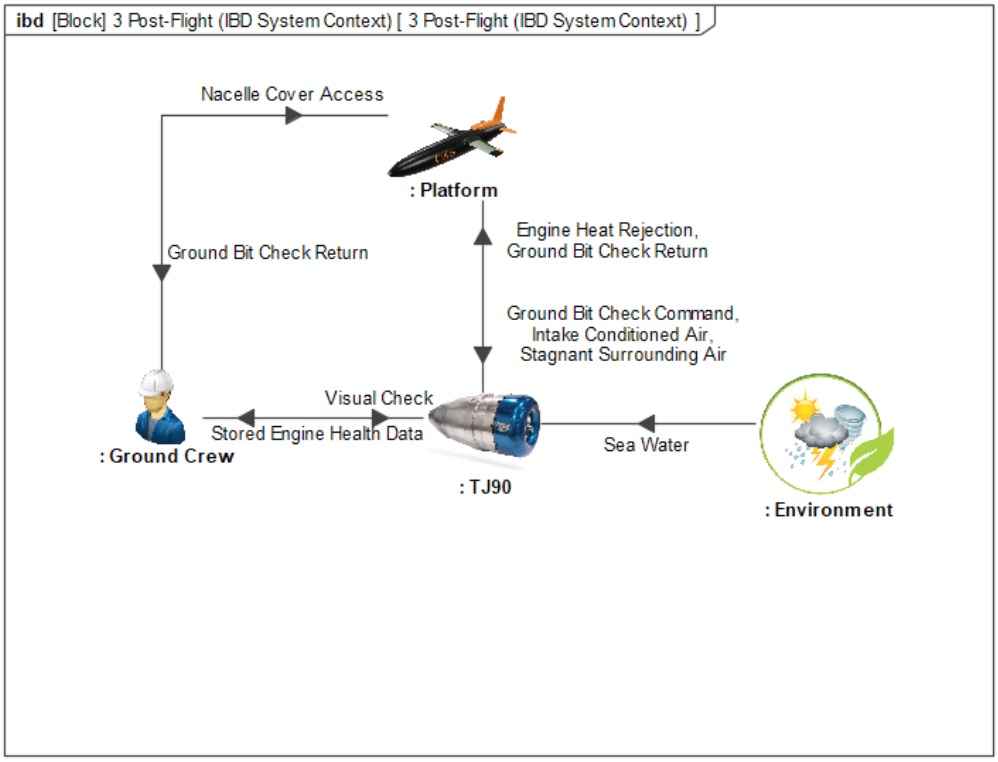



**Figure 6.12:** ibd[Block] 3 Post-Flight (IBD System Context

Figure 6.12 defines the flowed interface between the end users and the platform within the Post-Flight context, which is still within the operational context.

Figure 6.13 defines the flowed interface between the service users within the 'Maintenance Context', which is not within the operational context.

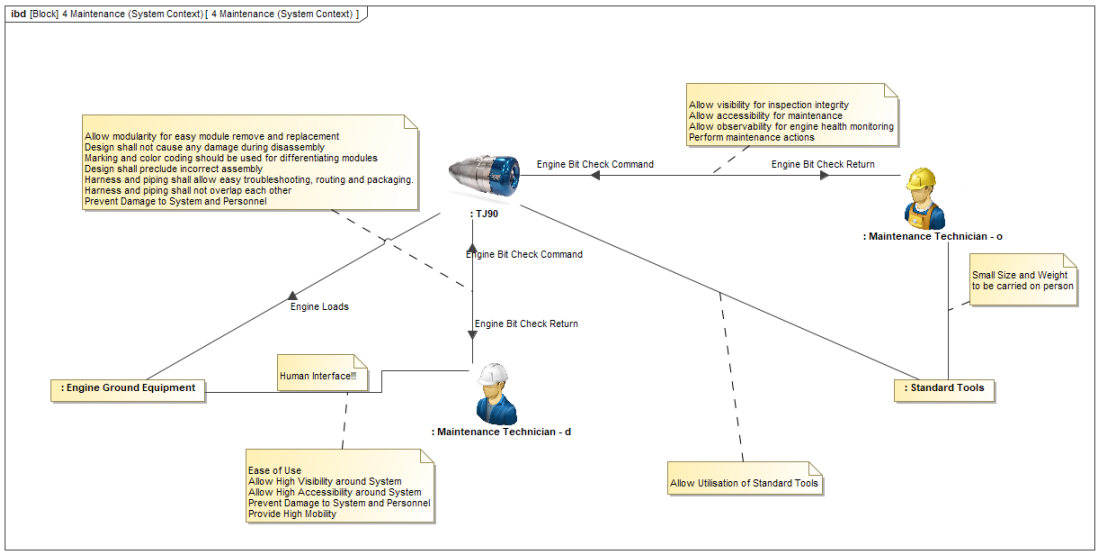



**Figure 6.13:** Maintenance Context

By determining each  operating environment;

- Basic functions specific to each operation environment have been extracted.
- Subfunctions are defined hierarchically under the basic functions;
- Requirements can be classified and broken down in a meaningful way,
- Enabled models to be created in the correct order and structure
- By modeling the interaction of the engine with the platform over time under the basic functions;
- Details of external interfaces have been determined.
- Function tree could be created.
- Functions that were not defined in customer requirements were determined.

System contexts are vital for understanding how our System of Interest (SoI) behaves, especially as discussed in the Use Cases section. We start analyzing behavior by identifying the use cases of our SoI, each linked to one or more system contexts. In these contexts, the participants are like actors in a play, helping to illustrate the scenarios. Exchange items identified earlier play a crucial role in these scenarios, moving around as needed. Once we've defined the SoI, we can dive into specifics, like how well it performs. This leads us to the Measures of Effectiveness section, where we gauge how effective and efficient our system is.

#### 6.1.1.3. Use Cases

This cell focuses on enhancing functional stakeholder needs through the utilization of use cases and use case scenarios. Unlike stakeholder needs, use cases offer a more detailed perspective on user expectations and desired outcomes from the system's usage. It's essential for each use case to align with one or multiple system contexts outlined in the System Context section.

Key outcomes of this process include:

- Development of functional use cases that deliver tangible value to users.
- Creation of use case scenarios illustrating the anticipated interactions between the system and users or other systems.

Within the modeling tool, the system's use cases can be depicted using the SysML use case diagram framework. Each diagram should correspond to one of the system

contexts outlined in the System Context section. Use cases are represented as elements of the use case type. It's important to note that a single use case may occur within different contexts.

Entities involved in performing one or more use cases within the system context are illustrated as blocks in the use case diagram. It's crucial to recognize that these block entities serve as part properties of the system context.

Additionally, as depicted in the accompanying figure, even human entities, such as vehicle occupants, can be symbolized as blocks, with or without accompanying images.

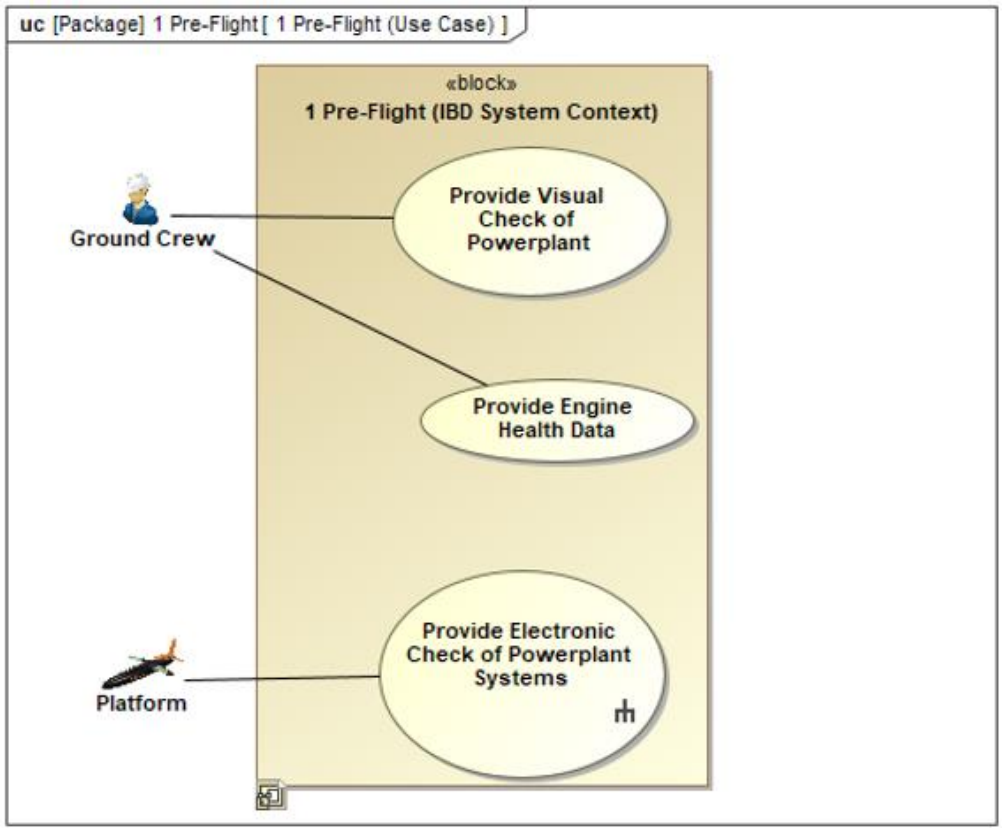



**Figure 6.14:** uc[Package] 1 Pre-Flight

Above figure (Figure 6.14) defines the high level functions that are provided to the platform and end users within the pre-flight check context.

Every use case should be assigned a name and a primary scenario, with alternative scenarios being considered optional. Use case scenarios can be depicted using either a SysML activity diagram or a sequence diagram. In the activity diagram, the System of Interest (SoI), intended users, and/or other systems can be visualized as swimlanes,

while in the sequence diagram, they are represented as lifelines. Throughout these representations, the SoI is treated as a black box, focusing solely on its external interactions.

In accordance with the MagicGrid framework, use cases and their corresponding scenarios, depicted as SysML activity or sequence diagrams along with the associated elements, are to be organized within a distinct package housed within the Black Box package. It is advised to label this package in alignment with the respective cell, denoted as "Use Cases."

To commence the documentation of use cases pertaining to the engine context, a SysML use case diagram must be created. Within this diagram, it is essential to include the block representing the system context and the platform block, symbolizing the user of the System of Interest (SoI). These elements should be visually represented on the diagram pane, ensuring a comprehensive depiction of the system context and user involvement.

Figure 6.15 shows the top-level functions and their actors called out within their respective diagrams, inside the “Use Case” folder.

To depict a use case scenario, one can utilize either a SysML activity or sequence diagram. The SysML activity diagram is preferable for delineating abstract behavior definitions, providing a broader overview of the interactions involved. Conversely, the sequence diagram is best suited for situations requiring a clear distinction between synchronous and asynchronous interactions. Careful consideration of these diagram types ensures the accurate representation of the use case scenario, aligning with the specific requirements of the system under analysis.

Swimlanes within the activity diagram serve as visual representations of the participants associated with the pertinent system context, as detailed in System Context. These lanes facilitate the establishment of allocation relationships between the structural and behavioral components within the model. Leveraging these relationships in the black-box analysis model enables the specification of each participant's role in executing the corresponding steps of the relevant use case scenario.

It's imperative to note that participants within the system context are captured as part properties thereof. Consequently, when creating swimlanes within the activity diagram of the nested use case, the modeling tool automatically identifies and presents the part

properties associated with the encompassing system context for selection. Furthermore, activating the allocation to usage mode in the activity diagram is essential. This mode ensures that allocations are delineated within the context of the system, thereby establishing a clear link between the allocated tasks and the relevant system context. Without this mode enabled, allocations remain generic and disconnected from any specific system context.

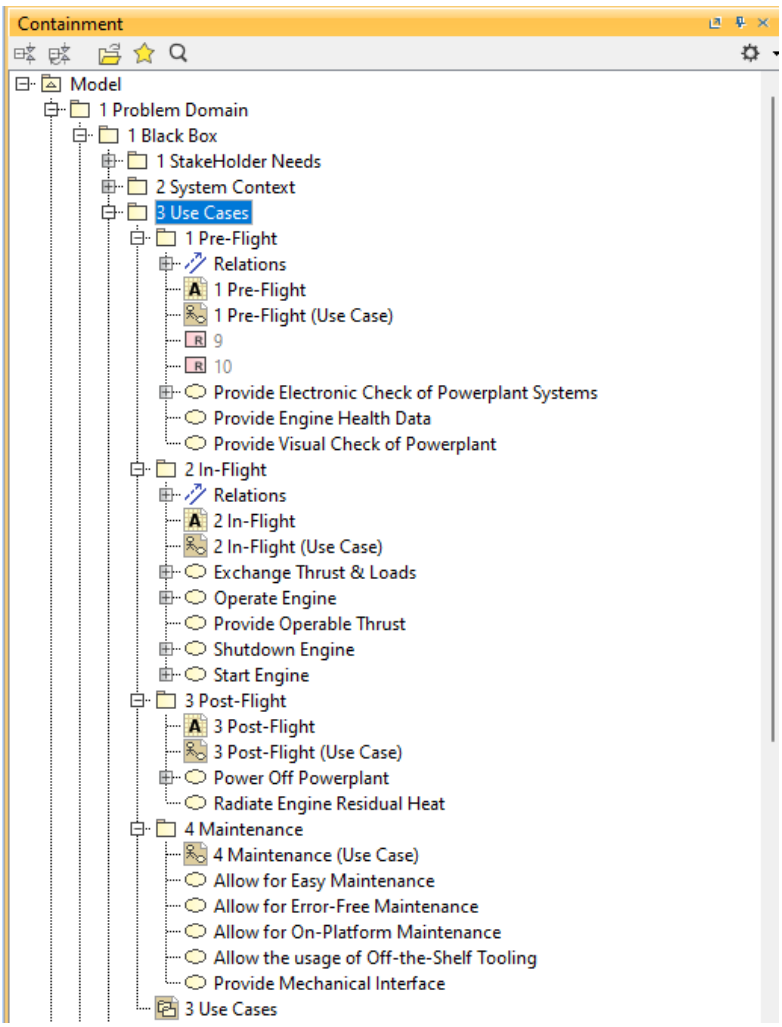



**Figure 6.15:** Use Cases in the containment tree

The processes indicated in the use case scenario can be represented in the activity diagram using call behavior actions. According to SysML specifications, a call behavior action may encompass an assigned behavior, represented in the model as either an activity, state machine, or interaction. When utilizing an activity as the behavior, it can be concurrently generated with the referencing call behavior action,

provided the Behavior Creation Mode is activated within the model. Embracing this mode is recommended for several reasons:

- Streamlined Specification of Recurrent Steps: Activities formulated once within the model, and subsequently assigned to call behavior actions, can be readily referenced by other call behavior actions across one or more activity diagrams. This approach fosters efficiency by eliminating the need for redundant step creation.
- Simplified Decomposition Process: Decomposing a step becomes more straightforward as the referenced activity is already established and designated as the behavior for the corresponding call behavior action. This eliminates the necessity to create the activity anew. It's important to note that the Activity Decomposition Map does not incorporate call behavior actions. Therefore, to ensure comprehensive decomposition of the use case steps, they should be encapsulated as activities.

***1 Pre-Flight***

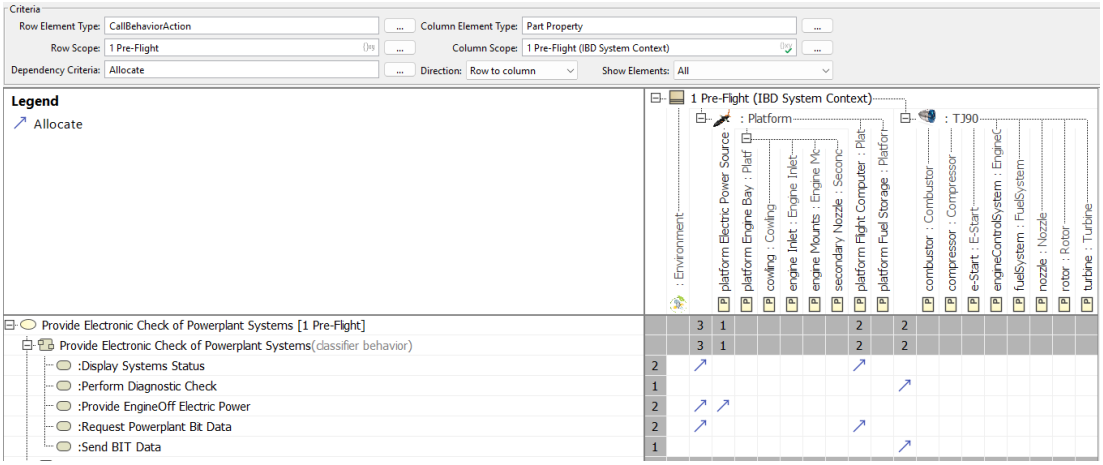



**Figure 6.16:** Allocation Table

Figure 6.16 shows the allocation between the first two levels of functions and the subsystems of the platform, within the Pre-Flight context. Functions and subsystems without any allocation means that they are either not being utilized in this context, or more analysis needs to be done later in the design process in order to define the missing allocations.

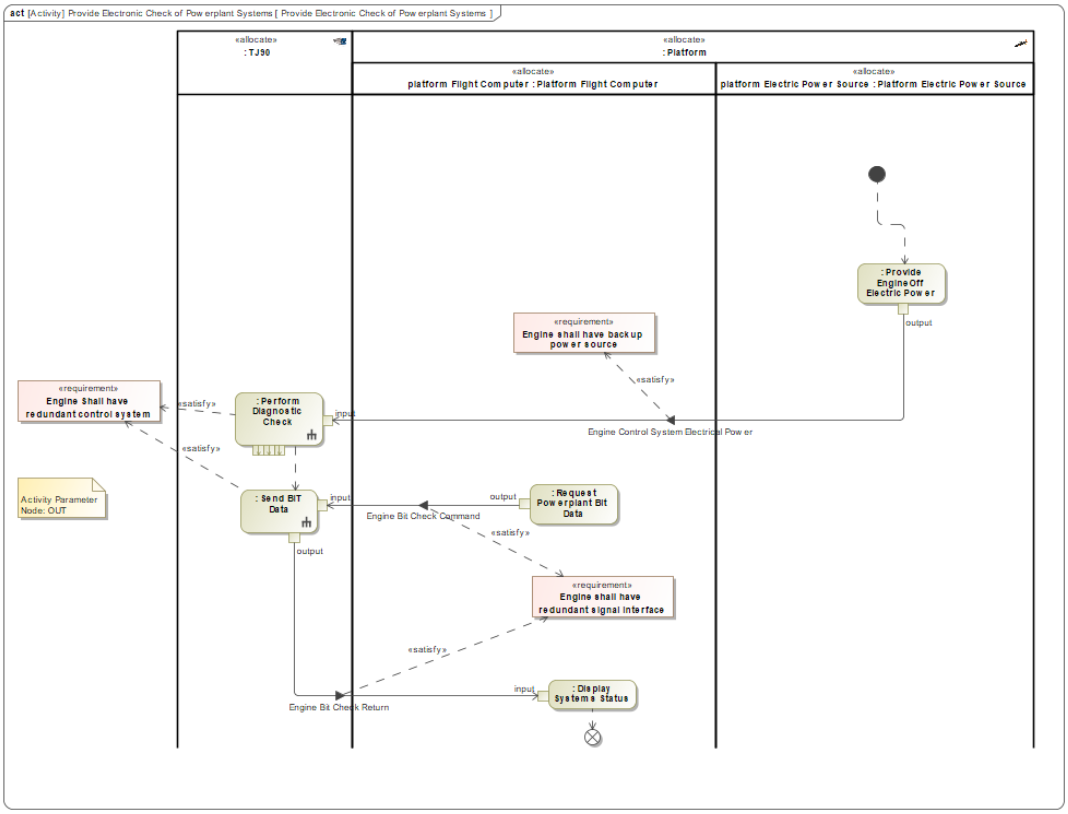



**Figure 6.17:** act[Activity] Provide Electronic Check of Powerplant Systems

Figure 6.17 activity diagram defines the second level functions under "Provide Electronics Check of Powerplant Systems".

## ***2 In-Flight***

As it can be seen in the below figure (Figure 6.18) defines the first two level of functions that are being utilized by the platform, within the In-Flight context.

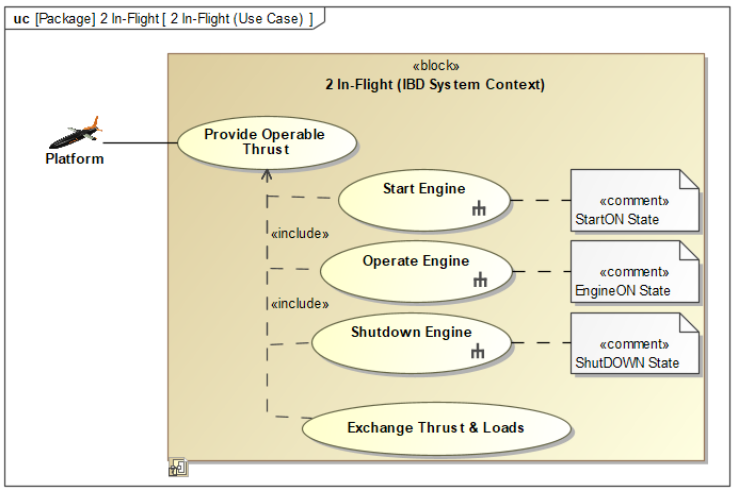



**Figure 6.18:** uc[Package] 2 In-Flight (Use Case)

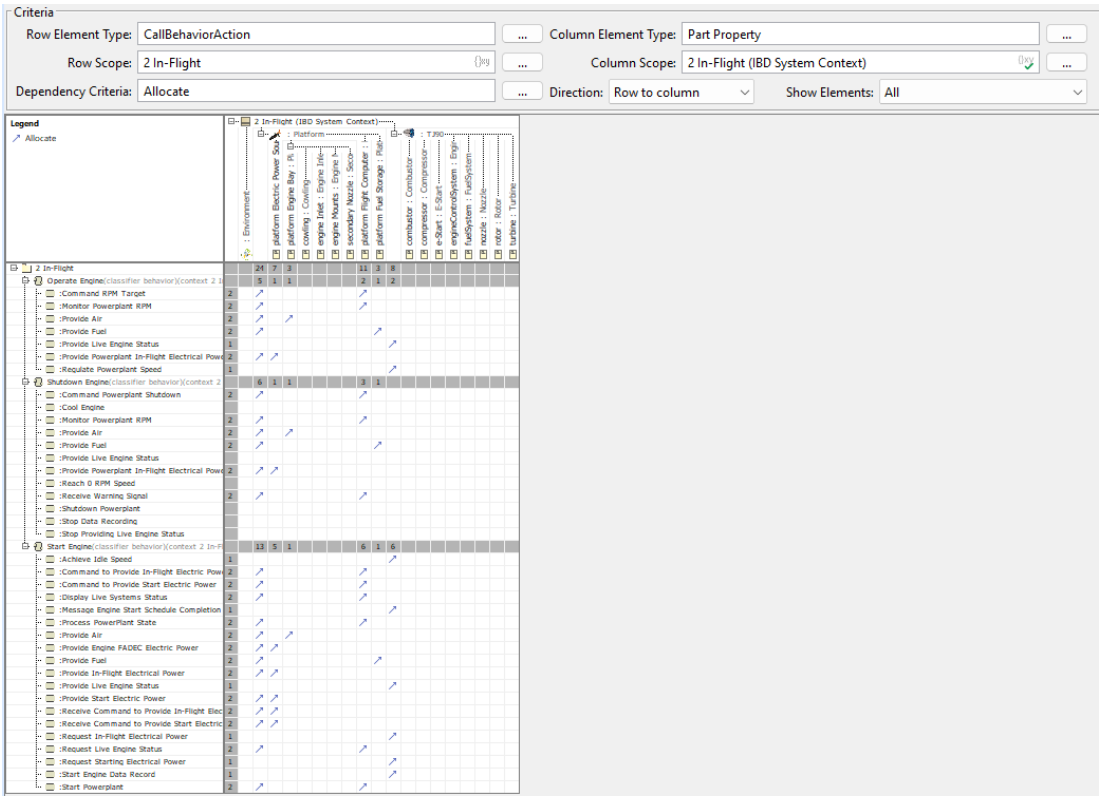



**Figure 6.19:** Allocation Table

Figure 6.19 shows the allocation of first two level of functions to the platform's subsystems.

Figure 6.20 defines the third level functions under the "Operate Engine" function.

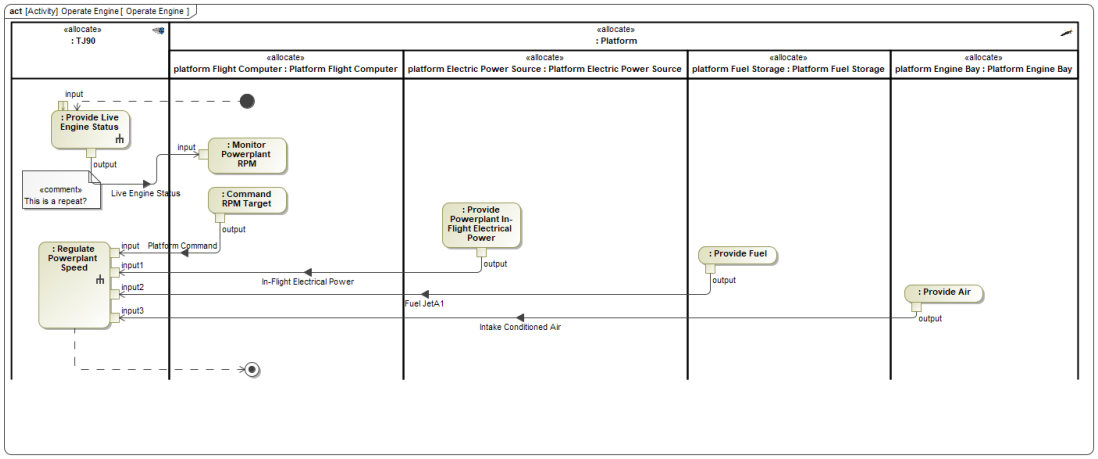



**Figure 6.20:** act[Activity] Operate Engine

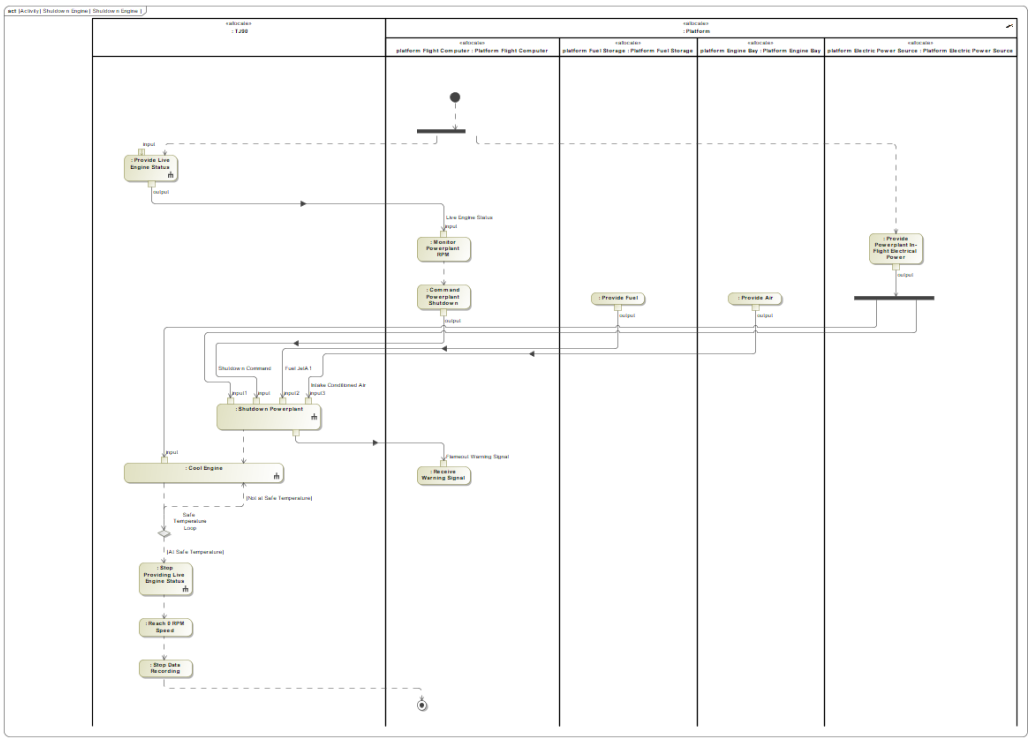


**Figure 6.21:** Stop Engine Function

The third level functions under the “Stop Engine” function is defined in Figure 6.21.

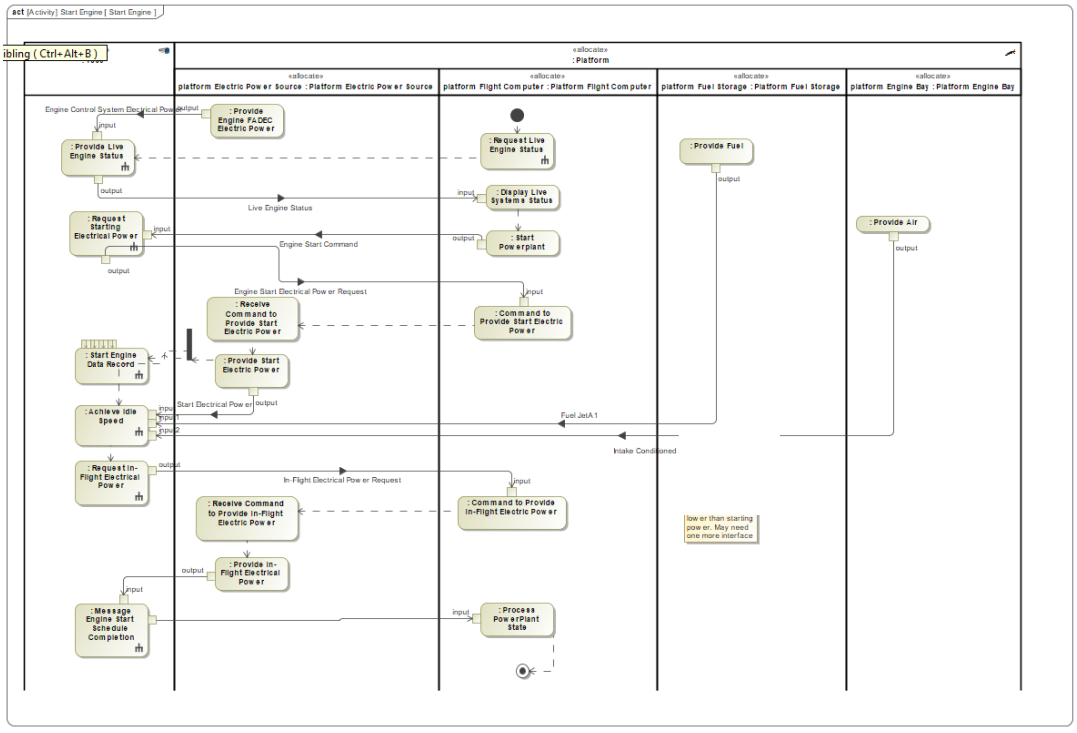


**Figure 6.22:** Start Engine Function

Figure 6.22 defines the third level functions under the “Start Engine” function.

## ***3 Post-Flight***

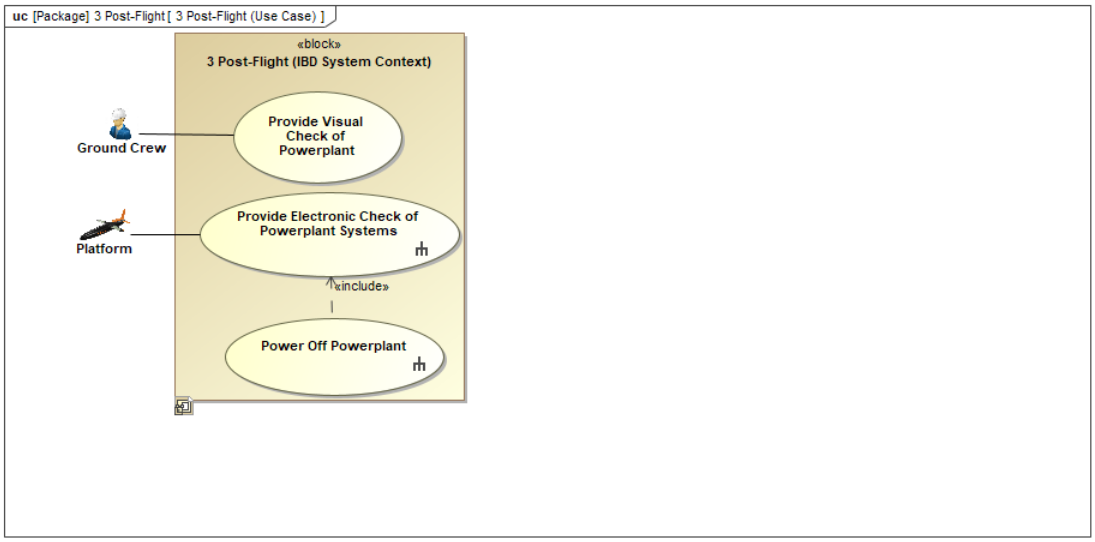



**Figure 6.23:** uc[Package]3 Post-Flight (Use Case)

Figure 6.23 defines first two level of functions, provided to the platform and end user within the Post-Flight context.

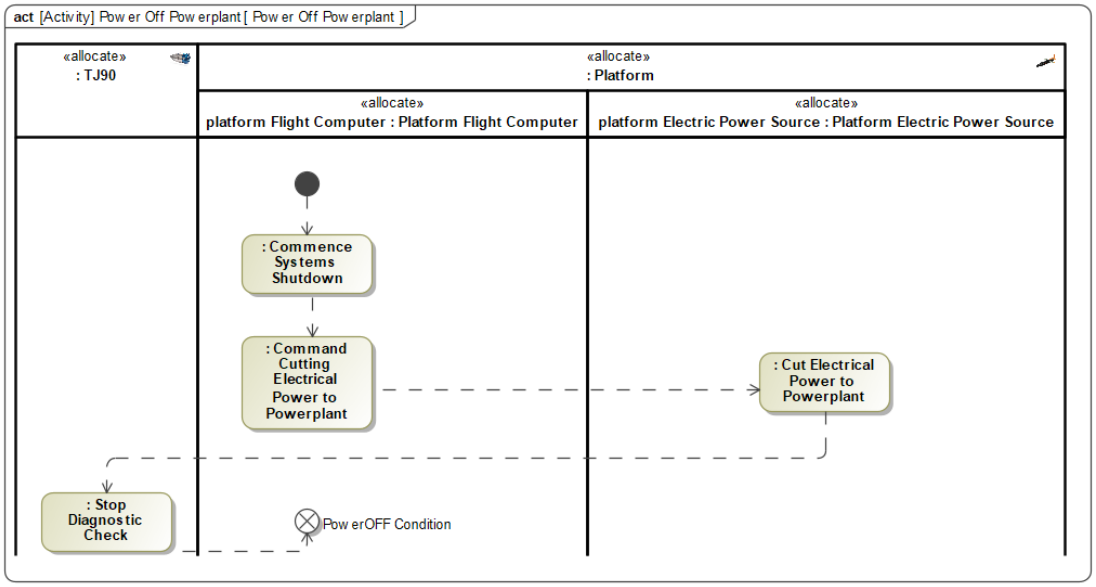



**Figure 6.24:** act[Activity]Power Off Powerplant

Figure 6.24 defines the third level functions, within the "Power Off Powerplant" function.

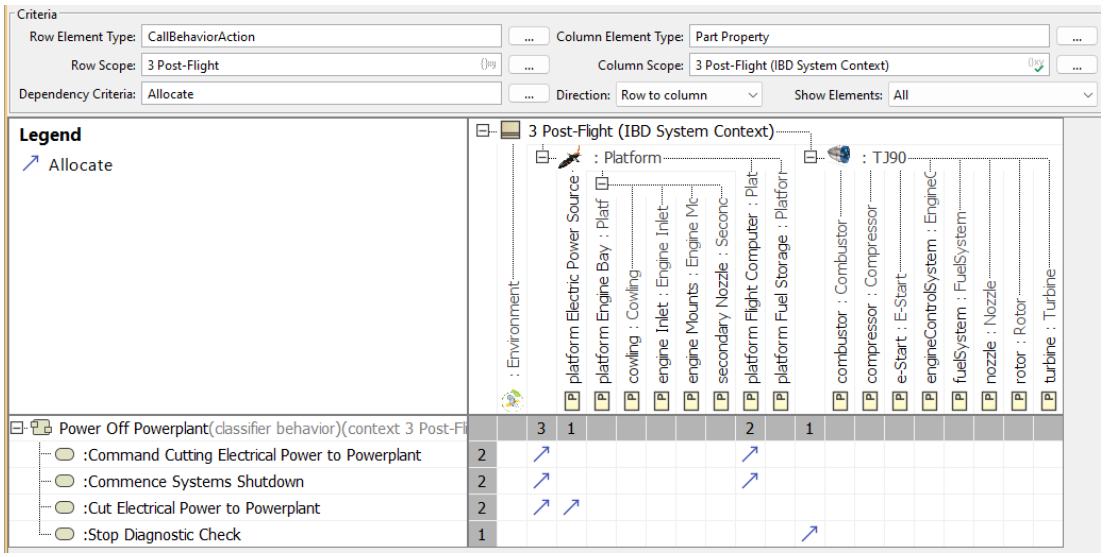



**Figure 6.25:** Allocating Table

Figure 6.25 shows the allocation between the first two levels of functions and the subsystems of the platform, within the Post-Flight context. Functions and subsystems without any allocation means that, they are either not being utilized in this context, or more analysis needs to be done later in the design process in order to define the missing allocations.

***4 Maintenance***

Figure 6.26 defines the first level functional interactions between the SOI and the service personnel, ground tools and equipment within the Maintenance context.

Once one or more use case scenarios have been developed, the subsequent phase involves functional analysis. Activities attributed to the System of Interest (SoI) within the SysML diagram typically delineated within the swimlane designated for the SoI can be decomposed further. This decomposition aids in specifying the internal behaviors expected from the SoI, facilitating a detailed examination of its functional aspects.

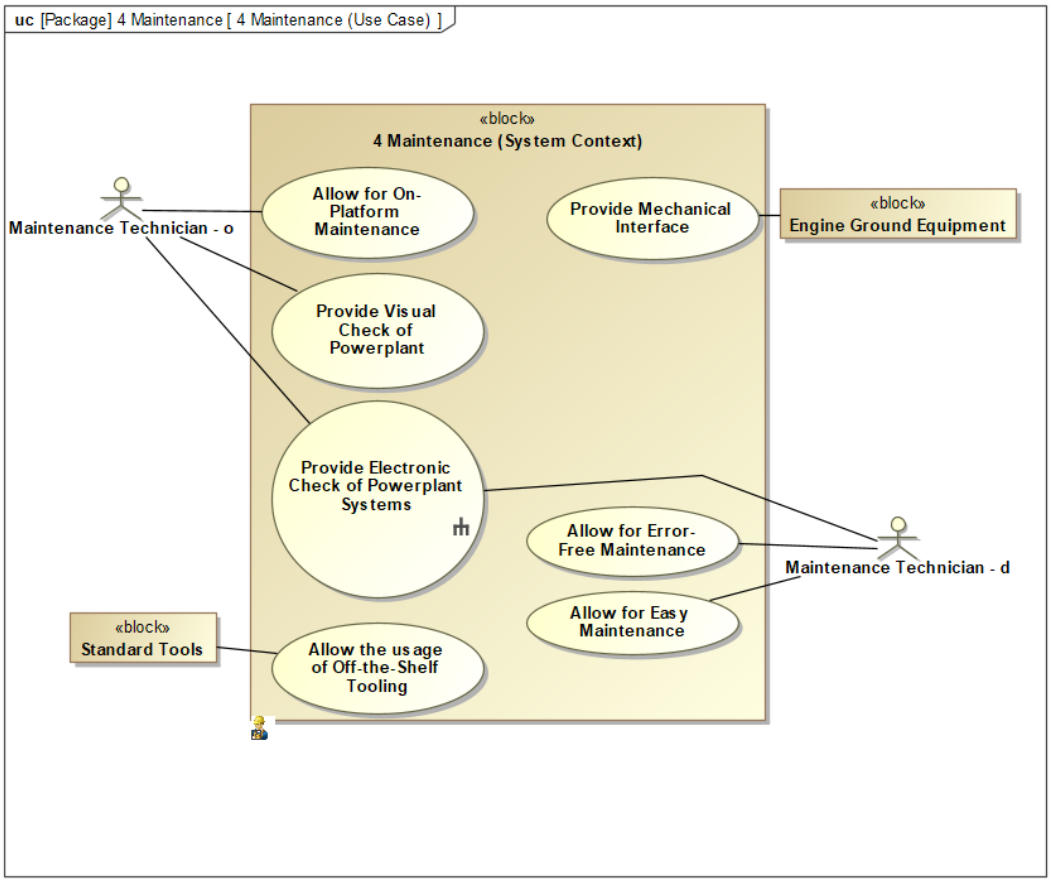



**Figure 6.26:** uc[Package]4 Maintenance (Use Case)

### 6.1.1.4. Measure of Effectiveness

Within this cell, non-functional stakeholder needs undergo refinement through the integration of measures of effectiveness (MoEs), which encapsulate quantifiable attributes of the System of Interest (SoI) in a numerical format. MoEs, a conventional term prevalent in systems engineering, delineate the efficacy of a system in executing a task within a defined context. Whitin the problem domain model, MoEs serve as overarching key performance indicators that are automatically validated inside the solution domain model. Through MoEs, design constraints can be articulated, facilitating the verification of system design. Moreover, a consistent set of MoEs can be applied across various models to articulate numerical characteristics pertinent to other Systems of Interest.

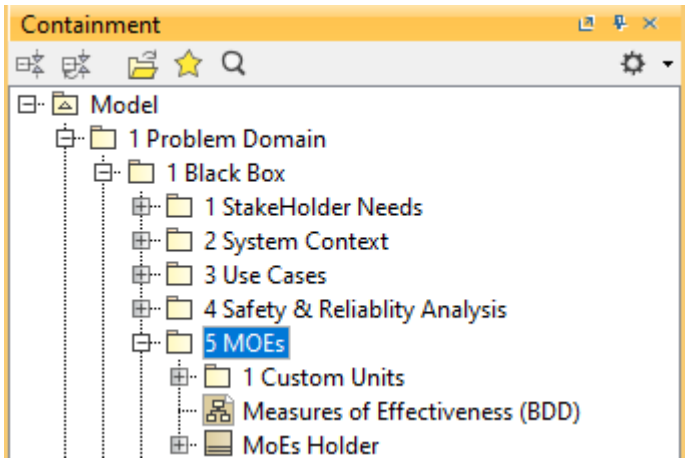



**Figure 6.27:** MOEs in the Containment Tree

To document MoEs within the modeling tool, a SysML block definition diagram (BDD) proves effective. Establishing a distinct block for MoEs allows for the creation of a reusable set, with this block designated as the superclass of the SoI representation block.


**Figure 6.28:** bdd[Package]5 MOEs[Measures of Effectiveness(BDD)]

With measures of effectiveness (MoEs) in place, design constraints can now be specified, which can be used to verify whether the system design is correct.

## 6.1.1.5. Safety and Reliability Analysis

System safety analysis can also be performed within the same modeling environment, utilizing the requirements and functions and system definitions readily available for the assessment.

In the following example, Failure Modes Effects Analysis is performed for the engine; reusing stakeholder requirements, system's block definition parts and high level functions within use cases in relation with the appropriate failure modes.

| # | Id | Name | Item | Failure Mode | Cause Of Failure | Local Effect Of Failure | Final Effect Of Failure | Refines | Mitigation |
|---|---|---|---|---|---|---|---|---|---|
| 1 | F-1 | Dust and debris ingestion | : TJ90 | Engine unable to prevent dust an... | Insufficient seperation of dust and | Seizure of rotating c<br>Decrease in engine<br>Decrease in engine | Decrease in operational<br>Loss of platform | SN6.1 Protection | 1 Engine air intake shall have |
| 2 | F-4 | Release of high energy debris | : TJ90 | Engine unable to contain high en... | Insufficient case structure strength | Rupture of case stru<br>Loss of engine struc | Damage to platform<br>Loss of platform | SN6.3 High Energ | 2 Engine external structures s |
| 3 | F-5 | Injured technician | : TJ90 | Engine unable to provide safe co... | Insufficient human interface capat | Foreign object dam | Unable to perform insta<br>Unable to perform main | SN6.4 Contact by | 3 Engine handling points shal |
| 4 | F-3 | Fire outbreak | : TJ90 | Engine unable to survive uncontr... | Insufficient fire proofness capabilit | Damage to engine c | Damage to platform<br>Loss of platform | SN6.2 Fireproof | 4 Engine subsystems in conta |
| 5 | F-6 | Failure from single effect | : TJ90 | Engine unable to survive single p... | Failure to contain loss or partial of | Damage to engine c | Loss of platform<br>Damage to platform | SN6.5 Single to H | 5 All sensors and actuators sh |
| 6 | F-17 | Unable to receive Bit Check Con | : TJ90 | Loss of Interface | Severed physical connection | Unable to initiate | Failure to fulfill mission | Provide Electroni | 7 Engine shall have redundan |
| 7 | F-18 | Unable to provide Engine BIT ch | : TJ90 | Loss of Interface | Severed physical connection | Unable to initiate | Failure to fulfill mission | Provide Electroni | 7 Engine shall have redundan |
| 8 | F-19 | Unable to Receive Engine Contr | : TJ90 | Loss of Interface | Severed physical connection | Unable to initiate | Failure to fulfill mission | Provide Electroni | 8 Engine shall have backup p |
| 9 | F-21 | Unable to Perform Diagnostic C | : TJ90 | Loss of Function | Loss of processing capability | Unable to initiate | Failure to fulfill mission | | 8 Engine Shall have redundan |
| 10 | F-22 | Unable to Send BIT Date | : TJ90 | Loss of Function | Loss of processing capability | Unable to initiate | Failure to fulfill mission | | 8 Engine Shall have redundan |

**Figure 6.29:** Failure Modes

The mitigation solutions are later implemented into the future iterations of the system model, as derived requirements and new functions.

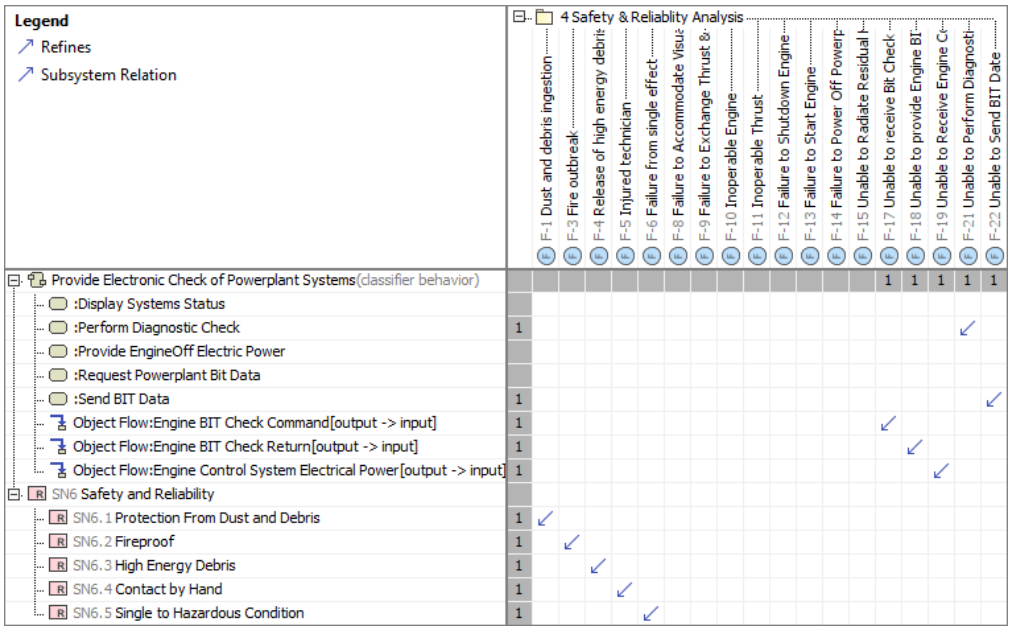



**Figure 6.30:** Allocation Table

Utilizing an allocation matrix, all discovered failure modes can be traced back to their mitigation solution as in requirements, functions or interface definitions. Any gaps seen in the table, will guide the systems engineers to develop and implement additional solutions in the architecture as the design progresses.

This process allows for the safety concerns for the system to become directly integrated with the system architecture definitions.

### 6.1.2. White-Box Perspective

Upon completion of the black-box problem definition phase, entailing the operational analysis of the System of Interest (SoI), the transition to a white-box perspective ensues. This transition facilitates a deeper comprehension of the expected behavior of the SoI, laying the groundwork for delineating its logical solution architecture. The subsequent phase of the problem domain definition necessitates a thorough functional analysis to pinpoint the conceptual subsystems, often termed as functional blocks in certain methodologies, inherent within the SoI. These conceptual subsystems serve as the initial building blocks for formulating the logical solution architecture of the SoI. The functional analysis delves into every function performed by the SoI, as identified in the Use Case cell, and decomposes them further in the Functional Analysis cell. Each function is then allocated to conceptual subsystems, indicating their respective responsibilities in executing one or more functions. This iterative functional decomposition process ensures the attainment of the requisite granularity in defining the problem domain. Additionally, the decomposition cascade extends to conceptual subsystems, breaking them down into more granular structures.

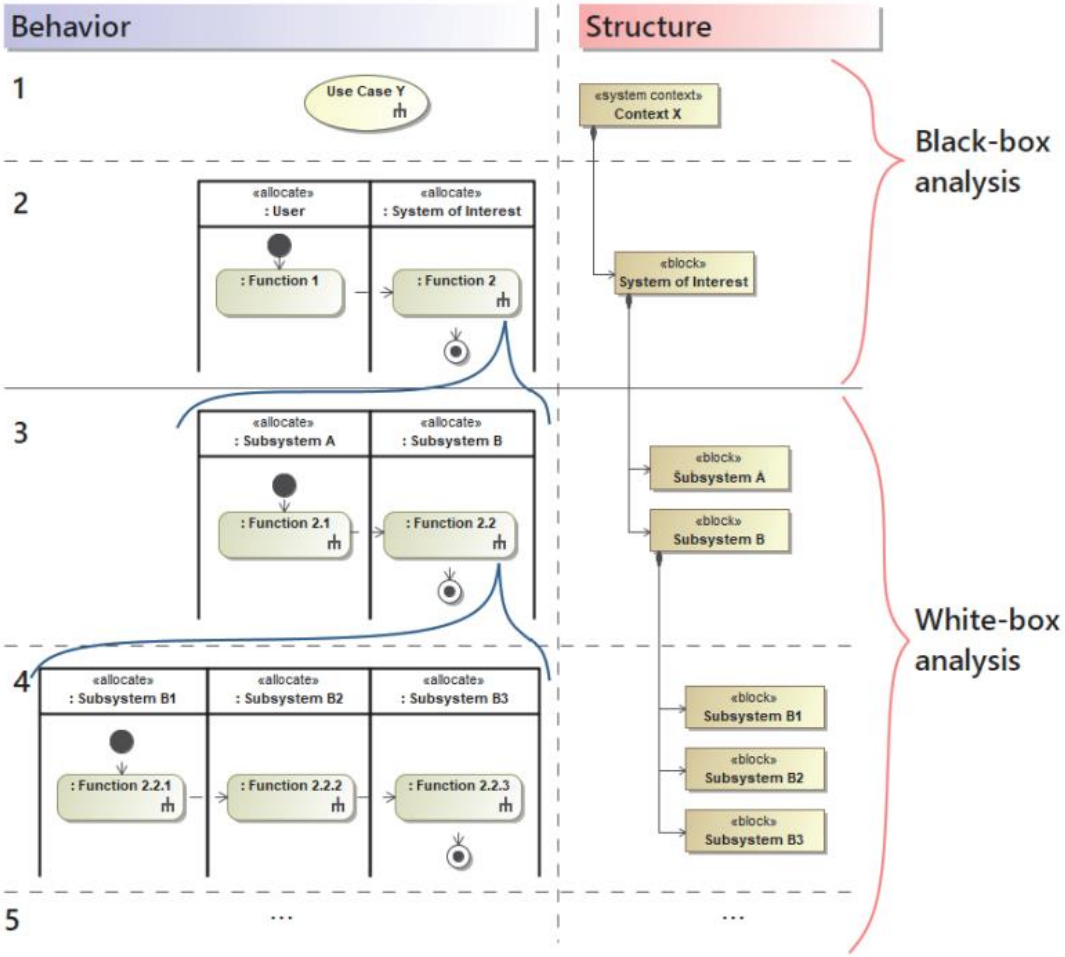



**Figure 6.31:** Functions at each layer of detail can be decomposed into even more detailed behavior [34].

Once functional subsystems have been identified, attention turns towards specifying interactions among them. Each subsystem may possess its unique quantitative characteristics, referred to as measures of effectiveness (MoEs). These MoEs, alongside the functions of the System of Interest (SoI) and its functional subsystems, collectively form the foundation for delineating system requirements. This pivotal step lays the groundwork for subsequent chapters, particularly the one addressing System Requirements.

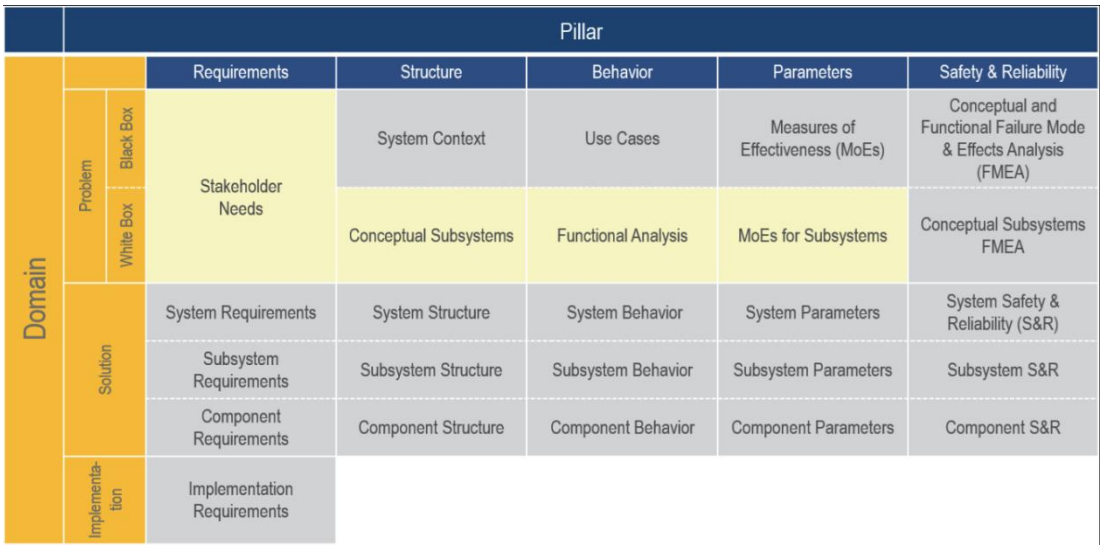



**Figure 6.32:** MagicGrid Framework – White Box [34]

### 6.1.2.1. Functional Analysis

In this stage, a more detailed analysis of the anticipated system behavior is conducted by deconstructing each function identified during the black-box analysis, as discussed in subject Use Cases. This decomposition process aims to uncover the underlying conceptual subsystems, also known as functional blocks, responsible for executing these functions. These conceptual subsystems are then elaborated upon in the subsequent white-box analysis stage, detailed in Chapter Conceptual Subsystems. It's essential to emphasize that as the analysis deepens, subfunctions of decomposed functions may themselves undergo further decomposition. This iterative process continues until the desired level of granularity is achieved to effectively address the problem at hand. Each decomposed subfunction is treated as a function in its own right, contributing to the overall understanding of system behavior. The Functional Analysis cell`s outcome is a model depicting an expected system behavior, from which an activity decomposition map can be derived, delineating the hierarchical structure of functions from the SoI's overarching goals down to its specific functions.

Functional analysis represents a natural progression from refining use case scenarios using SysML activity diagrams. For each function allocated to the System of Interest (SoI), as outlined in Use Cases, a dedicated SysML activity diagram is established. While these diagrams may feature multiple swimlane partitions, only functions nested

under the partition corresponding to the block capturing the SoI usage should be selected for inclusion.

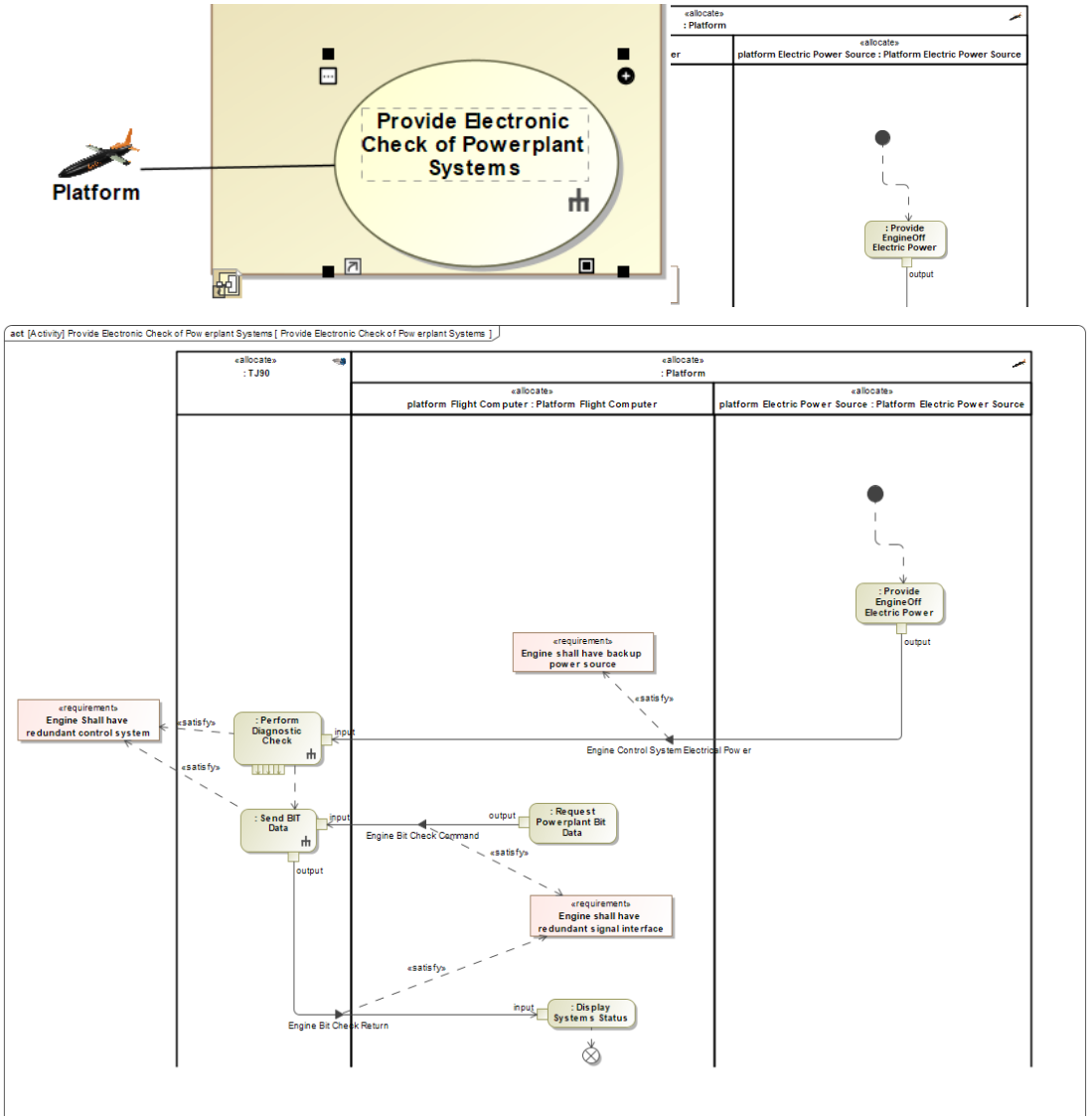



**Figure 6.33:** System functions are organized within packages.

In alignment with the MagicGrid framework's design, artifacts detailing decomposed system functions are organized within packages, as illustrated in the accompanying diagram. Here, the top-tier package signifies the domain, the intermediate package denotes the perspective, and the lower-level package signifies the cell.

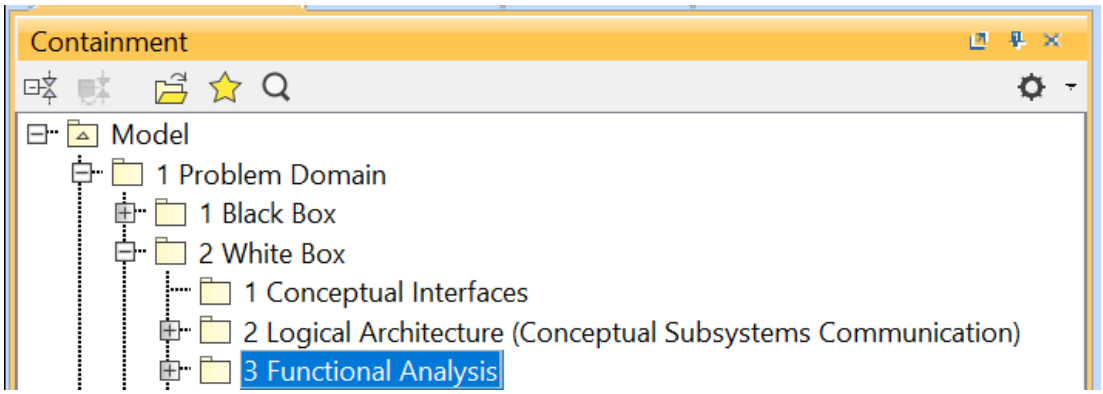



**Figure 6.34:** Functional Analysis in the Containment Tree


**Figure 6.35:** Cool Engine Function

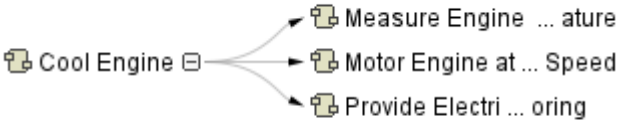



**Figure 6.36:** Cool Engine Function

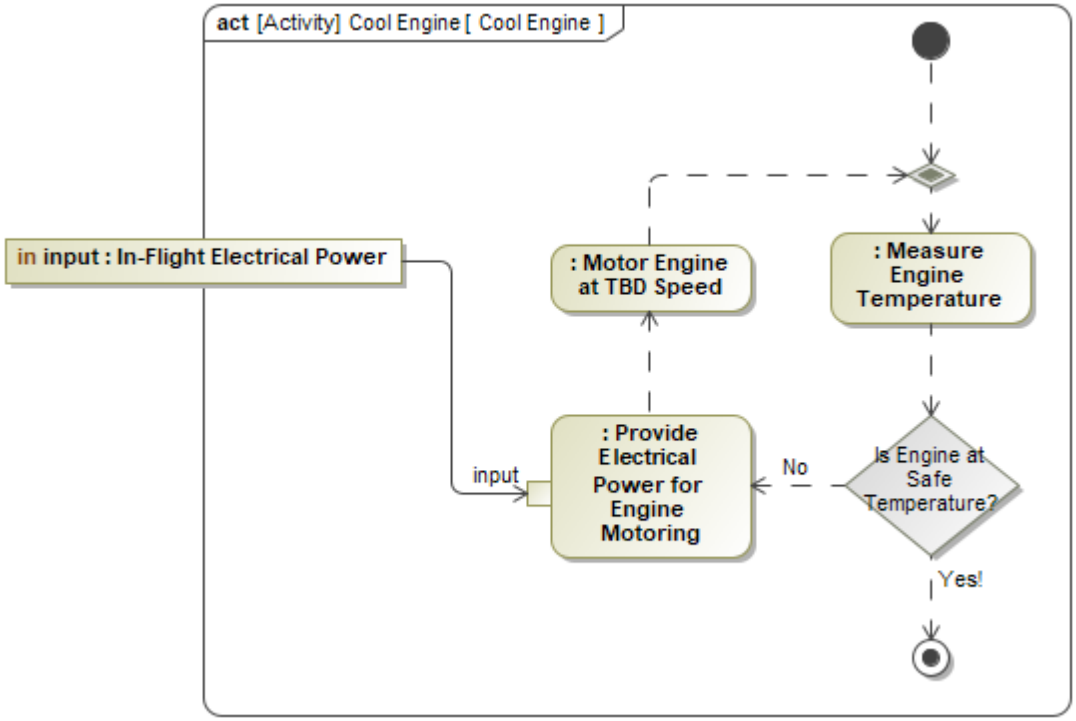



**Figure 6.37:** act[Activity]Cool Engine

The subsequent stage, Conceptual Subsystems, is prompted by white-box scenarios, facilitating the identification of conceptual subsystems within the System of Interest (SoI).

### 6.1.2.2. Conceptual Subsystems

Functional analysis aids in the identification of conceptual subsystems, which are also referred to as functional blocks. This stage serves to encapsulate these identified subsystems within the model. Conceptual subsystems are conceptualized as a cohesive ensemble of interconnected and interdependent components tasked with executing one or multiple anticipated functions of the System of Interest (SoI). It's essential to recognize that each conceptual subsystem can be further broken down into a more rudimentary arrangement. The extent of decomposition iterations is contingent upon the desired granularity level of the conceptual architecture under consideration. Every subsystem is approached as a system in its own right, viewed from the perspective of its internal constituents. Upon the integration of conceptual subsystems into the model, the subsequent step involves specifying the system functions they undertake. This entails the allocation of functions to the conceptual subsystems. It is imperative to maintain consistency in the granularity of expected system behavior and structure

across all levels of detail. Consequently, assigning the function of a subsystem to its constituent or attributing the function of a constituent to the subsystem is impermissible. A deeper analysis is only feasible once the current level of granularity has been fully addressed. The recommended alignment between the granularity levels of behavioral and structural decomposition is elucidated in Chapter White-box perspective. Furthermore, this stage accounts for the identification of inputs and outputs of the SoI. These are delineated through an examination of the system context and use case scenarios, supplemented by insights garnered from the functional analysis.

In summary, this stage yields:

- Identification of inputs and outputs of the SoI.
- Establishment of definitions pertaining to conceptual subsystems within the SoI.
- Interactions encompass:
- Relationships between conceptual subsystems and entities external to the SoI.
- Interactions among conceptual subsystems.
- Assignment of anticipated system functions to the conceptual framework of the SoI.
- Depiction of the structure decomposition map.

It should be kept in mind that it is not always necessary to specify the conceptual architecture of the SoI.

To document the inputs and outputs of the System of Interest (SoI) or any of its conceptual subsystems, the infrastructure of the block definition diagram (BDD) can be utilized. These inputs and outputs are defined as SysML flow properties. According to SysML standards, these flow properties can be grouped into interface blocks, which can be stored within the model or in an external library. Proxy ports are employed to associate interface blocks with the blocks that capture the structure of the SoI.

The accompanying figure illustrates the inputs and outputs of the the turbojet engine and some of its conceptual subsystems.

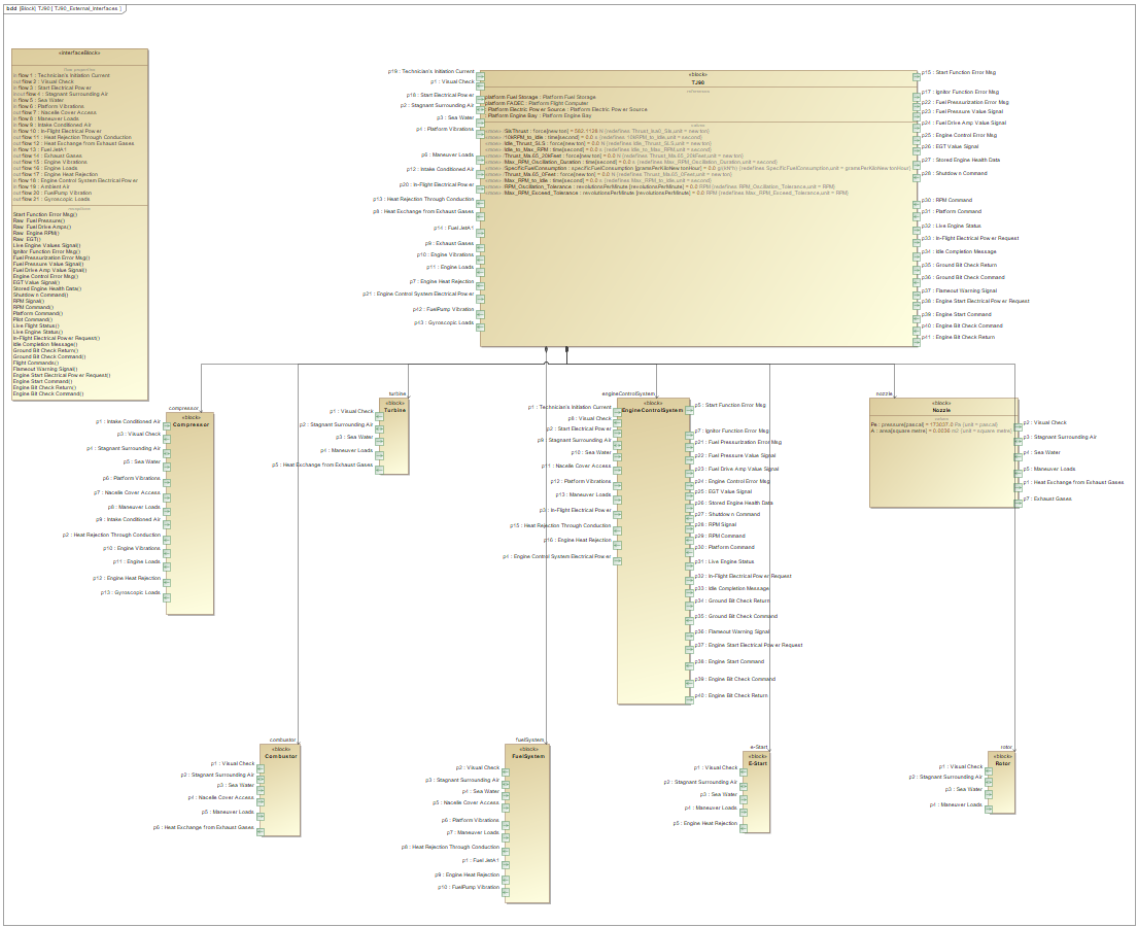


**Figure 6.38:** bdd [Block]TJ90[TJ90_External_Interfaces]

For defining the conceptual subsystems of the SoI and their interactions, the internal block diagram (IBD) infrastructure created for the block representing the SoI in the model can be used. Conceptual subsystems, identified during the functional analysis, can be specified as part properties of the block representing the SoI. Connectors with one or more item flows between these part properties denote interactions between the relevant conceptual subsystems. Connectors with item flows between part properties and the diagram frame signify interactions between conceptual subsystems and external elements of the SoI. These connections should be established using proxy ports.

The following figure shows the conceptual subsystems of the turbojet engine as well as their communication with the outside and inside.

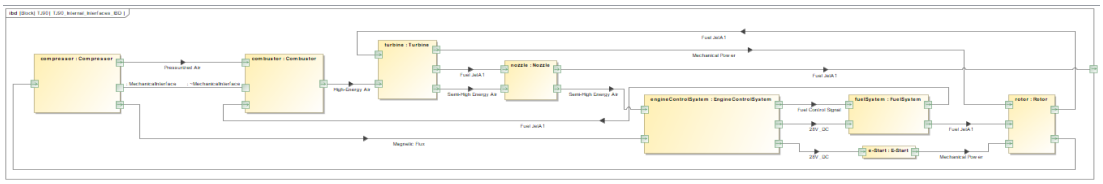


**Figure 6.39:** ibd[Block]TJ90[TJ90_Internal_Interfaces_IBD]

After all conceptual subsystems have been captured in the model, the white-box scenario can be revisited to allocate the white-box functions of the SoI to its conceptual subsystems, indicating that each conceptual subsystem (also known as a functional block) is responsible for one or more white-box functions. Conceptual subsystems can be represented as swimlanes in the activity diagram.

The following figure displays the subfunctions of the *Start Engine* function, allocated to the conceptual subsystems of the SoI.

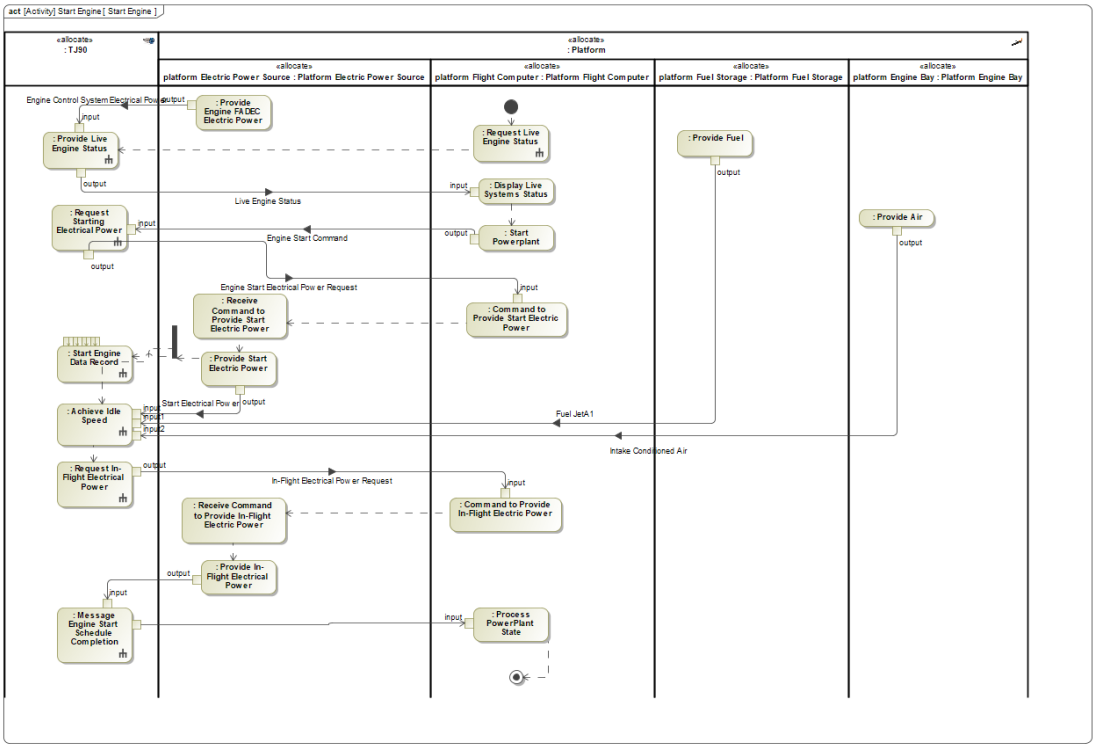



**Figure 6.40:** act[Activity]Start Engine[StartEngine]

According to the MagicGrid framework, artifacts capturing the communication between conceptual subsystems should be organized in a distinct package within the White Box package. It is advisable to name this package Conceptual Subsystems Communication.

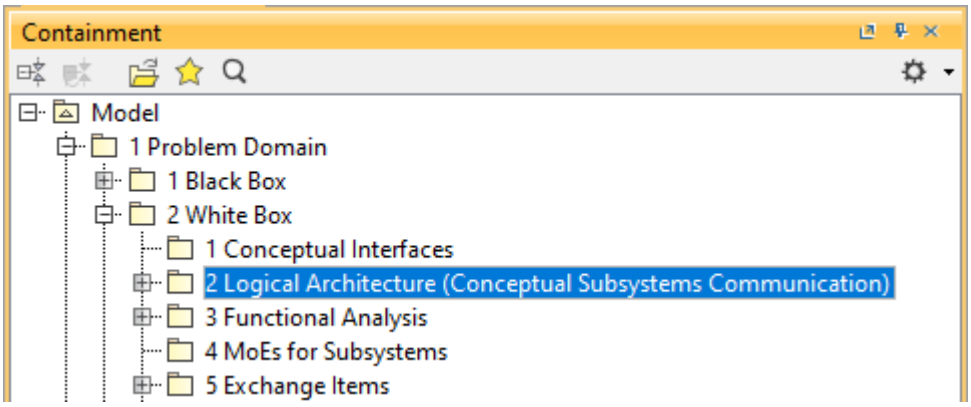



**Figure 6.41:** Conceptual Subsystems Communication in the containment tree

To begin defining conceptual interfaces, which capture the inputs and outputs of the System of Interest (SoI), a block definition diagram (BDD) should be created, and the block representing the SoI should be displayed in the diagram pane. Prior to this, the block must be relocated to the Conceptual Subsystems Communication package within the model. This step marks the transition from its previous black-box state to the white-box analysis phase.

The following figure illustrates the Blackbox ICD Table, offering an alternative view of the turbojet engine interfaces.

Criteria

Element Type: Connector ... Context: 0 External Interfaces ... Filter:

| # | Part A | Port A | Port A Features | Item Flow | Port B | Port B Features | Part B |
|---|---|---|---|---|---|---|---|
| 1 | : TJ90 | inout p2 : Stagnant Surrounding Air | | ◀ Stagnant Surrounding Air<br>▶ Stagnant Surrounding Air | inout p2 : ~Stagnant Surrounding Air | | : Platform |
| 2 | : TJ90 | in p4 : Platform Vibrations | | ◀ Platform Vibrations | out p4 : ~Platform Vibrations | | : Platform |
| 3 | : TJ90 | in p6 : Maneuver Loads | | ◀ Maneuver Loads | out p6 : ~Maneuver Loads | | : Platform |
| 4 | : TJ90 | in p12 : Intake Conditioned Air | AirSpeed : velocity[metre per second]<br>AirTemperature : celsiusTemperature[k<br>MassFlow : mass flow rate[kilogram pe | ◀ Intake Conditioned Air | out p12 : ~Intake Conditioned Air | AirSpeed : velocity[metre per second]<br>AirTemperature : celsiusTemperature[k<br>MassFlow : mass flow rate[kilogram pe | : Platform |
| 5 | : TJ90 | in p14 : Fuel JetA1 | FuelFlow : mass flow rate[kilogram pe<br>FuelTemperature : celsiusTemperature | ◀ Fuel JetA1 | out p14 : ~Fuel JetA1 | FuelFlow : mass flow rate[kilogram pe<br>FuelTemperature : celsiusTemperature | : Platform |
| 6 | : TJ90 | in p18 : Start Electrical Power | | ▶ Start Electrical Power | out p18 : ~Start Electrical Power | | : Platform |
| 7 | : TJ90 | in p20 : In-Flight Electrical Power | | ◀ In-Flight Electrical Power | out p20 : ~In-Flight Electrical Power | | : Platform |
| 8 | : TJ90 | in p21 : Engine Control System Electr... | | ▶ Engine Control System Electrical Pc | out p21 : ~Engine Control System El... | | : Platform |
| 9 | : TJ90 | p28 : Shutdown Command | | ▶ Shutdown Command | p28 : ~Shutdown Command | | : Platform |
| 10 | : TJ90 | p30 : RPM Command | | ▶ RPM Command | p30 : ~RPM Command | | : Platform |
| 11 | : TJ90 | p31 : Platform Command | | ◀ Platform Command | p31 : ~Platform Command | | : Platform |
| 12 | : TJ90 | p36 : Ground Bit Check Command | | ◀ Ground Bit Check Command | p36 : ~Ground Bit Check Command | | : Platform |
| 13 | : TJ90 | p39 : Engine Start Command | | ▶ Engine Start Command | p39 : ~Engine Start Command | | : Platform |
| 14 | : TJ90 | p40 : Engine Bit Check Command | | ◀ Engine Bit Check Command | p40 : ~Engine Bit Check Command | | : Platform |
| 15 | : TJ90 | out p7 : Engine Heat Rejection | | ▶ Engine Heat Rejection | in p7 : ~Engine Heat Rejection | | : Platform |
| 16 | : TJ90 | out p8 : Heat Exchange from Exhaus... | | ▶ Heat Exchange from Exhaust Gases | in p8 : ~Heat Exchange from Exhaus... | | : Platform |
| 17 | : TJ90 | out p9 : Exhaust Gases | | ▶ Exhaust Gases | in p9 : ~Exhaust Gases | | : Platform |
| 18 | : TJ90 | out p10 : Engine Vibrations | | ▶ Engine Vibrations | in p10 : ~Engine Vibrations | | : Platform |
| 19 | : TJ90 | out p11 : Engine Loads | | ▶ Engine Loads | in p11 : ~Engine Loads | | : Platform |
| 20 | : TJ90 | out p13 : Heat Rejection Through Co... | | ▶ Heat Rejection Through Conductio | in p13 : ~Heat Rejection Through Co... | | : Platform |
| 21 | : TJ90 | p32 : Live Engine Status | | ▶ Live Engine Status | p32 : ~Live Engine Status | | : Platform |
| 22 | : TJ90 | p33 : In-Flight Electrical Power Requ... | | ▶ In-Flight Electrical Power Request | p33 : ~In-Flight Electrical Power Req... | | : Platform |
| 23 | : TJ90 | p34 : Idle Completion Message | | ▶ Idle Completion Message | p34 : ~Idle Completion Message | | : Platform |
| 24 | : TJ90 | p35 : Ground Bit Check Return | | ▶ Ground Bit Check Return | p35 : ~Ground Bit Check Return | | : Platform |
| 25 | : TJ90 | out p43 : Gyroscopic Loads | | ▶ Gyroscopic Loads | in p43 : ~Gyroscopic Loads | | : Platform |
| 26 | : TJ90 | p38 : Engine Start Electrical Power Re... | | ▶ Engine Start Electrical Power Reque | p38 : ~Engine Start Electrical Power ... | | : Platform |
| 27 | : TJ90 | out p42 : FuelPump Vibration | | ▶ FuelPump Vibration | in p42 : ~FuelPump Vibration | | : Platform |
| 28 | : TJ90 | p41 : Engine Bit Check Return | | ▶ Engine Bit Check Return | p41 : ~Engine Bit Check Return | | : Platform |
| 29 | : TJ90 | p37 : Flameout Warning Signal | | ▶ Flameout Warning Signal | p37 : ~Flameout Warning Signal | | : Platform |

**Figure 6.42:** Blackbox ICD Table

To maintain a well-structured model, store all interface blocks in a distinct package. Create the Conceptual Interfaces package under the Conceptual Subsystems Communication package and move all interface blocks into it.

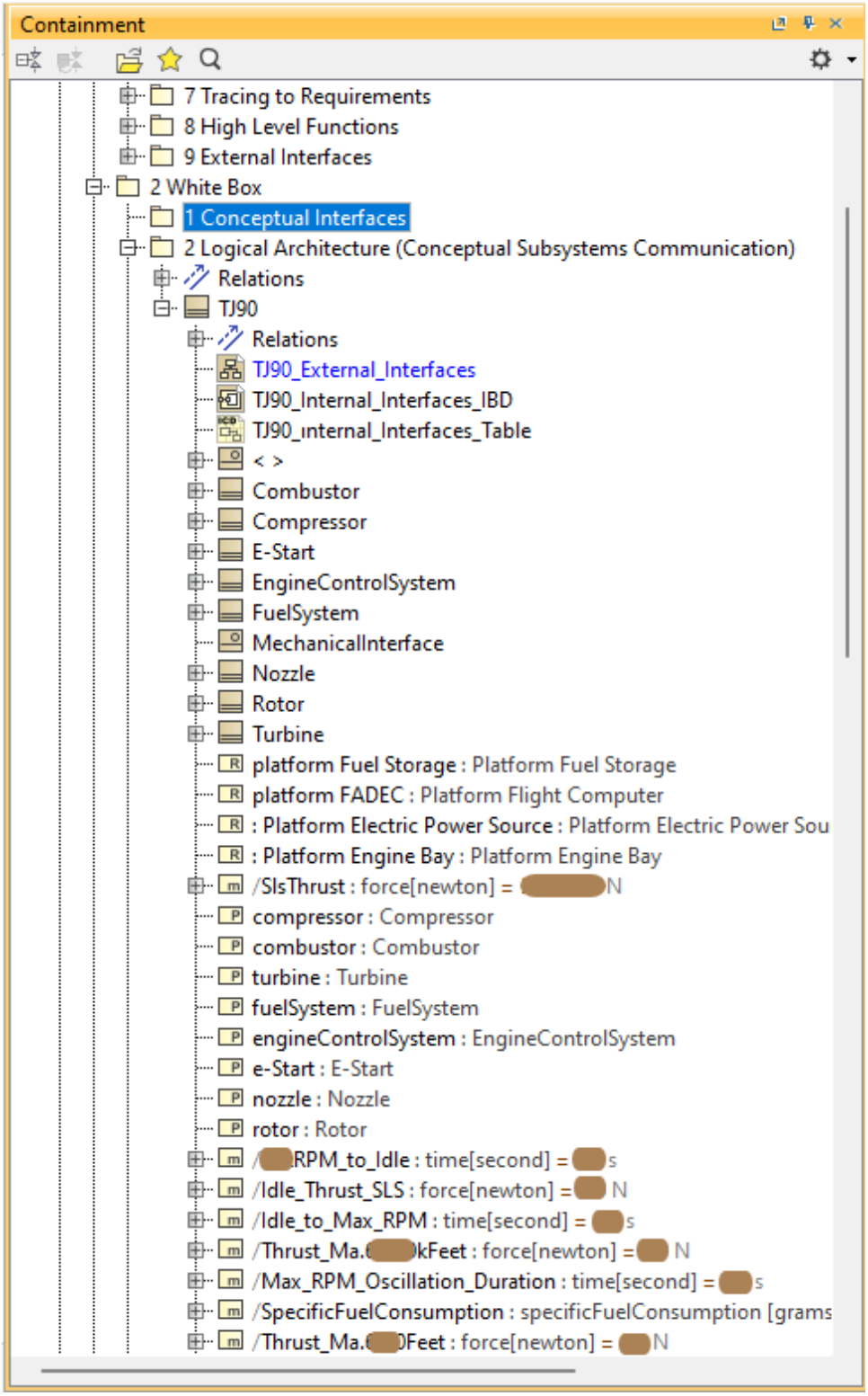



**Figure 6.43:** Conceptual Interface of TJ90

The conceptual subsystems of the turbojet engine can be specified within an internal block diagram (IBD) owned by the TJ90 Block. According to SysML, the diagram's frame represents the SoI's boundaries, allowing the specification of interactions

between internal parts of the SoI and external elements via proxy ports shown on the frame.

Functional analysis has identified the following conceptual subsystems within the turbojet engine:

- Combustor
- Compressor
- E-Start
- EngineControlSystem
- Fuel System
- Nozzle
- Rotor
- Turbine

These subsystems should be defined as part properties of the TJ90 block. While part properties do not require names, they should be typed by blocks that represent the conceptual subsystems mentioned above.

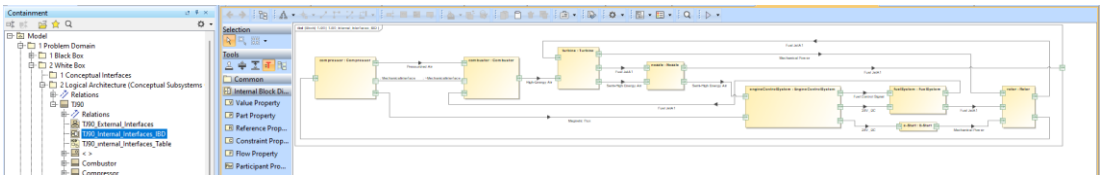


**Figure 6.44:** Part properties of the TJ90 Block.

For specifying interactions between conceptual subsystems and the outside of the SoI, it is focused on defining interactions between the conceptual subsystems (also known as functional blocks) of the turbojet engine and its external environment. An interaction between a subsystem and the exterior of the System of Interest (SoI) can be depicted as a connector with one or more item flows, linking the relevant part property to the diagram frame (the diagram frame in the internal block diagram delineates the SoI boundaries). Unlike the System Context cell, connectors here are established using compatible proxy ports. The exchange of information, matter, or energy over these connectors is represented by items conveyed through item flows.

The system functions and conceptual subsystems, with or without quantifiable characteristics of the SoI, refine stakeholder needs. Upon completion, establish traceability relationships to stakeholder needs (Traceability to Stakeholder Needs). It may be also needed to specify quantifiable characteristics for one or more conceptual

subsystems, which should be more detailed than those defined for the entire SoI (Measures of Effectiveness). If so, proceed to the MoEs for Subsystems cell.

### 6.1.2.3. MoEs for Subsystems

In this step, it may be chosen to specify the Measures of Effectiveness (MoEs) for one or more conceptual subsystems (also known as functional blocks) of the System of Interest (SoI) to further refine non-functional stakeholder needs. This step is optional, as additional MoEs beyond those defined for the entire SoI may not be necessary.

To capture MoEs for conceptual subsystems in the modeling tool, the SysML block definition diagram (BDD) infrastructure should be utilized, similar to how MoEs for the SoI were captured in the Measures of Effectiveness cell. For each conceptual subsystem, MoEs can be represented as value properties with the «moe» stereotype applied.

Following the MagicGrid framework, model elements capturing MoEs for subsystems should be organized in a dedicated package within the White Box package.

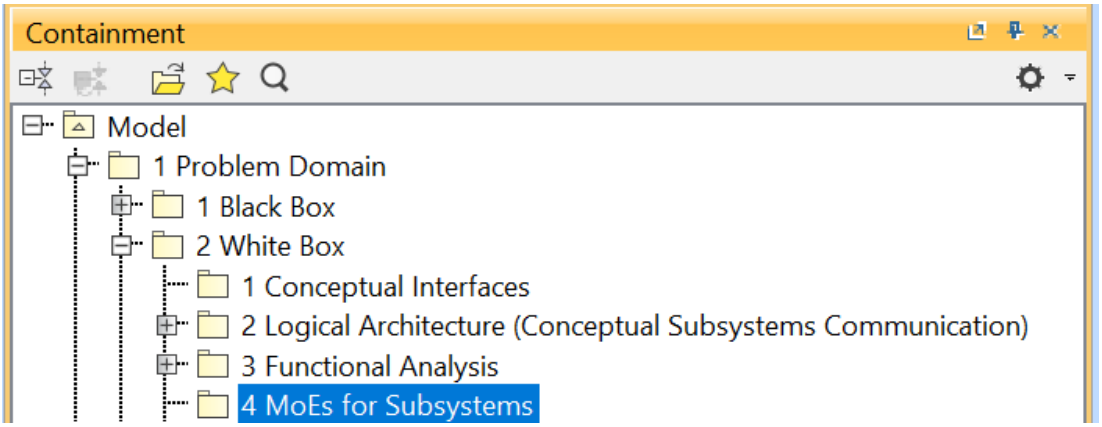



**Figure 6.45:** MoEs for Subsystems in the containment tree

Upon defining the MoEs for subsystems, next step is to specify which stakeholder needs they refine (Traceability to Stakeholder Needs).

### 6.1.2.4. Traceability to Stakeholder Needs

Establishing traceability relationships between the elements that represent white-box functions, conceptual subsystems, conceptual interfaces, and non-functional

characteristics of the System of Interest (SoI) and those that capture stakeholder needs is crucial to achieving a comprehensive problem domain model. These connections ought to start with the most minute components, such movements that represent the functional decomposition model's leaf functions. Ensuring that every stakeholder need is refined by one or more elements of other SysML types is crucial. Failure to do so implies that not all stakeholder needs are addressed in the problem domain model, potentially leading to incorrect system requirements, which can result in developing an inadequate system.

SysML diagrams are generally not ideal for handling a large number of cross-cutting relationships, such as refine. For this purpose, matrices or maps are more effective. The modeling tool provides various predefined matrices for specifying cross-cutting relationships. A Refine Requirement Matrix is particularly useful for capturing refine relationships.

Traceability relationships and their corresponding views should be stored in a separate package, created directly under the Problem Domain package. This package should be the final internal package to reflect the modeling workflow.

To start capturing refine relationships, first create a Refine Requirement Matrix. This matrix is designed to represent requirements in its columns and refine relationships in its cells. The rows of the matrix should display the following:

- Call behavior actions, representing functions of the SoI
- Part properties, representing conceptual subsystems of the SoI
- Proxy ports, representing the interfaces of the SoI
- Value properties with the «moe» stereotype, capturing MoEs of the SoI or its subsystems

The matrix also displays implied refine relationships (dotted arrows). Once a leaf requirement is refined, all higher-level requirements within the same branch are also considered refined. This is the rationale behind establishing implied refine relationships.

The problem domain analysis is considered complete when at least 90 percent of stakeholder needs (this threshold may vary by organization) are refined by one or more elements from the problem domain model.

The complete problem domain analysis model, which includes stakeholder needs, expected functions, and the conceptual structure of the SoI (with or without MoEs), can serve as the initial system requirements specification in a model based format. If a textual system requirements specification is needed (which is common), the problem domain analysis model can be converted into a SysML requirements model, either manually or automatically using transformation algorithms. This requirements model can then be provided to system architects as input for developing the logical architecture of the SoI in the solution domain.

## 6.2. Solution Domain

Upon completion of the problem domain analysis and the transfer of stakeholder needs for the SoI into the SysML model, attention should shift to developing the solution domain model for the SoI. This model delineates a detailed, cross-disciplinary logical architecture intended to resolve the issues identified in the problem domain analysis. It is important to distinguish this model from detailed design models, which are developed in later stages of SoI development and fall outside the scope of the MagicGrid framework.

Given that a single problem definition can lead to multiple potential solutions, a trade-off analysis is advisable to determine the most effective solution for system implementation.

As the following table illustrates, defining the requirements, structure, behavior, and parameters of the SoI is the first step in developing the solution domain model. Furthermore, creating a logical architecture typically requires several iterations, descending from the system level to the subsystem level, the subsystem level to the component level architecture, and, if necessary, even below. The number of iterations determines how accurate the system architecture model is.

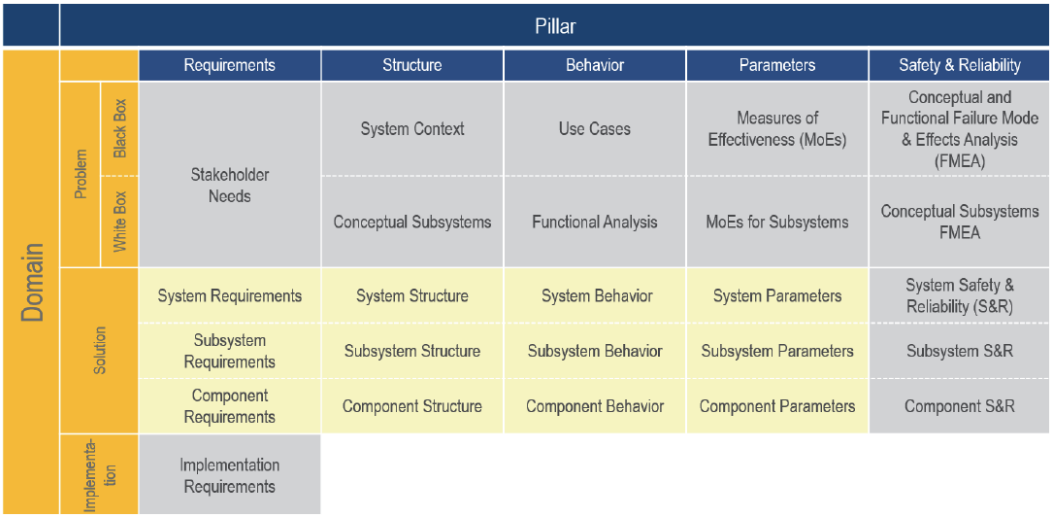



**Figure 6.46:** MagicGrid Framework – Solution Domain [34]

## 6.3. Implementation Domain

The logical architecture is designed and established by systems engineering disciplines within the “Functional Baseline” configuration domain for the SOI, with the purpose of providing the necessary inputs to the systems designers.

The system is prepared for deployment once the logical architecture, overall system design, and ideal configuration have been determined by trade-off analysis.

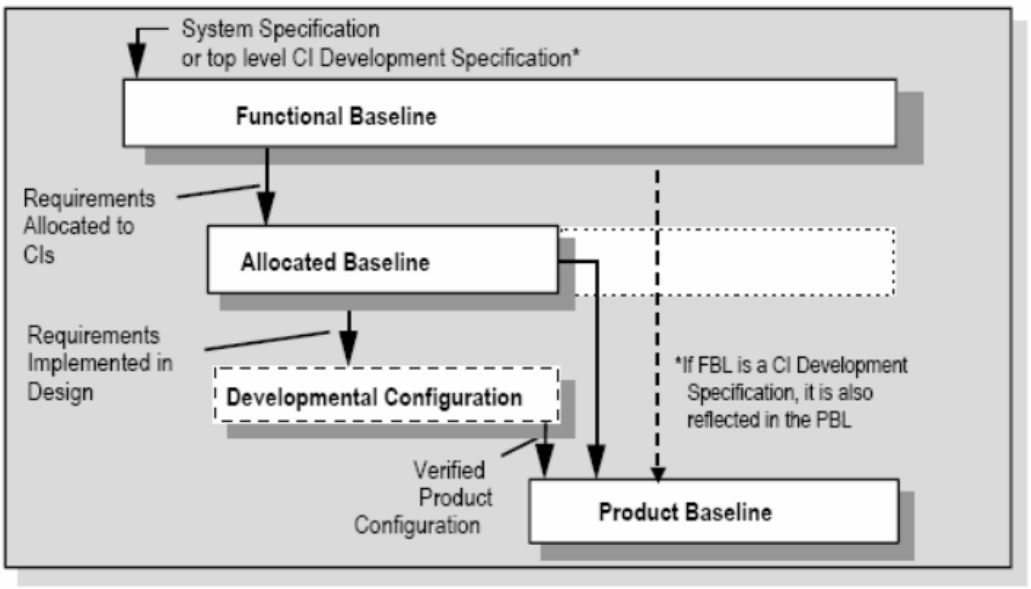



**Figure 6.47:** MIL-STD-973, Notice 3 Baseline Concept

At this stage, the system should be regarded as tangible rather than conceptual, in contrast to the earlier phases of problem and solution domain analysis, where the Allocated Baseline for the SOI will be introduced.

As illustrated in the following table, the implementation domain is only partially addressed by the MagicGrid framework. The framework outlines the specification of implementation requirements for the SoI, but it does not encompass the detailed physical design. Subsequent steps in the SoI development can be carried out using various tools, such as electrical/mechanical CAD software or automated code generation tools.

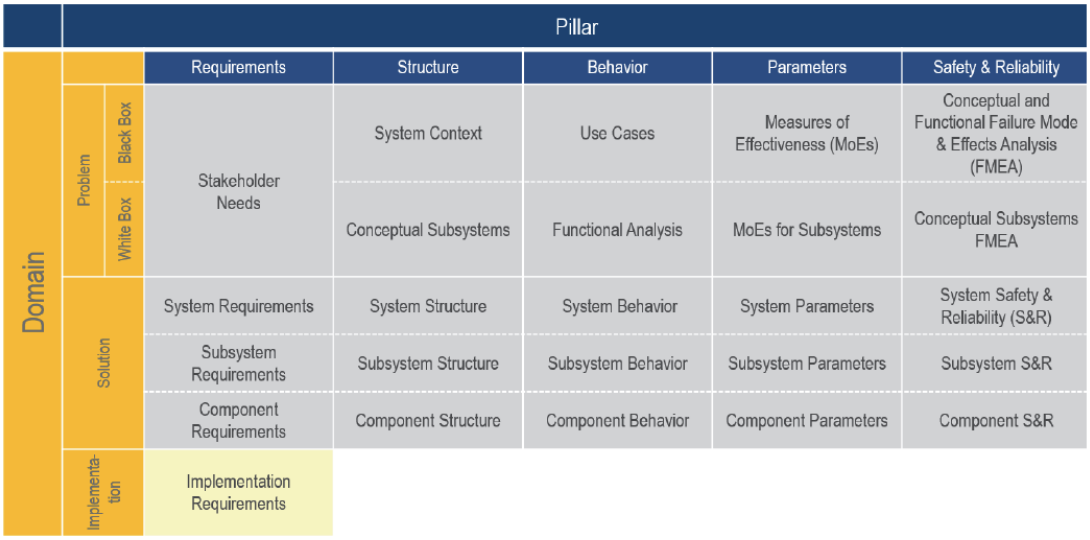



**Figure 6.48:** MagicGrid Framework – Implementation Domain [34]

To implement the System of Interest (SoI), detailed requirements are essential to guide the creation of the system's detailed physical design. These requirements are derived from the logical architecture of the SoI, encompassing the logical architectures of all its subsystems and more specific components. The implementation requirements specification is sent to engineers from various disciplines who are in charge of designing the various system components.

As seen by the matrix analysis, the implementation requirements definition is currently incomplete. It will be regarded complete when one or more implementation requirements improve each element in the system configuration model and address at least one solution domain requirement. The inclusion of derivation links between needs in different domains allows for simple tracing from extremely specific to more abstract requirements, shedding light on how distinct objects impact the development of others. It is advised that this effect study, whether downstream or upstream, be carried out using the Requirement Derivation Map infrastructure, one of the modeling tool's preconfigured relationship maps.

# 7. RESULTS

The pilot application aims to create architectural representations that describe the integration of TA's Şimşek platform with TEI's TJ90 turbojet engine.


**Figure 7.1:** Şimşek (TA) & TEI – TJ90

The use case diagrams have been developed in accordance with the requirements and existing external interfaces. During the detailing process, the need for additional external interfaces was also documented by capturing existing tacit knowledge.(Figure 7.2)

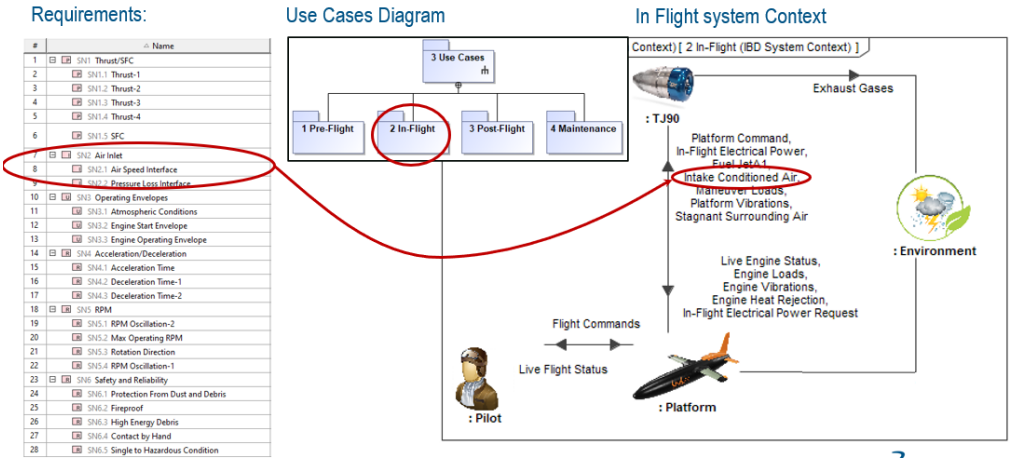



**Figure 7.2:** TJ90 Diagrams

Use case diagrams were developed to capture the top two levels of functions utilized by the platform. Subsequently, the third level of functions was recorded onto activity diagrams, detailing the interactions between the engine and the platform's subsystems. This process allowed for the elaboration of external interfaces, the addition of

previously missing external interfaces to earlier diagrams, and the creation of fourth-level functions for the engine subsystems. (Figure 7.3)

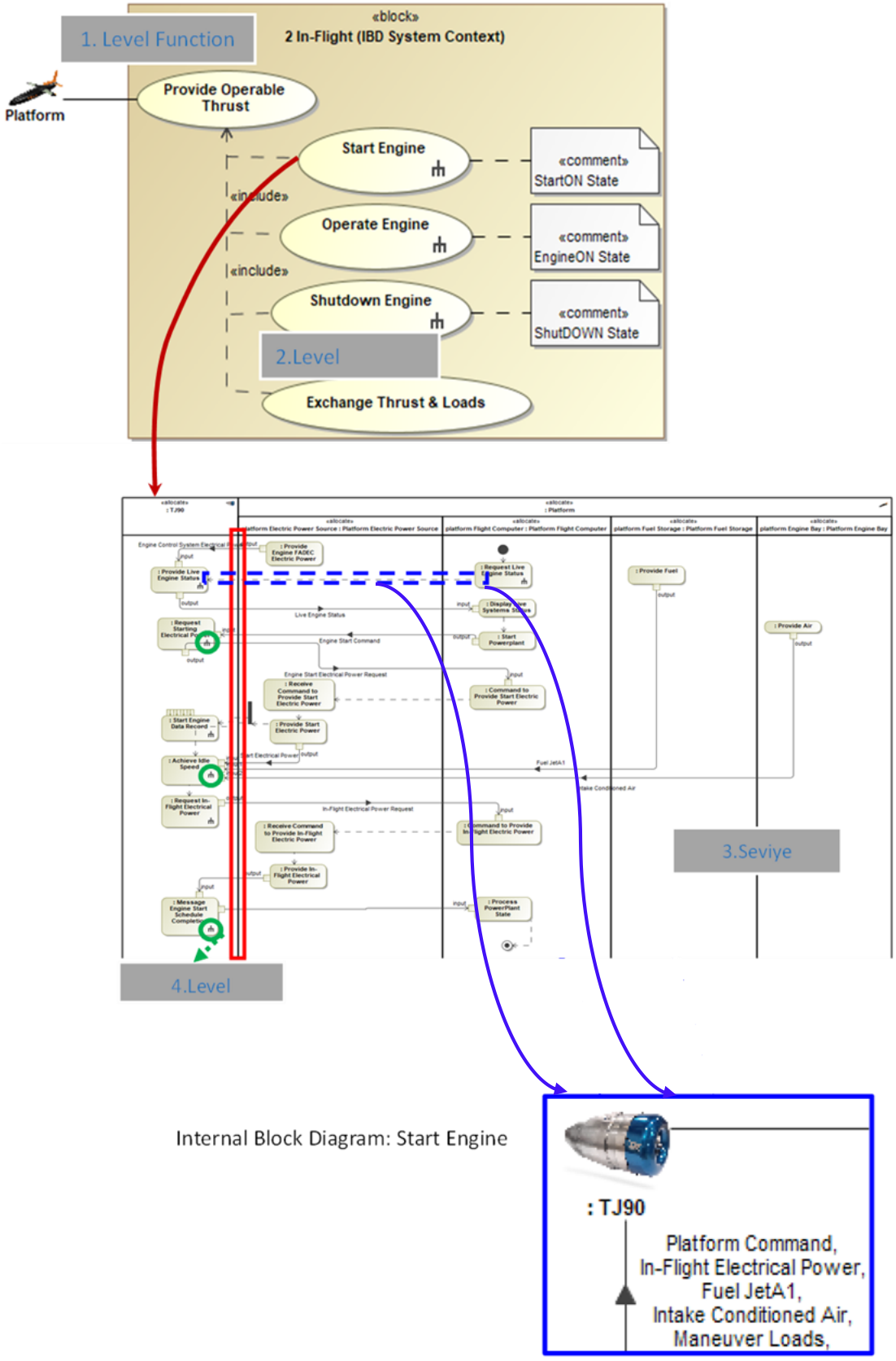



**Figure 7.3:** Fourth-level functions for the engine subsystems

Functions were allocated to the imported requirements, indicating the need for writing new derived requirements which are shown inside the red boxes. (Figure 7.4)

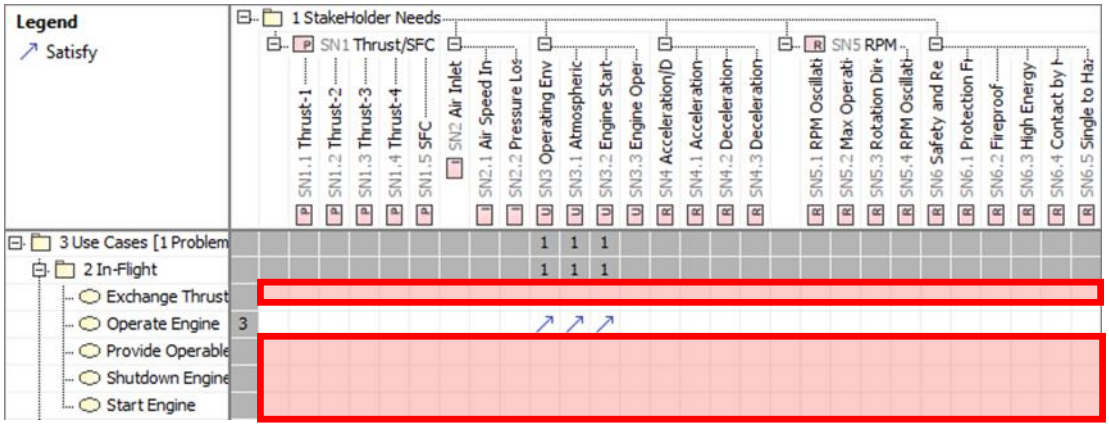



**Figure 7.4:** Allocation Matrix

Functions were successfully allocated to their respective subsystems automatically, which also allowed for creating an automatic function tree. (Figure 7.5)

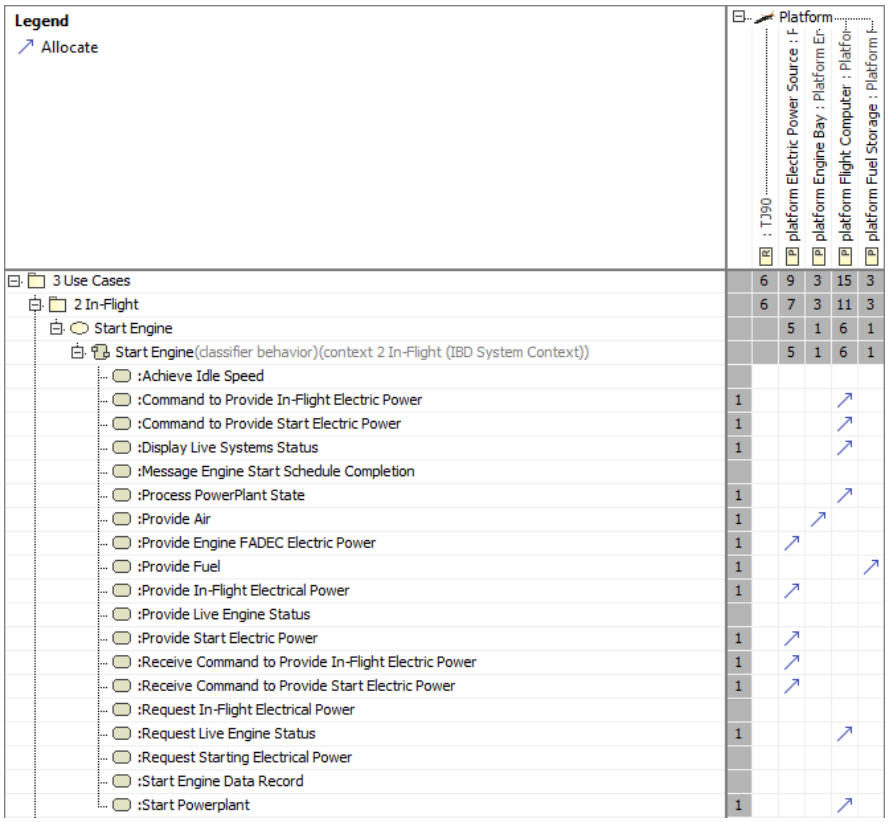



**Figure 7.5:** Allocation Matrix

From the specified contexts, all external interfaces consolidated into an automated interface diagram. The areas marked by red rectangles indicate external interfaces which need to be discussed with the stakeholder and users. (Figure 7.6)

| # | Part A | Port A | Port A Features | Item Flow | Port B | Port B Features | Part B |
|---|---|---|---|---|---|---|---|
| 1 | : TJ90 | inout p2 : Stagnant Surrounding Air | | Stagnant Surrounding Air<br>Stagnant Surrounding Air | inout p2 : ~Stagnant Surrounding Air | | : Platform |
| 2 | : TJ90 | in p4 : Platform Vibrations | | Platform Vibrations | out p4 : ~Platform Vibrations | | : Platform |
| 3 | : TJ90 | in p6 : Maneuver Loads | | Maneuver Loads | out p6 : ~Maneuver Loads | | : Platform |
| 4 | : TJ90 | in p12 : Intake Conditioned Air | AirSpeed : velocity{metre per second}<br>AirTemperature : celsiusTemperature{k<br>MassFlow : mass flow rate{kilogram pe | Intake Conditioned Air | out p12 : ~Intake Conditioned Air | AirSpeed : velocity{metre per second}<br>AirTemperature : celsiusTemperature{k<br>MassFlow : mass flow rate{kilogram pe | : Platform |
| 5 | : TJ90 | in p14 : Fuel JetA1 | FuelFlow : mass flow rate{kilogram per<br>FuelTemperature : celsiusTemperature | Fuel JetA1 | out p14 : ~Fuel JetA1 | FuelFlow : mass flow rate{kilogram per<br>FuelTemperature : celsiusTemperature | : Platform |
| 6 | : TJ90 | in p18 : Start Electrical Power | | Start Electrical Power | out p18 : ~Start Electrical Power | | : Platform |
| 7 | : TJ90 | in p20 : In-Flight Electrical Power | | In-Flight Electrical Power | out p20 : ~In-Flight Electrical Power | | : Platform |
| 8 | : TJ90 | in p21 : Engine Control System Electr... | | Engine Control System Electrical Pc | out p21 : ~Engine Control System El... | | : Platform |
| 9 | : TJ90 | p28 : Shutdown Command | | Shutdown Command | p28 : ~Shutdown Command | | : Platform |
| 10 | : TJ90 | p30 : RPM Command | | RPM Command | p30 : ~RPM Command | | : Platform |
| 11 | : TJ90 | p31 : Platform Command | | Platform Command | p31 : ~Platform Command | | : Platform |
| 12 | : TJ90 | p36 : Ground Bit Check Command | | Ground Bit Check Command | p36 : ~Ground Bit Check Command | | : Platform |
| 13 | : TJ90 | p39 : Engine Start Command | | Engine Start Command | p39 : ~Engine Start Command | | : Platform |
| 14 | : TJ90 | p40 : Engine Bit Check Command | | Engine Bit Check Command | p40 : ~Engine Bit Check Command | | : Platform |
| 15 | : TJ90 | out p7 : Engine Heat Rejection | | Engine Heat Rejection | in p7 : ~Engine Heat Rejection | | : Platform |
| 16 | : TJ90 | out p8 : Heat Exchange from Exhaus... | | Heat Exchange from Exhaust Gases | in p8 : ~Heat Exchange from Exhaus... | | : Platform |
| 17 | : TJ90 | out p9 : Exhaust Gases | | Exhaust Gases | in p9 : ~Exhaust Gases | | : Platform |
| 18 | : TJ90 | out p10 : Engine Vibrations | | Engine Vibrations | in p10 : ~Engine Vibrations | | : Platform |
| 19 | : TJ90 | out p11 : Engine Loads | | Engine Loads | in p11 : ~Engine Loads | | : Platform |
| 20 | : TJ90 | out p13 : Heat Rejection Through Co... | | Heat Rejection Through Conductio | in p13 : ~Heat Rejection Through Co... | | : Platform |
| 21 | : TJ90 | p32 : Live Engine Status | | Live Engine Status | p32 : ~Live Engine Status | | : Platform |
| 22 | : TJ90 | p33 : In-Flight Electrical Power Requ... | | In-Flight Electrical Power Request | p33 : ~In-Flight Electrical Power Req... | | : Platform |
| 23 | : TJ90 | p34 : Idle Completion Message | | Idle Completion Message | p34 : ~Idle Completion Message | | : Platform |
| 24 | : TJ90 | p35 : Ground Bit Check Return | | Ground Bit Check Return | p35 : ~Ground Bit Check Return | | : Platform |
| 25 | : TJ90 | out p43 : Gyroscopic Loads | | Gyroscopic Loads | in p43 : ~Gyroscopic Loads | | : Platform |
| 26 | : TJ90 | p38 : Engine Start Electrical Power Re... | | Engine Start Electrical Power Reque | p38 : ~Engine Start Electrical Power ... | | : Platform |
| 27 | : TJ90 | out p42 : FuelPump Vibration | | FuelPump Vibration | in p42 : ~FuelPump Vibration | | : Platform |
| 28 | : TJ90 | p41 : Engine Bit Check Return | | Engine Bit Check Return | p41 : ~Engine Bit Check Return | | : Platform |
| 29 | : TJ90 | p37 : Flameout Warning Signal | | Flameout Warning Signal | p37 : ~Flameout Warning Signal | | : Platform |

**Figure 7.6:** External Interfaces List

Safety analysis can be performed in an integrated manner with the system architecture. Architectural elements derived from the models, such as requirements, functions, interfaces, and components, were interlinked with unique failure modes. An allocation matrix was used to identify future safety activities, highlighting the need to allocate failure mode solutions within the system architecture, as shown in red (Figure 7.7).

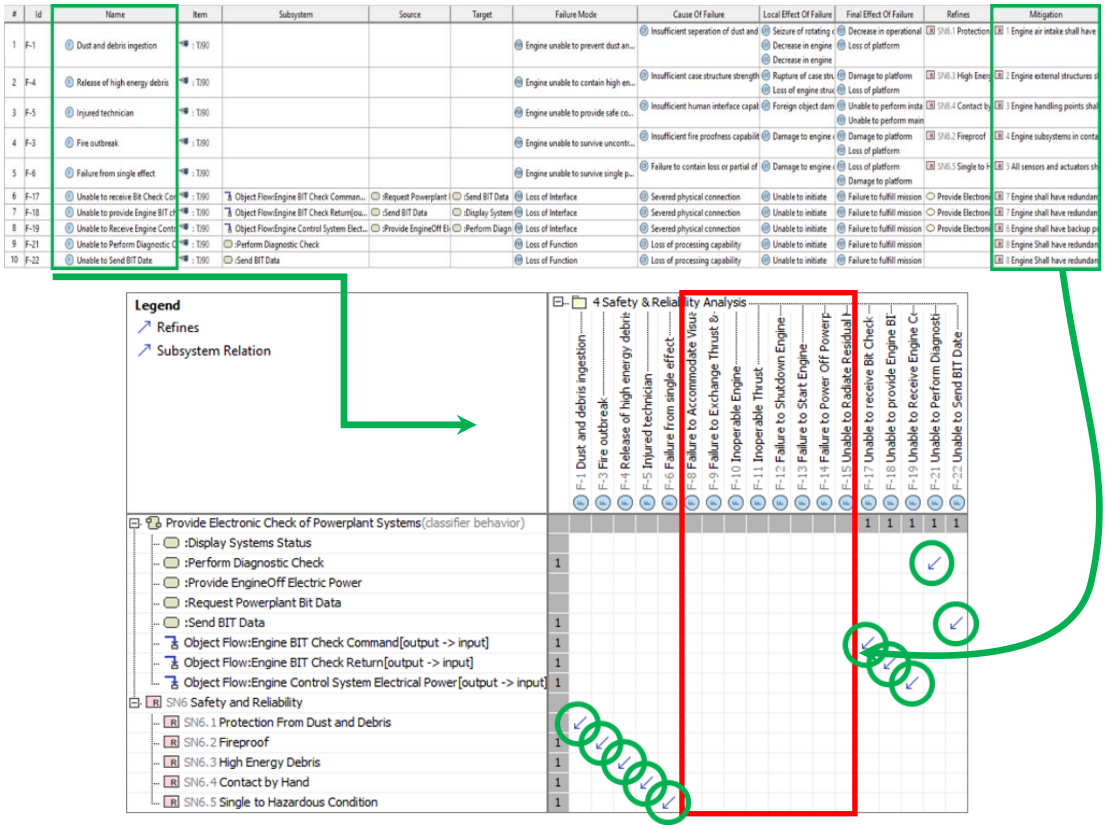



**Figure 7.7:** Allocation Matrix

Through this study conducted in the problem domain, it has been possible to thoroughly explore the content that needs to be addressed in the solution domain. The architecture established within MBSE has effectively guided the systems engineering teams and provided high-value outputs to the interfacing teams.

## 7.1. Case Studies

Upon retrospective analysis of the system model developed for the TJ90 pilot project through traditional systems engineering methodologies, the discernible disparity and augmented value conferred by Model Based Systems Engineering (MBSE) vis-à-vis traditional approaches is rendered more conspicuous through case studies encountered throughout the project's duration. MBSE facilitates enhanced management of Impact Analysis, Consistency Analysis, Error Detection, and Early Stage Verification activities, thereby engendering significant temporal and fiscal economies compared to conventional systems engineering practices.

The tangible benefits afforded by MBSE are palpably evident in the context of the TJ90 Turbojet Engine Pilot Project, where real-life illustrations underscore its

efficacy. To underscore the error detection prowess inherent in model based systems engineering within the purview of solution and application domains, a selection of case studies is scrutinized.

### 7.1.1. CASE-I : RPM Oscillation Error

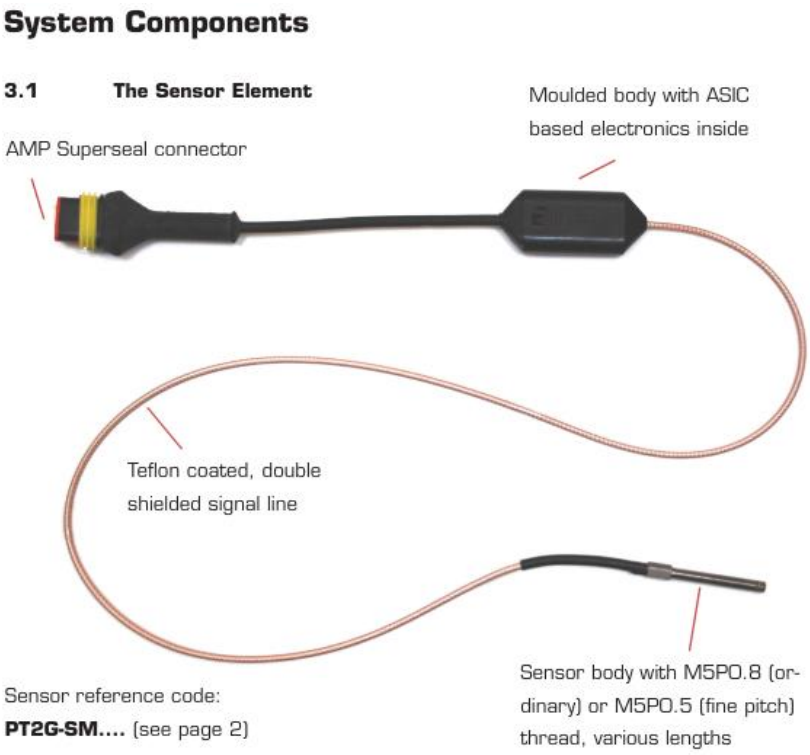
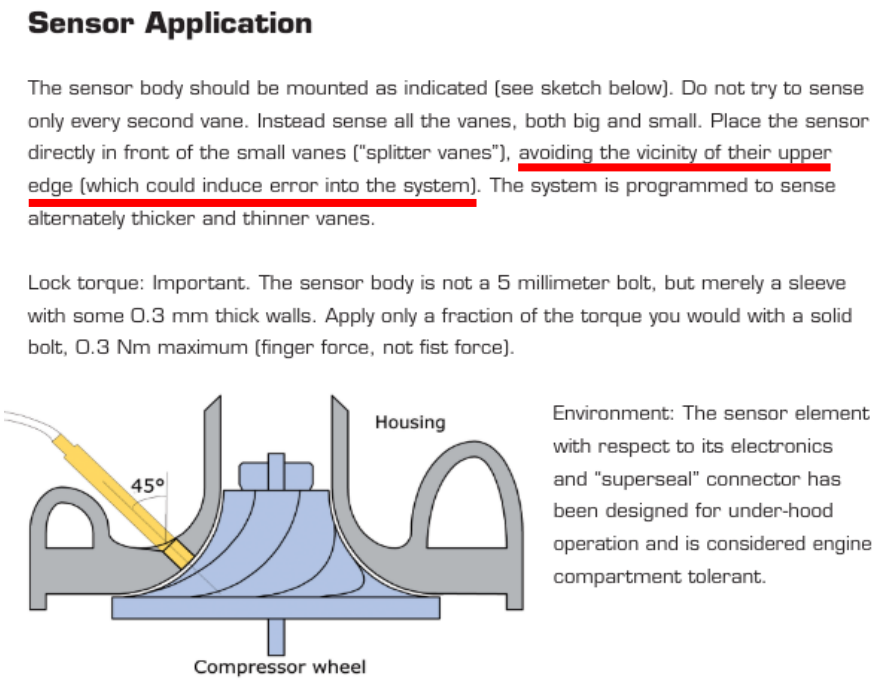



**Figure 7.8:** RPM Sensor

An RPM sensor is a device used to measure the speed in rotating systems. In the TJ90 project, the RPM sensor, which had been functioning correctly, began to fail in accurately measuring speed and caused sudden oscillations after a design modification. To address this issue, a fault report was generated, and a committee was convened to conduct root cause analysis aimed at identifying the root cause of the error. While evaluations were ongoing, various potential causes for the sensor's oscillations were tested by altering relevant parameters. However, none of the modifications and tests resolved the oscillation error.

The RPM sensor is an off-the-shelf product, and such products come with technical specification documents. Upon contacting the supplier for support regarding the

possible cause of the error, it was determined that the error was due to a known issue documented in the supplier's product catalog.



**Sensor Application**

The sensor body should be mounted as indicated (see sketch below). Do not try to sense only every second vane. Instead sense all the vanes, both big and small. Place the sensor directly in front of the small vanes ("splitter vanes"), avoiding the vicinity of their upper edge (which could induce error into the system). The system is programmed to sense alternately thicker and thinner vanes.

Lock torque: Important. The sensor body is not a 5 millimeter bolt, but merely a sleeve with some 0.3 mm thick walls. Apply only a fraction of the torque you would with a solid bolt, 0.3 Nm maximum (finger force, not fist force).



Environment: The sensor element with respect to its electronics and "superseal" connector has been designed for under-hood operation and is considered engine compartment tolerant.

**Figure 7.9:** Datasheet of the Sensor

As evident from the highlighted section in the product catalogue, the supplier has duly acknowledged the potential failure of the RPM sensor to accurately read near the edges of compressor components, attributing this issue to the gradual thinning of blade tips at these extremities, thereby impeding sensor detection due to the decreased blade thickness.

Subsequently, within the TJ90 turbojet engine framework, the root cause of RPM sensor failure in the subsequent configuration following a design modification was traced to a transformation in the form of the inlet part housing the RPM sensor. This modification inadvertently altered the sensor's viewing angle, redirecting it towards the blade edge.

In the prior design iteration, the distance measured approximately 8 mm, whereas in the revised design, this was notably reduced to approximately 2.5 mm, thereby serving as the root cause of the ensuing error.

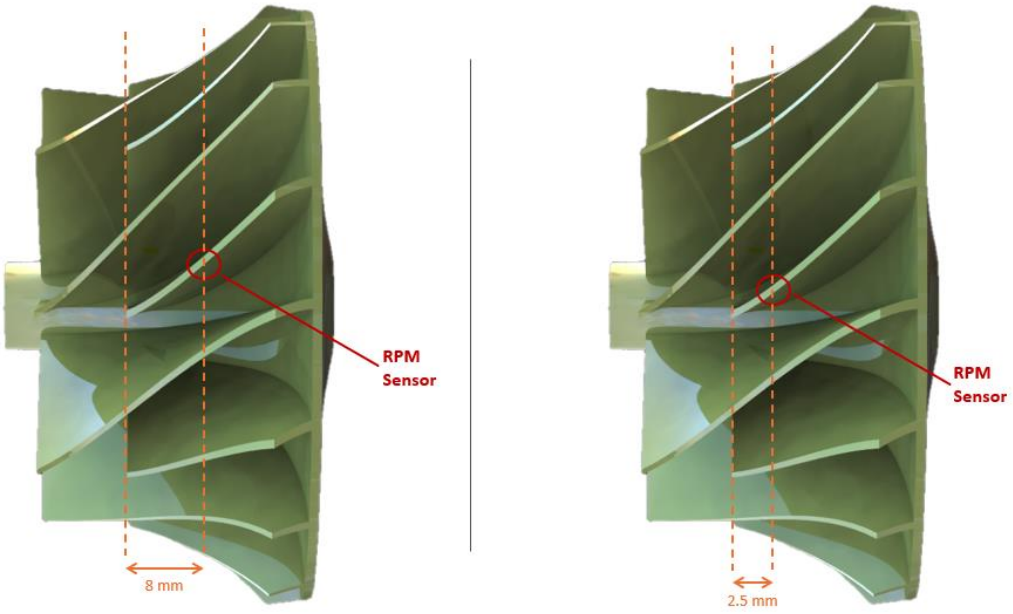



**Figure 7.10:** Sensor Location of prior design vs later design

Had the TJ90 project employed MBSE methodologies at the outset, the error could have been pre-emptively identified during the design phase through a systematic analysis of the system model, averting its manifestation during testing. Upon discovery of an error during testing, the ensuing processes entail a cyclical sequence, commencing with error documentation, convening of a committee, root cause analysis, formulation of temporary/permanent solution proposals, testing and validation of proposed solutions, identification of the optimal remedy, implementation of design alterations to incorporate the solution, issuance of revised drawings and diagrams, fabrication of new part designs, reversion of the engine from testing to assembly, disassembly and subsequent reassembly with the new component, comprehensive re-testing of the entire engine, validation of permanent error resolution through rigorous testing, documentation of outcomes, and final error closure. This intricate process results in considerable labour expenditure.

In the present case study, extensive testing procedures were conducted, necessitating the involvement of 8-10 personnel over a duration of 3-4 months solely dedicated to resolving the identified issue. Considering the associated fuel consumption, electricity usage, utilization of testing infrastructure, and labour costs, it becomes apparent that even a singular error eventuates in significant temporal and fiscal repercussions.

The detection of the RPM oscillation error leveraging the MBSE methodology within the system model context is outlined as follows:

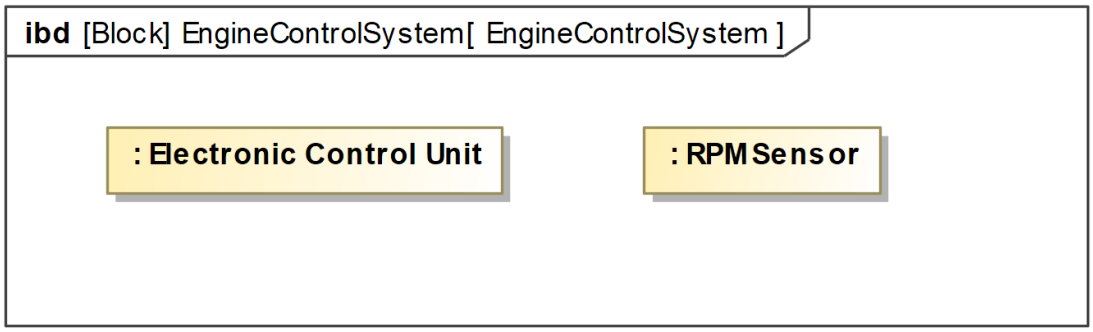



**Figure 7.11:** ibd[Block] EngineControlSystem

The RPM sensor, along with the Electrical Control Unit responsible for its regulation, are designated as subcomponents encapsulated within the broader Engine Control System.

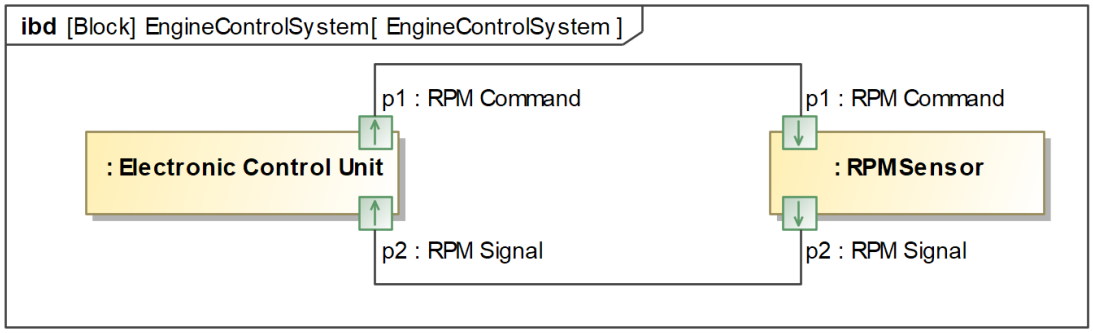



**Figure 7.12:** ibd[Block] EngineControlSystem

Flows are delineated to depict the exchange of data, material, and energy between the aforementioned components. Specifically, signal and command data are transmitted between the two entities.

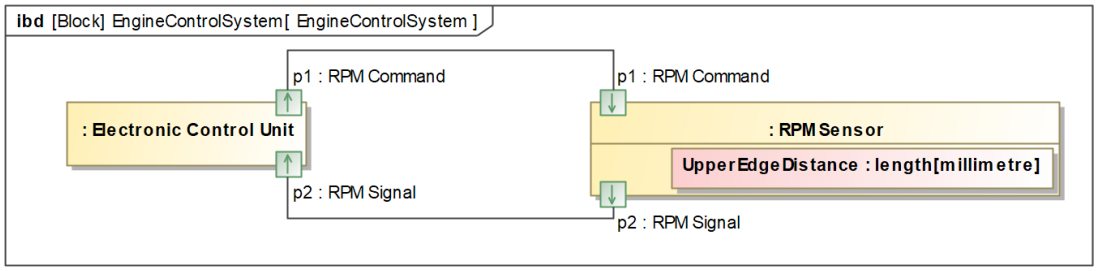



**Figure 7.13:** ibd[Block] EngineControlSystem

The error mode concerning the RPM sensor, as gleaned from the product catalog, is designated as a parameter defining the sensor's operational state.

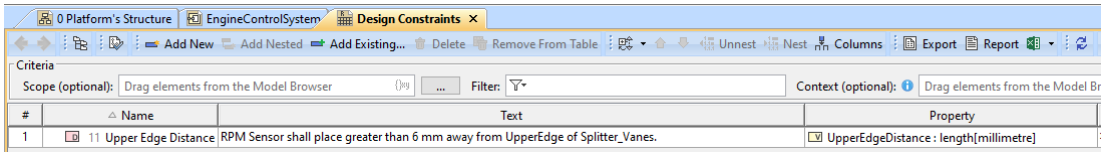



**Figure 7.14:** Design Constraint

The design constraint, stipulating the requisite value, is incorporated into the requirements table dedicated to design constraints within the Solution Domain. This value may either be sourced from the supplier or deemed obligatory, given the sensor's successful prior utilization in the preceding configuration. Furthermore, the establishment of a small-scale test arrangement enables the determination of the distance at which the sensor begins to exhibit malfunction. For instance, a constraint is outlined dictating that the RPM sensor should maintain a minimum distance of 6 mm from the edge of the compressor part.

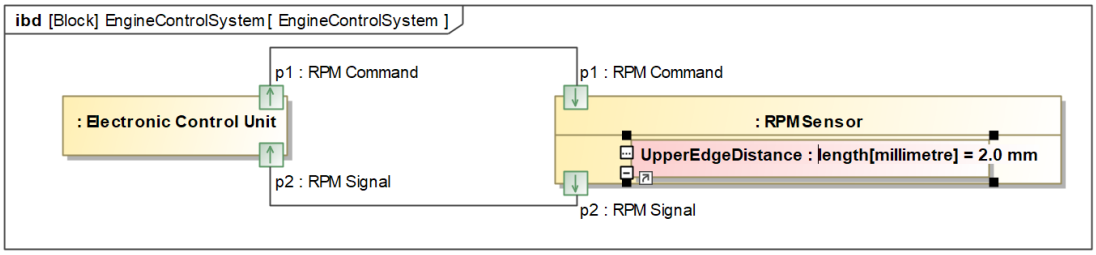



**Figure 7.15:** ibd[Block] EngineControlSystem

Following this, consultation with the mechanical design engineer ensues to ascertain the distance within the design, utilizing the 3D CAD solid model of the pertinent component. The ascertained value is subsequently designated as the nominal value at the requisite location.

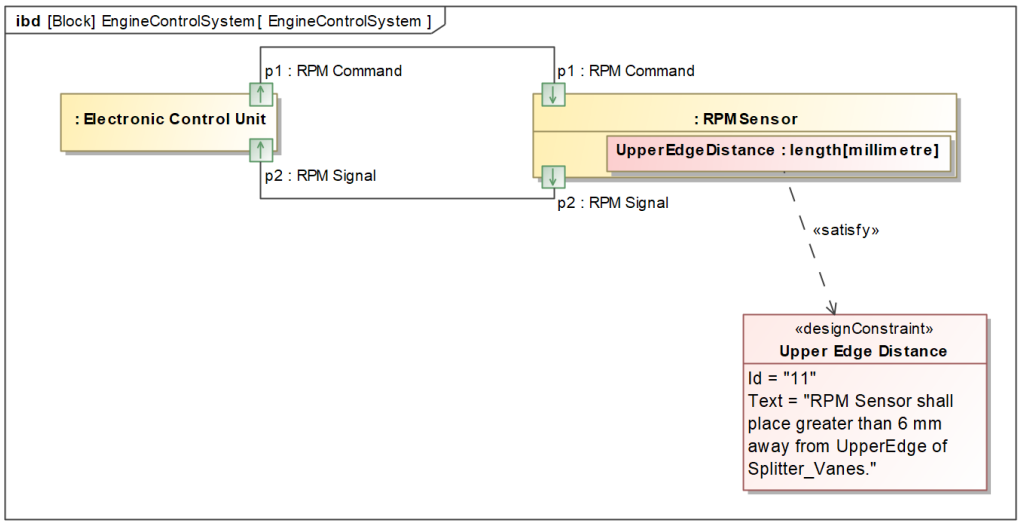



**Figure 7.16:** ibd[Block] EngineControlSystem

The system model functions through interconnections. However, when distinct inputs for the RPM sensor are delineated within the system model, MBSE tools do not inherently identify and correlate them solely based on identical nomenclature. Hence, it becomes imperative to define the pertinent RPM sensor value within the system model by establishing a <<satisfy>> relationship, denoting its adherence to the corresponding design constraint requirement.

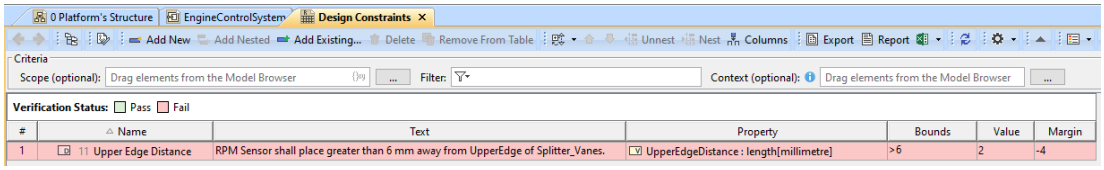



**Figure 7.17:** Design Constraint

The MBSE tool adheres to its proprietary set of requirements composition guidelines, enabling the transformation of verbal expressions into written requirements and subsequently into mathematical expressions. Upon automatic generation of a mathematical expression such as ">6 mm," it can be linked to the respective requirement, facilitating verification of compliance through the validation condition feature within the requirement table. Notably, in instances where the requirement is not satisfied, it is flagged in red, signaling the failure to fulfill the stipulated criterion. This preemptive alert system serves as an early warning mechanism, highlighting potential system errors pertaining to the RPM sensor during the design phase, prior to the commencement of testing procedures.

### 7.1.2. CASE-II: E-Start Clutch and DC Motor Error

The E-Start mechanism, situated at the forefront of the engine, functions as a pivotal component initiating the rotor's motion prior to ignition, facilitating the engine's transition into the ignition cycle. Employing spring-loaded mechanisms, it propels a DC motor utilizing voltage sourced from the power battery via the electronic control unit. Activation of the DC motor induces forward movement in the spring-loaded mechanism, driven by centrifugal force, thereby engaging the rotor and exerting pressure on the initial rotor nut, thus initiating its rotation.

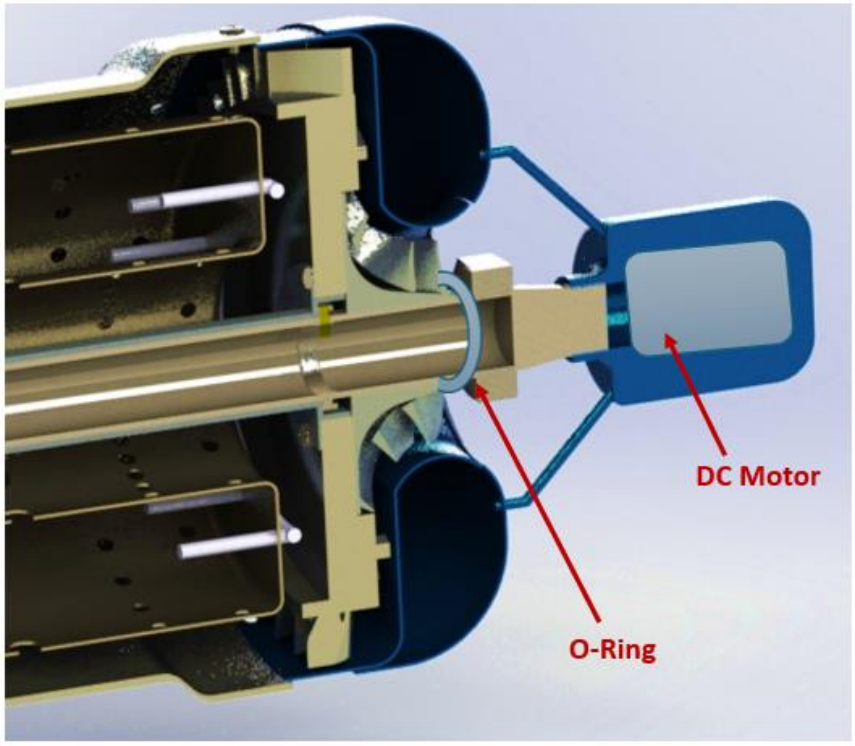



**Figure 7.18:** TJ90 DC Motor and O-Ring (representative)

With the cooling of the weather, it has been observed during tests that a gripping problem has occurred in the E-start mechanism during startup. Subsequent root cause analysis revealed that the error stemmed from the rubber o-ring, which, under cold conditions below 0°C, hardened and failed to perform its penetration function, resulting in it solidifying and spinning freely on the rotor nut. Consequently, an alternative o-ring was sought, and a new o-ring product covering the entire operating temperature range of the engine was adopted. The error here lies in the ready-made o-ring not being compatible with the operating temperature range of the engine. All

subcomponents incorporated into a system are expected to fulfill the requirements of the system. Just as a chain is only as strong as its weakest link, a system is limited to the operating range of its narrowest subcomponent.

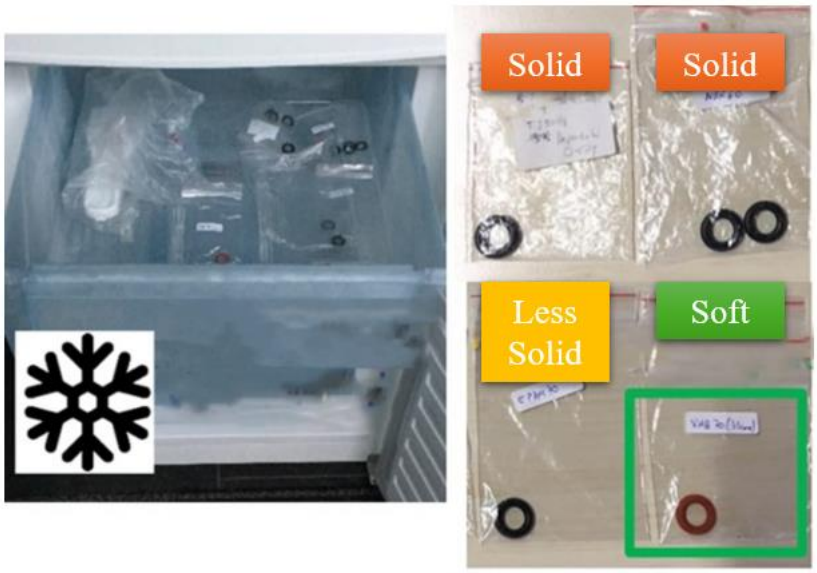



**Figure 7.19:** New O-Ring that maintains its flexibility under cold conditioning

Another issue arose concerning the swelling of the DC motor coil windings, leading to the motor's incapacity to endure prolonged operation, resulting in frequent malfunctions occurring within 25 hours of use. Upon discussions with the supplier regarding the pertinent DC motor, it was conveyed that the 'A-max 32 motor' possesses a permissible maximum speed of 6000 RPM. Exceedingly twice the specified speed threshold of 6000 RPM engenders excessive centrifugal force on the windings, causing them to expand outward. Consequently, the adhesive bonding securing the windings disintegrates under heightened pressure, leading to the windings rubbing against the motor's body, thereby causing malfunction.

Subsequent investigation identified two distinct discrepancies as the underlying causes of the error. Firstly, it was determined that the supply voltage of the E-Starter contributed to the issue, alongside the significant deviation from the product's maximum rotational speed limit in the current design.

| | | | with terminals<br>with cables |
|---|---|---|---|
| **Motor Data** | | | |
| | **Values at nominal voltage** | | |
| 1 | Nominal voltage | V | 6 |
| 2 | No load speed | rpm | 5870 |
| 3 | No load current | mA | 154 |
| 4 | Nominal speed | rpm | 4110 |
| 5 | Nominal torque (max. continuous torque) | mNm | 36.5 |
| 6 | Nominal current (max. continuous current) | A | 3.95 |
| 7 | Stall torque | mNm | 127 |
| 8 | Stall current | A | 13.2 |
| 9 | Max. efficiency | % | 78 |
| | **Characteristics** | | |
| 10 | Terminal resistance | Ω | 0.454 |
| 11 | Terminal inductance | mH | 0.066 |
| 12 | Torque constant | mNm/A | 9.58 |
| 13 | Speed constant | rpm/V | 996 |
| 14 | Speed / torque gradient | rpm/mNm | 47.2 |
| 15 | Mechanical time constant | ms | 21.9 |
| 16 | Rotor inertia | gcm² | 44.2 |

**Figure 7.20:** Data Sheet Information

The manufacturer's designated maximum rotational speed limit is observed to be 6000 RPM, as depicted below.

| | | |
|---|---|---|
| | **Mechanical data (ball bearings)** | |
| 23 | Max. speed | 6000 rpm |
| 24 | Axial play | 0.12 - 0.22 mm |
| 25 | Radial play | 0.025 mm |
| 26 | Max. axial load (dynamic) | 7.6 N |
| 27 | Max. force for press fits (static) | 110 N |
| 28 | Max. radial load, 5 mm from flange | 32 N |
| | **Mechanical data (sleeve bearings)** | |
| 23 | Max. speed | 6000 rpm |
| 24 | Axial play | 0.12 - 0.22 mm |
| 25 | Radial play | 0.012 mm |
| 26 | Max. axial load (dynamic) | 5.0 N |
| 27 | Max. force for press fits (static) | 110 N |
| 28 | Max. radial load, 5 mm from flange | 10.5 N |

**Figure 7.21:** Data Sheet Information

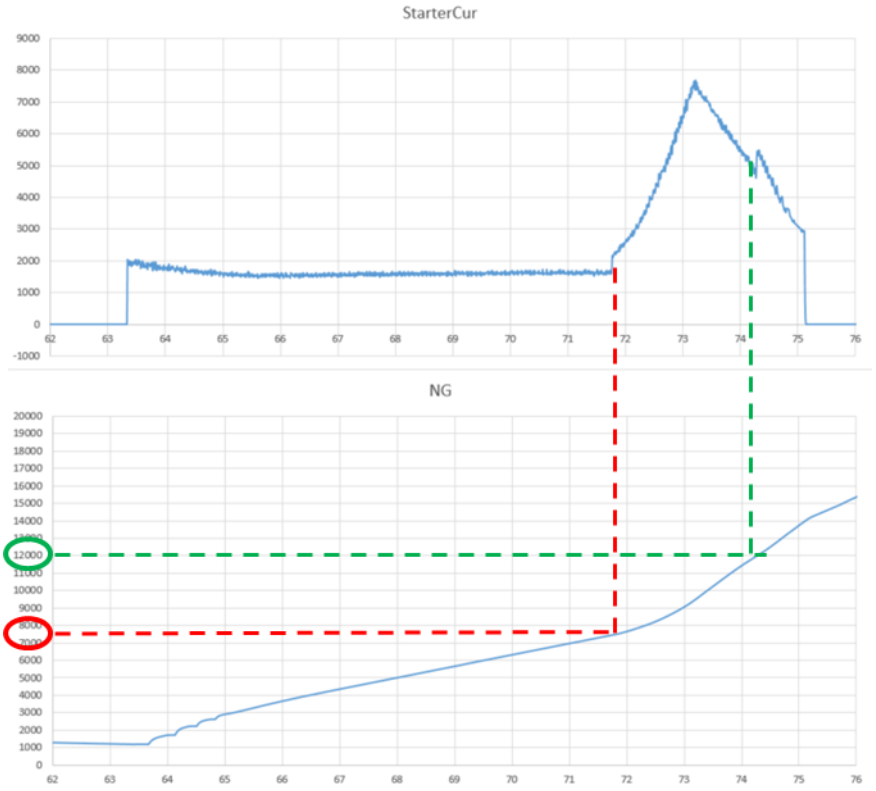



**Figure 7.22:** 7500 RPM to 12000 RPM

As depicted in the above figure, engagement initiates at approximately 7500 RPM. The DC motor persists in operation up to levels around 12000 RPM, precisely double the specified 6000 RPM speed limit that the windings can endure.

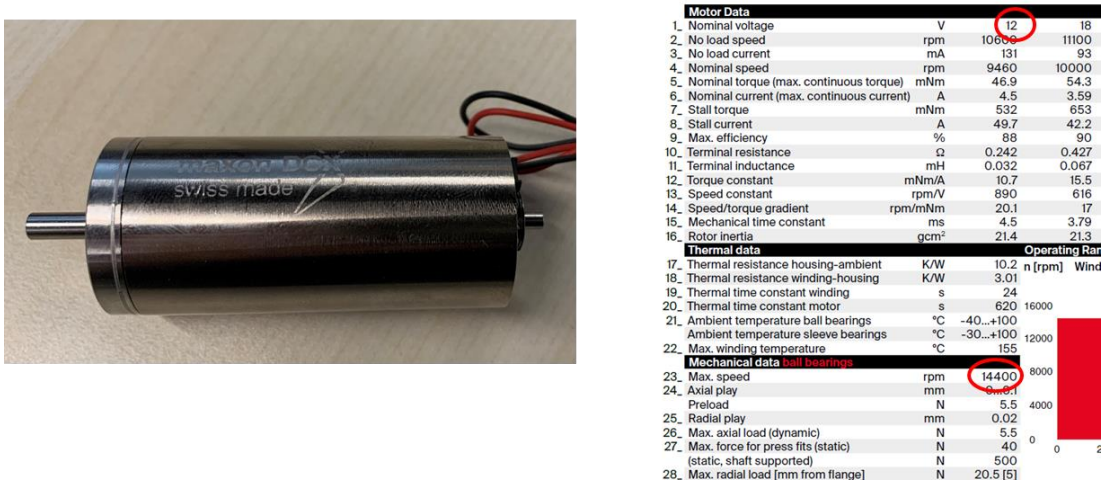



**Figure 7.23:** DC Motor and Data Sheet

The error was remedied by opting for an alternative DC motor that aligns with the specifications of the TJ90 turbojet engine. Now, let's demonstrate how this error could have been captured utilizing MBSE.

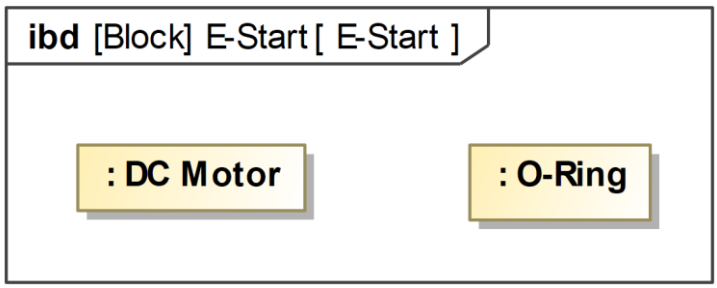



**Figure 7.24:** ibd [Block] E-start

The Internal Block Definition diagram can be used to illustrate the interactions and values between the subcomponents within the E-start block, which is defined in the Block Definition Diagram for the TJ90 turbojet engine. Similarly, on the BDD, these values related to the E-start can be shared. The team modeling the system can decide based on the design solution.

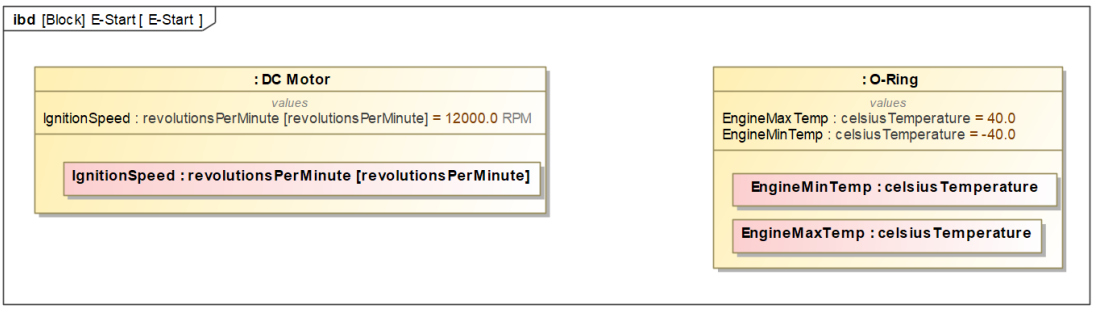
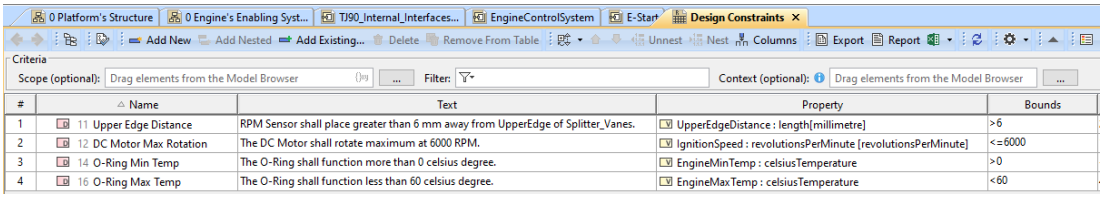



**Figure 7.25:** ibd[Block]E-Start

The operational range parameters for both the DC Motor and O-Ring were sourced from product catalogs and incorporated into the system model. As parameters like length, diameter, weight, and volume, retrievable from 3D CAD designs, were unavailable, a comparison was conducted between the values provided in the product catalog and those derived from the motor's performance cycle and analysis outcomes. Both the motor's performance cycle and test data from the preceding configuration confirm an ignition cycle of 12000 RPM. To ascertain whether the DC motor of the E-Start mechanism can attain these speeds, the maximum speed data for the DC motor was extracted from the product catalog.

The prescribed limit values are then integrated into the requirement table, positioned alongside design constraints, serving as system requirements.




| # | Name | Text | Property | Bounds |
|---|---|---|---|---|
| 1 | 11 Upper Edge Distance | RPM Sensor shall place greater than 6 mm away from UpperEdge of Splitter_Vanes. | UpperEdgeDistance : length[millimetre] | >6 |
| 2 | 12 DC Motor Max Rotation | The DC Motor shall rotate maximum at 6000 RPM. | IgnitionSpeed : revolutionsPerMinute [revolutionsPerMinute] | <=6000 |
| 3 | 14 O-Ring Min Temp | The O-Ring shall function more than 0 celsius degree. | EngineMinTemp : celsiusTemperature | >0 |
| 4 | 16 O-Ring Max Temp | The O-Ring shall function less than 60 celsius degree. | EngineMaxTemp : celsiusTemperature | <60 |

**Figure 7.26:** Requirement Table

Similar to the instance of the RPM Sensor error, a correlation is established between the pertinent parameter values and the design constraint requirements that these parameters must adhere to.

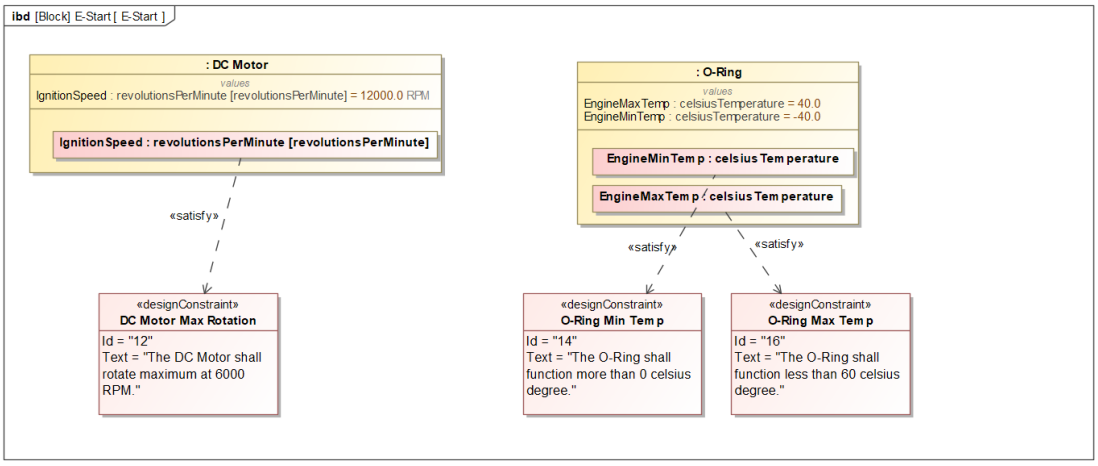
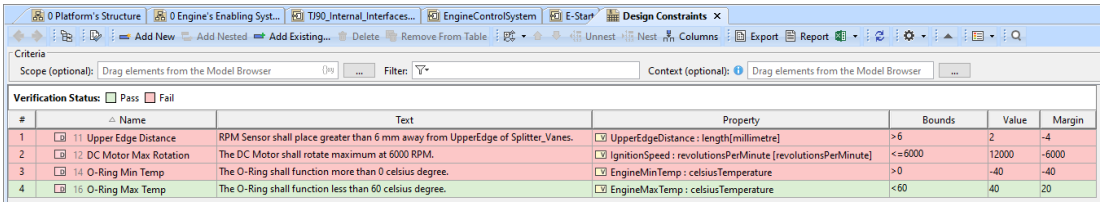



**Figure 7.27:** ibd[Block]E-Start

Consequently, during verification checks on the requirement table, criteria that are fulfilled are depicted in green, whereas criteria that are not satisfied are indicated in red.




| # | Name | Text | Property | Bounds | Value | Margin |
|---|---|---|---|---|---|---|
| 1 | 11 Upper Edge Distance | RPM Sensor shall place greater than 6 mm away from UpperEdge of Splitter_Vanes. | UpperEdgeDistance : length[millimetre] | >6 | 2 | -4 |
| 2 | 12 DC Motor Max Rotation | The DC Motor shall rotate maximum at 6000 RPM. | IgnitionSpeed : revolutionsPerMinute [revolutionsPerMinute] | <=6000 | 12000 | -6000 |
| 3 | 14 O-Ring Min Temp | The O-Ring shall function more than 0 celsius degree. | EngineMinTemp : celsiusTemperature | >0 | -40 | -40 |
| 4 | 16 O-Ring Max Temp | The O-Ring shall function less than 60 celsius degree. | EngineMaxTemp : celsiusTemperature | <60 | 40 | 20 |

**Figure 7.28:** Verification checks on the requirement table

From this table, several conclusions can be drawn: Firstly, the selected DC Motor may not have the capability to initiate the engine's ignition cycle. Even if it does, it might

operate at speeds significantly surpassing its intended capacity, posing a risk of potential damage. Secondly, the O-ring component, tasked with gripping the rotor, is composed of a material with a tolerance for temperatures up to 60°C, exceeding the engine's maximum operational temperature of 40°C by a considerable margin. However, there is no pre-existing product selection suitable for the motor to function in cold conditions. It is evident that the motor would fail to meet the minimum operating temperature of -40°C, starting from 0°C, and may consequently fail to initiate in cold environments. Hence, it is imperative to convey to the relevant mechanical design and materials departments that both the DC Motor and O-Ring, lacking suitable product selections, require replacement.

### 7.1.3. CASE-III: Compressor Blade Count Change Interface Error

In this case study, the issue at hand is an interface error resulting from a design change. Interfaces are the interaction points between different components of a system, and these interactions can be physical, functional, or electrical.

A functional interface needs to be defined between the Electronic Control Unit, which manages RPM measurement, and the Compressor. This is because the rotor's rotational speed is measured using the following formula:

$$RPM = \frac{Fr \,.\, 60}{Blade\ Number} \qquad (7.1)$$

In the TJ90 Turbojet engine, a design change increased the number of impeller (compressor part) blades from 6 to 7. During an Interface Evaluation Meeting held with the TJ90 Turbojet Engine Project Team and the Electrical, Electronic, Control, and Embedded Systems Teams, it was discovered that the software of the controller still reflected the blade count as 6.

The maximum speed of the TJ90 Turbojet engine is 96,000 RPM. Thus, the frequency value can be calculated as:

$$96000 = \frac{Fr \,.\, 60}{6} \qquad (7.2)$$

$$Fr = 9{,}600 \text{ Hz}$$

For the same frequency value, if the blade count was not updated in the software, when attempting to reach the maximum speed, the engine would achieve the following RPM:

$$RPM = \frac{9600 \,.\, 60}{7}$$
$$RPM = 82285 \; RPM \qquad (7.3)$$

Fortunately, even if this interface error was detected during testing, the engine speed would remain within the operational range, causing no damage. However, if the blade count had been reduced from 6 to 5 and this change was detected during testing, the engine, when attempting to reach its maximum speed of 96,000 RPM, would achieve the following RPM:

$$RPM = \frac{9600 \,.\, 60}{5}$$
$$RPM = 115200 \; RPM \qquad (7.4)$$

Turbojet engines must withstand up to 10% above their maximum speed, known as the redline speed. For the TJ90 Turbojet Engine, this speed is calculated as:

$$96000 \; RPM + 96000 \; RPM * \%10 = 105600 \; RPM \qquad (7.5)$$

As seen, if the blade count change was not managed through the interface and was detected during testing, the engine would exceed not only its maximum speed of 96,000 RPM but also its redline speed, leading to potential damage and loss of structural integrity.

To understand how this change can be analysed through impact analysis and how relevant stakeholders can be informed via the interface within the system model, we can explore the following steps:

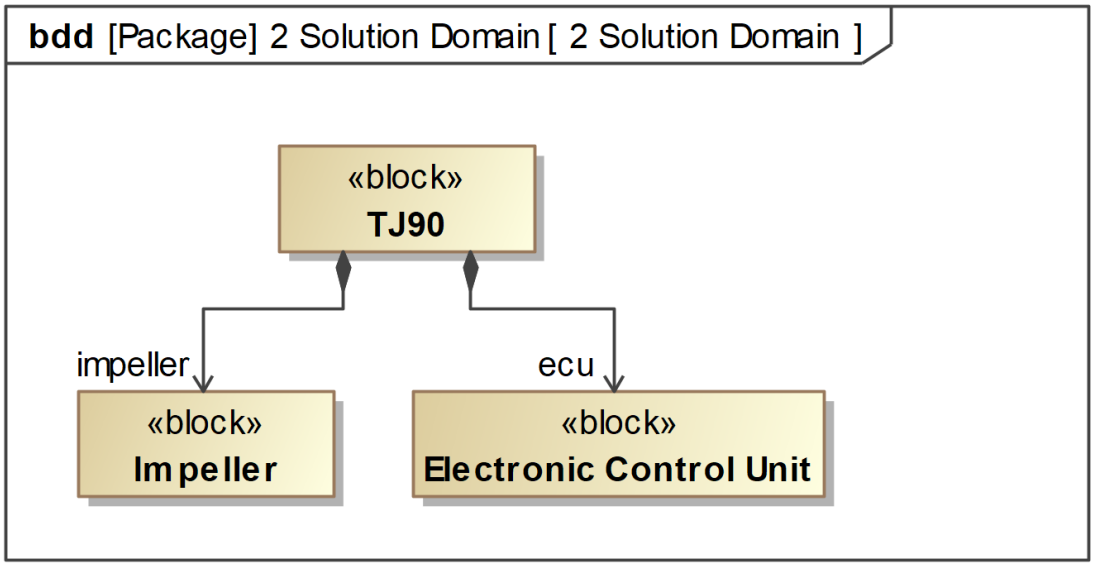



**Figure 7.29:** bdd[Package] 2 Solution Domain

In the system model, other system components have been hidden in the display to focus solely on the relevant section.

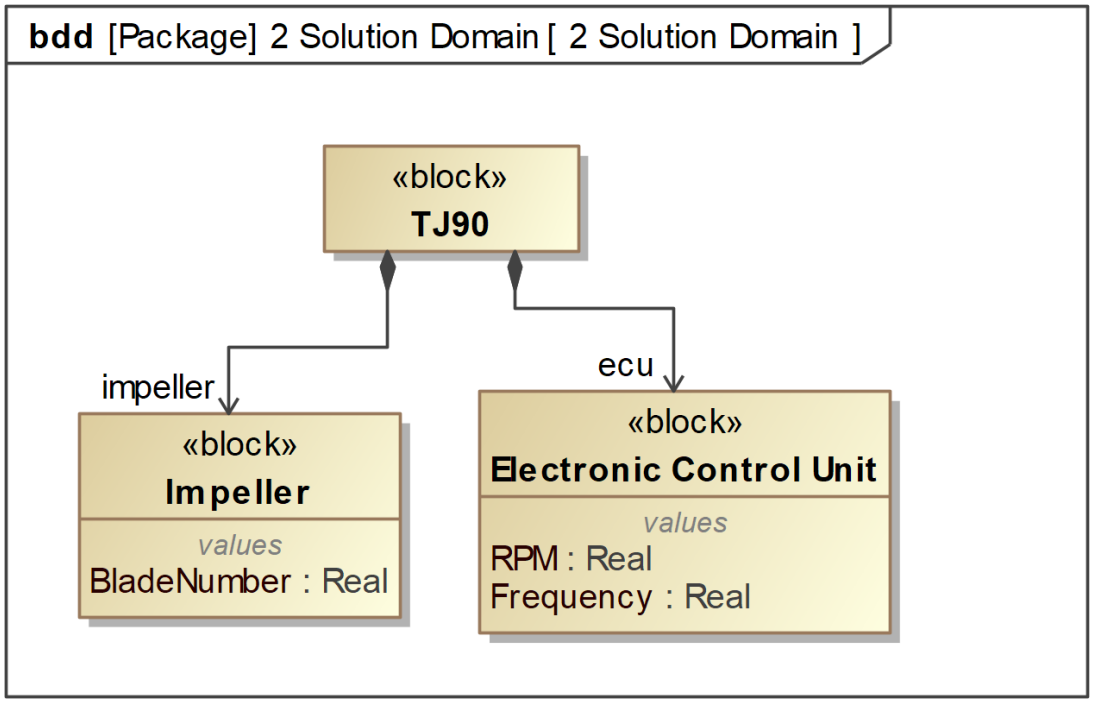



**Figure 7.30:** bdd[Package] 2 Solution Domain

Values and properties related to the relevant components for the case study are defined. As stated above, other values and properties of the components that are not pertinent to the case study have been removed from the display.

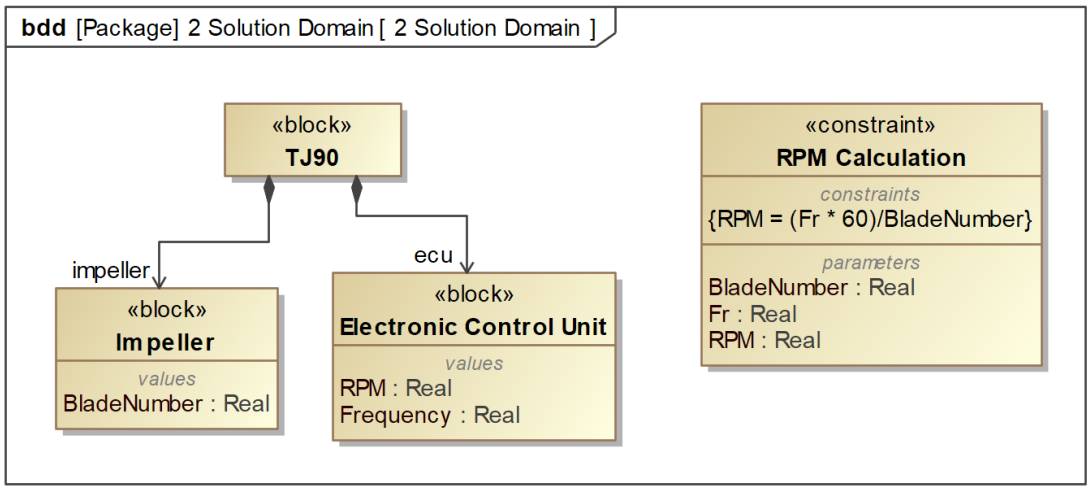



**Figure 7.31:** bdd[Package] 2 Solution Domain

The block where the calculation will be performed is defined as a constraint block. The inputs for the calculation can also be directly defined under the relevant blocks. The advantage of using a constraint block is that if the same calculation needs to be used in multiple places within the system model, it only needs to be defined once. Otherwise, the calculation would have to be individually defined under each required block every time.

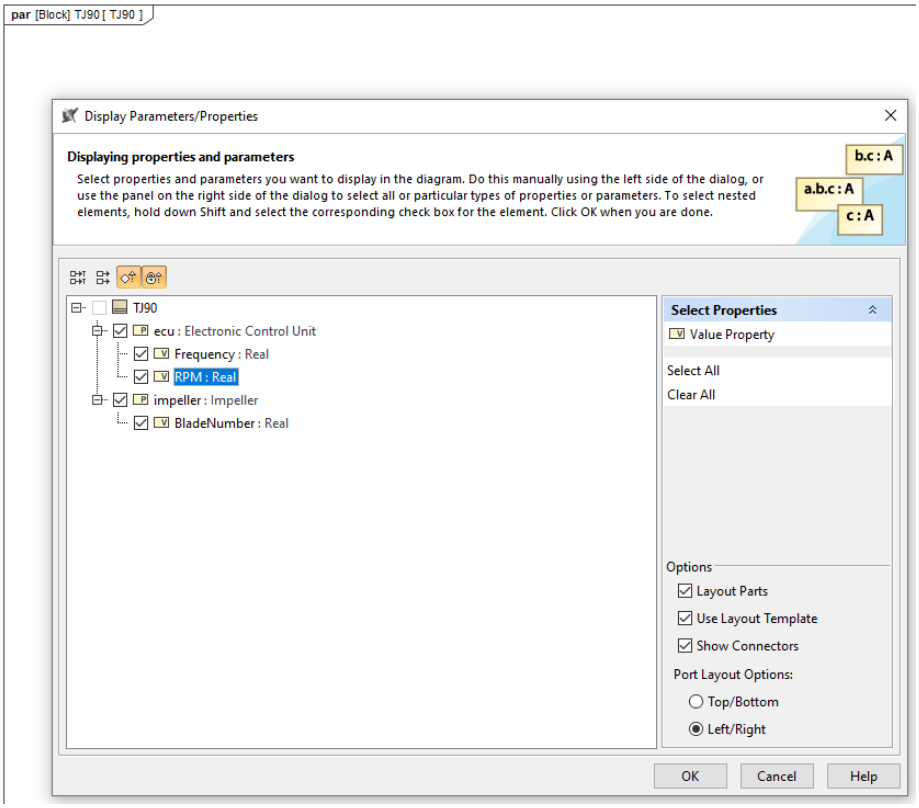



**Figure 7.32:** Parametric Diagram

A parametric diagram is designed for the TJ90 system block, and in the resulting window, all relevant parameters to be included in the diagram are selected. When adding the corresponding equation to the diagram, the program will ask which values should be matched with the parameters in the equation. This is necessary because the program does not automatically recognize parameters and their corresponding values even if they have identical names. For instance, if "Frequency" is shortened to "Fr" in the equation, the program will not inherently understand that "Frequency" corresponds to "Fr." It must be specified by the system modeler.

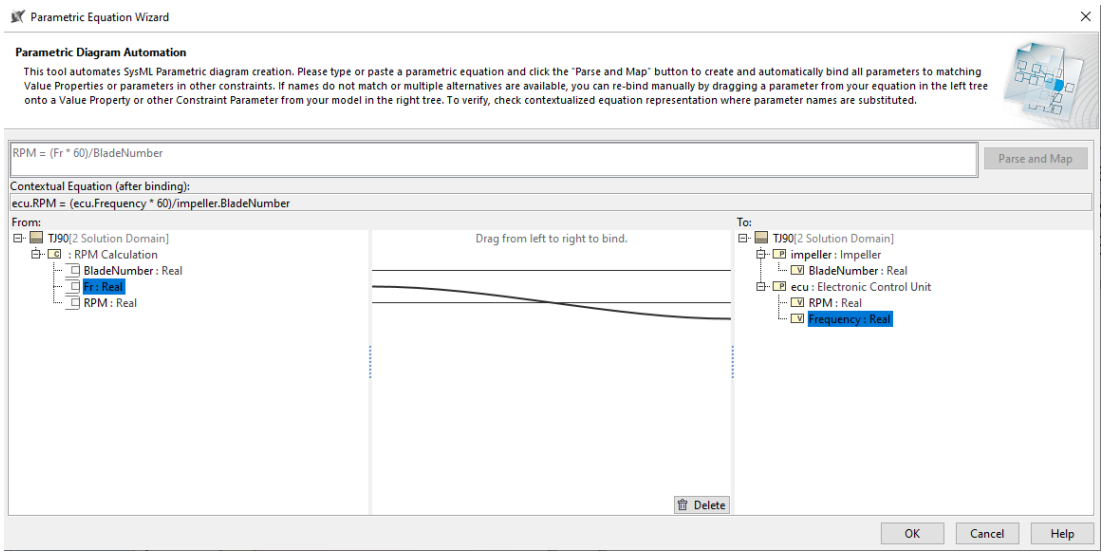



**Figure 7.33:** Parametric Diagram

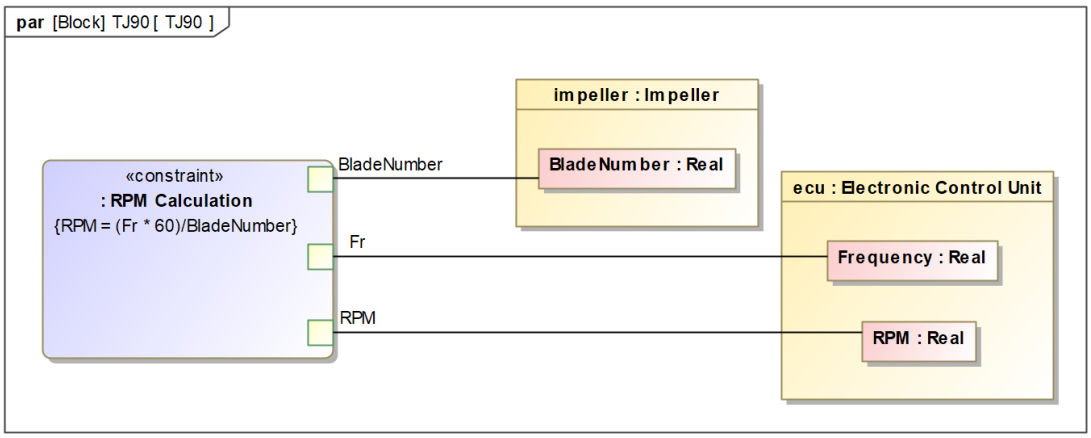



**Figure 7.34:** par[Block]TJ90

Once the matches are completed, the parametric model is automatically established by the program as shown above.

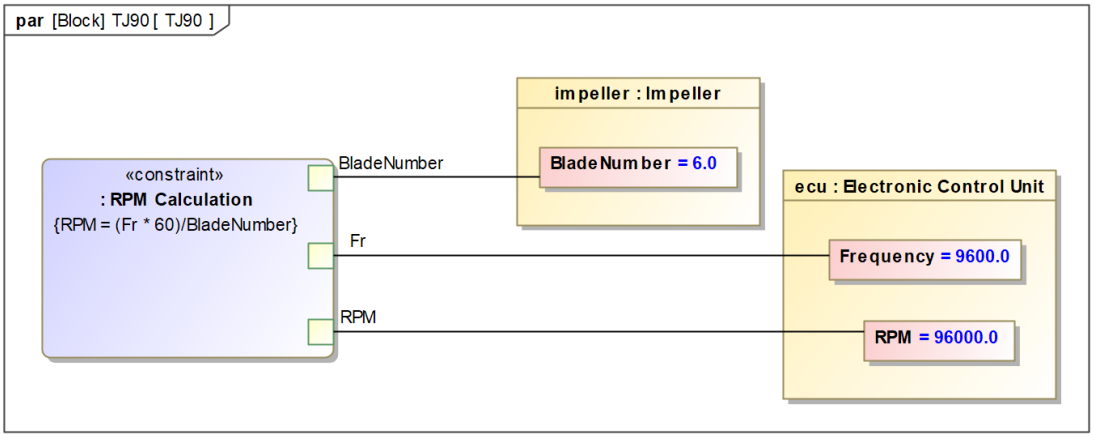



**Figure 7.35:** par[Block]TJ90

When the blade count and frequency values are entered, the RPM value is calculated by the parametric model through the equation.

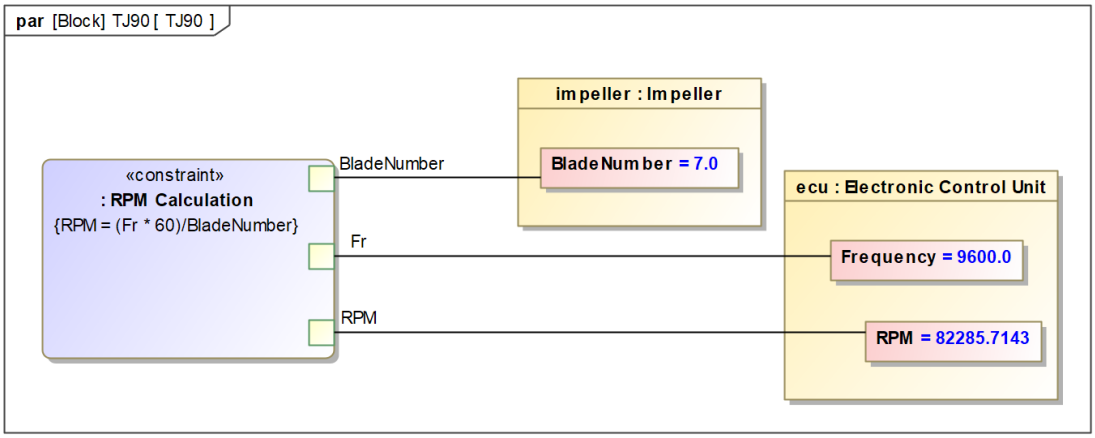



**Figure 7.36:** par[Block] TJ90

In the case study, the design change increased the splitter blade count of the impeller part from 6 to 7. When this value is entered into the model without adjusting the corresponding frequency value, the achievable speed of the motor drops to 82,000 RPM, as demonstrated in the manual calculation.

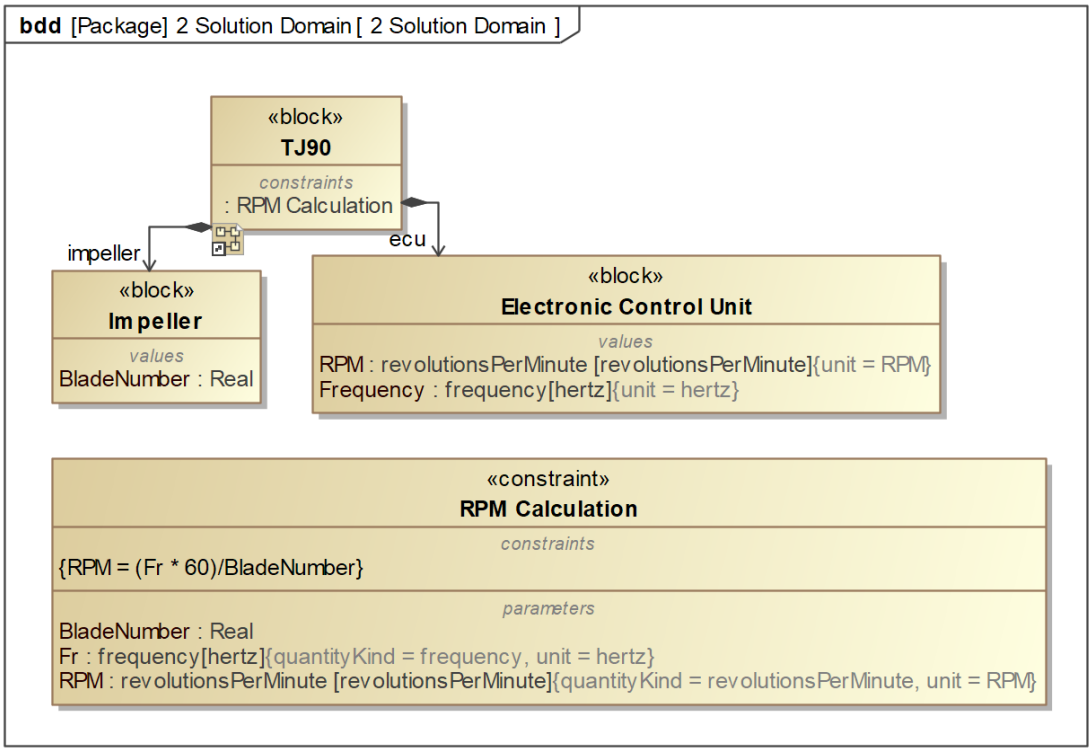



**Figure 7.37:** bdd[Package] 2 Solution Domain

In a real system model, to ensure the compatibility of work done by different disciplines, it is essential that units are accurately defined within the system model.

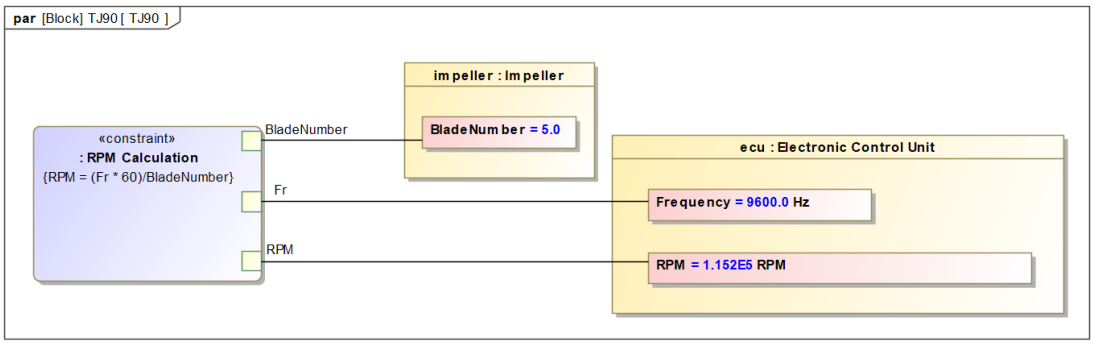



**Figure 7.38:** par[Block]TJ90

As demonstrated in the manual calculation, if the blade count had been reduced from 6 to 5 due to a design change, and this alteration had gone unnoticed by the Electronic Control Unit (ECU), the motor could have exceeded its maximum speed and redline speed limit during testing. Consequently, it could have exploded and disintegrated.

To ensure that the motor does not exceed its maximum speed value, the error mode should be identified for the system, and the effects of this error mode should be

analyzed. Necessary actions should then be taken during the design phase to prevent this from happening.

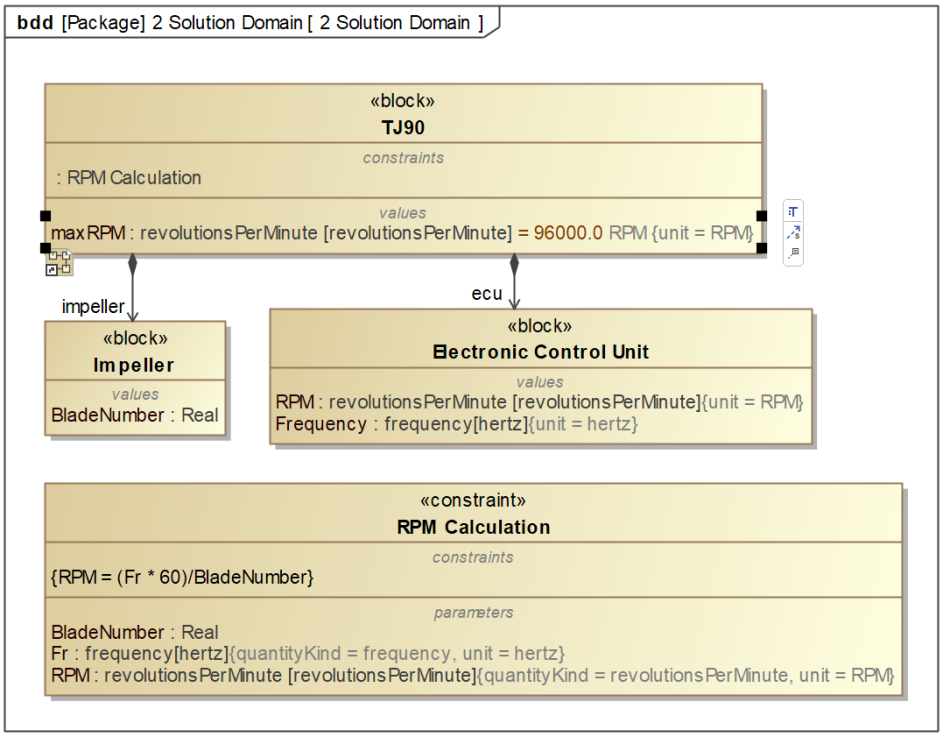



**Figure 7.39:** bdd[Package] 2 Solution Domain

Within the system model, the maximum speed limit for the TJ90 system can be defined, and the control of whether this limit is exceeded or not can be carried out through the parametric diagram.

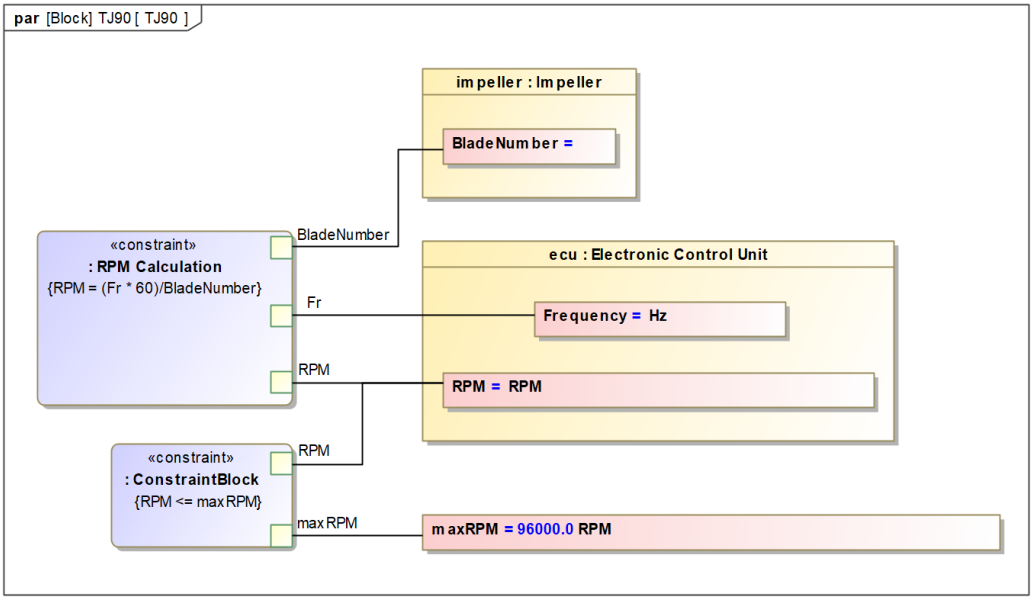



**Figure 7.40:** par[Block]TJ90

Another equation is defined regarding the necessity for the calculated speed value to always be less than or equal to the maximum speed for the maximum speed control.

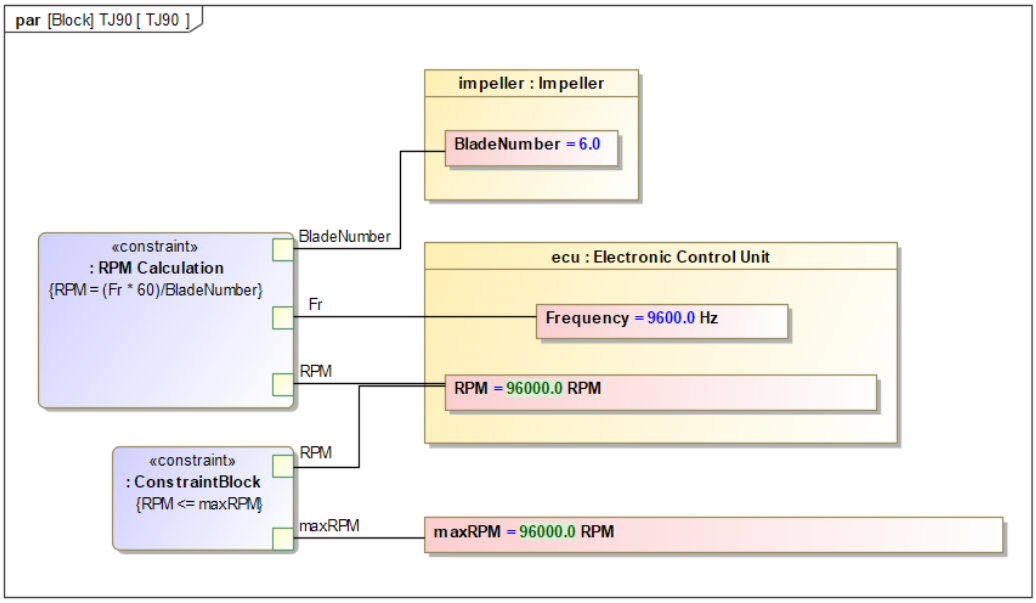



**Figure 7.41:** par[Block]TJ90

In the current design, with a blade count of 6, we observe that we are within the relevant limits for the maximum speed calculation.

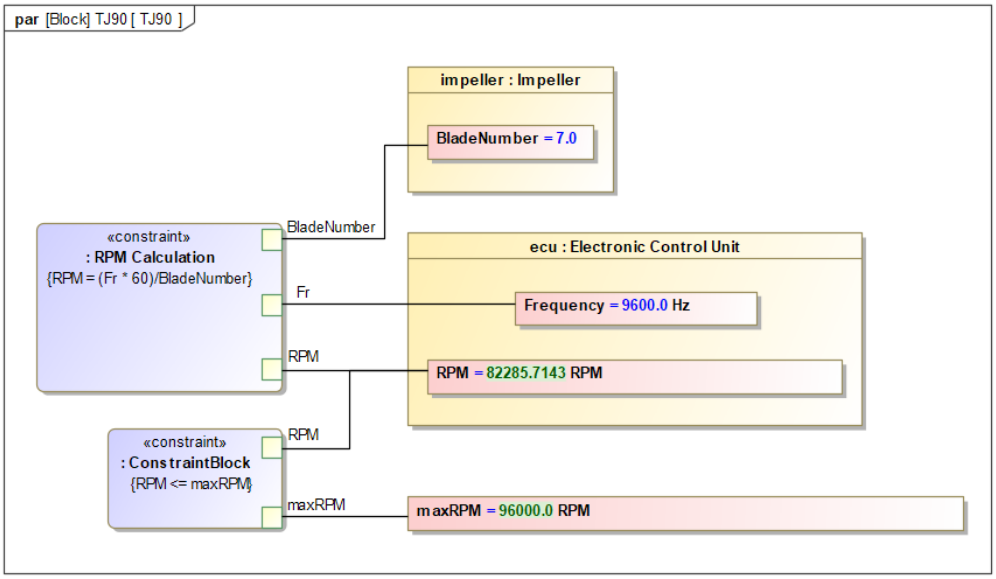



**Figure 7.42:** par[Block]TJ90

In the event that the change in blade count is not communicated to the interface stakeholder as part of interface management, we see that despite the increase in blade count, we are still within the operating range by chance. In this scenario, as mentioned earlier, if the motor were to be pushed to the maximum speed of 96,000 RPM during testing, it would only reach a value of 82,000 RPM.

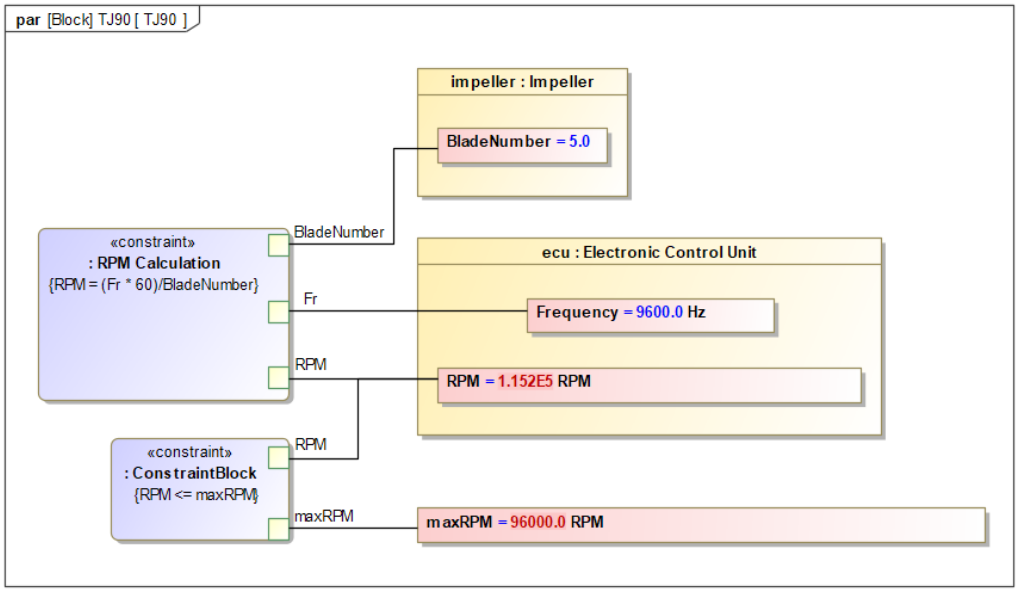



**Figure 7.43:** par[Block]TJ90

In the event of a decrease in blade count due to design changes, the motor exceeds its maximum speed value. Through this parametric model, the system model alerts us with a red indication, indicating that the value cannot be met.

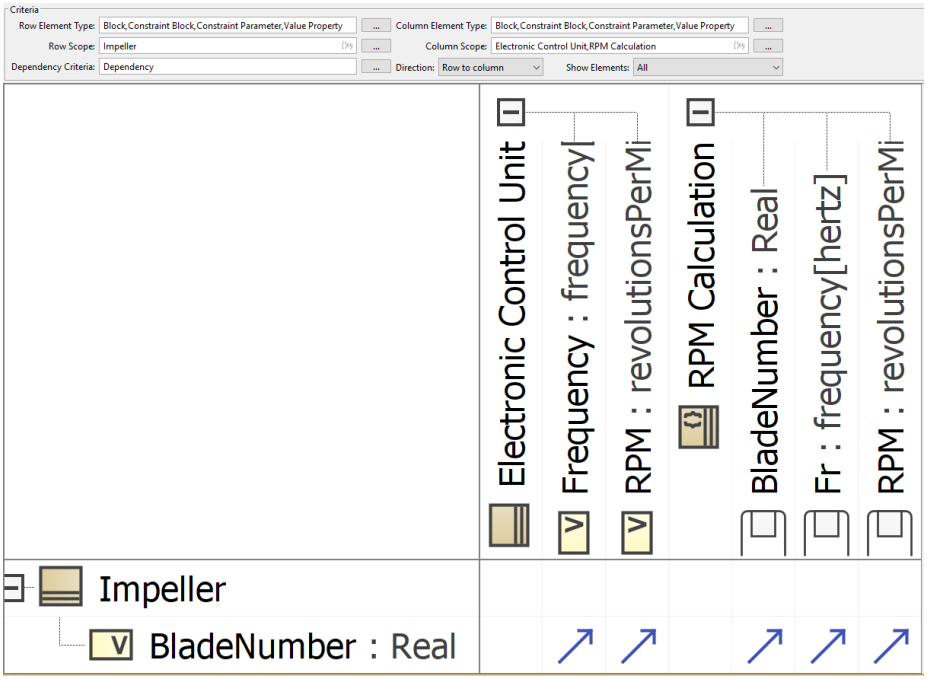



**Figure 7.44:** Dependency matrice

Simultaneously, associating parameters that are interrelated within the system through dependency matrices makes subsequent impact analyses more manageable. Thanks to such matrices in the system model, any change made to the system is not overlooked, and its effects on the rest of the system can be readily captured.

### 7.1.4. CASE-IV: System Total Weight Exceedance

System requirements are linked parent-child relationship, where the System Requirement represents the parent part, and the requirements assigned and defined for subsystems represent the child requirements.

For instance, there might be a requirement related to the total weight of the system. All subsystems of the system must adhere to this requirement. Therefore, weight requirements are allocated to subsystems to fulfil the total weight requirement. If subsystems meet their weight requirements, the system will meet its weight requirement. However, this ideal scenario may not always hold true. As the design progresses, weight targets set at the beginning may become unattainable for some

subsystems. In such cases, trade-offs need to be made among subsystems to maintain the total weight of the system. By transferring weight from subsystems exceeding their targets to those meeting or even being below their targets, the system's total weight target can still be achieved.

Failure to monitor this connection from a system perspective can lead to inconsistencies, as the total system weight may exceed the sum of its subsystems` weight. Therefore, when such inconsistency arises, it is crucial to capture it within a system model using MBSE methodology.

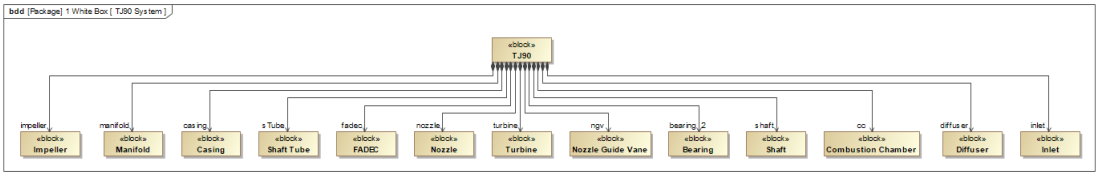



**Figure 7.45:** bdd[Package]1White Box [TJ90 System]

The relevant system is structurally defined along with all its subsystems. Here, the dry weight of the system is calculated, without considering fuel. Additionally, the weights of fasteners such as nuts, bolts, and instrumentation parts such as sensors in the engine are neglected as they are expected to be minimal.

Add New | Add Nested | Add Existing... | Delete | Remove From Table

Criteria

Scope (optional): Drag elements from the Model Browser ... Filter:

| # | Name | Text |
|---|---|---|
| 1 | 17 Total Mass | The engine's total dry mass shall be less than 5000 g. |
| 2 | 17.7 Bearing Mass | The Bearing mass shall be less than 75 g. |
| 3 | 17.13 Casing Mass | The Casing mass shall be less than 250 g. |
| 4 | 17.4 Combustion Chamber Mass | The Combustion Chamber mass shall be less than 450 g. |
| 5 | 17.3 Diffuser Mass | The Diffuser mass shall be less than 300 g. |
| 6 | 17.11 FADEC Mass | The FADEC mass shall be less than 1000 g. |
| 7 | 17.2 Impeller Mass | The Impeller mass shall be less than 200 g. |
| 8 | 17.1 Inlet Mass | The Inlet mass shall be less than 250 g. |
| 9 | 17.12 Manifold Mass | The Manifold mass shall be less than 150 g. |
| 10 | 17.8 Nozzle Guide Vane Mass | The Nozzle Guide Vane mass shall be less than 550 g. |
| 11 | 17.10 Nozzle Mass | The Nozzle mass shall be less than 350 g. |
| 12 | 17.5 Shaft Mass | The Shaft mass shall be less than 650 g. |
| 13 | 17.6 Shaft Tube Mass | The Shaft Tube mass shall be less than 400 g. |
| 14 | 17.9 Turbine Mass | The Turbine mass shall be less than 300 g. |

**Figure 7.46:** Weight-related requirements table

Weight-related requirements are defined within the requirement table in the system model. (The given values are arbitrary and do not reflect reality. They are provided for illustrative purposes only.)

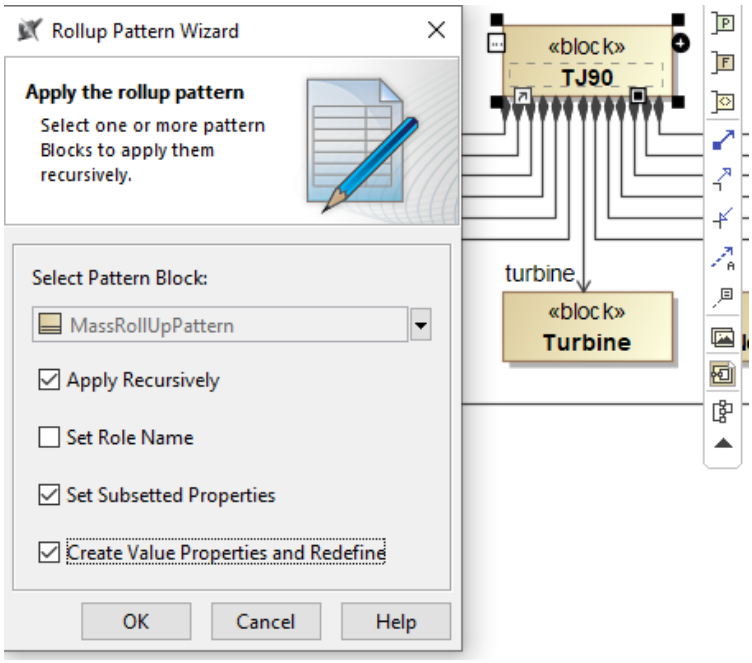



**Figure 7.47:** Rollup Pattern Wizard - The MassRollUpPattern is applied.

To calculate the total weight of the system, the TJ90 system block is selected, and the MassRollUpPattern is applied. The option "create value properties and redefine" is also selected to automatically assign weight values to the blocks.

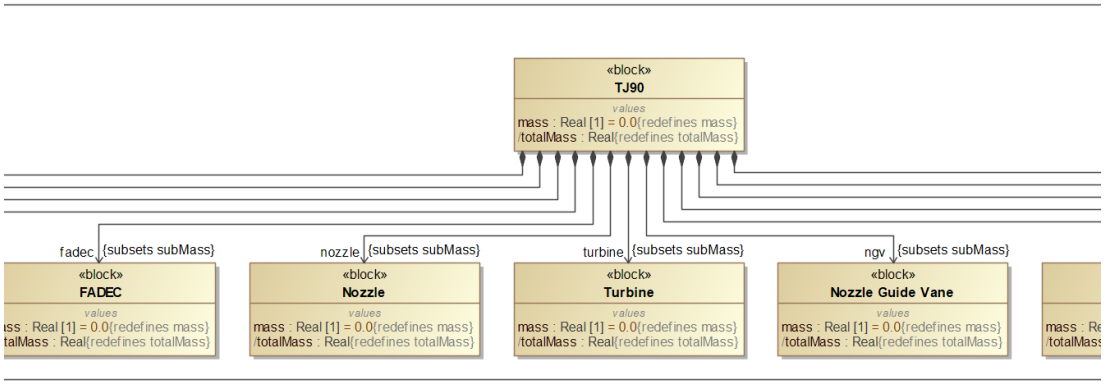



**Figure 7.48:** bdd[Package]1White Box [TJ90 System]

One of the key issues in consistency analysis is ensuring that the units used by different disciplines are compatible with each other. For instance, if one discipline uses meters while another uses inches for the same calculation, it will introduce inconsistency in

the system. In this example, it is preferred to use grams, which are defined in the requirements and accepted throughout the system, instead of the Real type automatically assigned by the program.

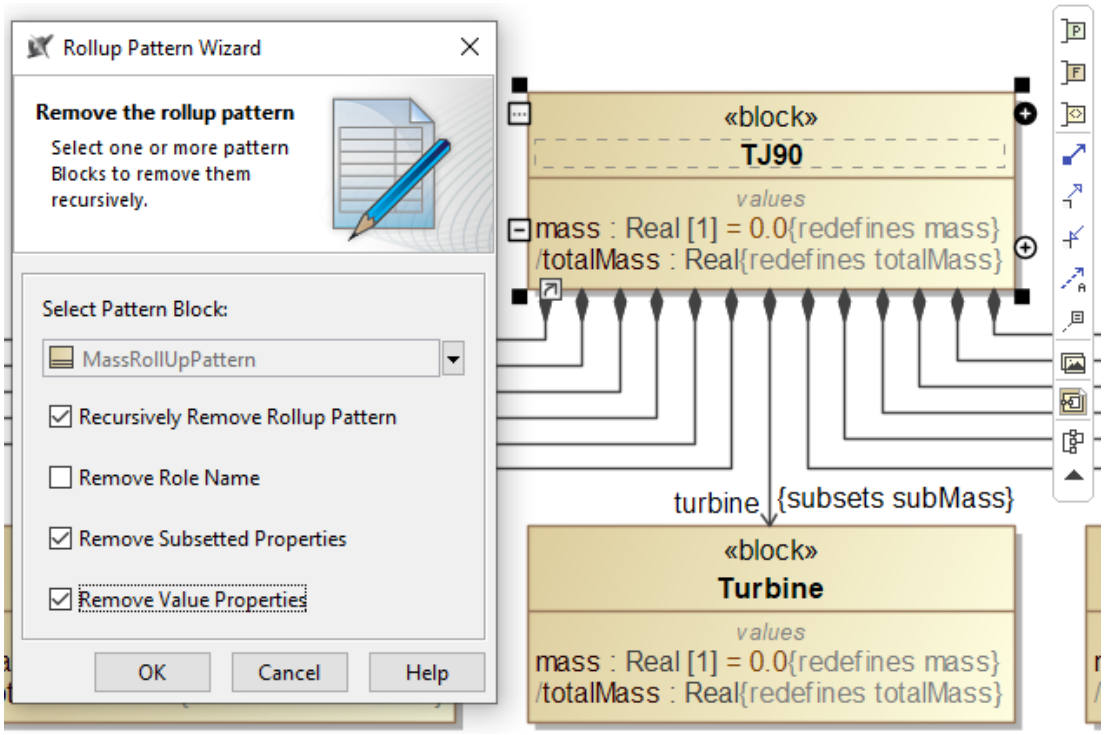



**Figure 7.49:** The assigned MassRollUpPattern is removed.

Firstly, the assigned MassRollUpPattern is removed.

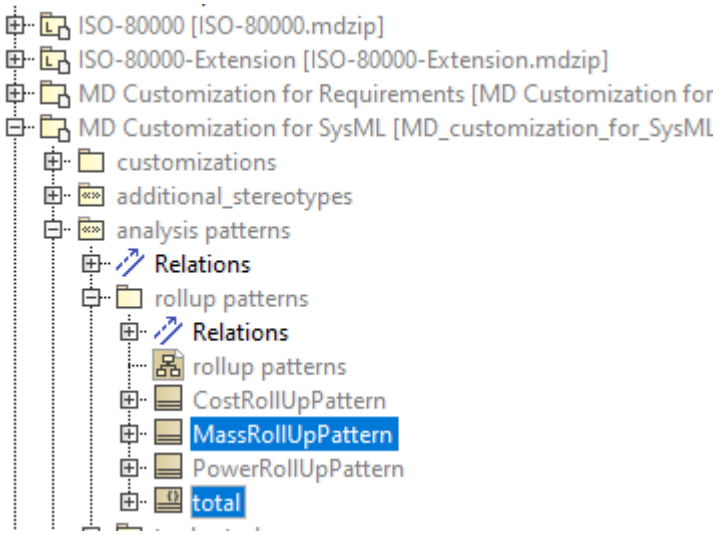



**Figure 7.50:** MassRollUpPattern in the containment tree

The search bar within the tool is used to locate the massrolluppattern block defined in the tool's own library. Subsequently, the total block is copied and pasted into the relevant folder within the system model.

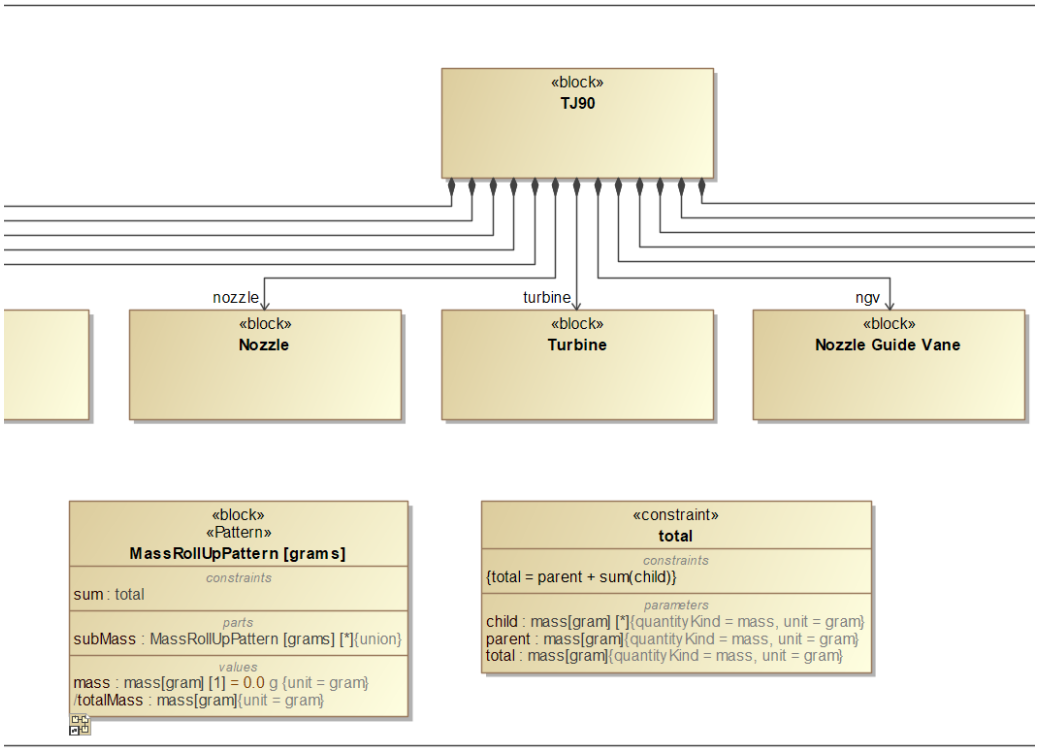



**Figure 7.51:** bdd[Package]1White Box [TJ90 System]

The name of the massrolluppattern block moved under the system model tree is changed. This is done to ensure that the properties of the default block in the library are not altered. Both the massrolluppattern and total blocks are dragged into the BDD (Block Definition Diagram) where the TJ90 system block is located, and the types of values in the compartments are changed to mass [gram].

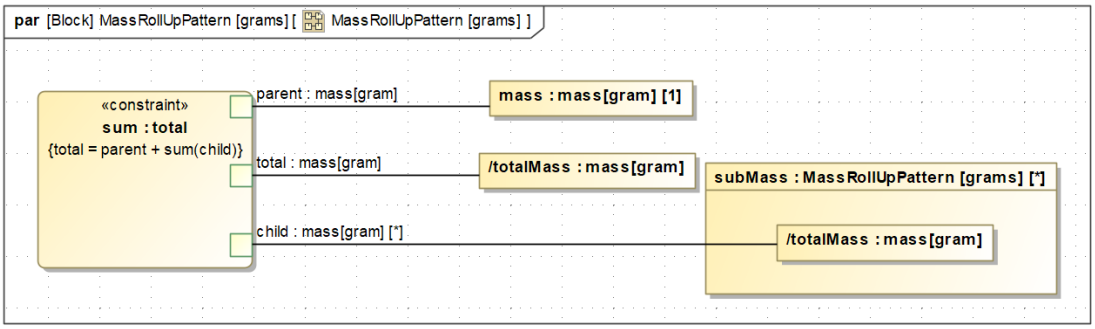



**Figure 7.52:** par[Block] MassRollUpPattern[grams]

Under the MassRollUpPattern block, a parametric diagram for weight calculation is automatically generated by the tool.

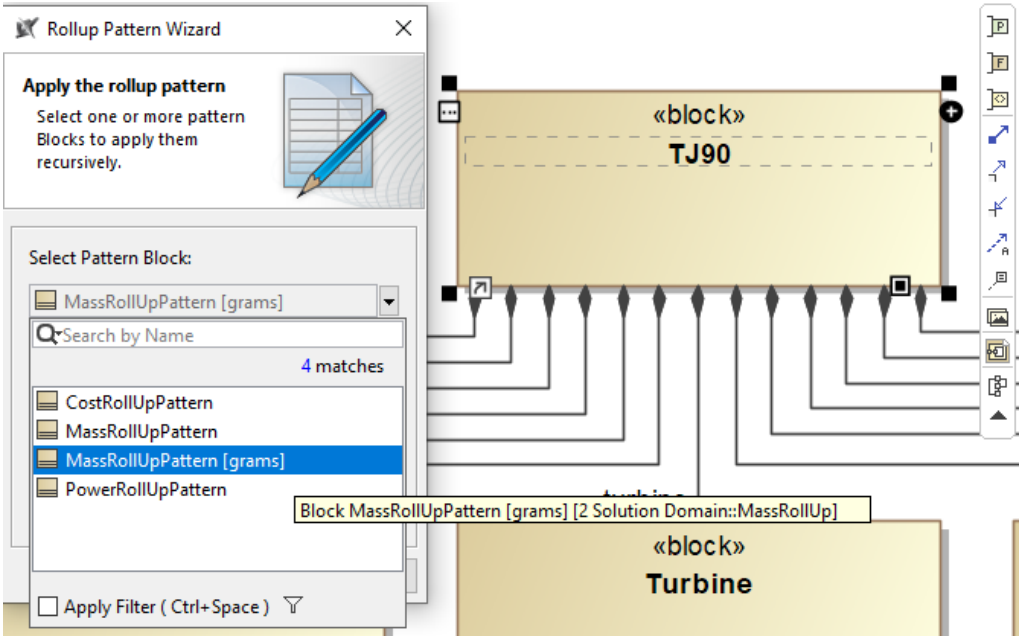



**Figure 7.53:** Rollup Pattern Wizard

The MassRollUpPattern is applied again to the TJ90 system block. However, this time, instead of the default block defined in the library, a newly defined block that collects weights in grams is assigned.

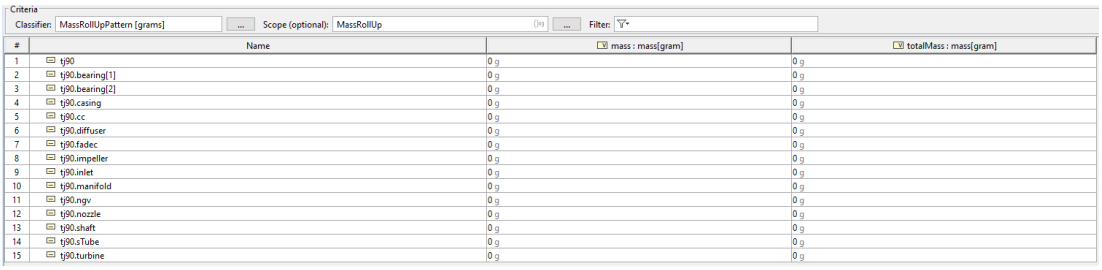



**Figure 7.54:** An instance table

Once simulation is initiated for the TJ90 system block and an instance table is created, after entering the necessary inputs, the weight table shown above can be obtained.

| # | Name | mass : mass[gram] | totalMass : mass[gram] |
|---|---|---|---|
| 1 | tj90 | 0 g | 0 g |
| 2 | tj90.impeller | 0 g | 0 g |
| 3 | tj90.impeller.inlet | 0 g | 0 g |
| 4 | tj90.impeller.diffuser | 0 g | 0 g |
| 5 | tj90.cc | 0 g | 0 g |
| 6 | tj90.cc.casing | 0 g | 0 g |
| 7 | tj90.cc.manifold | 0 g | 0 g |
| 8 | tj90.shaft | 0 g | 0 g |
| 9 | tj90.shaft.sTube | 0 g | 0 g |
| 10 | tj90.shaft.bearing[1] | 0 g | 0 g |
| 11 | tj90.shaft.bearing[2] | 0 g | 0 g |
| 12 | tj90.turbine | 0 g | 0 g |
| 13 | tj90.turbine.ngv | 0 g | 0 g |
| 14 | tj90.turbine.nozzle | 0 g | 0 g |
| 15 | tj90.fadec | 0 g | 0 g |

**Figure 7.55:** The system hierarchy is applied.

The system hierarchy can also be structured according to the BOM (Bill of Material) and PBS (Product Breakdown Structure). For instance, if the shaft were considered as a subsystem, with bearings and the shaft tube part included within it, and the same approach was applied to the compressor, combustion chamber, and turbine modules, the hierarchy would be modified as illustrated above.

However, when transforming the hierarchy in this manner, it is essential to adjust the assigned weights in the weight requirements and the parent-child relationships among the requirements accordingly. In this hierarchy, for instance, the impeller ceases to be an individual component and instead represents a subsystem, with its weight being the sum of its subcomponents, the inlet and diffuser. In other words, if we define the subsystem as the compressor module, a more encompassing term, and include the impeller part within this module, the calculation will still function correctly. For now, let us revert all the subcomponents of the TJ90 system to the same level, consistent with how we have defined the weight requirements, and proceed with the weight assignments.

Criteria

Classifier: MassRollUpPattern [grams] ... Scope (optional): MassRollUp

| # | Name | mass : mass[gram] | totalMass : mass[gram] |
|---|---|---|---|
| 1 | tj90 | 0 g | 4994 g |
| 2 | tj90.impeller | 172 g | 172 g |
| 3 | tj90.cc | 424 g | 424 g |
| 4 | tj90.shaft | 575 g | 575 g |
| 5 | tj90.turbine | 250 g | 250 g |
| 6 | tj90.fadec | 1300 g | 1300 g |
| 7 | tj90.nozzle | 314 g | 314 g |
| 8 | tj90.nozzle Guide Vane | 525 g | 525 g |
| 9 | tj90.inlet | 223 g | 223 g |
| 10 | tj90.diffuser | 340 g | 340 g |
| 11 | tj90.shaft Tube | 422 g | 422 g |
| 12 | tj90.bearing | 72 g | 72 g |
| 13 | tj90.casing | 230 g | 230 g |
| 14 | tj90.manifold | 147 g | 147 g |

**Figure 7.56:** Proceeding with weight assignments

The weight assignments for each component can be calculated in the design phase based on the material information assigned in the CAD environment. Once all weights are entered, although some components may fail to meet the weight requirements at the subsystem level, the overall system weight target of 5000 grams is achieved due to the components that fall below the weight requirement.

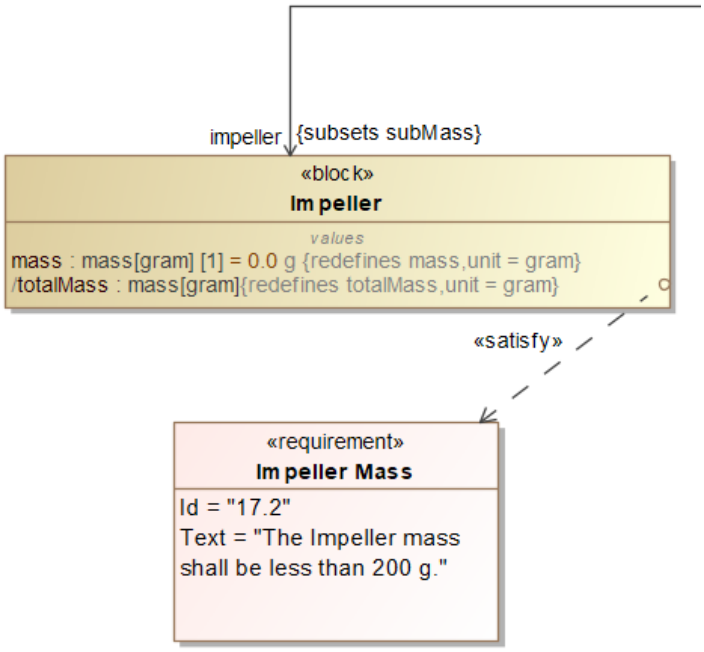



**Figure 7.57:** Weight Requirement

When the relevant value of each component is linked to its corresponding requirement, the instance table will display which requirements are met and which are not.

Criteria

Classifier: MassRollUpPattern [grams] ... Scope (optional): MassRollU

**Verification Status:** Pass Fail

| # | Name | totalMass : mass[gram] |
|---|---|---|
| 1 | tj90 | 4994 g |
| 2 | tj90.impeller | 172 g |
| 3 | tj90.cc | 424 g |
| 4 | tj90.shaft | 575 g |
| 5 | tj90.turbine | 250 g |
| 6 | tj90.fadec | 1300 g |
| 7 | tj90.nozzle | 314 g |
| 8 | tj90.nozzle Guide Vane | 525 g |
| 9 | tj90.inlet | 223 g |
| 10 | tj90.diffuser | 340 g |
| 11 | tj90.shaft Tube | 422 g |
| 12 | tj90.bearing | 72 g |
| 13 | tj90.casing | 230 g |
| 14 | tj90.manifold | 147 g |

**Figure 7.58:** Verification Chart

As observed, despite the FADEC, diffuser, and shaft tube components exceeding their weight targets, the overall system weight target is still met because the other components meet their weight targets and fall significantly below them.

Criteria

Classifier: MassRollUpPattern [grams] ... Scope (optional): MassRollU

**Verification Status:** Pass Fail

| # | Name | mass : mass[gram] | totalMass : ma |
|---|---|---|---|
| 1 | tj90 | 0 g | 5055 g |
| 2 | tj90.impeller | 172 g | 172 g |
| 3 | tj90.cc | 424 g | 424 g |
| 4 | tj90.shaft | 575 g | 575 g |
| 5 | tj90.turbine | 250 g | 250 g |
| 6 | tj90.fadec | 1300 g | 1300 g |
| 7 | tj90.nozzle | 375 g | 375 g |
| 8 | tj90.nozzle Guide Vane | 525 g | 525 g |
| 9 | tj90.inlet | 223 g | 223 g |
| 10 | tj90.diffuser | 340 g | 340 g |
| 11 | tj90.shaft Tube | 422 g | 422 g |
| 12 | tj90.bearing | 72 g | 72 g |
| 13 | tj90.casing | 230 g | 230 g |
| 14 | tj90.manifold | 147 g | 147 g |

**Figure 7.59:** Verification Chart

For instance, if the nozzle component also exceeds its weight target, resulting in a total weight of 5055 grams, surpassing the 5000 gram limit, inconsistency arises within the

system. The combined weight of the system's subcomponents exceeds the total system weight. This inconsistency can be easily detected through the system model.

To simplify the example, the two bearings within the system were considered as a single component. It should be remembered that subcomponents present in multiple quantities, such as bearings, must be accounted for individually.

Additionally, by defining a weight limit within the parametric diagram, the system's total weight target exceeding could be observed, similar to the RPM limit example mentioned earlier.

During system modeling, it is the responsibility of the modelers to determine which types of diagrams, elements, matrices, and tables will be used and for what purposes.

## 7.2. Advantages and Disadvantages

The transition from old document-centric procedures to model-based approaches in systems engineering is underway. Model Based Systems Engineering (MBSE) is becoming more popular in both business and government, and it continues to be a primary research priority in the systems engineering field [35]. As a matter of fact, MBSE was established by the International Council on Systems Engineering (INCOSE) as a key aspect of its systems engineering vision for 2020 [38] and remains so for 2025 [39]. And its 2035 vision (Figure 7.60).

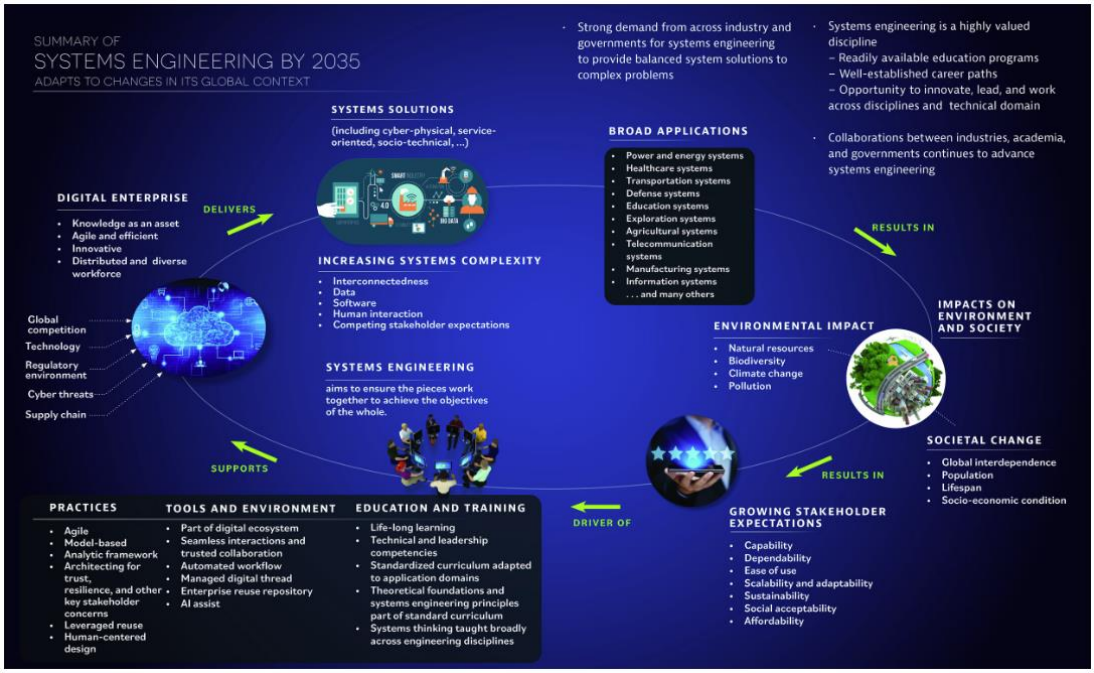



**Figure 7.60:** Summary of Sytem Engineering By 2035[36]

Given that the successful adoption of MBSE necessitates evidence of its value, it can be inferred that the strong interest in MBSE is driven by its advantages over document-based methods. However, a preliminary review of existing literature suggests a possible gap in comprehending the specific benefits of MBSE. This raises the question of whether MBSE truly offers advantages compared to a document-based approach [37].

The genesis of this study was to conduct a real-time case study to help decide whether or not to support the change process, investment, training, and tools required to implement an MBSE approach for a complex machine such as a turbine engine, as well as to raise awareness about the difficulties and benefits of this process. Additionally, it aimed to explain the context of using the framework for gas turbine engine design through a practical example.

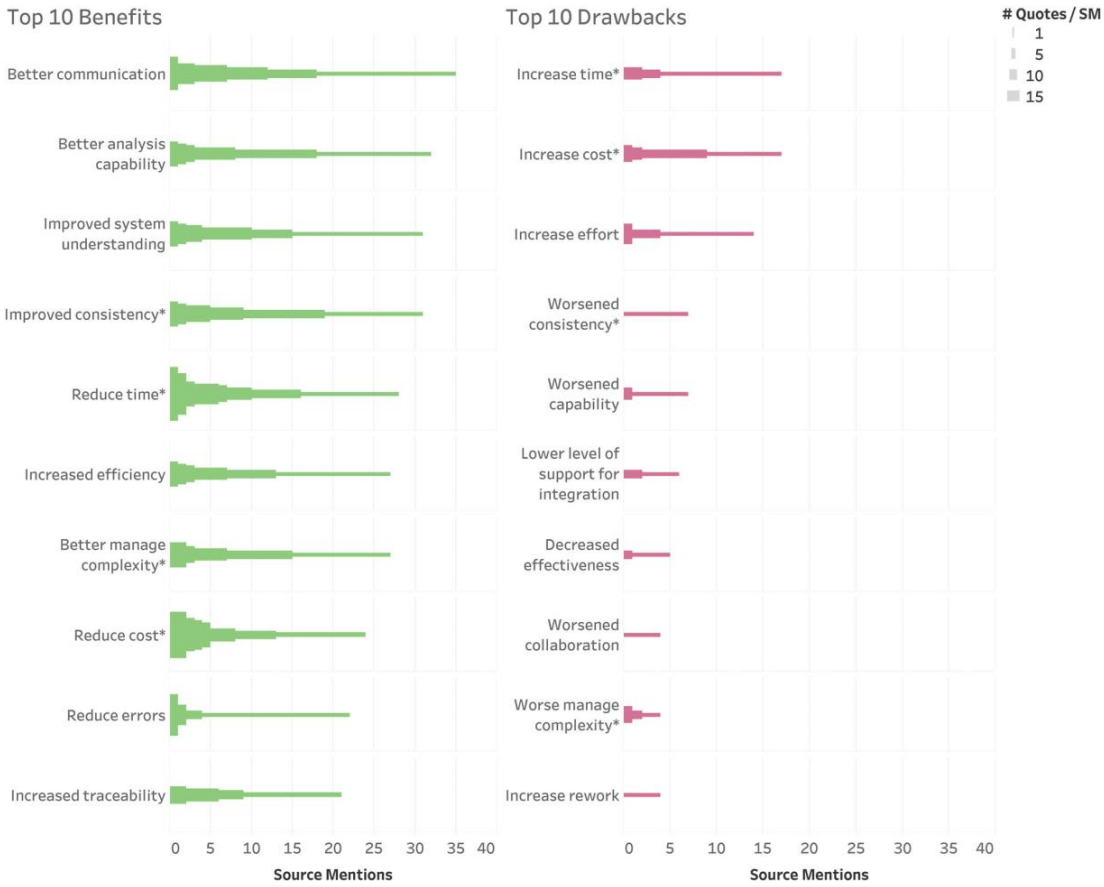



**Figure 7.61:** First 10 benefits (left) and 10 drawbacks (right) ordered by source mentions[38].

Figure 7.61 illustrates the most frequently mentioned benefits and drawbacks, with the height of the bars indicating how many times each was cited by individual sources. Benefits like reduced time are, on average, cited more positively and more often than comparable drawbacks such as increased time. This pattern is observed both across different sources and within individual sources, highlighting a tendency in the literature to emphasize positive attributes and benefits more than negative aspects and drawbacks [38].

Figure 7.62 depicts a percentage distribution chart comparing the data used to support both the advantages and downsides, as well as the positive and bad aspects of MBSE. Although the number of identified advantages much outnumbers the number of downsides, it is worth noting that benefits are supported by Author Opinion/Experience nearly twice as frequently as Case Studies. In contrast, both Author Opinion/Experience and Case Studies support downsides equally. This disparity suggests that the higher count of benefits might stem from authors' tendency to highlight the positive aspects of MBSE based on their perceptions and experiences,

while being more cautious about mentioning drawbacks unless backed by case study evidence [38].

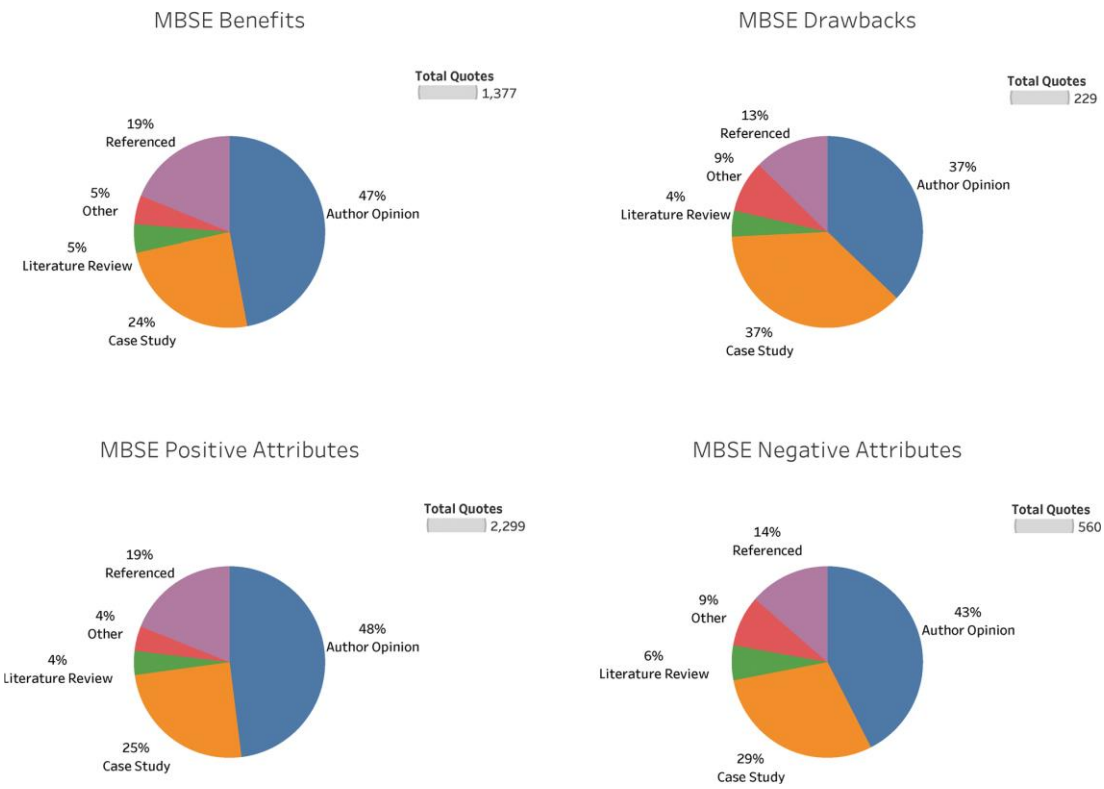



**Figure 7.62:** MBSE perceived benefits (top left), drawbacks (top right), positive qualities (bottom left), and negative traits (bottom right) ordered by evidence type [38].

## 7.2.1. Advantages

The MBSE community is increasingly recognizing the potential to quantify measures related with Model Based Systems Engineering (MBSE). However, practical implementation, lessons learned, and experience in this area are still at early stages. A thorough framework for MBSE measurements has been built following an intensive assessment of the literature and survey data, which includes essential aspects such as quality, velocity/agility, user experience, adoption, and knowledge transfer.

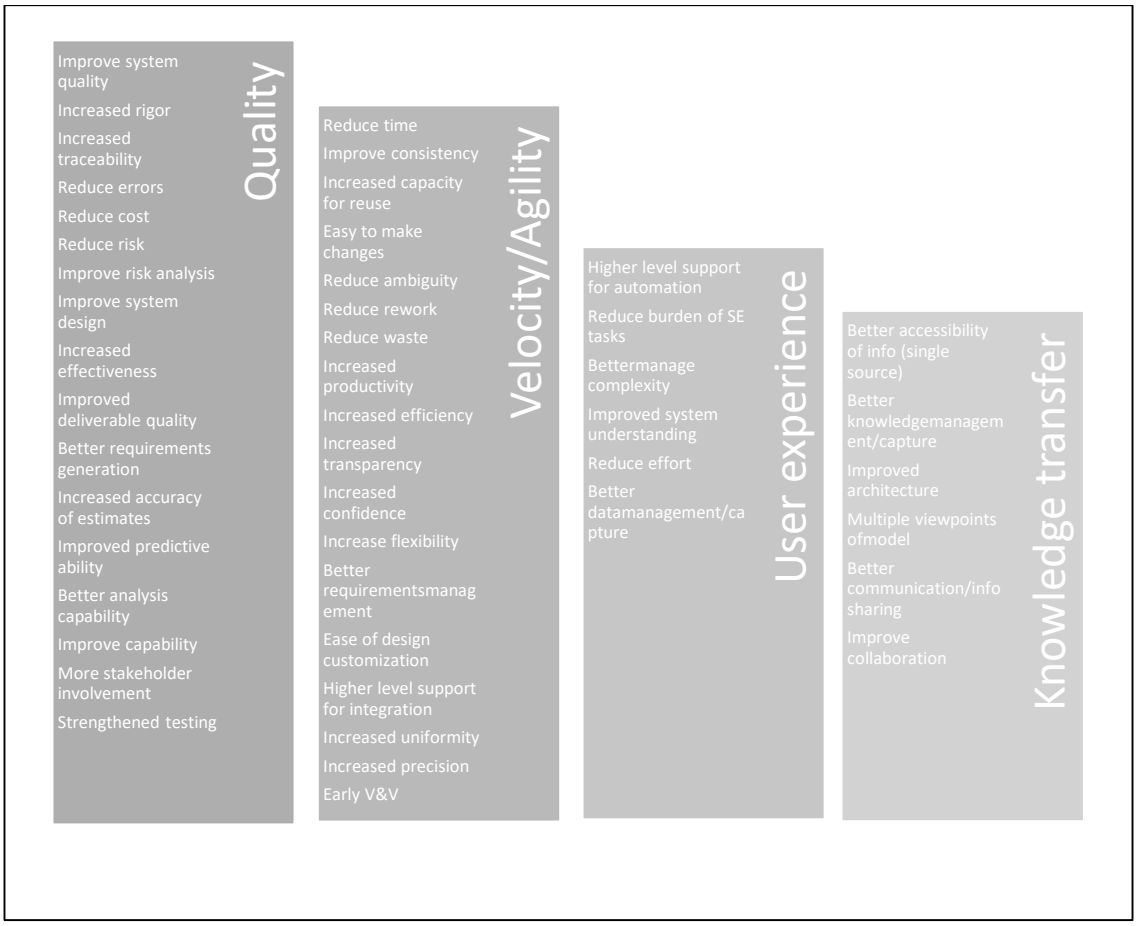



**Figure 7.63:** Expected benefits of MBSE [4]

Figure 7.63, adapted from the classic Systems Engineering textbook by Blanchard and Fabrycky (1998), illustrates a critical issue in program development where significant decisions regarding technologies, materials, and processes are often made prematurely. These early commitments can lead to rigid architectural and contractual constraints that make future changes difficult and expensive, particularly when more specific system knowledge becomes available. This often results in major cost overruns, as documented in both governmental and commercial projects. The figure serves as a roadmap for improving this situation by suggesting strategies such as Model Based Engineering (MBE). These strategies defer early architectural and technology commitments by investing more in upfront systems engineering efforts to build system-specific knowledge. This knowledge enables the creation of more change-adaptive architectures, as recommended by experts like Parnas (1979) and Baldwin-Clark (2000), thereby reducing the steep decline in the ease of making changes over time. Further investments in technology, market analysis, threat trend analysis, and autonomous environment monitoring, along with recommendation

engines, can enhance system-specific knowledge and facilitate the design of systems that are significantly easier to modify. This comprehensive approach ensures a more flexible and cost-effective system design, mitigating the risks associated with premature commitments [39].

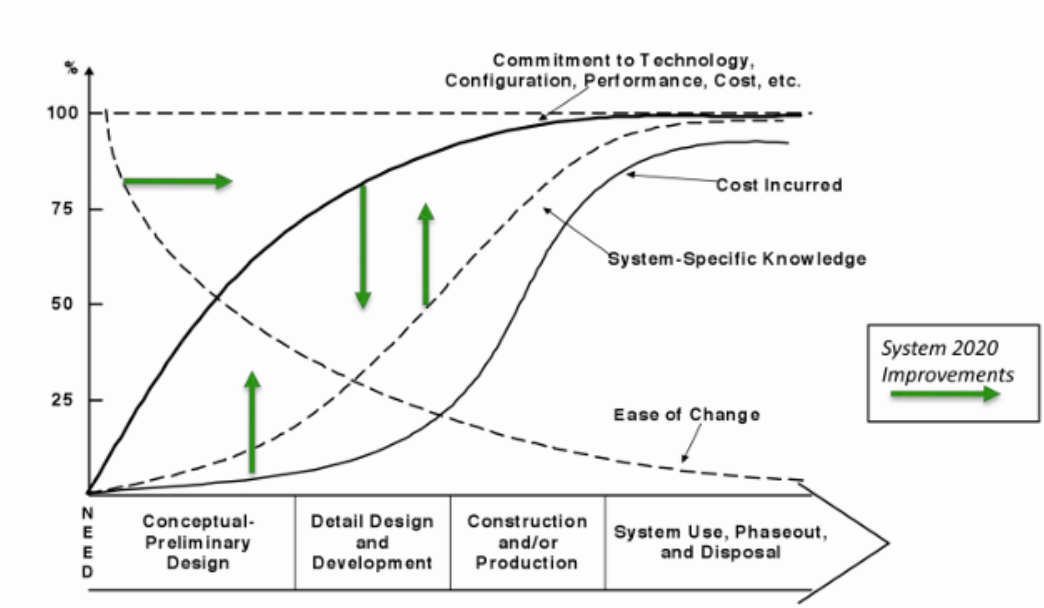



**Figure 7.64:** Systems 2020 Systems Engineering Improvements [39]

### 7.2.1.1. MBSE is an Improvement Over Systems Engineering

Figure 7.64 depicts comparable development costs for the three methodologies on the left and comparable project on-time delivery on the right. For example, projects employing the MBSE technique cost 55% less than those using the standard SE approach. Furthermore, 62% of projects utilizing an MBSE method were completed on schedule, compared to 59% for standard SE approaches. Tommasi and Vacca used data from an independent survey conducted by Embedded Market Forecasters (EMF) of 667 SE respondents working on software-intensive product delivery initiatives [21].

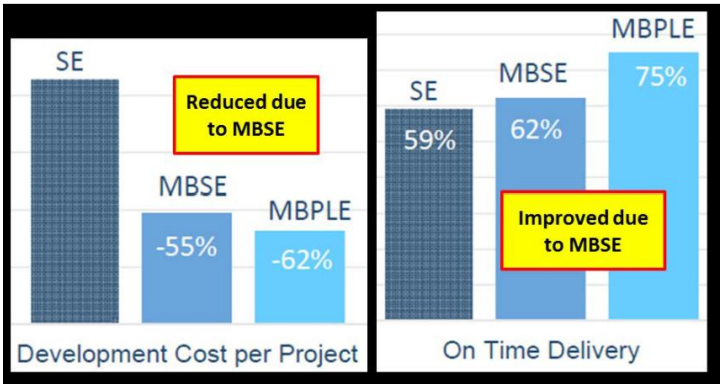



**Figure 7.65:** Effectiveness of MBSE [21]

### 7.2.1.2. A Significant Drop in Defects Specially Linked to MBSE

Saunders [22] validated an MBSE method in his case study, "Does a Model Based Systems Engineering Approach Provide Real Program Savings? - Lessons Learnt," by comparing defect rates on four projects conducted using a document based systems engineering approach to three programs done under an MBSE approach. Figure 7.65 shows a substantial 68% reduction in faults when MBSE procedures were implemented. Defects were recorded for five years, beginning before an MBSE technique was established [21].

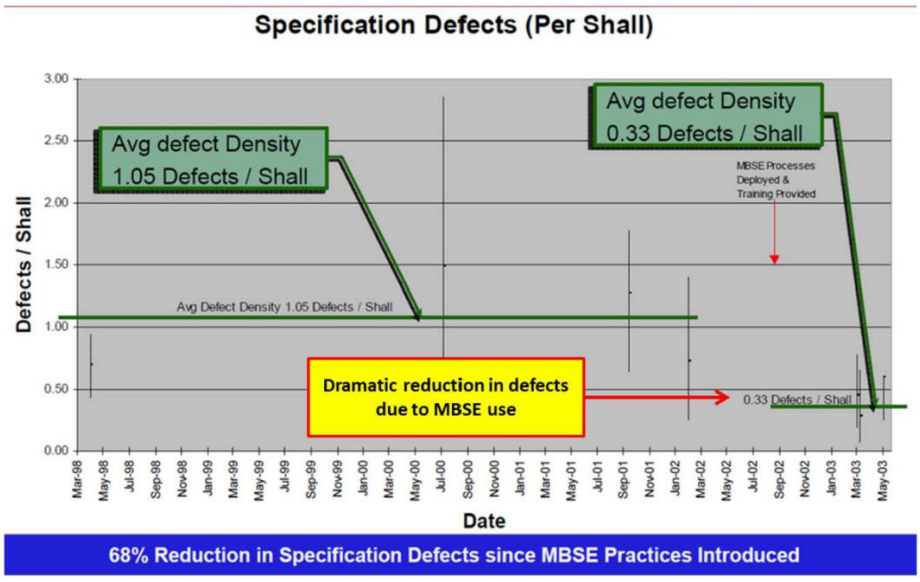



**Figure 7.66:** Defects Were Lower After MBSE Was Implemented [21]

Saunders explains that the Document-Based Systems Engineering (DBSE) approach centers on producing specifications, often using a requirements management tool,

which he does not consider an MBSE tool. He points out that DBSE's manual methods provide poor traceability reports and erroneous specs. In contrast, Saunders emphasizes that the Model Based Systems Engineering (MBSE) method gives a thorough knowledge of system behavior. According to him, "an MBSE approach focuses on evolving the system model until all views functional, requirements, architecture, and test are fully integrated." Saunders emphasizes that the functional behavior cannot be fully understood in order to produce a thorough and consistent specification using the DBSE method.

#### 7.2.1.3. System models must be used to rationally and functionally evaluate early design decisions

The cost of design modifications rises dramatically with the stage of the product lifecycle at which they occur. modifications linked to manufacturing are very costly, surpassed only by the expenses of product recalls. Errors in design have the potential to bring down whole operations, such space exploration projects. Verifying design choices early and consistently throughout the lifespan is therefore essential. For example, automatic rule checking of formal circuit specifications (such as VHDL) in electrical design enables the industry to maintain high rates of innovation and high-quality products.

Therefore, logical validation and model-based simulation two aspects of Computer Aided Engineering (CAE) must be fundamental to systems engineering. As system design progresses, these validations should address increasingly specific aspects well before the product is built and utilized.

#### 7.2.1.4. Increasing complexity of products and production systems requires new development processes, methods, and tools

Innovation and the demand for customization drive the development of increasingly complex products and processes. To manage this complexity, three strategies are employed: reduction, avoidance, and mastering. While reduction and avoidance are preferred when they do not compromise functionality and safety, mastering complexity is essential when it is inevitable. Advanced multi disciplinary technologies further highlight the significance of mastering complexity.

Separation of concerns, hierarchical decomposition, abstraction, encapsulation, and standardization are effective techniques for handling complexity. Integrating the work of different engineering disciplines into a common context is key, guided by defined semantics.

To realize this integration, interdisciplinary methods and modeling languages are needed. Model Based Systems Engineering (MBSE) outlines the interaction of these methods, languages, and tools. Tools should be user friendly to ensure adoption, given that complex tools often fail due to their non-intuitive interfaces and high training requirements.

We recommend supporting the development process with accessible, easy to use tools that allow practitioners to choose appropriate methods for their domain while ensuring results are integrated into the system model. Users should be able to reference information across tools without duplication.

#### 7.2.1.5. “Internet of Things” "Internet of Things" and "Industries 4.0" are made possible by MBE.

Currently, Product Lifecycle Management (PLM) and Model Based Systems Engineering (MBSE) are distinct approaches to product development, each arising from different needs. Model Based Engineering (MBE) is viewed as the integration of PLM and MBSE.

Product Data Management (PDM) was first created with an emphasis on version control, change procedures, and product configuration in order to handle product specifications, including requirements papers and CAD files. In the past, 2D CAD models were frequently printed and kept in physical storage. With the introduction of 3D CAD models, virtual administration was required, and PDM systems were born out of this need.

PLM developed from PDM to handle all data associated with a produced product at every stage of its life. With the increasing need to manage virtual models encompassing new engineering aspects, enhanced model management capabilities are required. Currently, PLM systems utilize a document based approach, handling documents and associated metadata. Future PLM systems, however, need to take a more organized, model based strategy that includes relevant data. This represents a

significant paradigm shift, necessitating that next generation PLM systems be inherently model based.

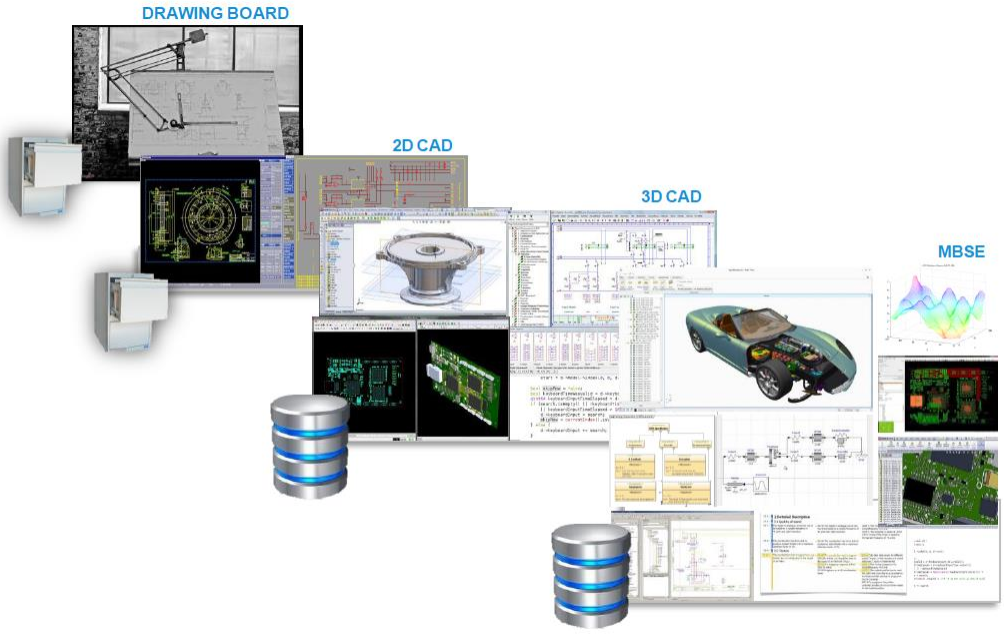



**Figure 7.67:** Model-based engineering has replaced the drawing board in product development [40].

The need to assure proper design early in the development process and to conceptually comprehend goods gave rise to Systems Engineering (SE). Its main objective is to provide a comprehensive, coherent, and workable system specification that will direct the development of the product. Systems engineers have resorted to models to mimic structural stability, behaviour, cost, and other elements as products become more complex; they frequently concentrate on functional verification and safety assessment.

Currently, SE is primarily applied in the early stages of product development and remains disconnected from Product Lifecycle Management (PLM) data in subsequent stages such as manufacturing and maintenance. However, developing, manufacturing, and maintaining complex interconnected products without understanding their functional behavior and environmental interactions poses significant challenges. This is particularly pertinent as smart manufacturing plants ("Industrie 4.0") emerge, requiring model based development methodologies.

### 7.2.1.6. Future PLM systems must consider a product holistically as a multidisciplinary system

All information generated throughout a product's life cycle, from inception to recycling, must be managed in a shared context to effectively meet the challenge. In the context of today's Product Lifecycle Management (PLM) systems, this is not adequately achieved, as disparate work products from mechanical, electrical, and software engineering are managed without regard to their logical interdependencies. Although PLM systems are intended to serve as the interdisciplinary backbone of modern virtual product development, they often lack support for non-mechanical domains and multidisciplinary integration. The overarching concept remains unofficial and poorly documented.

Conversely, complex products should be viewed as interdisciplinary systems, consisting of integrated and interrelated work products from all relevant fields. Therefore, it is proposed that PLM systems be enhanced to manage Model Based Systems Engineering (MBSE) artifacts and constructs. This entails a transition from document-centric to atomic and logically connected artifacts.

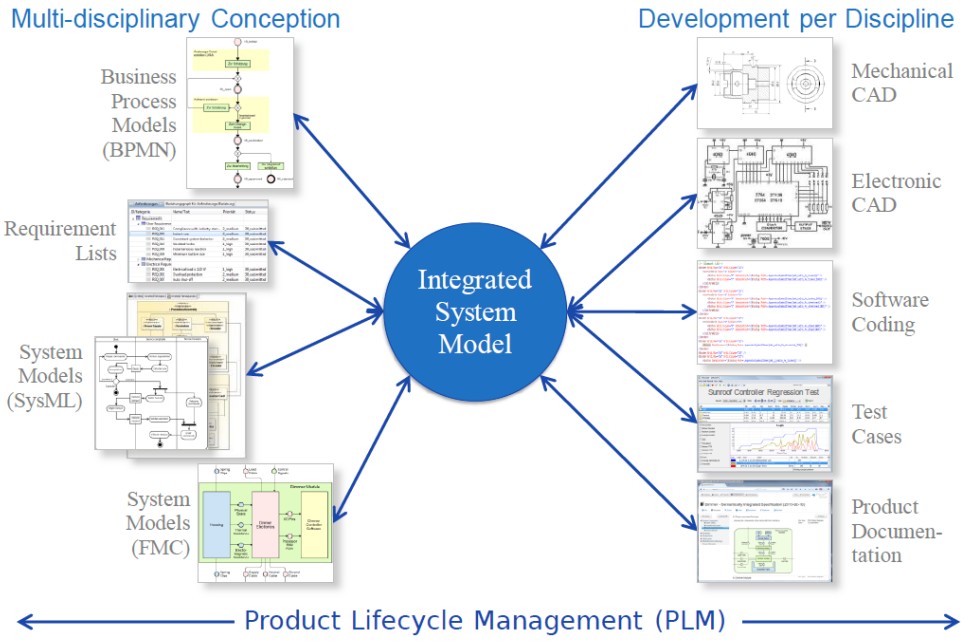



**Figure 7.68:** An integrated system model serves as a development reference by integrating data from many sources [41].

MBSE integrates engineering information elements to create a semantic network or graph of entities interpretable by computers. This decreases the work required to

produce, manage, distribute, and preserve information, resulting in an effective foundation for information lifecycle management. It offers effective information traceability and accountability, as well as continuous data quality improvement, through automated, rule-based data governance. This approach reduces recurring human effort, allowing more focus on engineering design. While documents remain essential as official records, they are derived from system models, enhancing design and documentation consistency.

To make the Integrated System Model usable throughout the system's lifecycle, certain objects within it must undergo configuration control. PLM systems have long supported versioning and configuration control for documents containing various information items. Atomic elements of the integrated system model itself will eventually require configuration control.

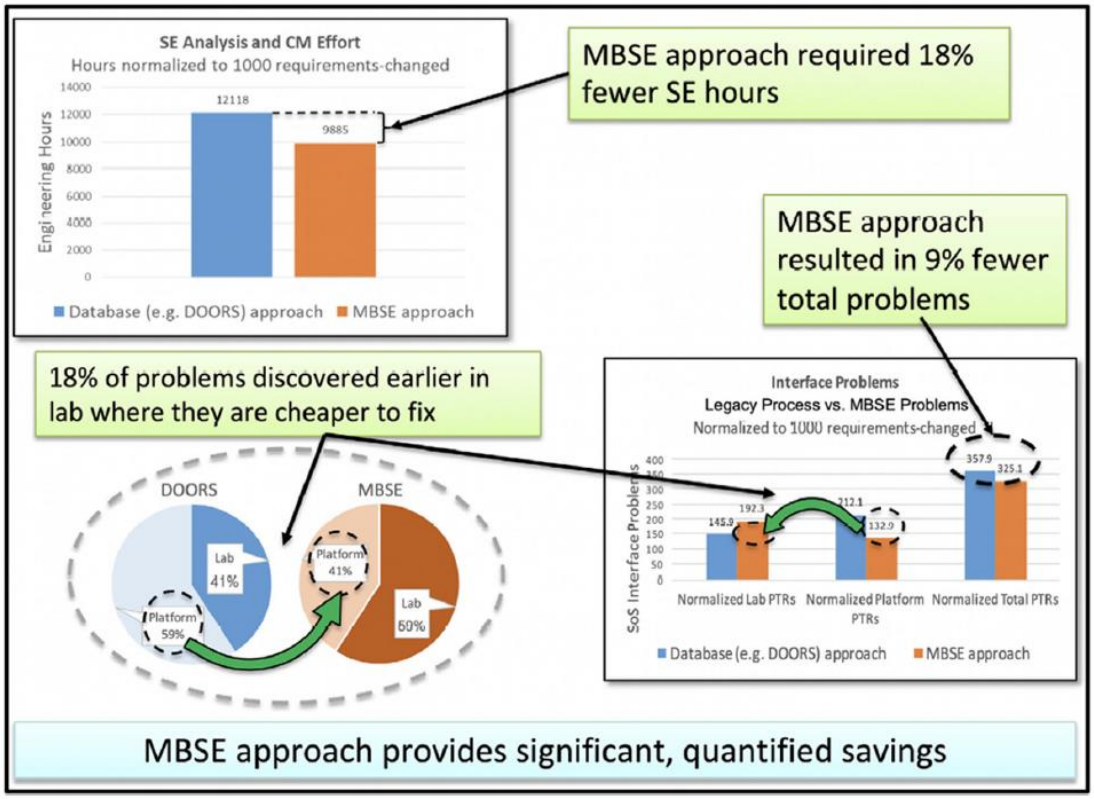



**Figure 7.69:** Quantified benefits [4]

This section summarizes the six capabilities of Model Based Systems Engineering (MBSE), which are:

1. Facilitating communication by offering a single, current, and dependable information source.

2. Improving cooperation both vertically and horizontally by creating a common knowledge of the system.
3. Making it possible for engineering and architectural work to be reused by utilizing information stored in models and standardizing documentation.
4. Using model dependencies to improve traceability and transparency and enabling quick design iterations.
5. Facilitating the whole lifetime by offering readily available and up-to-date information.
6. Promoting a thorough grasp of the problem by producing extensive information about the system from several angles [8].

### 7.2.2. Disadvantages

An exploratory literature review found that firms already implementing MBSE face five common problems challenges [8]. Each of these challenges is briefly described.

#### 7.2.2.1. High Initial Cost and Resource Requirement

The cost of MBSE tools and training is substantial. Furthermore, the effective use of MBSE necessitates adequate human and financial resources. This presents a significant disadvantage, particularly for small and medium sized enterprises

- **Lack of Guidelines on Implementing MBSE:** A survey conducted with 356 engineers revealed that 59% of respondents find it challenging to convince others of the value of MBSE for their organization due to a lack of implementation guidelines. This finding indicates that the primary challenge for survey participants is the implementation of MBSE, suggesting that future efforts should focus on developing guidelines to facilitate the adoption and integration of MBSE practices.
- **There is a need for skilled engineers:** One of the primary challenges identified in several case studies is the need for skilled engineers, particularly in systems engineering and MBSE. This lack of skilled personnel poses a significant barrier to the successful implementation of MBSE methodologies. MBSE models need to be properly maintained and updated in order to be useful

later in a program. For design engineers to properly utilize the data supplied by systems engineers, a fundamental grasp of MBSE concepts is necessary. But instead of looking at MBSE model artifacts, engineers and stakeholders are used to looking at traditional papers. Bringing in the core idea that "first change the model, the model is the design" might face opposition from the organization's culture. Overcoming this cultural resistance represents a significant challenge in the successful adoption of an MBSE approach.

### 7.2.2.2. Complexity and Learning Curve

The implementation of MBSE can be complex and time-consuming. For beginners, learning MBSE can be challenging, potentially leading to slow progress in the initial stages of projects.

- **Differing Understandings of MBSE Capabilities:** Assessing MBSE is complicated by its relatively recent emergence as a discipline [42]. Various sources offer distinct perspectives on the capabilities and values of the methodology. A goal for MBSE to become a "standard practice and [...] integrated with other modeling and simulation as well as digital enterprise functions" by 2025 was stated by the International Council on Systems Engineering (INCOSE) [39]. In order for MBSE to become this accepted standard practice, system acquisition managers, program managers, and systems engineers working on large scale projects involving a variety of real world systems need to be persuaded of the benefits of the technology [43].

### 7.2.2.3. Organizational Resistance and Cultural Change

Organizations may exhibit resistance to change. The successful implementation of MBSE necessitates changes in organizational culture, which can lead to significant resistance.

**Changes in organization, methodology and tooling are necessary for MBSE**

To successfully implement Model Based Systems Engineering (MBSE) within an organization, several key actions are required. Beyond adopting new tools and methodologies, MBSE necessitates significant organizational changes. Companies that lack systems engineers in their development teams must hire individuals with

these skills and establish new roles and responsibilities. The construction of Integrated Product Development Teams (IPDTs), where a lead system architect is in charge of the overall system idea and organizes contributions from all development teams, is recommended by ISO/IEC 15288 as a useful structure. The outcomes are recorded in an extensive model of the system that is being developed [44].

For organizations to adopt Model Based Systems Engineering practices, they need to:

- Improve the product development process (PDP) to encourage multidisciplinary cooperation and incorporate Systems Engineering activities and outputs into main development milestones.
- Ensure interdisciplinary collaboration is considered not only during the development phases but also throughout manufacturing and maintenance.
- Revise engineering methods to enhance cooperation among engineers with diverse capabilities across worldwide locations.
- Establish the flow and reference of data, such as requirements, functionalities, behavior, geometry, software code, electrical/electronic layouts, and test cases, between various tasks and tools.
- Address product data management issues such as version control, access policies, variants, release and change management.
- Provide the technical framework required to facilitate teamwork and simultaneous engineering.

These steps are essential for effectively integrating MBSE into an organization's operations and realizing the full benefits of this approach.

### 7.2.2.4. Patchwork Environments Not Fulfilling Expectations

Currently, many transformation approaches lack a clear, overarching value proposition connected to a coherent methodology. This absence results in digital patchwork environments [45][46] ,that fail to meet the expectations of digital transformation. The evaluation and design of a successful Model Based development environment are difficult jobs since such environments are complex sociotechnical systems used in a variety of scenarios. However, this deficiency restricts an organization's ability to select and implement effective components of its MBSE methodology [8].

# 8. CONCLUSION

In conclusion, this pilot project has demonstrated that the use of Model Based Systems Engineering (MBSE) in the design of complex machinery, such as gas turbine engines, presents certain disadvantages as highlighted in existing research. Nevertheless, notable advantages have also been observed. These include the ability to avoid issues encountered in case studies due to the inherent nature of MBSE, reduced documentation, and enhanced communication between teams, facilitating access to accurate information at any time and place. In addition to the anticipated cost and timing benefits, further advantages have been realized.

Advantages include:

- **Improved Communication and Collaboration**: MBSE enhances communication and collaboration among stakeholders, enabling a better understanding and management of system requirements, designs, and tests. Its visualization capabilities make complex systems easier to comprehend.
- **Increased Traceability and Quality**: MBSE improves traceability of requirements, designs, and tests, which enhances quality in systems engineering processes and allows for the early detection of errors. Standardization and advanced visualization aid in more efficient process management.
- **Time and Cost Savings**: MBSE can reduce project durations and costs. The efficiency gains provided by MBSE, particularly in complex systems, allow projects to be completed more quickly and at lower costs.
- **Better Risk Management**: MBSE facilitates better risk management by enabling the pre-identification and management of potential risks through system models, thereby increasing project success and preventing unexpected issues.

Disadvantages include:

- **High Initial Cost and Resource Requirements**: The tools and training required for MBSE are costly, necessitating significant human and financial

resources, which can be a major disadvantage for small and medium sized enterprises.

- **Complexity and Learning Curve**: MBSE applications can be complex and time consuming. For beginners, learning MBSE can be challenging, potentially slowing project progress initially.
- **Data Management Challenges**: Managing large datasets can be difficult. MBSE requires handling vast amounts of data, which can create challenges in data management processes.
- **Organizational Resistance and Cultural Change**: There may be resistance to change within an organization. Successful implementation of MBSE may require cultural shifts within the organization, which can lead to resistance.

# REFERENCES


[1] TEI, “TEI - TEI-TJ90 Turbojet Engine,” TEI. 2024, Accessed: Jun. 12, 2024. [Online]. Available: https://www.tei.com.tr/en/products/tei-tj90-turbojet-engine

[2] TUSAŞ, “Şimşek,” TUSAŞ. 2024, Accessed: Jun. 12, 2024. [Online]. Available: https://www.tusas.com/urunler/iha/hedef-ucak-sistemleri/simsek

[3] Madni, A. M., Purohit, S., “Economic Analysis of Model-Based Systems Engineering,” Syst. 2019, 7, 1.

[4] Henderson, K., McDermott, T., Van Aken, E., Salado, A., “Towards Developing Metrics to Evaluate Digital Engineering,” Systems Engineering. 2023, 26, 1, 3-31.

[5] Bayramoğlu, A., “Industrial Adoption of Model-Based Systems Engineering: An Investigation Within The Defence Industry,” The Middle East Technical University, Ankara. 2022.

[6] Haveman, S. P., Bonnema, G. M., Van Den Berg, F. G. B., “Early insight in systems design through modeling and simulation,” CSER 2014, Science Direct, Elsevier B.V. 2014, 171-178.

[7] Lee, R. M., Gilb, T., “Managing Complexity Within the Engineering of Product and Production Systems,” Security and Quality in Cyber-Physical Systems Engineering, Springer International Publishing. 2019, ch. 3.

[8] Schöberl, M., “A Guideline for The Implementation of Model-Based Systems Engineering,” Technical University of Munich, Garching b. München. 2019.

[9] INCOSE, “Systems Engineering Handbook: A Guide for System Life Cycle Processes and Activities,” *Systems Engineering*. John Wiley & Sons Inc. 2015.

[10] ISO, ISO/IEC/IEEE 15288:2015 Systems and software engineering — System life cycle processes. 2015.

[11] IEEE, 1220-2005 - *IEEE Standard for Application and Management of the Systems Engineering Process*. 2005.

[12] NASA OCE, “Systems Engineering Handbook,” *NASA*. 2019, Accessed: Jun. 14, 2024. [Online]. Available: https://www.nasa.gov/reference/systems-engineering-handbook/

[13] Oliver, D. W., “Systems Engineering and Object Technology,” *INCOSE International Symposium*. 1994, 4, 1.

[14] Kalawsky, R. S., et al., “Bridging The Gaps in A Model-Based System Engineering Workflow By Encompassing Hardware-In-The-Loop Simulation,” *IEEE Syst J*. 2013, 7, 4.

[15] Fernandez, J. L., Hernandez, C., *Practical Model-Based Systems Engineering*. Norwood, MA 02062: Artech House. 2019.

[16] Ramo, S., St.Clair, R. K., The Systems Approach: Fresh Solutions to Complex Civil Problems Through Combining Science and Practical Common Sense. 1998.

[17] Holt, J., Perry, S., Don't Panic! The Absolute Beginner's Guide to Model-Based Systems Engineering. INCOSE UK. 2017.

[18] Nder, C., Military Standart Engineering Management, MIL-STD-499A. Washington, D.C. 20301: Department of Defense. 1974. [Online]. Available: http://www.everyspec.com

[19] EIA, Processes for Engineering a System. Arlington, VA 22201: EIA. 1999.

[20] Lightsey, B., "Introduction to Systems Engineering Management," Systems Engineering Fundamentals, The Defense Acquisition University Press. 2001, ch. 1, 1-8.

[21] Carroll, E. R., Malins, R. J., "Systematic Literature Review: How is Model-Based Systems Engineering Justified?," Albuquerque, New Mexico 87185. 2016. [Online]. Available: http://www.ntis.gov/help/ordermethods.asp?loc=7-4-0#online

[22] Honour, E. C., "Understanding The Value of Systems Engineering," INCOSE International Symposium. 2004, 14, 1.

[23] Lightsey, B., Systems Engineering Fundamentals. Fort Belvoir, Virginia 22060-5565: The Defense Acquisition University Press. 2001.

[24] Friedenthal, S., Moore, A., Steiner, R., A Practical Guide to SysML: The Systems Modeling Language. 2008.

[25] Long, J. E., "Relationships Between Common Graphical Representations in System Engineering," The 5th Annual INCOSE International Symposium. 1995. Accessed: Jun. 15, 2024. [Online]. Available: www.vitechcorp.com

[26] Parrott, E., "SysML Distilled: A Brief Guide to the Systems Modeling Language," INSIGHT. 2014, 17, 2.

[27] Friedenthal, S., Moore, A., Steiner, R., "OMG Systems Modeling Language (OMG SysMLTM) Tutorial," INCOSE International Symposium. 2008, 18, 1.

[28] Husung, S., Weber, C., Mahboob, A., Kleiner, S., "Using Model-Based Systems Engineering for Need-Based and Consistent Support of The Design Process," Proceedings of the Design Society. 2021, 1, 3369-3378.

[29] Younse, P., "Comparative Analysis of Model-Based Systems Engineering And Traditional Systems Engineering Approaches For Architecting Robotic Space Systems Through Knowledge Categorization, Automatic Information Transfer, And Automatic

Knowledge Processing Measures," Colorado State University, Fort Collins, Colorado. 2021.

[30] Douglass, B., "Introduction to Model-Based Engineering."

[31] Estefan, J. A., "Survey of Model-Based Systems Engineering (MBSE) Methodologies," California, U.S.A. 2008.

[32] Weilkiens, T., Systems Engineering with SysML/UML: Modeling, Analysis, Design. 2008.

[33] Hoffmann, H. P., Systems Engineering Best Practices with the Rational Solution for Systems and Software Deskbook Release 3.1.2 Engineering. IBM Corporation. 2011.Accessed:Jun.15,2024.[Online].Available:https://www.slideshare.net/slideshow/ibm-rational-harmony-deskbook-rel-312/7122641

[34] Aleksandravičienė, A., Morkevičius, A., MagicGrid Book of Knowledge - A Practical Guide to Systems Modeling using MagicGrid, 2nd ed. Kaunas, Lithuania: Vitae Litera. 2021.

[35] McDermott, T. A., Hutchison, N., Clifford, M., Van Aken, E., Salado, A., Henderson, K., "Benchmarking The Benefits and Current Maturity of MBSE Across The Enterprise," 2020.

[36] INCOSE, "Systems Engineering Vision 2035," 2021. [Online]. Available: www.incose.org/sevision

[37] McDermott, T., Henderson, K., Salado, A., Bradley, J., "Digital Engineering Measures: Research and Guidance," INSIGHT. 2022, 25, 1.

[38] Campo, K. X., Teper, T., Eaton, C. E., Shipman, A. M., Bhatia, G., Mesmer, B., "Model-Based Systems Engineering: Evaluating Perceived Value, Metrics and Evidence through Literature," INCOSE Systems Engineering. 2023, 26, 1.

[39] Boehm, B., "Systems 2020 Strategic Initiative," California. 2010. Accessed: Jun. 20,2024.[Online].Available:https://www.researchgate.net/publication/235066727_Systems_2020_Strategic_Initiative

[40] Maheshwari, A., "Industrial Adoption of Model-Based Systems Engineering: Challenges and Strategies," Theses and Dissertations Available from ProQuest. 2015. Accessed:Jun.18,2024.[Online].Available:https://docs.lib.purdue.edu/dissertations/AAI10061186

[41] Kaufmann, U., Schuler, R., "10 Theses About MBSE and PLM - Challenges and Benefits of Model Based Engineering (MBE)," 2015.

[42] Reichwein, A., Paredis, C. J. J., "Overview of Architecture Frameworks and Modeling Languages for Model-Based Systems Engineering," Proceedings of the ASME Design Engineering Technical Conference. 2011.

[43] Madni, A. M., Sievers, M., “Model-Based Systems Engineering: Motivation, Current Status, and Research Opportunities,” Systems Engineering. 2018, 21, 3.

[44] Von Dungern, O., “Übergreifende Konzeption von Geräten für die Gebäudeautomation – Methodik und Management,” Tag des Systems Engineering. 2014.

[45] Tabaković, N., Durakovic, B., “Impact of Industry 4.0 on Aerospace and Defense Systems,” Defense and Security Studies. 2021, 2, 63-78.

[46] Long, D., “Evolving MBSE to Enable the Digital Future’ with David Long,” [Online]. Available: https://www.youtube.com/watch?v=2g00H9SdAWA

# BIOGRAPHY

Tuğçe COŞKUNTUNA DAŞDEMİR graduated from Kırşehir Science High School in 2006 and completed her B.Sc. in Electronics and Communication Engineering at Yıldız Technical University in 2011. Simultaneously, she obtained a degree in Business Administration from Anadolu University in 2012.

After her undergraduate studies, she began her career in 2011 as a product engineer for secondary protection relays at Aktif Group. From 2012 to 2021, she worked at Ford Otosan in various roles, contributing to different projects as an R&D engineer for light, medium, and commercial vehicles. In 2021, she joined TUSAŞ Engine Industry, Inc. (TEI) as a systems engineer and currently serves as a staff engineer.

Concurrently, she is pursuing her M.Sc. in the Energy Technologies Institute at Gebze Technical University. Additionally, in 2023, she began her studies in Visual Communication Design at Anadolu University.

Tuğçe Coşkuntuna Daşdemir is married and has four cats.

# PUBLICATIONS AND PRESENTATIONS FROM THE THESIS

[1] Dasdemir Coskuntuna T., (2024), “Implementing Partial MBSE In Turbofan Engine Architecture Development And Its Reported Advanteges And Disadvanteges”, 8th Gebze Technical University Graduate Research Symposium, Kocaeli, Turkey, 30-31 May.