DLR-IB-SL-HF-2021-218

A Model-Based Methodology for the Integration of a System Architecture in a Digital Aircraft Design Process

Masterarbeit

Yassine Ghanjaoui

Deutsches Zentrum
für Luft- und Raumfahrt

Dokumenteigenschaften

Titel	A Model-Based Methodology for the Integration of a System Architecture in a Digital Aircraft Design Process
Betreff	Masterarbeit
Institut	Institut für Systemarchitekturen in der Luftfahrt (SL)
Erstellt von	Yassine Ghanjaoui
Beteiligte	
Geprüft von	M.Sc. Mara Fuchs (DLR-SL) und Dr. Jörn Biedermann (DLR-SL)
Freigabe von	Prof. Dr.-Ing. Jutta Abulawi (HAW Hamburg)
Datum	01.12.2021
Version	1.0
Dateipfad	

Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Master Thesis

Yassine Ghanjaoui

A Model-Based Methodology for the Integration of a System Architecture in a Digital Aircraft Design Process

Fakultät Technik und Informatik
Department Fahrzeugtechnik und Flugzeugbau

Faculty of Engineering and Computer Science
Department of Automotive and
Aeronautical Engineering

Yassine Ghanjaoui

A Model-Based Methodology for the Integration of a System Architecture in a Digital Aircraft Design Process

Master Thesis submitted as part of the Master's Examination
in the Master of Science Degree Programme Aeronautical Engineering
at the Department of Automotive and Aeronautical Engineering
of the Faculty of Engineering and Computer Sciences
at the Hamburg University of Applied Sciences

First Examiner: Prof. Dr. -Ing. Jutta Abulawi
Second Examiner: Dr. -Ing. Jörn Biedermann

Date of submission: 15^{th} November, 2021

Yassine Ghanjaoui

Title of Thesis

A Model-Based Methodology for the Integration of a System Architecture in a Digital Aircraft Design Process

Keywords

MBSE, SysML, Aircraft Design, CPACS, System Architecture, Simulation, KBE, MDO, VR

Abstract

Due to the current trend towards sustainable, environmentally friendly, and digitally networked aircraft, it is necessary to integrate new revolutionary technology. This increases complexity and necessitates the investigation of novel aircraft designs and configurations early, quickly, and cost-effectively. Innovative concepts and approaches are necessary to handle this complexity. Model-based Systems Engineering (MBSE) is a fundamental approach to support and manage complex system development. Notable benefits are achieved compared to more traditional document-based methods. Therefore, researchers at the DLR Institute of System Architectures in Aeronautics are developing a process to fully digitalize and virtualize an aircraft. This enables it to be completely represented as a virtual product, allowing for the rapid implementation, visualization, and validation of novel design concepts. This work extends the digital design process with a model-based methodology for developing and integrating the functional system architecture. An application use case on passenger service functions serves as a proof of concept (PoC) during the methodology evaluation. At the beginning, the system is analyzed and it's architecture is modeled using the Systems Modeling Language (SysML). The system requirements are defined and all model elements are linked together. This improves the traceability and enables early error detection as well as the validation of requirements. The system model is then linked to existing models. Model integration allows the system architecture to be configured with cabin design parameters from CPACS and the architecture data to be used for geometrical cabin design. To illustrate advantages of architecture integration, multidisciplinary optimization is investigated based on the interaction between the different models. A trade-off analysis is performed using multidisciplinary design parameters regarding electrical power distribution and cable length. The interactions and effects between the design domains are therefore identified and analyzed.

Yassine Ghanjaoui

Thema der Arbeit

Eine Modellbasierte Methodik zur Integration einer Systemarchitektur in einen digitalen Flugzeugsentwurfsprozess

Stichworte

MBSE, SysML, Flugzeugentwurf, CPACS, Systemarchitektur, Simulation, KBE, MDO, VR

Kurzzusammenfassung

Aufgrund aktueller Trends zu nachhaltigen, umweltfreundlichen und digital vernetzten Flugzeugen, müssen neue revolutionäre Technologien integriert werden. Dies steigert die Komplexität und erfordert eine frühzeitige, schnelle und kostengünstige Untersuchung neuer Flugzeugdesigns und Konfigurationen. Um diese Komplexität zu beherrschen sind neue Konzepte und Methoden erforderlich. Das Model-based Systems Engineering (MBSE) ist ein grundlegender Ansatz für die Entwicklung von komplexen Systemen, welcher deutliche Vorteile gegenüber klassischen dokumentbasierten Ansätzen erzielt. Am DLR-Institut für Systemarchitekturen in der Luftfahrt entwickeln die Forschenden ein Verfahren zur vollständigen Digitalisierung und virtuellen Darstellung des Flugzeugs. Neue innovative Konzepte im Flugzeug- und Kabinendesign können somit schnell umgesetzt, visualisiert und validiert werden. Diese Arbeit erweitert den digitalen Entwurfsprozess um eine modellbasierte Methodik zur Entwicklung und Integration der funktionalen Systemarchitektur. Zur Bewertung der Methodik dient die Anwendung an Passagierservicefunktionen als Proof of Concept (PoC). Zunächst wird das System analysiert und die Systemarchitektur mit der Systems Modeling Language (SysML) modelliert. Dabei werden die Systemanforderungen definiert und alle Modellelemente miteinander verknüpft. Somit wird die Rückverfolgbarkeit von Informationen verbessert, sowie die frühzeitige Fehlererkennung und die Validierung von Anforderungen ermöglicht. Anschließend wird das Systemmodell mit bestehenden Modellen verbunden. Die Modellintegration ermöglicht es, die Systemarchitektur mit Kabinenentwurfsparametern aus CPACS zu konfigurieren und die Architekturdaten für die Kabinenauslegung zu nutzen. Um Vorteile der Architekturintegration zu verdeutlichen, wird eine multidisziplinäre Optimierung anhand der Interaktion zwischen den verschiedenen Modellen untersucht. Dabei wird eine Trade-off Analyse mithilfe von multidisziplinären Designparametern zur elektrischen Leistungsverteilung und Länge der Versorgungskabeln ausgeführt. Die Wechselwirkungen zwischen den Designdomänen werden identifiziert und analysiert.

Acknowledgment

This thesis was written at the Hamburg University of Applied Sciences in cooperation with the German Aerospace Center's (DLR) Institute for System Architectures in Aeronautics.

To begin, I want to express my heartfelt gratitude to my advisor, Prof. Dr.-Ing. Jutta Abulawi, for her unwavering support of my Master's thesis, for her valuable time, motivation, enthusiasm, and immense knowledge. Without her recommendation and guidance, this work would have never emerged.

My sincere thanks go to Dr.-Ing. Björn Nagel for providing me with the opportunity to conduct this research at the DLR institute. I would also like to express my profound thanks to my two supervisors, Dr.-Ing. Jörn Biedermann and M.Sc. Mara Fuchs, for their precious suggestions, valuable advice, and active support throughout the entire process of writing this thesis.

I would like to express my appreciation to my colleagues for their kind welcome into the team and for our enjoyable collaboration during my Master's thesis. My heartfelt special gratitude also go to Prof. Dr.-Ing. Marc Wiegmann, who had an open ear and offered me his precious advise.

I would also like to thank my parents and sister for the opportunities they've provided me in life, as well as their unwavering love and support throughout my journey. Last but not least, I'd like to express my sincere thanks to my girlfriend, Leonie Rörup, and my best friend, Rim Benouahi, for always being there for me and supporting me in all my endeavors.

I dedicate this work to Beate Rörup, a truly remarkable, loving, and supportive mother.

Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

FACULTY OF ENGINEERING AND COMPUTER SCIENCE
DEPARTMENT OF AUTOMOTIVE AND AERONAUTICAL ENGINEERING
Prof. Dr.-Ing. Jutta Abulawi

Master Thesis Assignment

Name: Yassine Ghanjaoui

A Model-based Methodology for the Integration of a System Architecture in a Digital Aircraft Design Process

Introduction

Model-based Systems Engineering (MBSE) is a fundamental approach for the development of complex systems in today's and tomorrow's industry. Especially in the Aeronautics field, where the complexity of aircraft systems is constantly increasing, new concepts and methodologies are required to face environmental and socio-economic challenges. This makes the digital system modeling and the interaction between the generated virtual models and physical systems promising methods, that offer advantages compared to classic development methods.

At the DLR's Institute of System Architectures in Aeronautics, researchers are developing a methodology and process for the full digitalization and the virtual representation of the aircraft. Thus, the aircraft is completely depicted as a virtual product and new innovative concepts in aircraft and cabin design can be rapidly implemented, visualized and validated. Moreover, the aircraft digitalization and the communication between models and physical systems deliver valuable and relevant data that help enhance the models themselves and optimize the digital representation, or the so called **"Digital Twin".**

Tasks

This work is a contribution to the existing digitalization methodology at the DLR Institute. It aims to detect where enhancement is needed and beneficial expansion of the process through the integration of executable system architecture models. Hence, appropriate modeling languages and tools will be selected. Knowledge resulting from these analyses will serve as an input for the definition of the modeling methodology and the integration approach. This methodology will then be applied. Therefore, the focus will be on cabin systems and the detailed modeling activities will be applied on a subsystem or component level. The consideration level will be defined during this work. Finally, the resulting model and its interaction with the digitalization process will be tested and the methodology evaluated. Following tasks summarize the main purposes of this work:

- **Analysis and identification of enhancement abilities and process expansion**
- **Development of a modeling and interaction methodology**
- **Implementation of system modelling and connection to digitalization process**
- **Test and validation of the system model and the interaction**
- **Evaluation and assessment of designed methodology**

Contact Person: M.Sc. Mara Fuchs
German Aerospace Center e.V. (DLR) | Institute of System Architectures in Aeronautics |
c/o ZAL TechCenter | Hein-Saß-Weg 22 | 21129 Hamburg | Tel.: +49 40 24 89 64 13 71 |
mara.fuchs@dlr.de | www.DLR.de

Contents

List of Figures

List of Tables

List of Acronyms

A/C	Aircraft
act	Activity diagram
AI	artificial intelligence
API	Application Programming Interface
APU	Auxiliary Power unit
ATA	Air Transport Association
bdd	Block Definition Diagram
CAD	Computer-Aided Design
CBP	Circuit Breaker Panels
CCS	Cabin Core System
CE	Concurrent Engineering
CIDS	Cabin Intercommunication Data System
CMS	Cabin Management System
CPACS	Common Parametric Aircraft Configuration Schema
CPS	Cyber-Physical System
CSM	Cameo Systems Modeler
DAL	Development Assurance Level
DEU	Decoder/Encoder Unit
DEU-A	Decoder/Encoder Unit type A
DEU-B	Decoder/Encoder Unit type B
DLR	German Aerospace Center (Deutsches Luft- und Raumfahrt Zentrum)
DSL	Domain Specific Language
EASA	European Aviation Safety Agency
EC	Economy Class
FAL	Final Assembly Line
FAP	Flight Attendant Panel
FAS	Functional Architecture for Systems Method
FC	First Class
FEM	Finite Elements Method
GPU	Ground Power Unit
HTTP	Hypertext Transfer Protocol
ibd	Internal block diagram
ID	Identification
IEEE	Institute of Electrical and Electronics Engineers
IFE	In-Flight Entertainment

INCOSE	International Council on Systems Engineering
IoT	Internet of Things
I/O	Input/Output
JSON	JavaScript Object Notation
KBE	Knowledge-based Engineering
LRU	Line Replaceable Unit
M2T	Model-To-Text
MBSE	Model-Based Systems Engineering
MDAO	Multidisciplinary Design Analysis Optimization
MDO	Multidisciplinary Design Optimization
NN	Network Nodes
OEM	Original Equipment Manufacturer
OHSC	Overhead Storage Compartment
OMG	Object Management Group
OO	Object Orientation
OOP	Object-Oriented Programming
OOSEM	Object-oriented Systems Engineering Method
OSLC	Open Services for Lifecycle Collaboration
par	Parametric diagram
PEPDC	Primary Electrical Power Distribution Center
PISA	Passenger Interface and Supply Adapter
pkg	Package diagram
PLC	Product Life Cycle
PoC	Proof-of-Concept
PSC	Passenger Service Channel
PSU	Passenger Service Unit
req	Requirements diagram
RIF	Requirements Interchange Format
ROI	Return on Investment
RTCA	Radio Technical Commission for Aeronautics
SDF	Smoke Detection Function
SE	Systems Engineering
seq	Sequence diagram
SimulML	Simulation Modeling Language
stm	State machine diagram
SPDB	Secondary Power Distribution Boxes
STA PISA	Stand-alone PISA
SoI	System-of-Interest
SWT	Semantic Web Technology
SysML	Systems Modeling Language
SYSMOD	Systems Modeling Toolbox
TPO	Top Performance Objectives

uc	Use case diagram
UI	User Interface
UML	Unified Modeling Language
XMI	XML Metadata Interchange
XML	Extensible Markup Language
URI	Uniform Resource Identifier
XSD	XML Schema Definition
VR	Virtual Reality
W3C	World Wide Web Consortium
WWW	World Wide Web

List of Symbols

Symbol	Unit	Description
P	$[W]$	Electrical power
$\bar{P}$	$[W]$	Mean electrical power
U	$[V]$	Electric voltage
R	$[\Omega]$	Ohmic resistance
A	$[mm^2]$	Cable cross section area
l	$[mm]$	Cable length
I	$[A]$	Electric current
ρ	$[\Omega \cdot m]$	Electrical resistivity
y	$[-]$	Objective function
X	$[mm]$	Input variable for optimization case
x	$[mm]$	Aircraft coordinate
S	$[-]$	Optimal X solution

Indexes

Symbol	Description
cu	Control Unit
$cable$	Cable for power transmission
$SPDB$	Secondary Power Distribution Boxes
i	Variable
n	End value of variable
k	End value of variable

1 Introduction

The aviation sector, like many other industries, is facing enormous challenges in meeting society's aspirations for mobility, safety, environment, and economics. To address these issues, the German government launched the "fourth industrial revolution", also known as Industry 4.0, accelerating the digitalization through a range of technologies such as the Internet of Things (IoT), cloud computing, and Cyber-Physical-Systems (CPS) [1]. Industry 4.0 is a term that refers to the intelligent networking of industrial machines and processes via information and communication technologies. It aims to create an interconnected manufacturing enterprise, which communicates, analyzes, and processes data to induce smarter behavior in the physical world. One crucial technology for Industry 4.0 is the so-called digital twin [2].

A digital twin is made up of three components: the virtual model, the physical product, and the interaction between them. Its objective is to create the most accurate virtual representation possible of the physical product [3]. Prior to establishing the digital twin, it is critical to integrate all data, models, and information collected during the product's life cycle (PLC) to create a high-fidelity depiction of this product. This data integration and interlinking is referred to as the Digital Thread. It enables to create a single source of truth, where all information from each discipline are made available in a overall consistent database. To implement digital thread for development and operation of high-quality complex products, the designs are optimized to depict all aspects of the product's life cycle. A huge number of models, such as PLC-models or engineering domains models, all of which are related but significantly distinct, are employed to address the optimization problem of the Digital Thread. These models enable to take use of the synergies inherent in product's interconnected components and subsystems [4].

Model-based approaches are crucial to follow and assist product development throughout its life cycle, especially when it comes to complex and highly interconnected and communicative systems, as in the aviation industry. These systems must be demonstrated to be exceptionally safe and reliable even when functionality is enhanced. Thus, proper management and traceability throughout the PLC is critical. The latter is supported by simulating and testing executable system models.

The German Aerospace Center's (DLR) aeronautics research is accelerating the aviation industry's digitalization. Interconnected models are used to map complex system interrelationships and to aid in the verification of plausibility and data consistency, as well as data traceability, life cycle monitoring, and reconfiguration. This work is a contribution to the DLR's effort in improving the digitalization in the aviation industry. It leverages model-based design and integration approaches to enhance the digital representation and interlinking of aeronautical products.

1.1 Motivation to Digitalization in Aircraft Design

Typically, the aviation industry's current trends dictate which technology should be implemented into the aircraft. These are often brand-new technologies that have never been implemented into an aircraft before. Current German government and international aviation laws require the development of sustainable aircraft and more climate-neutral aviation. This entails the integration of technologies such as hydrogen propulsion systems or fuel cells. The technology integration happens both by adapting existing airplanes as well as investigating and developing entirely new aircraft designs. Digitalization should support a rapid, early, and thorough knowledge of how new technologies can be integrated into aircraft and how they interact with the overall system. Both existing aircraft programs, such as the Airbus A320, as well as novel revolutionary aircraft designs, must be considered, because both cases require agile and effective design assessment. Digitalization should also uncover synergies and their effects on the whole system during technology integration. This synergy analysis enables correct design decisions to be made, the true effects of aircraft modification or new integration to be examined, and only beneficial design changes to be implemented. This would result in significantly cost-effective decisions.

A constructed airplane is the culmination of the expertise, knowledge, and wishes of the several engineers that comprise an aircraft company's different design and manufacturing departments. Therefore, design assessments must consider all interrelationships and linkages between various disciplines. Capabilities of each discipline must all be involved in order to create and optimize the overall product. This prevents each discipline's skill from becoming narrowly focused on itself and failing to consider the implications on other fields. Figure 1.1, taken from *C. W. Miller*'s cartoon "Dream Airplane", demonstrates in a comedic way, what might happen if each design discipline or manufacturing group were permitted to realize their ideas and desires and optimize the aircraft for their needs independently from other disciplines [5].

Aircraft cabins and cabin systems have always played an important role in civil aircraft design and are constantly evolving. Due to structural changes and aviation trends mentioned before, new cabin ideas must be promptly adopted and designed. There is a requirement for rapid development cycles from design to assembly in this case. The incorporation of new technologies has a direct impact on the cabin's architecture, as well as the organization and layout of the cabin systems. Simultaneously, breakthrough technologies enable new sorts of cabin system synergy. Airlines want highly customized, high-efficiency cabin configurations. However, development of novel cabin configurations must proceed more quickly than in the past. To do this, it is critical to understand the intricate connections between the different subsystems as thoroughly and as early as possible to assure a holistic and considerable gain in efficiency.

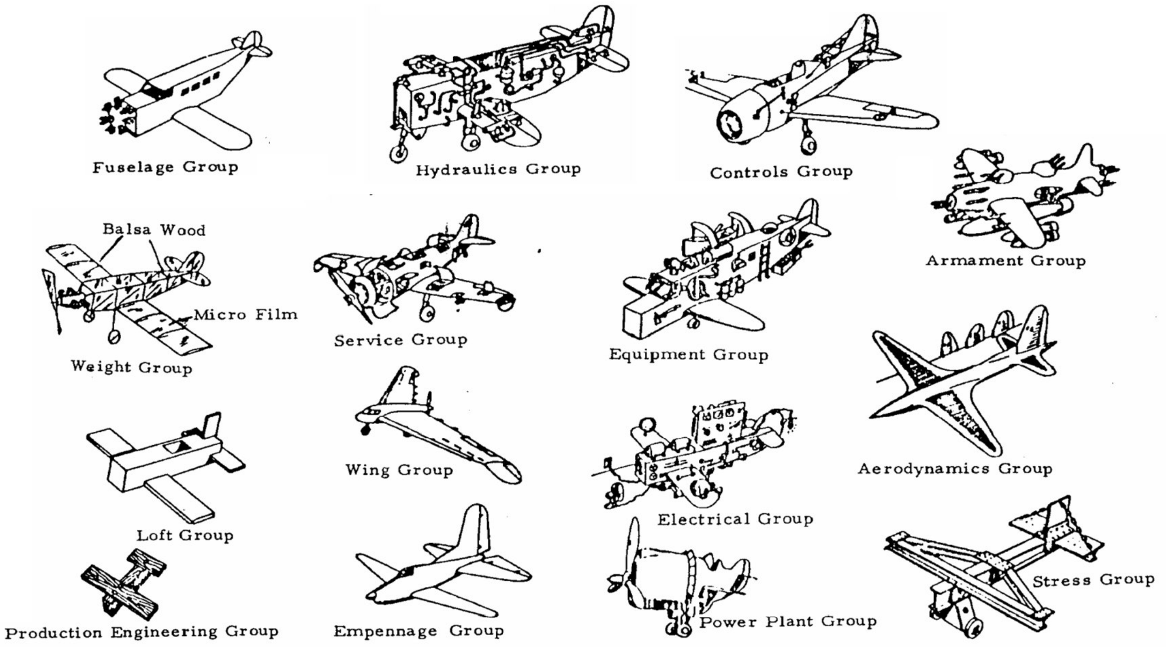


Figure 1.1: Aircraft design from each discipline's perspective (modified from [5])

The ongoing digitalization of the cabin systems development phase should result in ever-shorter development times and reduced costs. This is accomplished by collecting and formalizing the requirements and regulations for the individual systems, as well as the expert knowledge necessary for design interpretation. The difficulty here is to connect and interpret the resultant digitalized knowledge. The interdisciplinary approach of Model-based Systems Engineering (MBSE) employs a variety of techniques and modeling languages to overcome this challenge. This holistic approach to system development enables the creation of complex cabin systems by concurrently supporting the different phases of system development, from specification definition through system design, implementation, testing, and to product's entry into service. Furthermore, this approach should enable early error discovery, potentially lowering development costs. The consistent use of linked, digital system models and the interaction of models and physical systems offer many advantages over working with isolated digital sub-models, natural language documents and purely physical prototypes.

At DLR, researchers have devised a process for designing aircraft cabin systems, that automatically translates modeling findings to a virtual design platform. This interactive analytic environment offers early, virtual reality-based evaluation of design decisions. This thesis details how the process of developing aircraft cabin systems is extended to incorporate executable functional designs, as well as how the models are connected with one another within the digitalization process. It demonstrates how the model-based design approach facilitates the integration and reconfiguration of new technologies, fosters digitalization, and hence enables shorter development time frames.

1.2 Goals and Structure of this Thesis

The goal of this work is to provide a model-based methodology for functional system design and integration by means of the Passenger Service Channel (PSC). It will enable multidisciplinary model consideration and optimization based on top-level requirements and regulations. Before interconnecting the system model with other models in the cabin systems design process, a methodical derivation of a system architecture based on a grounded system analysis will be accomplished. Traceability will be realized in the system model by associating all system requirements and components within different abstraction levels. The approach will adhere to fundamental modeling concepts addressing the modeling process's agility and flexibility, as well as the exchange of knowledge between different models. Furthermore, the system architecture will be easily and quickly executable and reconfigurable, as well as interoperable within external models. Only so, an early design understanding and assessment can be achieved. Motivated by the goal of optimizing the overall aircraft, the modeling methodology will also enable the enhancement and optimization of each system representation in the various models through the exchange of relevant design parameters. Moreover, a suitable cabin system or component must be adopted to apply and evaluate the defined methodology for system modeling and integration.

This work consists of nine chapters. **Chapter 2** begins with an overview of the theoretical foundations of conceptual aircraft design and Model-based Systems Engineering. It covers both traditional and emerging digital aircraft design approaches. The cabin systems design process developed at DLR is then outlined. Additionally, an introduction to MBSE's emergence, advantages and challenges is provided. The modeling language and tool used in this work are detailed, as are existing modeling methodologies.

Chapter 3 outlines the methodology used to model and integrate the system architecture in this work. Therefore, it examines pertinent elements and procedures from established methodologies. The combination and extension of these elements in order to develop the methodology necessary to accomplish the work's objectives are then discussed.

Chapter 4 introduces the selected System-of-Interest (SoI), which is the Passenger Service Channel (PSC). A detailed explanation is provided for the selection of this complex, interdisciplinary and integrative system as a proof of concept for the work's objectives. The structure and various functions of this system, as well as the requirements for its design, are described. Furthermore, the cabin systems that interact with the PSC are discussed, as well as the corresponding interactions. The physical fundamentals underlying these interactions are also explained.

The methodology's application to the development of the PSC's model-based system architecture is detailed in **chapter 5**. How the model is organized and structured in order to facilitate concise and correct modeling is demonstrated. Additionally, the process of analyzing the system and deriving its functional architecture from the system requirements is explained. After presenting the emerging logical architecture, an overview of the model's traceability is provided.

Subsequently, **chapter 6** outlines the integration of the resulting system architecture in the cabin design process. The interface between the system model and external models is detailed. The exchange of design parameters and the interaction simulation is described.

Chapter 7 presents a trade-off study for the PSC's architecture. It defines a use case to assess and demonstrate the practical benefits of PSC's architecture modeling and integration. This use case comprises the PSC's optimization in connection with the electrical design. The implementation and results of this optimization study are outlined in this chapter.

Chapter 8 discusses the developed model-based methodology in light of the PSC application results. The advantages that were explored during the system analysis and architecture modeling methodologies are discussed. The findings from integrating the system architecture into the cabin design process are described and compared to similar integration research. Additionally, results of the trade-off and optimization study are analyzed and discussed.

Finally, **Chapter 9** summarizes the thesis' findings. Additionally, a roadmap for future work is described, including the use of MBSE for knowledge-based Multidisciplinary Design Optimization (MDO), the real-time interaction of system models with Virtual Reality (VR), and the evaluation of futuristic and revolutionary system designs and configurations in aircraft cabins.

2 Theoretical Basics of Conceptual Aircraft Design and MBSE

2.1 Digital Aircraft Design Process

The goal of this work is to extend and improve the existing digital design process. Therefore, a thorough understanding of the process as well as the theoretical fundamentals of aircraft design and knowledge-based engineering are essential. This will help situating the goals of this work within its scope and context, as well as identifying the necessary extensions in the process. The sections that follow provide an overview of these aspects.

2.1.1 Introduction to Conceptual Aircraft Design

Todays' aircraft design is interlinked with national infrastructure, global politics and natural resources. Thus, the trends in the aviation sector are closely related to the economic-political situation in the world. Recession, fuel price rises, the spread of pandemics, and international terrorism are all impacting civil aviation and influence the aircraft design strategies [6, p. 8].

Despite all of these external influences, the primary goal of aircraft design remains, in theory, to synthesize something new. Any new aircraft design must have a thriving edge over already manufactured ones. In recent developments, the major manufacturers have opted for a well established approach to aircraft design, the traditional one. To explain further, this process is primarily concerned with the geometric description of the new aircraft. Therefore, a three-view drawing, a fuselage cross-section, a cabin layout, and a list of aircraft parameters are used to describe the aircraft so that the requirements are optimally met [7]. Typically, this design process begins with a conceptual stage, where new concepts are investigated and evaluated based on a market analysis. The requirements are then defined and assessed based on their significance and the performance goals are established. Engine manufacturers also offer information on new engine options or answer to requests from the airframer. The most promising concept is chosen through trade-off analysis and the design engineers proceed to the preliminary sizing phase [8]. At this stage, design algorithms and codes that rely on aircraft physics and technologies, design assumptions and optimization rules, deliver characteristics of the aircraft as such as take-off and fuel masses, wing area, and take-off thrust. The detailed design of the aircraft takes place at the final stage of the process and can be performed on different detail levels depending on the level of accuracy sought and the available amount of data input and time. At this stage, not only the fuselage cross section and the cabin layout are specified, but also detailed parameters for aircraft structure and systems (e.g high lift system or landing gear) are defined. Already calculated masses, geometrical and aircraft performance parameters are then controlled and refined. Finally, the engineers can estimate the operation costs and continue with further certification, testing, and manufacturing activities [7].

Considering the traditional design process, the aircraft concept is chosen at an early stage, when information collected from detailed analysis in several disciplines, such as aerodynamics, propulsion, structures, or controls is unavailable. In these disciplines, high-fidelity models and studies, such as finite element method (FEM), are only performed once the configuration has been frozen. At that time, it is exceedingly difficult to make major adjustments to the designed aircraft concept and configuration. Additionally, classical design is defined by the fact that each of the specialist domains works independently from one another. Each discipline is optimizing its field based on its own interests and perspective without considering other disciplines. As a result, an optimal solution for aircraft design cannot be obtained. To overcome these challenges and enable a better configuration, new approaches for designing aircraft have been developed. The goal is to improve aircraft design patterns by combining high-fidelity analysis with numerical optimization techniques. In this context, the Multidisciplinary Design Optimization, or MDO, is gaining significant traction as a decision-support technique for digital aircraft design and optimization. MDO is defined by Martins and Lambe in [9] as a branch of engineering concerned with the application of numerical optimization to the design of optimal systems involving a number of disciplines or subsystems, taking into account that the performance of a multidisciplinary system is determined not only by the performance of the individual disciplines but also by their interactions. MDO has been widely used in the aviation industry to examine and optimize standard and non-conventional aircraft configurations. It addresses several design factors and disciplines and employs a variety of approaches depending on the optimization cases. Several recent applications of MDO in aircraft design are presented in [10], as well as the techniques and computational frameworks used. Furthermore, the European Union has funded the AGILE project to create the next generation of aircraft MDO processes, which aims to significantly reduce aircraft development costs and time to market, resulting in more cost-effective and environmentally friendly solutions [11].

Compared to standard design approaches, MDO or MDAO (Multidisciplinary Design Analysis Optimization) techniques and the technologies created to execute such an approach have been shown by numerous collaborative aircraft design and optimization applications to reduce the setup time by more than 40% [12]. The new project AGILE 4.0, which will run until 2022, builds on the preceding project by including architectural trade-offs and requirements engineering into the development process. Thus, it bridges the usual upstream systems engineering or MBSE phases (more details in section 2.2) to MDAO by leveraging digital engineering. The project structure and the conceptual framework developed by *Ciampa et. al* bridging the MBSE and MDAO are described in [13] and the effect of this investigation on this work will be discussed in chapter 8.

One of the goals of MDO is to gather information sooner in the design process while retaining freedom for a longer period of time. By resolving the optimization problem early in the design process, the designer can enhance the quality and minimize the design time and cost. Figure 2.1 illustrates these concepts visually. In fact, the designer has considerable leeway to make significant modifications throughout the early design stages, although his knowledge at the time may be limited. The second aim is to make a large quantity of information from many designers accessible and to build mechanisms for sharing this knowledge in a prolific manner, so that it may be used by other designers during their early development phases. Knowledge-based Engineering (KBE) is a branch of engineering that investigates this specific issue and searches for ways to improve information and knowledge sharing. The next section discusses the fundamental principles of KBE and presents the information exchange platform of the aircraft design that is used in this work.

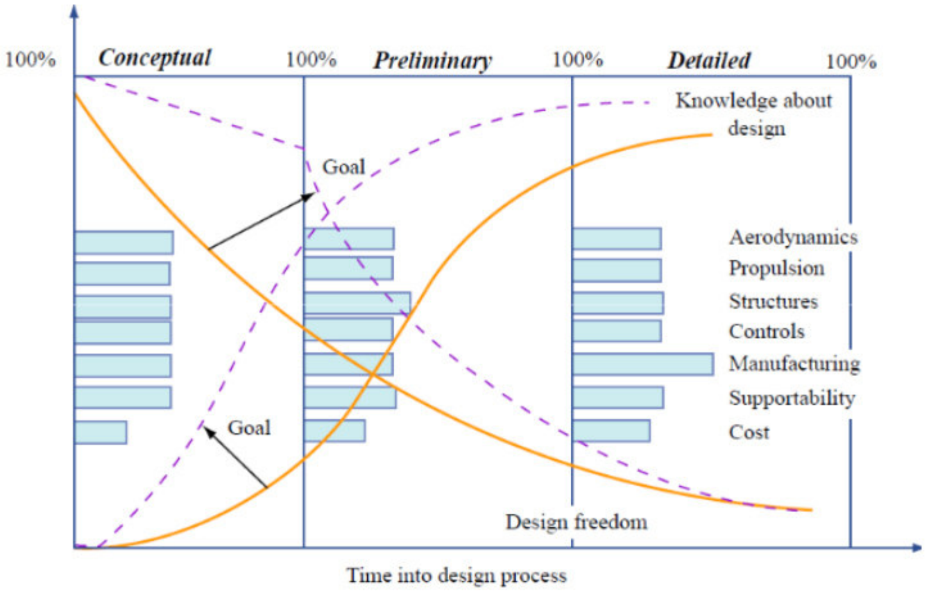


Figure 2.1: Design freedom and knowledge goals in the design process [14]

2.1.2 Knowledge-Based Engineering and CPACS

The success of a design project is based on two stages: the efficient exchange of information necessary to develop a productive solution and the efficient capture of the resulting solutions [15]. The conventional method of communicating and transmitting critical information does have some major limitations. In a multi-disciplined context, characterized by highly integrated products and processes, design efforts can be squandered if they do not consider exterior design concerns or are unable to be manufactured. Additionally, the accessibility of the overall product cycle experiences, including manufacturing, is very compromised. An engineering approach known as Concurrent Engineering (CE) allows not only for the different knowledge and expertise within the engineering team to be used to accomplish multiple activities simultaneously, but also to integrate project considerations early in the development process. Clearly, in an effort to improve efficiency.

However, CE cannot function unless all designers and engineers involved in the product cycle are in agreement in terms of proper information and knowledge exchange. In an engineering context, knowledge is regarded as processed information with the ability to act effectively. This information can be thought of as data inside a structured context, which is generated in different forms (documents, models etc.) during the product's development, production and operation [16]. Knowledge management and sharing within the different sectors of one company is crucial to benefit from competitive advantage in the market. Moreover, increasingly available information technology and a growing variety of knowledge projects have driven the growth of knowledge systems. These systems use research into managing, organizing and recirculating product life cycle (PLC) knowledge within an organization. To put the concurrent engineering concept into effect, it is critical to leverage the PLC knowledge early in the design process to better understand product lifespan. In this regard, knowledge-based engineering is one of the most appealing application domains.

La Rocca defines KBE as a technique that is based on specialized software tools called KBE systems, that enable the capture and reuse of product and process engineering knowledge [17]. KBE's primary purpose is to reduce the time and expense of product development by automating repetitive, non-creative design chores and assisting MDO throughout the design process. Therefore, object-oriented programming (OOP) (detailed definition in section 2.2), artificial intelligence[1] (AI) and computer-aided design[2] (CAD) technologies are used in this method to support the automated collection of data, the structuring of knowledge, and its maintainability and usability throughout the PLC [15] [16]. Ontology and logic are the two basic pillars of formal knowledge representation in knowledge based systems. An ontology is an explicit description of a domain that represents the domain's ideas and associations in a structured way. A formal language, such as predicate logic, is typically used to represent an ontology's concepts and relationships, allowing the structuring of knowledge bases on which reasoning mechanisms can execute logical operations [16].

Due to product complexity, diversity, and huge investment volumes, the aerospace sector is KBE's main driver alongside the automobile industry. Furthermore, the extended product cycles of aircraft necessitate long-term knowledge capture in an organization to minimize the loss of know-how due to staff fluctuation, rendering KBE suitable for aeronautics. As a strategy to enhance the interdisciplinary collaboration and the creation of decentralized MDO architectures, researchers highlighted the importance of knowledge-based data models. Meaning, the efficiency would greatly improve if a knowledged-base system acted as a common language. This concept has been validated by the use of the Common Parametric Aircraft Configuration Schema (CPACS) in advanced preliminary aircraft design activities, including MDO [20].

The data model CPACS has been introduced in 2005 by the German Aerospace Center (DLR)

[1] AI is the science and engineering of making intelligent machines, especially intelligent computer programs. Every aspect of learning or any other feature of intelligence can in principle be so precisely described that a machine can be made to simulate it [18].

[2] CAD is the use of computers (or workstations) to assist in the development, modification, analysis, or optimization of a design [19].

and enables the multi-disciplinary information exchange between different tools. It describes hierarchically the characteristics of air transportation systems and products (e.g. fuselage, engines, materials, fuels etc.) and also designs processes and workflows for analysis [20]. The CPACS schema is accessible for free on the GitHub platform and on a dedicated webpage. Along with substantial technical documentation, it also contains additional information about the standard's application. Regular updates to the CPACS schema are also provided, which include the addition of new elements and concepts [21].

The basis for CPACS is the XML (Extensible Markup Language) language, with CPACS implemented as an XML Schema Definition (XSD). XML is a markup language created and published by the World Wide Web Consortium (W3C) [22]. It defines a set of criteria for encoding texts in a human-readable and machine-readable manner. Programmers have created several application programming interfaces (APIs) to help process XML data. As the name suggests, XML schemas describe the structure and content of XML documents, beyond the fundamental syntactical limitations that are imposed by XML itself [23]. CPACS benefits from the strength of the open standard XML, which is widely recognized as a modeling language in the field of information technology. Because XML is so general, it can be used as a computer-processable meta-language, allowing for the development of an aircraft ontology as a markup language. Additionally, it decouples the data structure from the content and enables the semantic[3] rules to be defined in a separate XSD file. Thus, it enables in the scope of CPACS the combination of aviation-specific ontology and tool- and process-specific data, by supporting the use of a diverse set of open source editors and APIs. It is essential to note that XML is used solely for the purpose of storing and transporting data. Because executable program code is not included in an XML document, reading and writing XML data must be performed by an external program [20].

Using XML's hierarchical data representation, CPACS's structure decomposes a general definition (like an airplane) into more precise descriptions of its components (see Fig. 2.2). This comes from the conceptual and preliminary design of aircraft, where the degree of detail starts off low and increases over time. To create higher-level CPACS concepts, however, a bottom-up method is used, where components are first specified in depth, then connected together. The middle-out approach combines these two approaches to completely parameterize aeronautical systems. The challenge in developing and improving CPACS is to constantly find a balance between the flexible bottom-up and the strict but semantically more relevant top-down approaches [20].

As already mentioned, XML schema definitions are used to represent complex data models in XML documents. In addition to element and type definitions, the XML schemas also contain specifications for the hierarchy of elements. There are defined characteristics for each CPACS type definition, which are indicated in the schema visualization by a blue " @ " symbol (see fig. 2.2). A fundamental element property shared by nearly all CPACS elements that appear

[3] The distinction between syntax and semantics is that syntax refers to the rules that govern every statement in a programming language and may be thought of as the formal relationship between signs, whilst semantics refers to the meaning associated with any statement in a programming language [24].

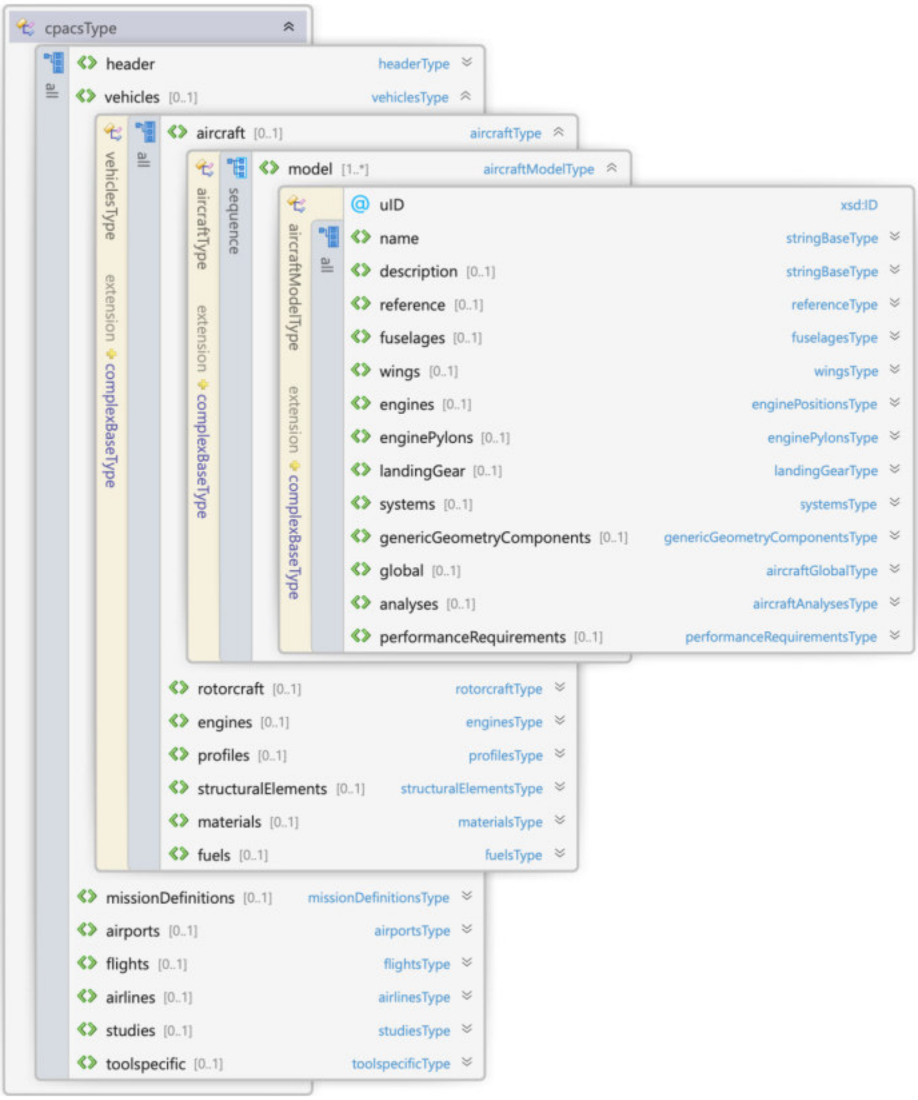

cpacsType
all
header headerType
vehicles [0..1] vehiclesType
vehiclesType extension complexBaseType
all
aircraft [0..1] aircraftType
aircraftType extension complexBaseType
sequence
model [1..*] aircraftModelType
aircraftModelType extension complexBaseType
all
uID xsd:ID
name stringBaseType
description [0..1] stringBaseType
reference [0..1] referenceType
fuselages [0..1] fuselagesType
wings [0..1] wingsType
engines [0..1] enginePositionsType
enginePylons [0..1] enginePylonsType
landingGear [0..1] landingGearType
systems [0..1] systemsType
genericGeometryComponents [0..1] genericGeometryComponentsType
global [0..1] aircraftGlobalType
analyses [0..1] aircraftAnalysesType
performanceRequirements [0..1] performanceRequirementsType
rotorcraft [0..1] rotorcraftType
engines [0..1] enginesType
profiles [0..1] profilesType
structuralElements [0..1] structuralElementsType
materials [0..1] materialsType
fuels [0..1] fuelsType
missionDefinitions [0..1] missionDefinitionsType
airports [0..1] airportsType
flights [0..1] flightsType
airlines [0..1] airlinesType
studies [0..1] studiesType
toolspecific [0..1] toolspecificType

Figure 2.2: Extract from the CPACS schema [25]

many times in the design is a unique identifier, abbreviated ID. The ID can be used to refer to items. This enables the usage of a previously specified profile in many locations across the CPACS specification.

In this work, the focus is on digital enhancement of cabin and cabin systems design. Although CPACS is used largely in aircraft preliminary design, simple descriptions of cabin are also feasible. A physical link between the airplane level and the cabin is required for a proper structural description. In this regard, CPACS serves as the connection between aircraft preliminary design and cabin design [26]. The following section discusses the process of moving from preliminary design data to detailed cabin design and optimization. It describes the current process that this work aims to expand and enhance.

2.1.3 Cabin Systems Design Process

It stands to reason that the aircraft cabin offers great potential for increased efficiency and new opportunities, as well as for exploiting system synergies. In other words, new cabin layouts and breakthrough technologies will be developed at a faster rate than before. However, there are too many subsystems and interdependencies, as well as safety constraints, to allow for rapid development cycles. In order to achieve a comprehensive and significant improvement in efficiency, a thorough understanding of the nested interactions between the many subsystems is required. The German Aerospace Center (DLR) is researching several approaches and methodologies for the digital design and development of aircraft cabins. The goal is to integrate new technologies faster, evaluate and better understand the interrelationships inside the cabin and better comprehend the systems architecture [27] [28].

An automated digital cabin systems design and visualization process has been implemented at the DLR's *Institute of System Architectures in Aeronautics* for this purpose and is shown in figure 2.3. The process begins with the import of preliminary aircraft design parameters. Because of the CPACS schema's versatile description options for primary structures, this one is used as a platform for design data import or exchange [29]. Taking into consideration that the information imported from CPACS has been generated in previous aircraft design activities and is required before starting with the cabin design process. Floor elements such as longitudinal and transverse beams, seat rails and supports can be defined in addition to stringers, frames and skin panels. There are also descriptions of pressure bulkhead and wing-fuselage transition structures. Openings in the structure such as doors, windows or loading hatches can also be described via cutouts, and optionally provided with border structure. For the cabin description with CPACS, there is an extensive node called "*decks*". Based on a simplified description of the outer boundary of the cabin, using contour lines, seats and large cabin modules and monuments, such as galleys and lavatories, can be placed and extended with information such as door positions and evacuation clearance areas [26].

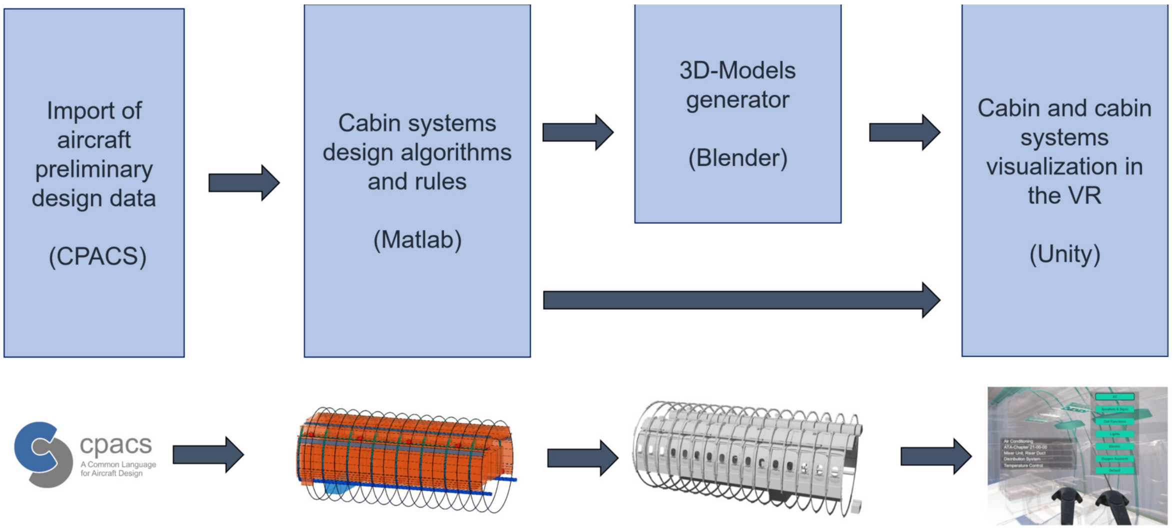

Import of aircraft preliminary design data
(CPACS)
Cabin systems design algorithms and rules
(Matlab)
3D-Models generator
(Blender)
Cabin and cabin systems visualization in the VR
(Unity)
cpacs
A Common Language for Aircraft Design

Figure 2.3: Cabin systems design and visualization process (according to [28])

The data of the aircraft structure can be automatically imported into Matlab's[4] cabin systems design and layout model via a python[5] interface that can read CPACS data, identify elements, extract target attributes and values and finally instantiate them in the Matlab Model. *Beckert* in [25] provides a detailed description of the implemented automation of data exchange interfaces in the whole process (each of the transition arrows in fig. 2.3). The cabin systems design model's goal is to produce the cabin architecture, including systems, by utilizing basic geometric forms to represent the cabin components. Thanks to its ability to handle numerical problems and to employ comprehensive graphical functions to illustrate results, the commercial software Matlab is used for this implementation. It also allows for the use of object-oriented programming principles, which play an important role in the design process [32]. An object-oriented structure is used to design the cabin and its systems. As a result, each cabin component type is given its own class. This is then used to create objects for the needed cabin components, with component-related information stored as attributes (properties). This provides information such as the produced cabin object's name, construction dimensions, location, or affiliation. In addition to the cabin component class, there are two other classes: links (relationships) and requirements (safety requirements and human factors). Specific characteristics are saved in them as well [27]. Moreover, algorithms and guidelines have been created that consider regulation authorities requirements and user needs, assembly costs as well as human factors and cabin functionalities. Each of these features is given its own model, allowing for multi-disciplinary design optimization in the next stage [33]. Several designs are generated by altering a number of parameters and then assessed by imposing constraints on the design variables. An optimization algorithm is used to select the design parameters that result in the best design by meeting the objective functions. *Fuchs et al.* provides a comprehensive example of using MDO to design and layout a cabin component in [34].

A virtual container manages all newly generated items. This contains the object's identification number (ID), which facilitates an accurate assignment. This allows particular objects to be accessible if they are required for the installation of additional components or for a link. In addition to central management of all cabin items, link objects and requirement objects are also kept in a separate virtual container. When two components have a relationship with each other, a link object is generated. The type of link is also saved as a property. The generated virtual containers are exportable into an XML format using an export function, allowing the data to be further processed in other programs [27].

The cabin system layout and design optimization results in objects with simple geometric shapes (simple boxes). In the following step, high-resolution 3D modeling of these objects

[4] Matlab is a commercial software program developed by the American company MathWorks that solves mathematical problems and graphically displays the results. Matlab is primarily intended for numerical calculations involving matrices, hence the name: MATrix LABoratory [30].

[5] Python is a high-level, general-purpose programming language that is widely interpreted. It promotes an easy-to-read, concise programming style. Python is compatible with a variety of programming paradigms, including object-oriented, aspect-oriented and functional programming. It also supports dynamic typing. Python, like many dynamic languages, is frequently used as a scripting language [31].

is automated using the Blender graphics software[6]. 3D models from a library are modified for each object, or scanned and digitized models of existing physical cabin elements are used [25]. The implemented method at DLR to scan and digitalize cabin objects is described in detail by *Rauscher et al.* in [36]. Finally, the generated data from the design model and geometric object models are imported into the virtual environment to be modeled, displayed and controlled in the virtual reality (VR) using appropriate hardware and the development software Unity[7]. A standard scene is initially created for the virtual cabin environment. The lighting of the scene, as well as the script for the import, are saved here. The XML file, which is generated from the design model in Matlab, and the constructed Blender CAD file can be imported using the import script. An additional script component enables interaction between the user and the virtual cabin. The functional procedures for using the controller, as well as methods for movement and camera settings, are specified. If the user walks in the virtual world and exhibits a particular item with a pointer, he can interact with it. Both the object's information (name, dimensions, ATA chapter) and relationships to other things can be shown. To implement this, the script returns to the saved XML file and searches for the appropriate links using the Object ID. When a matching ID is found in a link object, all subsequent related cabin objects are colored [27]. Further functions to interact with the cabin in the VR can be flexibly implemented. For example, *Fuchs et al.* describes in [28] how the cabin objects can be manipulated and directly visualized within the VR. In order to integrate new technologies more quickly and effectively, one objective of this approach using the VR is to make the interdependence between the different system components obvious. Consequently, the cabin system becomes easier to comprehend, and a variety of issues may be handled at an early stage. As a result, cross-system dependencies may be taken into account throughout the design phase, preventing costly adjustments in the future [28].

Obviously, the digitalization process described in this section is based on mapping different design aspects with the help of models and then linking these models with each other in a subsequent step. Similarly, the goal of this work is to extend this process with the help of additional models and to connect them with one another. These models will serve the cabin systems' functional analysis and representation and enable further optimization regarding other system properties (e.g. mass, power etc.). The use of models and their integration is thus critical. The section that follows explains why models have become so important in the digital development of future systems.

[6] Blender is a free and open source 3D creation suite that supports the entirety of the 3D pipeline modeling, rigging, animation, simulation, rendering, compositing and motion tracking, video editing and 2D animation pipeline [35].

[7] Unity is a game runtime and development environment developed by the San Francisco-based company Unity Technologies. Platforms targeted include not only PCs, but also game consoles, mobile devices and web browsers. The development environment enables the generation of computer games as well as other interactive 3D graphics applications [37].

2.2 Introduction to Model-Based Systems Engineering MBSE

This work is about the development of a model-based methodology to extend the digital design process, as the title implies. To understand why models are used in this notion, the emergence of MBSE must be explained. The sections that follow will illustrate this and provide an outline of the benefits of MBSE as well as the current issues in this domain.

2.2.1 Emergence of MBSE

A system is defined as a combination of interacting elements to achieve one or more set goals. These elements include products (hardware, software and firmware), processes, people, information, procedures, facilities and services. Systems are created by people and are designed to provide products or services in a determined environment for users and other stakeholders[8][39][40]. When it comes to managing systems, the complexity is increasing quicker than the ability to do so. Information about projects gets rapidly lost, development expenses and the possibility of late-discovery of design issues are on the rise. In addition, the rapid technological evolution combined with the incorporation of new technologies creates system vulnerability and possible obsolescence. The creation of a comprehensive methodology is required to overcome these problems and enhance system development [41]. In order to deal with these challenges, the so called Systems Engineering (SE) has emerged as an entire and independent engineering discipline and its importance in the development of todays' systems has increased significantly. The mission of Systems Engineering is to assist in the development of systems throughout their life cycle, from conception and implementation to integration and operation. By managing complexity and risk effectively, this multidisciplinary approach enables the development of successful overall systems and provides a solution that meets needs of the stakeholders. Currently, the International Council on Systems Engineering (INCOSE) is the world's largest organization for systems engineering. More than 17,000 members of this non-profit organization work together with aiming to develop and promote systems engineering on an international scale [42].

Not only in this work, but in the context of SE in general, the term "System Architecture" and the activities related to it play a significant role. *Dickerson et al.* defines a system architecture as the "*[...] organization of the system components, their relation to each other and to the environment, and the principles guiding its design and evolution*" [43, p. 8]. For a successful system, a system architect must carefully design the system architecture. He is in charge of synthesizing the system architecture by breaking it down into components and describing their interactions and interconnections. Allocating system needs and developing technical specifications for these components are also part of this process. Hence, numerous tasks must be completed, each utilizing an effective approach. At the end, the resultant system design is always just one solution to a particular issue. The resulting architecture must then be described and communicated to the appropriate stakeholders. An architectural description

[8] A stakeholder is a person or group that has a vested interest in the course or outcome of a process or project [38].

must explain how a design satisfies stakeholder requirements and needs. Additionally, it must demonstrate how architecture assessments result in both rationales and architectural decisions [44][45].

To facilitate in the comprehension of systems architecture and other aspects of systems engineering, a process model of the associated activities and stages is needed. Various models, often from the software development industry, have been modified to explain or define complex system development lifecycles, including models such as the traditional stage-wise and waterfall models, as well as the spiral model, or development standards such as VDI2221 [46]. However, the so-called V-Model has received the most significance and attention in SE applications. The V-Model, which illustrates the macrocycle of product development, represents the concept of interconnecting all disciplines engaged in engineering activities (see. fig. 2.4). An elemental system is decomposed into its constituents on the left thigh and gradually integrated into the entire technical system on the right thigh. Aside from these two "V" thighs, the product's characteristics are continually evaluated and verified. So the "right" system (validation) is created in the "right" way (verification). Requirements are displayed graphically as an input box and system implementation are shown at the bottom of the model. The V-Model represents the whole product, since mechanics, electrics/ electronics, software as well as pneumatics, hydraulics, optics and other disciplines are all used in mechatronic and cyber-physical systems (CPS) engineering. A model is graphically framed by a bracket representing modeling and model analysis that runs between the two thighs. Arrows between the two V-Model's thighs depict characteristics requiring verification and validation [47].

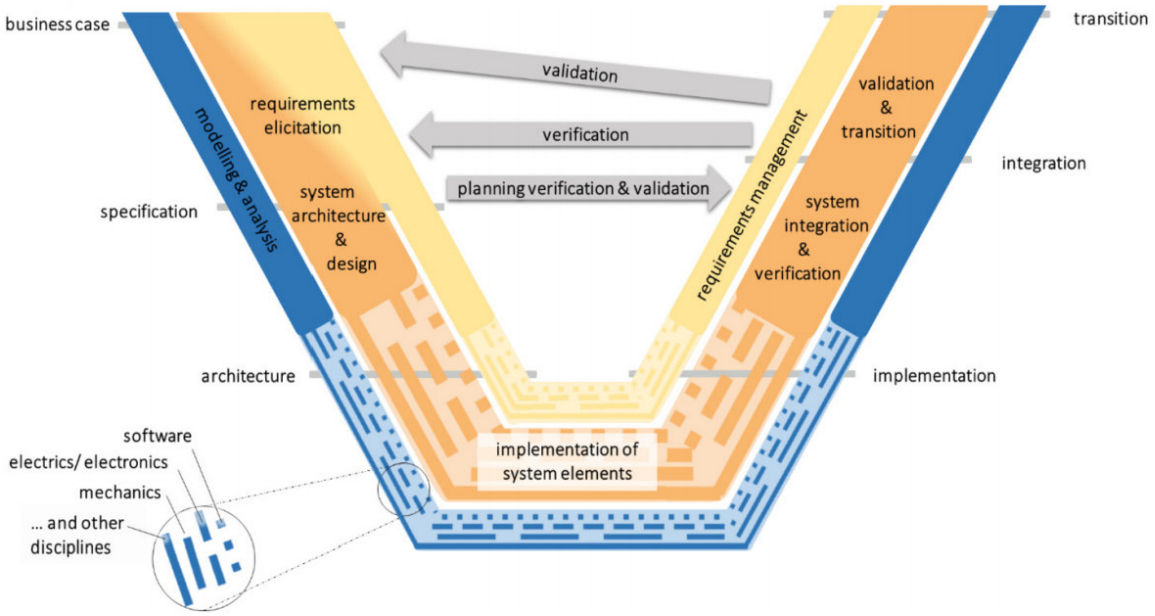


Figure 2.4: V-Model as presented at design 2018 conference in Dubrovnik (according to [47])

Initially, SE's primary focus was on projects that required a single highly innovative, very large and complex product or mission. This classic SE is highly process-oriented, with top-down decisions and a clear hierarchy. It focuses on high quality and adheres to the zero-defect quality principle in order to provide customized solutions and optimized systems [48]. The traditional

SE processes provide a proven framework for managing complexity, but they are no longer adequate for todays' market demands. The needs for speed, flexibility, personalized solutions, and cost-effective solutions (together referred to as mass customization) necessitate a renewal and pragmatic interpretation. SE needs modern features to complete the classical approach. Systems and processes that display agility are able to operate effectively in unpredictable, uncertain and changing environments. At this point, system agility must be distinguished from that of the systems engineering process. While agile systems are defined by their ability to change or be altered quickly and cost effectively, the term Agile Systems Engineering refers to something quite different [49]. The engineering process should be intended to be agile, or adaptable, in order to accept and apply new or amended requirements during development.

The principles of agile architecture design are defined by INCOSE in its Handbook for Systems Engineering [40]. It differentiates between the ideas of reusability, reconfigurability and scalability. Encapsulated modules and loosely linked modules that are reusable and reproducible and share well-defined interaction and interface standards are characteristics of reusability principles. The concepts of reconfigurability provide a greater emphasis on peer-to-peer interaction and communication across process modules, which are guided by aims rather than techniques, so that knowledge is linked locally and available globally. Work activity, response assembly and response deployment should thus be postponed until the last responsible minute to avoid costly lost effort that may also hinder a future successful reaction. Scalability principles advocate for developing standards for module compatibility, redundancy for better failure tolerance and method variety.

To support the different life cycles stages and implement the agility principles in modern systems engineering processes, new methods have been developed and implemented. The document-based technique has been employed for decades and has limitations in the development process. Achieving stakeholder alignment with requirements or system aspects is difficult when designing and building complex aviation systems (engines, control systems, etc.). Besides, the intricacy makes it tough for designers to choose the best system design option. Requirements documents, interface standards, development guidelines and similar artifacts have traditionally defined requirements. However, these papers are created independently by different departments inside the business, complicating effective and programmatic engineering decision making. Because the information is dispersed throughout the several documents, connecting the requirements, design and test is only achievable with enormous effort [50].

For this reason, the transition to Model-based Systems Engineering was carried out to enable improved support in all stages of the development process and synchronization between all aspects in the V-Modell [51]. INCOSE defines MBSE as the "*[...] the formalized application of modeling to support system requirements, design, analysis, verification and validation activities beginning in the conceptual design phase and continuing throughout development and later life cycle phases*" [52]. After all, the industrial usage of models is not a new trend, as model production has been used for years to not only support discipline-specific or cross-disciplinary design, but also enriched other fields. The model-based approach has not been reinvented with MBSE. However, the use of the term model or system model in the context of MBSE must be more explicitly specified at this point. In his book, *Stachowiak* defines it as an abstraction of an actual or to-be-realized system. By reducing information about a notion to the essential components, this abstraction recollects elements of the original system and can be used instead of the original for a specific purpose [53][44].

INCOSE and major industries regard MBSE as critical to the development of next-generation systems and products [52]. This is owed to various advantages and potentials of this approach, which will assist in overcoming future challenges. The following section discusses these benefits and describes MBSE's challenges based on actual industrial applications and experiences.

2.2.2 MBSE advantages and challenges

The interaction between components from various technical areas is one of the characteristics of aeronautical systems. Because of these interactions, the system is able to perform more functions than the sum of its components, resulting in a rapid increase in system complexity. The MBSE approach assists systems engineers in managing this complexity as well as enabling better quality, lower risk and lower costs. Many advantages of MBSE have been discussed in the literature and some of them are summarized in the following points:

- A common reference model for a project standardizes communication and collaboration across project stakeholders. It makes it easier for the teams to extensively explore the system's conceptual and configuration spaces as well as to discover and analyze critical factors in the evaluation of system alternatives [54].

- Small, iterative activities that can demonstrate incremental benefit and obtain early and ongoing feedback are supported by the modeling. It is possible to discover mistakes and inconsistencies sooner since requirements, rules and regulations are incorporated into the model. There is a clear correlation between the timeliness of mistake discovery in a development process and projected expenses for fixing these [41]. *Delligati* describes in his book proper MBSE practices as a remedy for inconsistency and as a method of doing systems engineering that offers a higher Return on Investment (ROI) than document-based SE [55].

- To avoid costly system redesigns at the end of the design life cycle, visualization and simulation help uncover incompatibilities in interfaces and assembly procedures early in the process, before hardware expenses are fully committed. By utilizing several integrated modeling technologies, MBSE assists companies in establishing digital design traceability across their systems [56].

- The use of models in SE activities allows for the detection of missing concepts or objects, as well as the emergence of capability from data (can be planned or identified by surprise). It enables more analytical capability and assists engineers in locating problems earlier, faster, better and cheaper [57].

Although most authors in the literature agree that MBSE, due to its numerous advantages, is the solution for developing systems of the future, its adoption in real-world applications continues to face significant challenges. Organizations and industries are unable to fully apply it without issues. *Chami et al.* gives a good overview about this in his survey on MBSE adoption challenges in [58]. Major challenges of MBSE are the clear definition of its purpose and scope, the awareness and change resistance within the organization, the clear set up of required methodology and needed extension, and the tool dependency and integration. As a result, research must be undertaken using a holistic, accurate and consistent methodology in order to implement and integrate MBSE. A unified and standardized modeling language must be made available to address obstacles in intercommunication between models in projects. In this context, the most extensively used language for MBSE application is known as Systems Modeling Language (SysML). The next section introduces the major language elements and concepts, as well as the software tools that facilitate the implementation of MBSE or the use of modeling languages.

2.3 Modeling Language and Tool

One of the key parts of this work is the use of a standard language. Like any other language, it contains semantics and elements that define it and enable its usage. The sections that follow provide an insight into SysML, starting with a brief historical overview. Besides, tools for applying the language's parts and concepts are also described. Because the model is designed to be integrated into this work, this section also includes descriptions of other research that examines comparable integrations.

2.3.1 A brief story of SysML

INCOSE decided in 2001 to make the Unified Modeling Language (UML) the standard modeling language for SE. UML was originally intended as a modeling language for software development, and it was already extensively distributed and utilized in this sector, as well as in SE to a lesser extent. To avoid complicating the language, they opted against adding a SE viewpoint to the UML for which tools, skilled engineers and best practices were already available. Instead, a new modeling language was created using the UML profile extension, and as a consequence, the Systems Modeling Language was released as an Object Management Group (OMG)[9] standard in 2006. The SysML is therefore a UML profile, that is, a particular extension of UML, even though it is considered as a modeling language in its own right [44]. However, many elements of the UML, which were primarily designed for software development, have been discarded in the definition of the SysML standard, while a few elements have been newly defined for the SysML (s. fig 2.5).

[9] OMG is an international, open membership, not-for-profit technology standards consortium. Founded in 1989, OMG standards are driven by vendors, end-users, academic institutions and government agencies. OMG Task Forces develop enterprise integration standards for a wide range of technologies and an even wider range of industries [59].

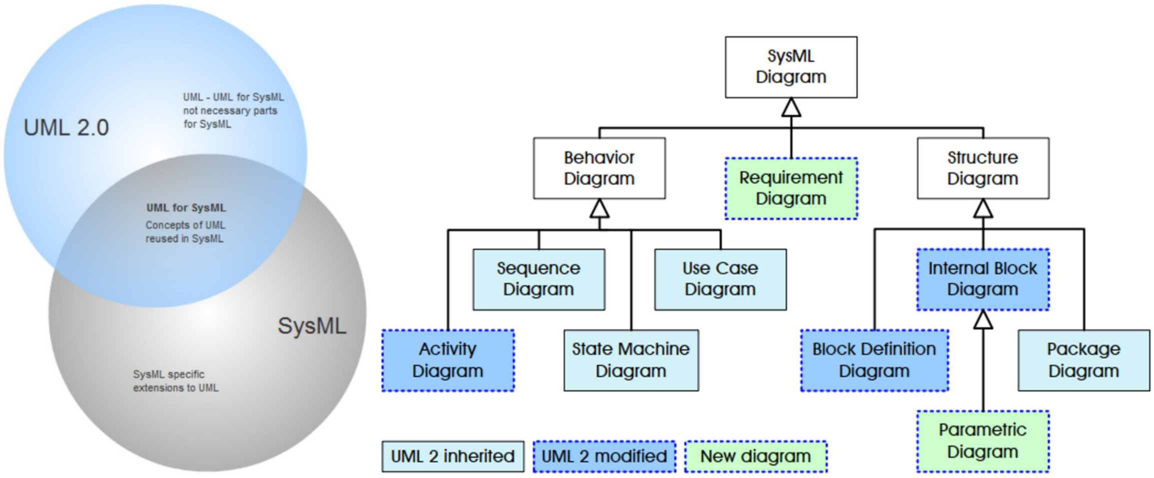


Figure 2.5: UML extension through the SysML (modified from [60][61])

OMG defines the SysML as "*[...] a general-purpose graphical modeling language for specifying, analyzing, designing and verifying complex systems that may include hardware, software, information, personnel, procedures and facilities*" [62, p. 27]. The language is meant to assist in the description and the architecting of systems. It is also there to specify components that can subsequently be developed using other domain-specific languages, such as UML for software design, VHDL[10] for electrical design and 3D geometric modeling for mechanical design. By using SysML, the MBSE technique can be applied to construct a coherent and consistent model of the system [51]. SysML should not be viewed as a replacement for existing system development tools, but rather as a useful supplement to them. SysML defines a shared basis and a common foundation for the many disciplines that collaborate in system development such as requirements management, electronics and mechanical development, design, system analysis. Based on this, further development can be done simultaneously to support concurrent engineering [45].

Since its first introduction, the SysML has continued to improve over time through numerous modifications and new specifications that introduce new concepts and aspects to the language. The goal remains to gain adaptability towards ongoing challenges and to better support the development of future systems. Current industry aims, such as digitalization and traceability, are critical in the creation of the new version of the system modeling language SysML.v2. This version of the SysML is an independently specified language for the purpose of SE and is no longer a profile of UML. Thus, the flavor of a software modeling language is no longer attached to it, thing that has caused many problems in past experiences and reduced the acceptance of the language in the user community. An incredible amount of time has been invested in analyzing SysML.v1. and to working out the disadvantages and obstacles in its applicability [64]. The new version is intended to correct the shortcomings of the previous language version's formalization and to provide an expressive and exact metamodel (detailed

[10] VHDL is a hardware description language that can represent the behavior and structure of digital systems at many levels of abstraction, from the system level down to logic gates, for design entry, documentation and verification. The Institute of Electrical and Electronics Engineers (IEEE) has been standardizing VHDL since 1987. [63].

explanation of metamodel in section 2.3.2). In addition to that, many new systems aspects are considered in the new language version such as the geometry for a better linking to CAD models. Furthermore, it specifies a standardized application programming interface (API) that enables integrated system modeling as part of a multidisciplinary development environment. As a result, interoperability across multiple and cross-disciplinary models is strengthened to allow a holistic digital product life cycle [65].

The version SysML.v1.6 is used in this work because full adoption of the SysML.v2 has not yet occurred and no tool supports the implementation of this version of the language. The following section introduces the essential ideas and concepts of SysML.v1.6 and defines the elements and diagrams required for its use.

2.3.2 The language concepts and elements

The object-oriented nature of SysML comes from the fact that it's built on UML, which is a language for describing object-oriented software. As a result, it is essential to understand the language and the fundamentals of object orientation (OO), particularly the terminology. The most important OO-terms and concepts are presented shortly below and are represented in figure 2.6:

- **Classes and objects:** objects are generated from the same blueprint, which is refered to as a class. Meaning, a class is a data type. It determines which variables the associated objects have. Objects produced with this blueprint may have different characteristics (i.e. the values of their variables) (see. figure 2.6).
- **Parts:** The difference between objects and part elements is that objects can exist on their own. Parts, on the other hand, are always a part of something and can thus exist only if the element containing the part element exists.
- **Encapsulation:** The combination of variables and methods in a class serves to hide the variables from the outside world in particular. The data type and name of the variables are unknown outside the class, and access is only allowed through the class's methods. Variables can therefore be safeguarded from being assigned wrong values.
- **Inheritance:** To enable modularization and thus simplification of modeling, OO provides the concept of inheritance, which allows such more general properties to be outsourced to separate classes and then used in more specific classes. So, classes can inherit properties from other classes, just as children inherit properties from their parents.

There are no classes, objects, or parts in SysML; they have been given alternative names. Here, classes were transformed into blocks, parts into so-called properties and objects into instance specification elements [45]. The properties play a significant role in the specification of a block and are classified as follows [55, p. 28][51]:

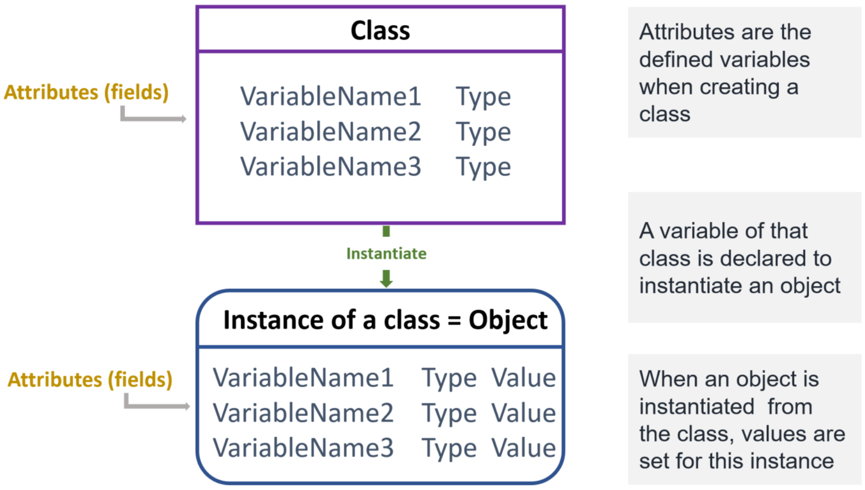

Class
Attributes (fields)
VariableName1 Type
VariableName2 Type
VariableName3 Type
Attributes are the defined variables when creating a class
Instantiate
A variable of that class is declared to instantiate an object
Instance of a class = Object
Attributes (fields)
VariableName1 Type Value
VariableName2 Type Value
VariableName3 Type Value
When an object is instantiated from the class, values are set for this instance

Figure 2.6: Concepts and terms in the context of OO (modified from [66])

- *part property:* composition connections between blocks are described by parts. A part specifies how its type is used in a context. The fundamental difference between a part and an instance is that a part describes the block properties in the context of its composite blocks, while an instance does not.

- *reference property:* represents an external structure to a block. Unlike a part property, it does not transfer ownership. A reference property is a "needs" association; a block with a reference property provides a service or exchanges material, energy, or information.

- *value property:* is a measurable characteristic that employs a certain value type. It is used to specify the quantitative properties (e.g. speed or volume) of a block. They can also be used to represent vector values like location or velocity. A value property is defined by a value type, which defines the range of acceptable values that the property may take. Although a value type does not need a quantity kind or unit, SysML provides these notions that may be used to further describe it. Value properties can be linked with default values, and they can also specify a probability distribution for their values.

- *constraint property:* is a relationship (an equation or an inequality) placed on a collection of value attributes and are used to build mathematical models of a system. This is a significant level of model fidelity that is often needed for modeling tasks.

The introduction of the term "stereotype" is necessary before defining a profile. Abstract syntax specifies the concepts in the language, their relations and a set of rules for how the concepts can be combined. A metamodel is used to explain the abstract syntax of a modeling language. Individual notions in a metamodel are represented by metaclasses, which are linked to one another using generalizations and associations in the same way that blocks may be associated to one another on a block definition diagram [51, p. 365]. Each metaclass contains a description, a collection of metaclass characteristics that describe the idea it represents, and a set of constraints that put restrictions on the values of those attributes. A stereotype is a customized version of any SysML element that exists within the SysML meta-model

[62, p. 87]. A stereotype must be based on an existing SysML element; creating a new element from scratch is not feasible. The elements can be given new names, qualities and symbols as a result of stereotyping. Thus, SysML can be customized to meet the needs of various domains and applications. UML reference elements (e.g. metaclass "Class") can be found at the top level, from which SysML elements have been generated. Finally, a profile is a sort of package that includes stereotypes, metaclasses and their interrelationships. Figure 2.7 illustrates an example of a profile (called SimplifiedReqML), and its sub profiles (SRMLNodes and SRMLRelationships), metaclasses (Class and Dependency) and defined stereotypes (Requirements and Refinement). The attributes of the stereotype are also defined and typed (e.g. the attribute "importance" with the enumeration type "ImportanceLevel").

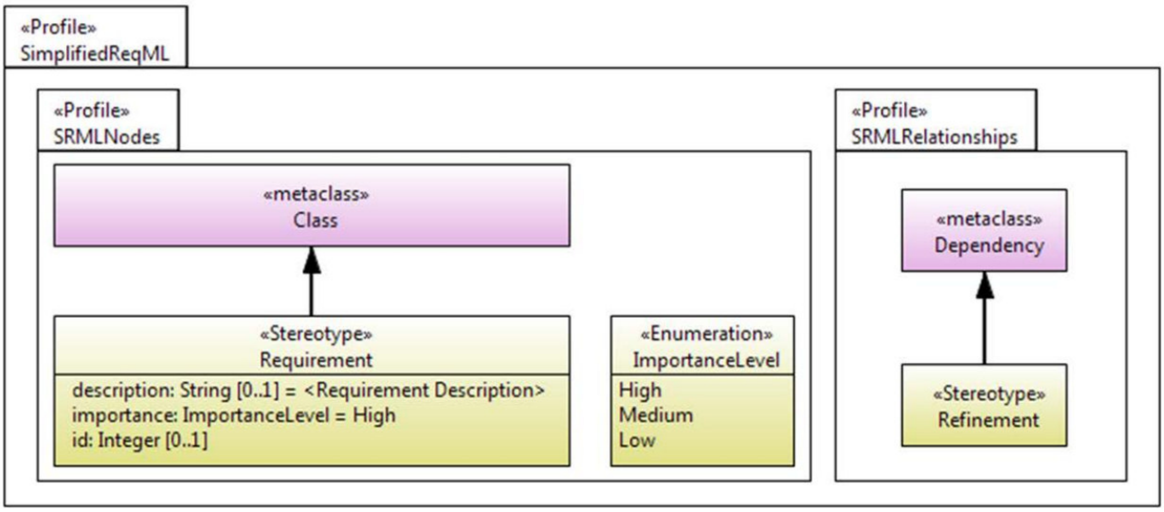


Figure 2.7: Example of a profile with associated metaclasses and stereotypes [67]

SysML follows the model-view separation principle (see fig. 2.8). On the one hand, there is a repository that contains the entire model. The model repository could be stored in a database. On the other hand, there are views of the model available. Diagrams in SysML are used to depict model sections. By projecting a view on them, the user can see specific model components in the repository and interact with the model. Because SysML separates model and view, certain model components in the repository may not be mirrored in any of the accessible views. Despite this, they are also incorporated in the model. In the modeling tools, a repository of all model components is always available and hierarchies are typically depicted as a tree. A single model element can thus be displayed in several perspectives. When generating numerous views of the same project, this can be used for a variety of objectives, such as providing different viewpoints to different project stakeholders [45].

SysML includes nine diagrams. Some of the diagrams were adapted from UML, while others were created from scratch (see fig. 2.5). Aside from the requirement diagram (req), which is a unique feature of SysML, structure diagrams (pkg, bdd, ibd, par) and behavioral diagrams (pkg, bdd, ibd, par) are roughly distinguished (uc, seq, act, stm). The SysML diagrams are shortly described as follows [45, p. 40][55, p. 15]:

- *Package diagram (pkg):* reflects the model's organizational structure and package hierar-

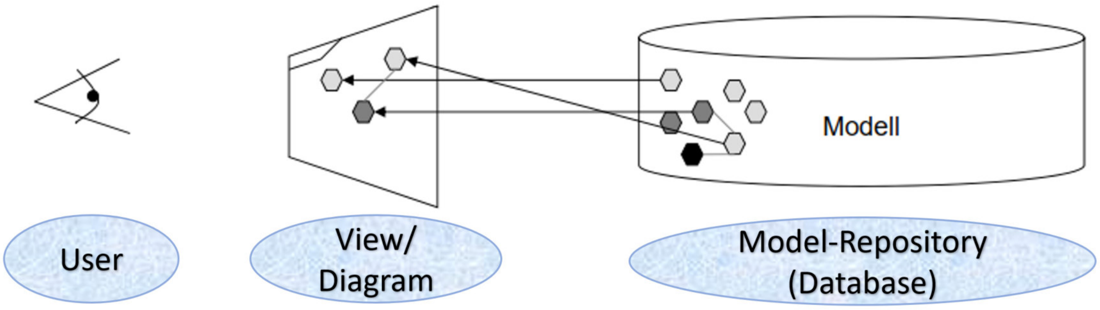


Figure 2.8: Separation of the Model from the View (modified from [45])

chy. Packages are model items that assist in the organization of all other model elements in the model database. They organize a user's access to information and model navigation by including model elements and dividing them into logical and cohesive parts.

- *Block Definition Diagram (bdd):* allows for the definition of block properties and relations. It is typically used to illustrate the structural hierarchy of system components. Specialization and inheritance of block attributes can be modeled here. Additionally, it may be utilized to identify the system configuration through block instances and their stated characteristics.
- *Internal block diagram (ibd):* is used to depict a block's internal structure by illustrating the interfaces and connections between blocks' internal parts. Together, the interfaces in the form of ports and the connections in the form of connectors provide an overview of the internal interaction and communication in a block.
- *Parametric diagram (par):* is a new diagram in SysML and a specialization of the internal block diagram. It can be used to link equations and inequalities to system attributes. Parametric diagrams are used to support engineering studies that include performance, reliability, availability, power, mass and cost. It can also be used to perform trade analyses on potential physical designs and in control applications and simulations for function-oriented analysis (like Simulink).
- *Use case diagram (uc):* the system's intended functionality is depicted in this diagram with the corresponding external users. This allows the stakeholders' interactions with the system to be documented. A use case diagram is a black-box representation of the services provided by a system in cooperation with its actors.
- *Sequence diagram (seq):* is used to define behavior, with a focus on how the actors and parts of a block interact with one another via operation calls and asynchronous signals. The sequence diagram is useful for modeling and simulating real-world events or demonstrating communication between systems or system components. The signals and information exchanged between the components are shown along a time axis.

- *Activity diagram (act):* is used to model the conversion of input into output via a predefined sequence of actions. Using the concept of tokens[11], flows are managed across the actions using various nodes. Activity diagrams are also often used as an analysis tool to comprehend and articulate a system's desired behavior.
- *State machine diagram (stm):* A state machine diagram can describe the behavior of a system component using states and the transitions between them. Specific behaviors can be assigned to a system's and its components' states (e.g. activities). State machines are well-known to modelers because they are similar to machine code and statecharts.
- *Requirements diagram (req):* is very essential and is used to show text-based requirements, connections between requirements and interactions between requirements and other model parts. This might imply that a component satisfies, refines, or associates with a need.

In addition to the above-mentioned diagrams, the SysML standard provides a few additional constructs derived largely from the system development context. The so-called allocation is one of these constructs. The main idea behind the allocation is to offer a way for models to be linked with one another, which depict the system from various abstract points of view. On the one hand, architecture components (e.g., block characteristics) from various abstract model portions can be assigned to each other. In this instance, the allocation is always directed from the more abstract model element to the more concrete one. The allocation of behavioral components to architectural elements is a second type of allocation described by the SysML standard. For example, this can be used to represent the assignment of a behavioral element, such as an activity or a state, to a specific architectural component [45, p. 61].

This section attempts to provide an overview of the major SysML concepts, constructs and constituents, that are essential for the understanding of this work. However, *Friedenthal et al.* in [51] and *Holt et al.* in [62] provide a full exposition of the language with examples and applications in their books. The next part will provide an overview of the existing tools that allow SysML usage and application, with an emphasis on the selected tool.

2.3.3 SysML Modeling Tools

An MBSE tool follows the principles of one or more modeling languages and is a tool-developer implementation of the language specification. It enables and assists the user in producing a decent model in this language. The MBSE tool is thus utilized for complex system modeling, visualization and simulation. The modeling tool serves as a conduct between the model data and the users, acting as a repository in the background when modeling. It should facilitate the use of linguistic elements, constructs and diagrams.

[11] A token is an executing thread that can be created and terminated. The token represents the logical flow's or data-/object flow's progress. It is possible to apply an automatic verification of Activity-Diagrams, namely a simulation, using this formal specification of the semantic of activity diagrams [68].

There are two approaches that explain how MBSE tools interact with other relevant tools in the end-to-end model-based development tool environment: federated and integrated MBSE. The federated method outlines the model-based interconnection of several tools through suitable interfaces and exchange formats. This guarantees that the most appropriate software is utilized for each development project and for each MBSE subtask. This often implies that the company's existing tool chain has not been totally changed. Individual models are managed and connected via a federation platform. However, the interface complexity and technological connection of the different applications are the drawbacks of this collaborative tool environment. For example, the Phoenix Integration's tool "ModelCenter" is a framework for creating and automating multi-tool workflows, optimizing product designs and thus enabling the federated approach for applying MBSE. Figure 2.9 shows the application of the federated approach by linking SysML Tools to multidisciplinary design and analysis tools using ModelCenter [69].

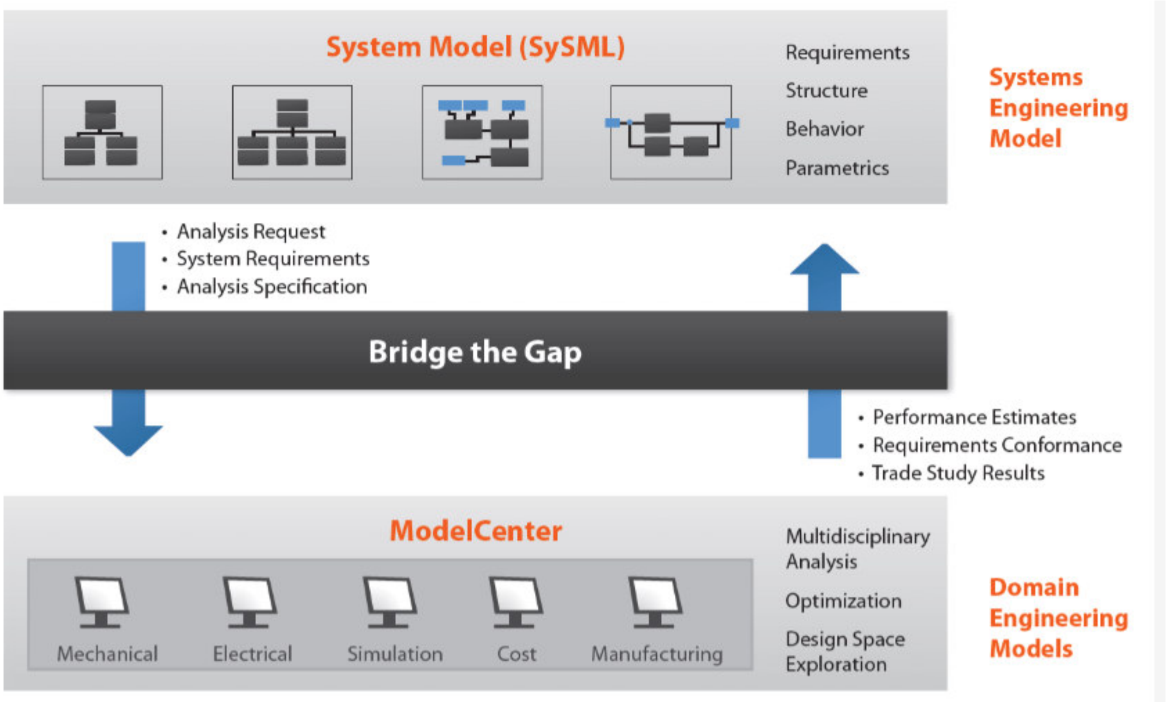


Figure 2.9: ModelCenter platform for federated approach [70]

The integrated tool approach is highlighted by the fact that all of the tools for the different MBSE use cases are provided by a single vendor. This implies that all models are kept in this manufacturer's data model, removing the above-mentioned interface complexity and tool migration. Consequently, the business becomes reliant on a single manufacturer and faces the danger of not being able to offer the optimal tool for specific tasks. Dassault Systemes, with its 3D Experience Platform, or Siemens are two potential vendors that offer integrated platforms [69].

In this work, a tool that supports the usage of SysML was needed. The SysML tools differ in a number of ways, making the choice of the correct tool of great importance. There are a number of factors and criteria that must be examined in the selection process. A decision must be made based on the project's intended usage and goals. Some of these criteria are the modeling language support, multi-user capability, code and test case generation, model execution and simulation or document generation and extensibility [45]. The most popular SysML tools are *PTC Integrity Modeler* from PTC, Sparx Systems' *Enterprise Architect*, IBM's *Rational Rhapsody* and No Magic's *Cameo Systems Modeler*. In his study, *Rosenow* gives a detailed overview of the MBSE tools characteristics [71]. He also presents a method for selecting an appropriate MBSE tool. The technique covers several steps, including a framework, and then recommends which tool is best suited for the user based on his weightings and evaluations of the tool requirements and features.

The tool Cameo Systems Modeler (CSM), which is accessible from the Institute of Systems Architectures in Aeronautics at DLR, was chosen for this study due to the following reasons:

The tool

- strictly respects SysML well-formedness standards for syntax (notation) and semantics,
- enables a good interoperability by integrating Requirements Management tools (e.g., DOORS, PTC Integrity) and Simulation tools (MATLAB/Simulink, Mathematica),
- is a user-friendly SysML modeling tool that supports Agile MBSE and intermediate-level model simulations,
- allows modelers to conduct engineering analysis for the purpose of evaluating creative decisions, verifying requirements and constantly checking model consistency.

Interoperability is crucial not only for the tool, but also for the methodology as a whole. This is because one the primary objective of this research is to integrate the SysML System architecture with other models used in the digitalization process (see section 2.1.3). Thus, synergies between models and information exchange can be leveraged to expedite analysis and validation during the early design phase. The following section discusses the current state of the art in terms of integrating other programs and tools with SysML models.

2.3.4 Integration of external models

Domain-specific modeling advancements have resulted in strong tools for their primary purpose. Many technologies have a restricted area of modeling or analysis, emphasizing the necessity to combine information models from several modeling areas. The integration of analytical and MBSE models has been considered for a long period of time. Typically, there are conflicts between the form of the analytical models and the form of the MBSE model. To define models in the MBSE model, a semantic language is required. However, the analytical models vary in nature and are not necessarily semantically correct. As a result, mapping between analytical and MBSE models remains difficult. This frequently results in MBSE models being unable to perform comprehensive analyses or advanced analytics that were not developed for use with MBSE.

Nevertheless, models and tools must be combined for larger-scale analysis or simulation (e.g. aircraft level). Text documents and spreadsheets in proprietary data formats currently prevail for information exchange in MSBE activities, but data exchange standards such as the AP233 - ISO 10303 Standard for exchanging systems engineering data, XML Metadata Interchange (XMI) defined for the UML framework and the Requirements Interchange Format (RIF) are evolving [72]. SysML makes use of the OMG XMI standard to transmit modeling data across tools [73]. Another significant interchange format is the Open Services for Lifecycle Collaboration (OSLC). It is an open initiative that develops standards for tool interaction. These standards enable the data exchange and the cooperation of separate tools from product lifecycle software [74]. Various working groups under the OSLC project are exploring the integration of tools for special cases such as change, test, requirements or configuration management. To enable these scenarios, they provide a standard vocabulary for the relevant lifecycle artifacts (see fig. 2.10). Each artifact is represented as an HTTP[12] resource, identifiable by a Uniform Resource Identifier (URI)[13], and can be accessed and modified using various methods [77]. Each tool contains information pertinent to its domain and internet URIs point to accessible resources in other tools (see fig. 2.10). Instead of using separate tools to handle templates, OSLC enables MBSE tools to act as web servers. To exchange model contents with other tools and users, online service requests are utilized [78].

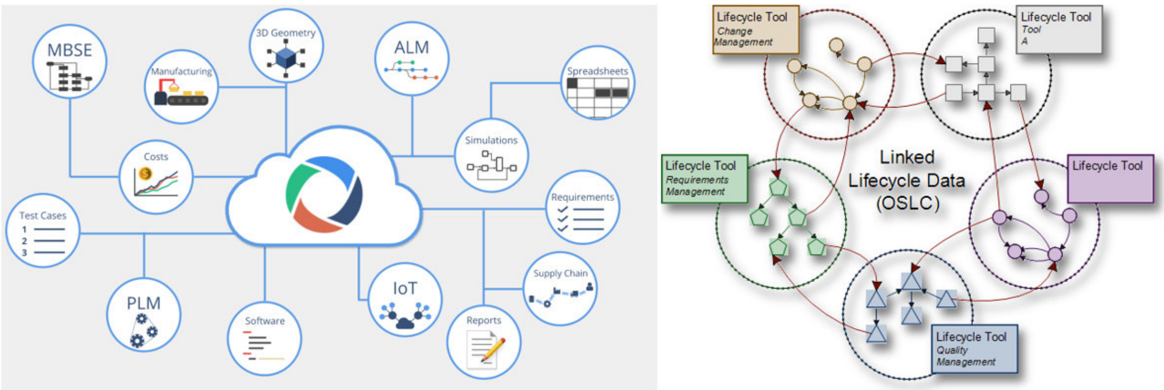


Figure 2.10: Data linking across domains and organizations using OSLC [74][77]

In this work, the focus is on the integration of Matlab/Simulink models in SysML. According to *Vanderperren et al.*, there are two ways to combine SysML and Matlab/Simulink models: co-simulation and integration using an executable language [79]. Simulink simulations communicate with SysML's via an intermediary coupling tool. Both simulations exchange signals and run concurrently during duplex synchronization, while running alternately with each other during sequential synchronization. Another option is to use a common execution language as a middleman. Due to the lack of support for Matlab code creation directly from SysML, it is possible to create C/C++ code using either Matlab Compiler or RealTime Workshop and

[12] The Hypertext Transfer Protocol (HTTP) is a stateless protocol for application-layer data exchange over a computer network. It is mostly used to download web pages (hypertext documents) from the World Wide Web (WWW) [75].

[13] A URI is a identifier made up of a string of characters that is used to refer to an abstract or physical resource. URIs are used to identify resources (web pages, other files, etc.) on the Internet, most notably the World Wide Web [76].

then connect it to a C++ implementation of the SysML model. The first method improves simulation speed, while the second improves the temporal accuracy of the signals exchanged. A few examples from practical applications demonstrating Matlab/Simulink integration with SysML models and the corresponding data exchange are provided as follows.

A model integration approach from SysML to Matlab/Simulink has been investigated by *Chabibi et al.* in [80]. Through the development of a Domain Specific Language (DSL), called Simulation Modeling Language (SimulML), the authors propose an integration method to unite the potential given by systems modeling languages and simulation environments. SimulML is defined by abstract and concrete syntaxes, as well as its semantics and enables the creation of a simulation model of a given system. This model is made up of components and/or subsystems that are linked together through ports. The behavior of a system is defined by the behavior of its components and their interactions. A component's behavior is described using mathematical models based on component variables and parameters. A Model-To-Text (M2T) method is then used to convert the model from this DSL to the Matlab/Simulink simulation environment. The method suggested is based on a Matlab code creation from SimulML models.

There are several other research work that investigated the interface between SysML Tools and Matlab/Simulink Models. At Saab Aerosystems in the context of the Gripen-Avionics-Demo project, a function prototyping was carried out in a model that included an architectural model in SysML, a display layout model in VAPS XT, and a Simulink model with scaling, filtering, and logical functions [72]. Whereas, *Rahman et al.* provided a unique workspace that could be used throughout the development cycle [81]. This was accomplished through the use of a Rhapsody-Simulink-Integration plugin that combined Rational Rhapsody with Matlab/Simulink. The system level Rational Rhapsody model includes a reference to the physical system's Matlab/Simulink behavior model. Finally, *Johnson et al.* merged modeling constructs from SysML and Modelica to improve support for Model-Based Systems Engineering [82]. They used a vehicle suspension to demonstrate the synergies between SysML and Modelica at three different levels. This study demonstrates dynamic co-simulation and shows the benefits of the MBSE method of using a SysML model and a calculation model. These integration examples are discussed and compared with this work's integration approach (cf. section 8).

In this section, SysML, as a modeling language, and Cameo Systems Modeler, as a modeling tool, were presented. However, both of them do not provide a methodology for system modeling. In the next section, the basics of MBSE methodologies, that are relevant in this work, are described.

2.4 Overview of Existing MBSE Methodologies

This section begins with some key definitions and distinguishes between some relevant terms like process, tool and methodology. *Martin* defines the process in his book as a "*[...] logical* *sequence of tasks performed to achieve a particular objective. A process defines "WHAT" is to be done, without specifying "HOW" each task is performed*" [83, p. 54]. In contrast, he claims that a method is made up of "*[...] techniques for performing a task, in other words, it defines the "HOW" of each task. At any level, process tasks are performed using methods*" [83, p. 55]. Finally he describes a tool as "*[...] an instrument that, when applied to a particular method, can enhance the efficiency of the task provided; it is applied properly and by somebody with proper skills and training. The purpose of a tool should be to facilitate the accomplishment of the "HOWs"*" [83, p. 55]. As a result, a methodology describes according to these definitions a set of processes, methods and tools that belong together. In general, a methodology may be considered a kind of plan for applying linked processes, methods and tools to a set of issues that are all similar or interrelated.

A more specific concept, the MBSE-methodology, can be defined as a set of processes, methods, and tools used to accomplish particular goals of Systems Engineering, driven by a model-based framework [84]. There is currently no widely accepted standard methodology for using SysML in model-based systems engineering. In 2007, a formal survey of MBSE methodologies has been published by INCOSE [85]. MBSE methodologies that have gained attention in industry forums and publications are briefly examined in this research. They are created to support as potential candidates for adoption and customizing to an organization's SE policies and processes, but they are not intended to be complete. Some of these methodologies, as well as others not mentioned in this survey, serve as a reference for the methodology developed and used in this work and are explained below:

- **System Exploration and Definition Approach [86]:** *Abulawi* devised a methodology for investigating novel system concepts by combining technology-oriented views with risk and user-oriented ones. This approach is built on the use of SysML models to support the many activities that comprise the core of the system exploration and definition process. The suggested method's conceptual framework addresses system analysis and exploration, as well as a specification emerging from a requirements engineering process that involves information from diverse analytical activities. Figure 2.11 depicts the main stages of the process.

  The method user is directed through the process of building a system concept from an original, sophisticated system idea first by using an hierarchical analysis, where system is being studied as a "black box". The attempt to combine a technical push with a market pull leads to system concepts. This is followed by a functional analysis with an abstracted picture of the system structure, referred to as the "grey box." The goal here is to analyze what the system should do to fulfill its functions and what to expect when specific events emerge. An initial risk analysis is undertaken in conjunction with the study of the system's intended purpose in all analysis stages. Finally, a structural analysis is performed and the system architectural framework is defined (i.e. the technical solution). To achieve the finest results, architectural ideas should be varied. Alternative concepts are researched and assessed in trade studies using predetermined criteria in order to arrive at a final, ideal solution. The tasks related to requirements analysis and

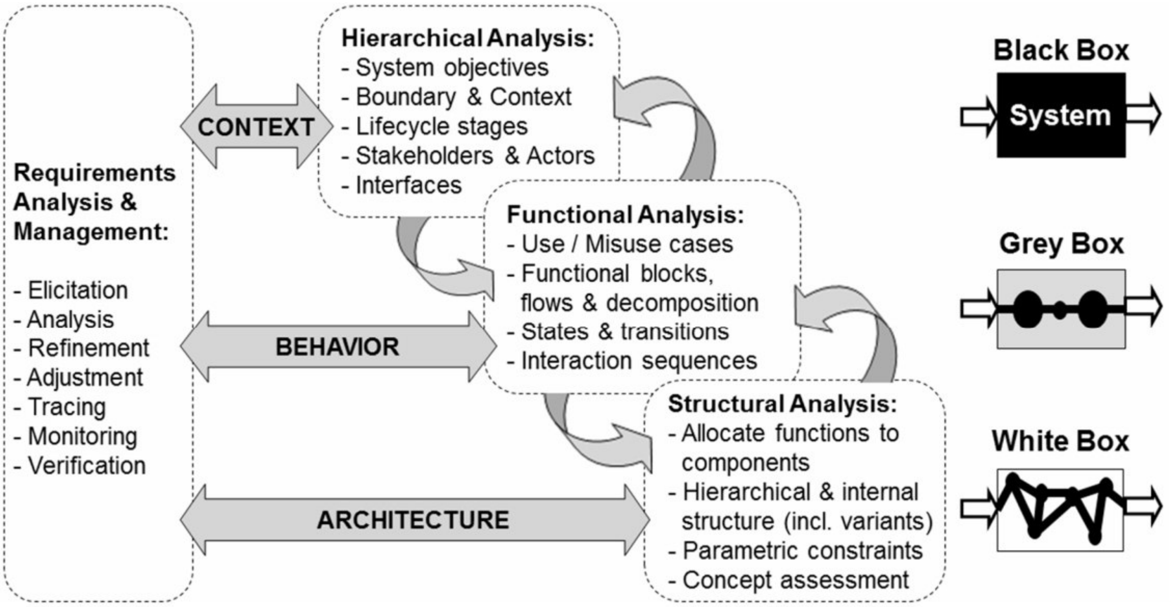


Figure 2.11: Main conceptual stages for the SysML-based system exploration methodology [86]

management support the whole process stages. It aims to generate, derive, refine and trace all system requirements at all levels of abstraction. Because the industry is still hesitant to utilize SysML for requirements engineering and management, the traditional method to requirement documentation is employed instead.

- **Object-oriented Systems Engineering Method (OOSEM) [51]:** is a model-based top-down approach that uses the concept of object orientation and the SysML. The method covers fundamental system engineering tasks such as stakeholder analysis, requirement analysis, architectural design and analysis and verification. Therefore, it leverages a variety of methodologies, including causal analysis, black-and white-box descriptions, logical decomposition, partitioning criteria, node distribution and variant design. To manage complexity, concerns are segregated and merged into a coherent system model. OOSEM is also designed to make it easier to integrate object-oriented software development, hardware development and test, to enable architects design more agile and scalable systems that can handle growing technologies and changing needs over time. Activities and artifacts used in the approach are shown in figure 2.12. These activities are clearly in line with the standard systems engineering V-Model, which may also be performed repeatedly at each level of the system hierarchy. SE principles like management processes and multi-disciplinary collaboration must be used for any of these activities to be effective. The development process can be used recursively across the many stages of development, such as conceptual design, preliminary design, detailed design, and subsequent stages [85].

- **Functional Architecture for Systems Method (FAS) [87]:** in 2010, the method was described in a paper for the german systems engineering conference TdSE. The FAS method can be applied regardless of any language or tool, however SysML is best suited

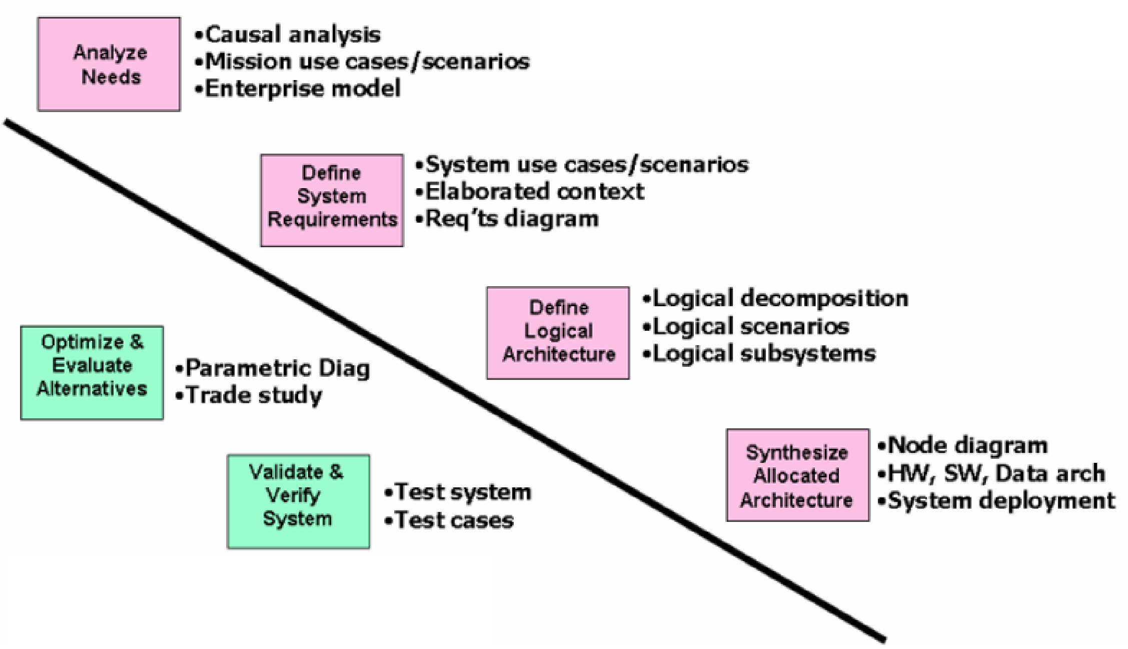


Figure 2.12: Activities and artifacts used in OOSEM [85]

and recommended due to it having all necessary modeling components to support the method. Through the grouping of use case activities, the approach generates functional architectures in a block oriented form from use cases. The model-based functional architecture is used to describe a system in which the goal technologies are independent of the system functional elements, that transmit modeled information (signal, data), materials, force or energy [44]. This functional element is stated as "*[...] an abstract system element that defines a relation between at least one input and at least one output by means of a function.*" [87]. This approach does not address the specification of functional requirements and use cases for the system. Nonetheless, these are the most important inputs when it comes to designing an architecture. The method starts by modeling the use cases and the flow of their associated activities that refine functional requirements. Using use cases, a clearer understanding of how a system's operations connect to system actors emerges. It also aids in determining which factors outside the system communicate with it. Users are clearly included, but so are other systems and environmental elements. The use case activities are then scraped together using functional groups to prevent developing several functional components for the same capability and to ensure that the system is characterized by an appropriate collection of functional components. This is the most significant step and necessitates therefore architect's expertise. *Lamm et al.* provide a detailed overview of heuristics that might be utilized to help in grouping [88]. For example, some heuristics include that abstract and secondary use cases form a functional group, that one functional group accepts functions linked to system actors and that functions that share data may be grouped. Lastly, functional blocks are generated from functional groups and linked through functional interfaces to create the final functional structure. On higher abstraction levels, the FAS approach works effectively but complex interactions between system operations at lower levels of abstraction may result in emergent behavior. Furthermore, plugins are available for some modeling tools such as enterprise architect and cameo systems modeler and have been developed to support and automate certain parts of the FAS approach tasks [89].

- **Systems Modeling Toolbox (SYSMOD) [90]:** is a technique for developing requirements and architectural specifications that is mostly used to define updated and new products. A collection of commonly used and well-known procedures (such as use case analysis) can be customized to meet special needs. The suggested modeling language for SYSMOD is SysML and can be used in conjunction with any SysML modeling tool. SYSMOD guides systems engineers from system goals, stakeholder needs, stakeholder requirements, domain, and functional analysis to architectural descriptions. SysML's SYSMOD specification defines the Model semantics and relationships explicitly. Different types of diagrams provide stakeholders with varying perspectives on the development of a model [91]. One of the main patterns that SYSMOD suggests, states that each development is predicated on an existing technical basic solution. This solution is referred to as the base architecture and it serves as the beginning point for the continuous evolution of requirements and architectural solutions (zigzag pattern). When developing a solution, it is constantly attempted to keep needs and technologies in alignment (see fig. 2.13).

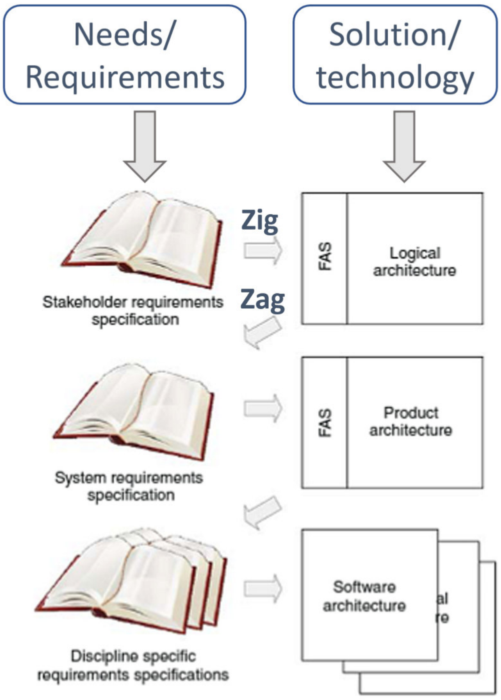

Needs/ Requirements
Solution/ technology
Zig
Zag
Stakeholder requirements specification
FAS
Logical architecture
System requirements specification
FAS
Product architecture
Discipline specific requirements specifications
Software architecture

Figure 2.13: Zigzag pattern of the SYSMOD toolbox (modified from [90])

Other methodologies and techniques used in industrial applications are presented and discussed in the MBSE survey by *Estefan et al.* [85]. These are not included here since they were not considered in the methodology of this research. The following chapter analyzes, combines, and expands on the major methodological aspects and concepts mentioned in this section in order to build a methodology that accomplishes the work's objectives.

3 Development of Methodology for System Architecture Modeling and Integration

In this chapter, the model-based methodology developed in this work is presented and explained. SysML is utilized as a modeling language in this study and its use is facilitated by the tool Cameo Systems Modeler (CSM). The goals of this work (given in chapter 1) define the problem and specify what is intended to be accomplished. Whereas the SysML describes which syntax, semantics and concepts are utilized to solve the defined problem. A methodology, for specifying how the SysML using the CSM-tool can be utilized to meet the goals of this work, is required. The methodology outlined in this chapter is the consequence of these concerns. However, it was not created out of thin air, rather, it emerged from previous methodological notions, which are presented in section 2.4.

The initial steps in developing an acceptable methodology are to analyze existing methods and to determine relevant concepts for the research's aims. Both *Abulawi*'s method for investigating novel system concepts and INCOSE's OOSEM method are top-down approaches in nature. Both begin at a high degree of abstraction and analyze the system in its context and according to its top-level requirements. These abstraction levels are classified based on the type of analysis, rather than the physical system hierarchy. *Abulawi* focuses on the system's functional features, allowing for additional function-driven analysis. OOSEM distinguishes between logical and physical abstraction levels in structural analysis. It describes how the functions can be carried out using certain logical components and scenarios. Physical systems or subsystems can also be logical components, but only their logical behavior is examined and classified at this level. At the lowest abstraction level, discipline-specific and detailed development, such as hardware, software, or data, occurs, allowing for system synthesis. The analysis throughout the complete system abstraction levels helps develop new knowledge and information about the system. Each analysis step also helps generate new systems requirements for the next level which together constitute at the end the system specification.

The core of this work's methodology is its systematic segregation into context-based, functional, logical and physical analysis and design elements. It allows agility by modularizing the system analysis and design stages, which is a requirement for this work. When these steps are separated from one another, it is simple to change one without impacting the others. It also allows for the investigation of buttom-up effects (e.g. the effects of the modification of physical architecture on the logical or functional architecture). To understand which modeling activities are conducted throughout each of these stages, an overview of the methodology is shown in figure 3.1, using a SysML activity diagram.

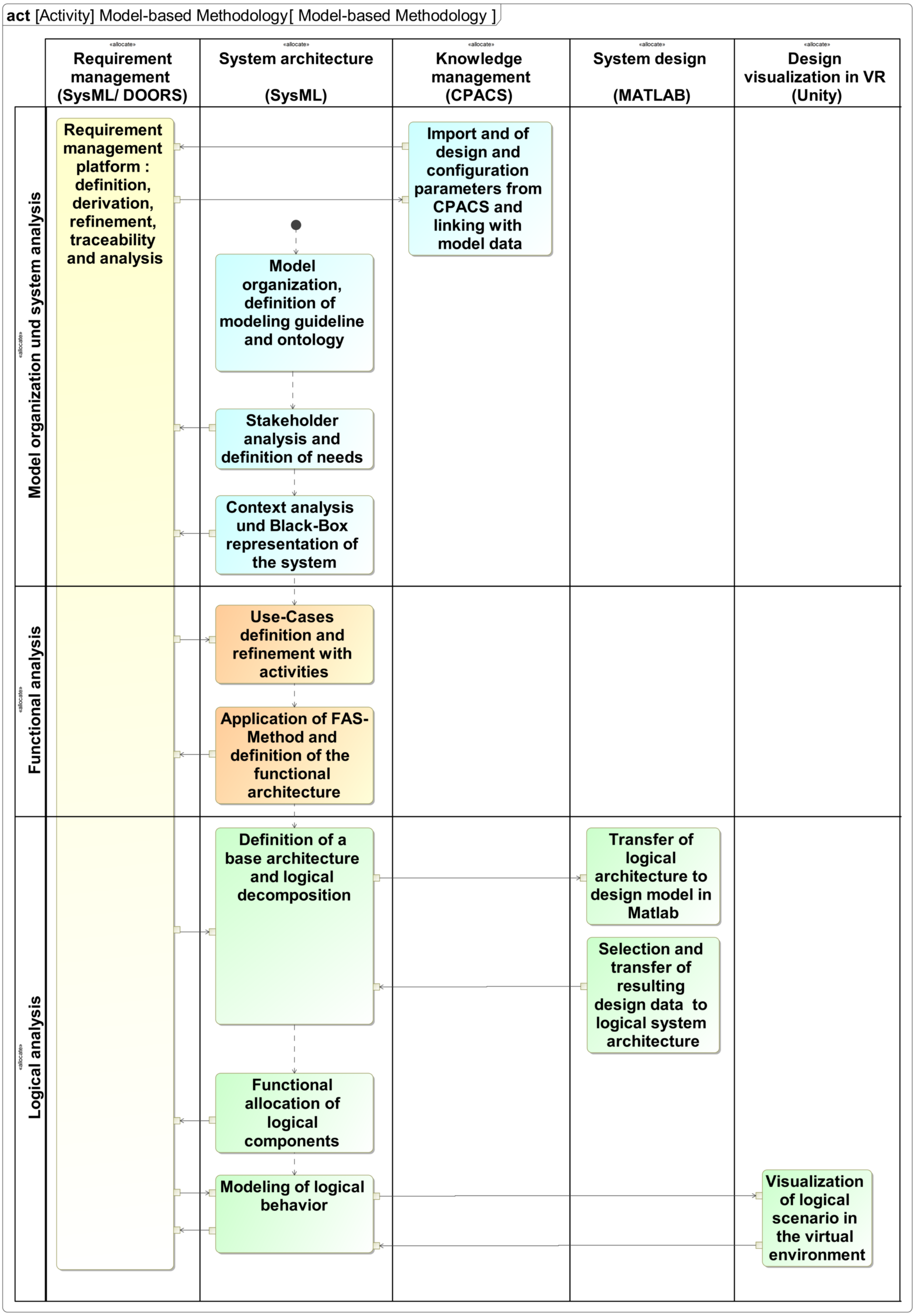

act [Activity] Model-based Methodology[Model-based Methodology]
«allocate»
Requirement management (SysML/ DOORS)
«allocate»
System architecture (SysML)
«allocate»
Knowledge management (CPACS)
«allocate»
System design (MATLAB)
«allocate»
Design visualization in VR (Unity)
«allocate»
Model organization und system analysis
«allocate»
Functional analysis
«allocate»
Logical analysis
Requirement management platform : definition, derivation, refinement, traceability and analysis
Import and of design and configuration parameters from CPACS and linking with model data
Model organization, definition of modeling guideline and ontology
Stakeholder analysis and definition of needs
Context analysis und Black-Box representation of the system
Use-Cases definition and refinement with activities
Application of FAS-Method and definition of the functional architecture
Definition of a base architecture and logical decomposition
Transfer of logical architecture to design model in Matlab
Selection and transfer of resulting design data to logical system architecture
Functional allocation of logical components
Modeling of logical behavior
Visualization of logical scenario in the virtual environment

Figure 3.1: Model-based methodology for the development and integration of the system architecture

The separation is based on two factors: the analysis and the model type. It is represented by partitions (swimlanes) that are both horizontal (for analysis) and vertical (for modeling). There is no consideration given to the type of system-of-interest (SoI), implying that this methodology can be implemented at the component, subsystem, or system level. However, similar to *Abulawi*'s approach, it is advised that a specific degree of SoI complexity exists in order to justify the effort of analysis and integration. A model organization and system analysis, a functional analysis and a logical analysis are all part of the analysis. Because the requisite specific data at this abstraction level is not available, the physical analysis is not addressed in this work. Furthermore, communication and data exchange with specialized models (such as aerodynamics, FEM, CAD or multi-physics) are not necessary in this research. Nevertheless, the methodology enables for the extension and integration of this physical layer in future research.

First, the aircraft design and configuration information is incorporated as system requirements for all design phases. This information is obtained from CPACS and imported into the requirement management platform. This includes top-level system requirements as well as configuration parameters like passenger number and aircraft class definitions. This interface also enables the evaluation of the effects of reconfiguration (for example, a new cabin configuration in CPACS) on the system architecture, as well as the automatic system customization and architecture adjustments. Thus, as specified in the objectives of the work, all information from the design are linked to the system architecture. This interface between CPACS and the system model is tool specific and shall be designed to automatically update and share data between the two platforms. Though, it is not possible to transfer the resulting system architecture information back to CPACS because the considerations regarding systems (specifically cabin systems) are still not given in the CPACS representation, it is an important step that should be considered in the future, as the knowledge exchange platform CPACS develops and expands.

Not only top-level requirements obtained or derived from CPACS data, but in general all requirements are continuously identified, derived, refined, traced, and analyzed in parallel with the architecture development process. They play a central role in the design and integration methodology. These requirements management activities are similar to *Abulawi*'s method but with the difference that here SysML is used for the platform for requirements management. It can be done with special requirements management programs (e.g. DOORS) and integrated into SysML models via corresponding interfaces. SysML contains a range of requirements elements, relationships, and diagrams to support the requirement management and analysis.

The digital cabin design and visualization process (cf. section 2.1.3) is examined before beginning the development of systems modeling methods. The goal is to identify the extensions required for a more consistent digital representation of cabin systems. The system architecture was adopted without any analysis, as evidenced by the process description. As a result, the structure, functions, and behavior of the system are assumed. Furthermore, the system design elements are not derived from a system analysis and it is impossible to trace why or how each architecture element exists. Hence, the emphasis in this work is mainly on the function-oriented system architecture. Its systematic derivation, development, and integration are the primary purpose of this work. The model preparation and setup (similar to OOSEM) is the initial step in the model organization and system analysis phase. In contrast to OOSEM, the scope is now reduced to several simple activities to simplify and permit a quick transition to further analysis stages. The initial step, modeling standards and rules related to the model format and style can be checked and determined. For example, OOSEM recommends using upper case for the first letter of each word in all definitions and types, such as blocks, value types, packages, and requirements, with a space between compound names that contain more than one word. Implementing automated model validation is an optional step depending on the required model consistency. This ensures that the modeling rules and guidelines are followed. Then, a system ontology must be defined and modeled. As stated in Chapter 2, a reference for this can be the CPACS ontology which is enhanced using common terminology in the context of an aircraft systems. To describe the ontology, a UML profile diagram can be used to extend SysML model elements with additional stereotypes and corresponding characteristics. Furthermore, the relations between the stereotypes must be specified in order to constrain the ontology application. Model organization activities take place throughout the modeling process, but the model structure, which defines how the model will be organized, must be specified before beginning modeling. As a result, the modeler has complete control over whether the model is structured according to the system hierarchy, analysis aspects or any other logical grouping. Packages and package diagrams are commonly used for this and they can be interconnected to enable model navigation.

The SoI stakeholders are then identified and analyzed in a subsequent step. There are some broad categories that can be used to identify the main stakeholders, such as user, manufacturer, operator, or regulator. Depending on the SoI modeling goals, the emphasis will be set on different types of stakeholders. Furthermore, the top-level requirements that are available can be used to identify which parts, organizations, or people are interested or involved. Following that, the identified stakeholders are modeled using SysML actor elements. Each of these actors should be assigned a requirement that represents the SoI requirements from their perspective. After that, the requirements are transferred to the requirement management platform.

The first system analysis, like *Abulawi*'s and OOSEM approaches, is based on a black box representation. The modeler identifies and associates in a block definition diagram actors, environmental elements or external components and systems that interact with the SoI. These associations are then refined in an internal block diagram and the interfaces are specified. The information, data, energy, or material that flows between the SoI and the outside world is defined at this stage. Based on these interactions, flow items are translated into context requirements from the perspective of each external element. The system analysis concludes with the collection and sorting of all requirements generated during the analysis stage. Only functional requirements are to be considered in this work. As a result, these must be filtered out of the requirements and used as input for the next analysis.

The system's functional requirements are interpreted in functional scenarios and use cases in the second part of the analysis, known as "functional analysis". The goal is to specify various use cases that, when combined, lead to the accomplishment of all required system functions. As a result, the use cases are modeled and functional requirements are defined to refine these use cases. Linking these two model elements is critical for the desired traceability. This is the first design and specification activity in the methodology that defines how the system will behave. The modeler can use use cases from previous system configurations or create new ones. Use case diagrams and related relationships are used to develop and relate the use cases with each other as well as with external actors. These use cases are very broad and need to be refined. The FAS method is used to achieve this goal. The actions in the activity diagram are used to specify which behaviors are executed in which chronological order in the use cases. The actions are assigned based on whether the system functions are internal or interfacial. The FAS partition "I/O" is used for latter and the corresponding actor is specified. Following that, the steps for function grouping and the development of a block-oriented functional architecture are applied (as described in section 2.4). This results in a pure functional representation of the system, which is generally valid and independent of the technical solution.

The transition to the logical abstraction level, like the functional specification, necessitates architecting actions. As a result, the SYSMOD pattern is employed to assist in the logical specification. If the SoI isn't entirely new (which is usually the case), the base architecture is either the standard architecture or a modified or upgraded version of it. To define the base architecture of a completely new system, engineering expertise and innovation are required. This does not, however, imply that this is the final solution in either case. On the contrary, the primary goal of the subsequent analysis is to refine and optimize it through a series of recursive steps in order to validate it. A bdd is used to decompose the logical components of the base architecture into subordinate hierarchical subsystems or components. An ibd is then used to node the logical interfaces, i.e. the flows or interfaces that exist between the logical components. It is important to note at this point that the logical system's boundary interface must conform to the ones already defined in the black box representation. Furthermore, the logical architecture is linked to the functional architecture. As a result, using allocation matrices, all logical components are linked to functional blocks. Here more than one function can be allocated to the same component and one function can be allocated to several logical components.

Finally, the system architecture is linked to the external models for the purpose of cabin design and visualization process extension. The CPACS-imported cabin design parameters and requirements are used to configure and instantiate the developed logical architecture. The logical architecture instance is passed to the design model in Matlab, where it is used as additional parameterization rules and design requirements. To accomplish this, the two models of ontologies and object-oriented structure must be adapted to one another in order to avoid communication conflicts. Each system component in the Matlab geometrical design model is now linked to a traceable architecture element in the CSM model. The geometric results are returned to the architectural description for further analysis after the design algorithm has placed the various systems components in the cabin. This necessitates a thorough examination of design data and information that may be useful for system analysis in the system architecture model. Thus, the interaction between the design and system models opens up a new opportunity for identifying and exploiting synergies between different design and analysis perspectives. The effect of geometrical design parameters can be used to optimize and refine the system architecture that has been implemented.

Moreover, SysML is used to model functional scenarios for the developed system. To depict the logical and chronological system behavior, structural and behavioral SysML modeling elements and diagrams (such as state machines, activities and sequence diagrams) are used. The modeling results must be converted into structured data that can be used by the VR platform. They are then rendered in the virtual environment for interactive exploration and modification. Feedback on visualization results is also provided to the system architecture for further tracking and validation of the specified requirements. The visualization of behavioral data in VR reduces complexity by representing complex interrelationships in a simple, fast and intuitive way. The system architecture and its interaction with the cabin design can thus be better understood by bringing together a large amount of data in the virtual environment.

Thus, the model-based methodology for achieving the work's objectives is established. To apply it, an appropriate system-of-interest must be selected. The following chapter discusses the rationale for choosing this work's SoI and describes its main technical aspects.

4 The System-of-Interest: Selection and Description

Understanding the system-of-interest (SoI) one of the most important prerequisites of any modeling effort and is a necessary condition for realistic modeling and analysis. The system is chosen for the application of the proposed model-based methodology in this work. As a result, a thorough justification of the choices is required. The sections that follow address the selection of the system and describe it. They also provide an overview of current industry alternatives as well as the key research and development in relation to them.

4.1 The Passenger Service Channel (PSC)

An appropriate cabin system is required to apply and validate the model-based system architecture integration methodology, which is developed in this work. Therefore, the Passenger Service Channel (PSC) was chosen for the application. The PSC is a critical cabin module that integrates several service and safety functions and is leveraged as an interface between the aircraft crew and the passengers. The European Aviation Safety Agency (EASA) distinguishes in the CS-25 document between primary and secondary PSC functions [92]. The primary functions are safety related and include oxygen masks, fasten seat belt indication, smoking allowance indication and announcement speaker. The secondary functions depend on the airline and related to passenger services. They include, among others, individual air supply, attendant call, reading lights or boarding music. Reading lights, attendant call, signs and speakers are integrated in the Passenger Service Unit (PSU), which in turn includes the Passenger Interface and Supply Adapter (PISA) [34]. This component is an integrated electronic circuit and enables the power supply and data communication for the control of the PSU components. Some cabin architectures also include a Stand-alone PISA (STA PISA), which is functionally identical with PISA but is used outside the PSC (e.g.in galleys or doors areas) [93]. Figure 4.1 shows an example of the interfaces of an A380 PSU (some of the terms related to the data buses or power supply are described in detail in section 4.2).

The cabin concepts vary between airlines and allow for customized designs according on the passengers' preferences. This has an impact on the positioning of the PSC's numerous components. PSCs are typically installed in the Overhead Storage Compartment (OHSC), which is located above the passenger seats in most modern designs. In addition, screens for In-Flight Entertainment (IFE) can be fitted whereas appropriate signal interfaces must

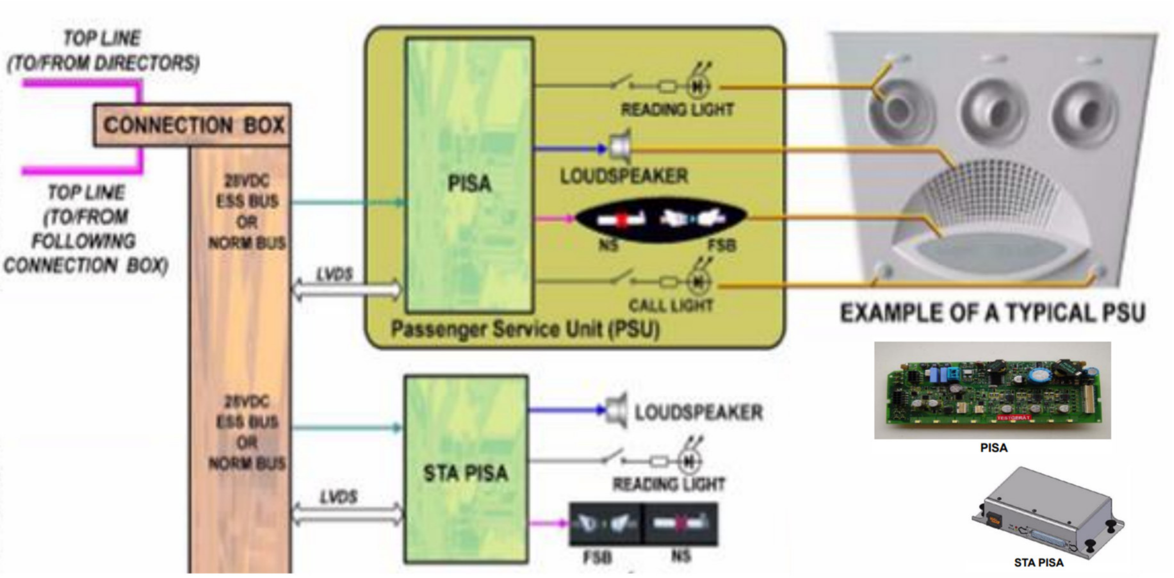


Figure 4.1: PSU components and interfaces (modified from [93])

be included into the PSC. Because of the growing tendency of more accompanying luggage in recent years, more storage space in the cabin was required, which resulted in innovative storage options and larger overhead storage compartments. Hence, the PSC has to evolve and must be designed as to remain customizable [34] [94].

The numerous interfaces to external systems, the variety of implemented functions, the close connection to the cabin design for installation and assembly or the required flexibility for customization: all of these aspects increase the PSC's complexity and make it an ideal candidate for the application of the MBSE methodology in this work (see fig. 4.2).

The methodology established in this study focuses on the context of the SoI, in order to gain a better understanding of how it interacts with other systems. It also allows for the examination of the influence of the PSC as well as the external systems on each other. As a result, an overview of these systems, their architectures and behaviors is important for modeling and analytic activities. The overview is covered in the next section.

4.2 Overview of Cabin Systems

Functionality distinguishes aircraft systems, that are often grouped together according to the Air Transport Association of America's Specification (ATA). Aircraft equipment is designated by an equipment identifier consisting of three two-digit components, according to ATA100. For example, a system 21, subsystem 11 and unit 01, are according to the identification specification, all part of 21-11-01. ATA 100 became a part of the ATA 2200 standard, which has also made minor adjustments and upgrades to aircraft system specifications [95].

As the name suggests, cabin systems are all aircraft systems that have an impact on the aircraft's cabin and hence have an effect on its passengers. Table 4.1 lists the partially

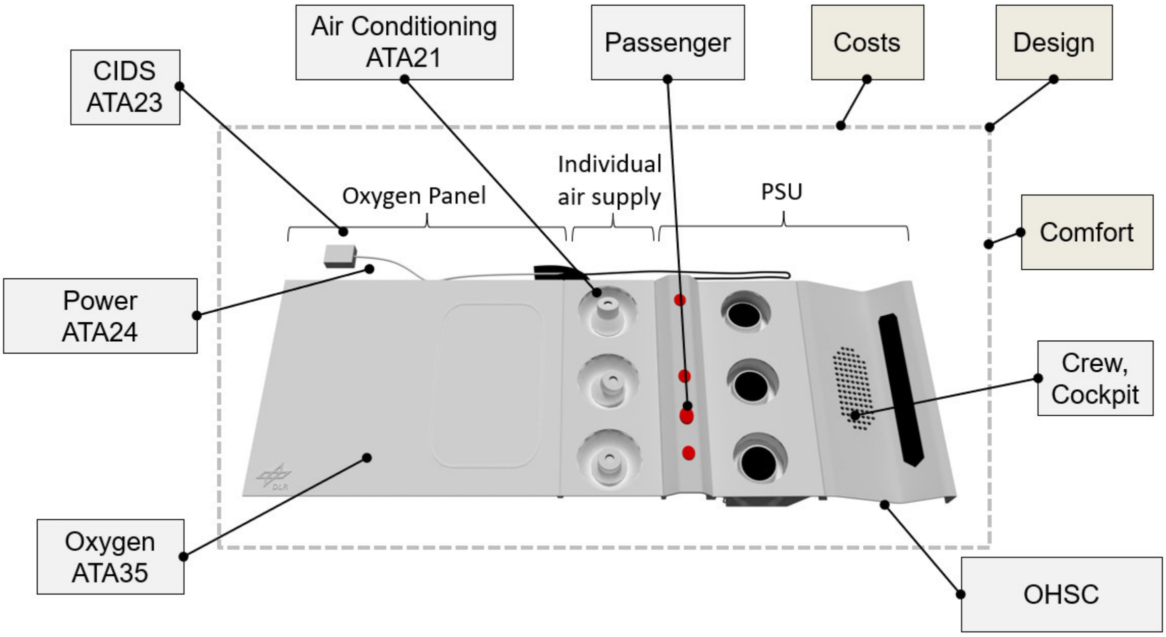


Figure 4.2: Complexity due to PSC Interfaces with other systems and disciplines (modified from [94])

included cabin systems according to ATA100. Not included here are new Chapters 44 and 50 from ATA 2200. According to the new specification, cabin systems (ATA 44) are defined as "*[...] those units and components which furnish means of entertaining the passengers and providing communication within the aircraft and between the aircraft cabin and ground stations. Including voice, data, music and video transmissions [...]*" [96, p. 57].

Table 4.1: Cabin systems according to ATA 100 [95]

Identifier	**Name of Cabin System**	**Type**
21	Air conditioning	Cabin system
23	Communications	Cabin system (partially)
25	Equipment/ Furnishings	Cabin system
26	Fire protection	Cabin system (partially)
31	Indicating/ recording systems	Cabin system (partially)
33	Lights	Cabin system (partially)
35	Oxygen	Cabin system
38	Water/ waste	Cabin system

Among all cabin systems, the Cabin Management System (CMS) is one of the most closely tied to the PSC, and its interface with it is complex. The cabin management system is one of the most complex and sophisticated systems onboard modern CS-25 passenger aircraft [97]. The cockpit and cabin crew use this system and integrated technologies to check the cabin's condition and interact with the passengers. Even though the highest design assurance level (DAL)[1] is C, the CMS must be reliable since it affects aircraft dispatchability as well as passenger

[1] DAL is evaluated during the safety assessment and hazard analysis processes by assessing the impacts of a system failure condition. The failure modes are classified according to the impact they have on the aircraft, crew, and passengers. Catastrophic (A), Hazardous (B), Major (C), Minor (D), and No Safety Effect are the classifications (E) [98].

comfort and satisfaction [99]. As a result, most CMS systems and designs include fault-tolerance features such as segregation, redundancy, and reconfiguration. Another source of complexity that challenges the design of this system is real-time requirement. Meaning, some realized features, such as audio-intercommunication, require real-time capabilities. CMSs are distinguished from other avionic systems by their need for flexibility. Before entering into service, the aircraft is customized for a certain airline, or a variation for a specific aircraft family is created. Besides, cabin layout modifications (e.g., short vs. mid-range arrangement) or increased capabilities during a retrofit impact the CMS after it enters into service.

The Airbus Cabin Intercommunication Data System (CIDS) is a typical CMS for the CS-25. The CIDS, which serves as a central control and display unit in Airbus aircraft, manages all cabin functions. It was first designed for the Airbus A320 in the 1980s. It only had simple, but essential functions, like passenger announcements and lighting control. System integration has expanded to include more than 30 functions, making it one of the most integrated systems on board [93][100]. Some functions such as emergency lighting are necessary for CS-25 aircraft, whilst others are based on passenger comfort and may differ from CIDS to CIDS, depending on the system context. In certain use cases, the CIDS is utilized as a user interface (UI) for control and monitoring activities (e.g. temperature control). The CIDS also has a Smoke Detection Function (SDF) for the cabin and cargo area. Because of its high safety relevance and needed reliability (particularly in the inaccessible cargo area), a separate power supply, hardware, and software are specified to manage this function in the CIDS to keep it separated from the other functions. CIDS has been implemented in the entire civil airbus fleet, with varying capabilities. Several data buses interconnect an array of redundant simplex processor modules and hundreds of peripherals [100].

Figure 4.3 illustrates an example of a A380 CIDS architecture. The Director is the most essential part of the CIDS. This central computer is a Line Replaceable Unit (LRU) that controls all system functions and is placed in the avionics section underneath the aircraft cockpit. The Director communicates with, monitors and controls all devices and components in the CIDS network. The Director is also capable of communicating and exchanging data with other aircraft systems. The Flight Attendant Panel (FAP) serves as a human-machine interface for configuring the system and its operations and displaying the system's and other cabin systems' status. It communicates with the Director via the panel network. The interface of various system components is necessary to realize the CIDS functionality. The Director can interact with the passenger and attendant-related components using the Decoder/Encoder Unit (DEU). The DEU-A serves as the interface for all components with passenger functions. PISA is linked to this component in order to share data with PSU components. The Top-Line connects the DEU-A to the Director. The DEU-B serves as the interface for all components that perform cabin crew functions. Handsets or display panels are examples of this. The DEU-B communicates with the Director via the Middle-Line. The Top- and Middle-Lines are bidirectional data buses with particular protocols that run along the aircraft cabin, supplying data to the Director and DEUs [93].

Another system, whose interface especially with the PSU is really important and interesting

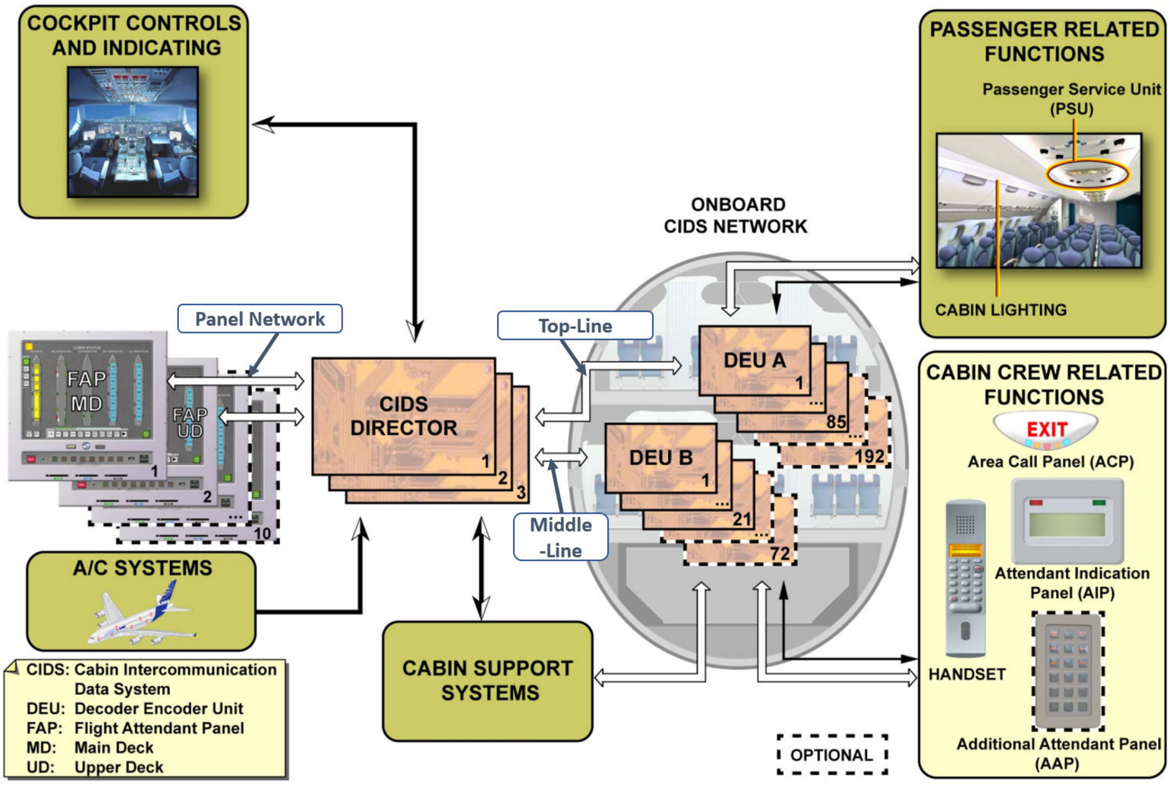

COCKPIT CONTROLS AND INDICATING
PASSENGER RELATED FUNCTIONS
Passenger Service Unit (PSU)
CABIN LIGHTING
ONBOARD CIDS NETWORK
Panel Network
Top-Line
FAP MD
FAP UD
1
2
10
CIDS DIRECTOR
1
2
3
DEU A
1
85
192
DEU B
1
21
72
Middle -Line
CABIN CREW RELATED FUNCTIONS
EXIT
Area Call Panel (ACP)
Attendant Indication Panel (AIP)
HANDSET
Additional Attendant Panel (AAP)
A/C SYSTEMS
CABIN SUPPORT SYSTEMS
OPTIONAL
CIDS: Cabin Intercommunication Data System
DEU: Decoder Encoder Unit
FAP: Flight Attendant Panel
MD: Main Deck
UD: Upper Deck

Figure 4.3: A380 CIDS Architecture (modified from [93])

for the analysis and optimization, is the electrical power system. It is also one of the aircraft's most complex systems and includes electrical power generation, electrical power distribution, electrical loads, as well as monitoring and protection activities. Technical and commercial loads are also distinguished [101][102]. In terms of flight and landing safety, the importance of technical loads can vary but stays relevant for safety in contrast to the commercial loads and functions. Alternating current is often produced by the aircraft power generating system. For some applications, a part of this power is converted to direct current. Conventional airplanes use 115V AC with a fixed frequency of 400 Hz and 28V DC to power the cabin and cargo loads.

Traditional airplanes (e.g. A320) use a centralized network design, which is the most prevalent. One of several power sources is used to deliver electricity to the aircraft's primary Electrical Power Distribution Center (PEPDC), which is positioned in the aircraft nose. Power is based on different types of sources, including primary engine generators, Auxiliary Power Units (APUs), and Ground Power Units (GPUs). Circuit Breaker Panels (CBPs) are used to connect systems to the PEPDC directly for load protection. This centralized architecture makes it simple to operate while still keeping each system in the electrical network distinct. A failure of the PEPDC or CBP, however, will cause the entire network to fail. A cable failure may have an effect on one system. This form of connection has the disadvantage of requiring longer cable lengths, potentially with greater weight, when compared to a decentralized topology [102].

Last year's passenger airliners have an electrical system with a more decentralized power system. This decentralized method was used for a variety of reasons, including decreased system weight and easier installation in the final assembly process. New components, such as the Secondary Power Distribution Boxes (SPDB), have been incorporated into the electrical power distribution network in recent aircraft development (see fig. 4.4). 115/200V AC and 28V DC electrical loads are supplied by the SPDBs in the cabin and cargo compartments.

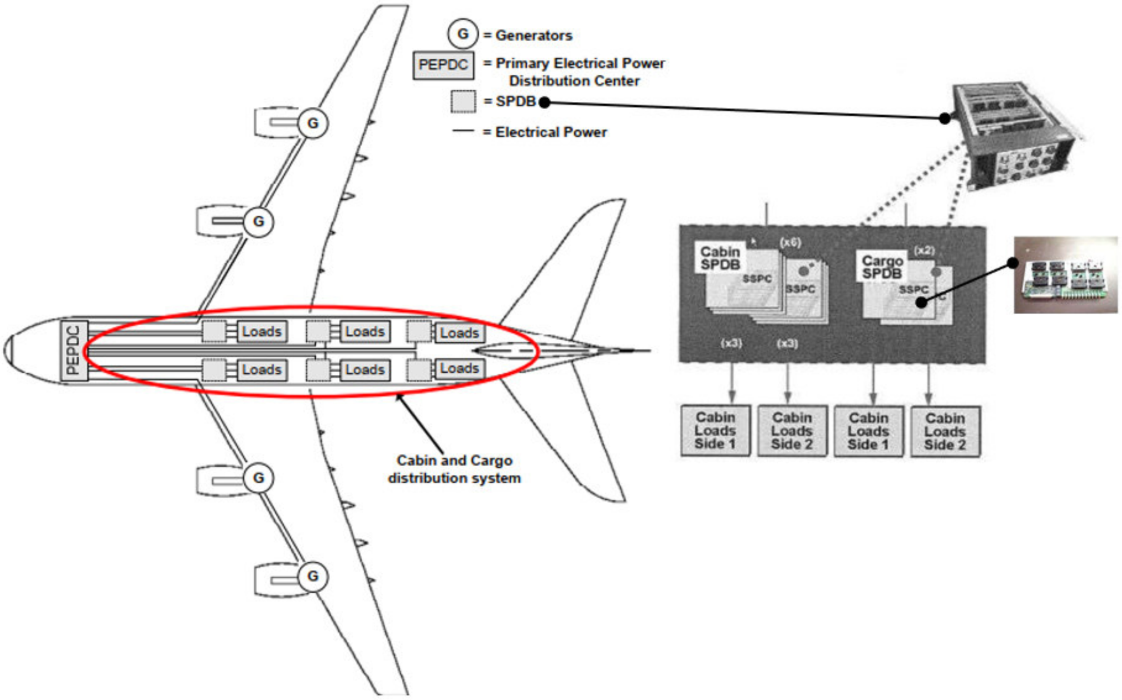


Figure 4.4: Decentralised architecture with SPDBs in modern aircraft architecture modified from([101][93])

Besides of the distributional function, the SPDB also protects cables of the system and performs modern maintenance activities and power management. Dispersion of power supply and smaller cable length requirement are the major benefits of the SPDB integration. Therefore, for non-essential loads, the SPDB use Solid State Power Controller (SSPC) technology rather than circuit breakers. This technology beholds the following features: digital protection, status monitoring, fusible ink backup and voltage transient protection [93]. A detailed description of investigation results of the optimization of the electrical system and also the weight evaluation of cabin power architecture is given in [101] and [102]. These analyses and investigations served as a model for the use cases in this work.

The Cabin Core System (CCS) also considers the combination of electrical power and data management. CCS is the primary management platform for the cabin's power supply, operation, control and monitoring, as well as system testing [103]. *Hintze et al.* have developed a new generation of platform (see fig 4.5) to satisfy top performance objectives (TPOs), such as maximum flexibility and performance with an improved installation concept, rapid processing in the final assembly line (FAL), and weight reduction.

The cabin backbone could be a single network that provides all of the data and power to all

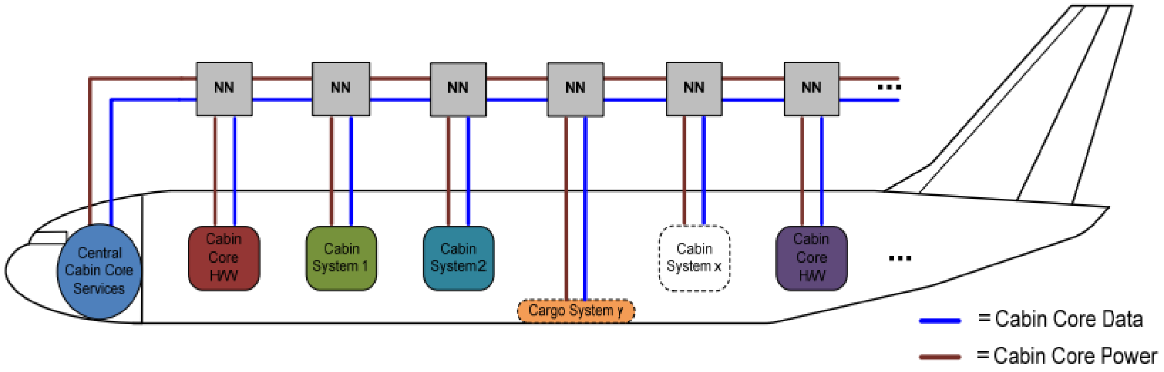


Figure 4.5: New CCS platform with parallel data and power network [103]

of the cabin devices. Each system is linked to the CCS via network nodes (NNs), which would connect the power supply and cabin systems as well as offer bi-directional data communication within the CCS backbone and with the attached systems. This platform architecture with integrated power and data services aims to reduce the overall complexity of CCS. This is due, in part, to the system's higher level of integration and usage of modern data network technologies.

In this work, a use case for PSC's architecture optimization for the electrical design is considered (cf. 7). The goal is to assess the PSC's intended architecture modeling and integration in the cabin design process. The next section presents the physical fundamentals needed in this use case.

4.3 Physical Fundamental for Electrical Design

As stated in section 4.2, the SPDB provides electrical power to cabin loads. The SPDBs are installed in specific locations throughout the cabin. The SPDB location is considered fixed in the intended use case, which is consistent with realistic aircraft development because the SPDB location is not part of the configuration parameters. Each PSC is linked to one of the SPDBs to receive power. Power cables with specific physical properties are used to connect the two components. The electrical model in figure 4.6 illustrates the interface between a PSC and a SPDB.

The SPDB power is required in this model to supply the connected loads. Thermal effects cause a portion of the provided power to be lost during power transmission via cables. The load then consumes the remainder of the power. The PSC's control unit is the component that receives energy and supplies it to the PSC's internal components. For each PSC configuration, the power rating, which is the maximum power input allowed to flow through the component, is used as a parameter.

Kirchhoff's circuit law, which addresses the potential difference (the voltage), is used to define the model mathematically. It asserts that the directed sum of potential differences (voltages) around any closed loop is equal to zero. The voltage from the source, in this case the SPDB,

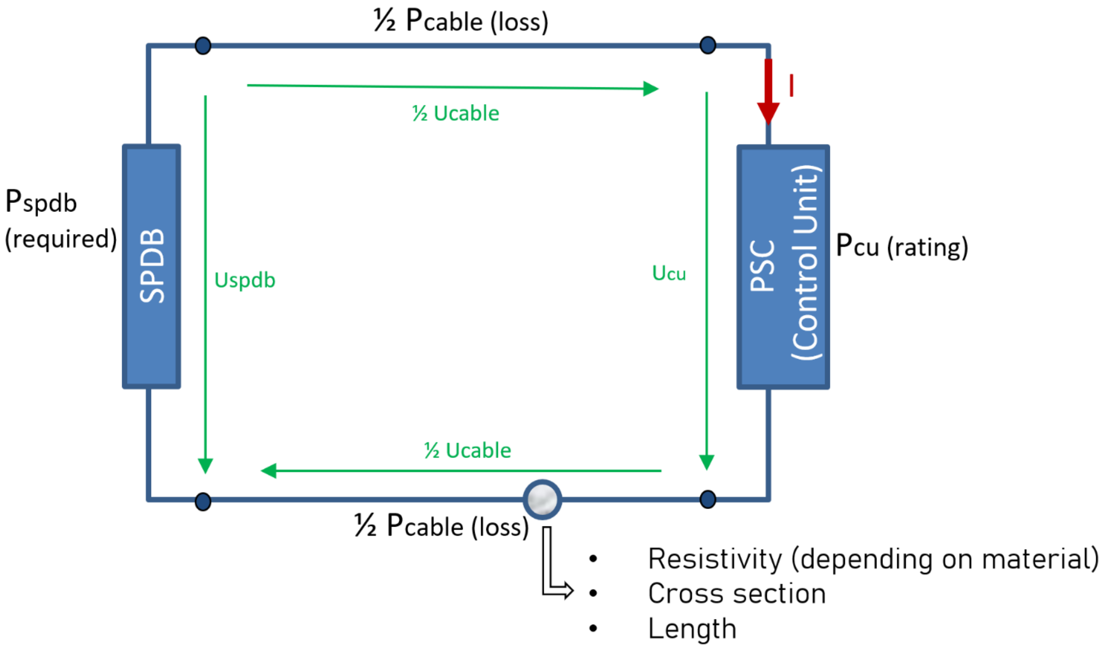


Figure 4.6: Electrical model for the PSC's power supply

is equal to the sum of the partial voltages in a mesh [104]. These are the voltages of the cable and the control unit. Thus, the voltage equality in this model is expressed as follows:

$$U_{SPDB} = U_{cable} + U_{cu}. \tag{4.1}$$

The SPDB, as indicated in section 4.2, can supply direct current to the loads (usually 28VDC). So, the following relationship between the power, voltage and current intensity can be considered [104]:

$$P = U \cdot I. \tag{4.2}$$

By means of the equations 4.1 and 4.2, the following power relationships for the model in figure 4.6 can be formulated as follows:

$$P_{SPDB} = P_{cable} + P_{cu}. \tag{4.3}$$

Given that the SPDB does not only deliver power to one PSC but multiple, the power relationship in 4.3 becomes:

$$P_{SPDB} = \sum_{i=1}^{n} P_{cable\ i} + P_{cu\ i}. \tag{4.4}$$

If all of the PSCs in the cabin are similar, the rating power of their respective control unit is the same. As a result, the sum of their rating power can be defined as follows:

$$\sum_{i=1}^{n} P_{cu\ i} = n \cdot P_{cu}. \tag{4.5}$$

Because the cable only experiences thermal power loss, this can be regarded as ohmic resistance. According to Ohm's law, the electric current flowing through an object is proportional to it's electric voltage [104]. Using the relationship in 4.2, the cable power loss is defined as follows:

$$\sum_{i=1}^{n} P_{cable\ i} = 2 \cdot \sum_{i=1}^{n} R_i \cdot I^2. \tag{4.6}$$

Because the cable voltage must consider both directions of power supply (to PSC and back to SPDB), it is multiplied by two in equation 4.6. Furthermore, the electrical resistivity is used to relate the cable's resistance to its physical parameters (length, cross section, and material). This is a fundamental feature of a material that indicates how well it resists current flow [104]. It is defined mathematically as follows:

$$\rho = R \cdot \frac{A}{l}, \tag{4.7}$$

with ρ representing the material's specific resistivity, l representing the cable length, and A representing the cable cross section's area. By substituting the resistivity relationship in 4.7 in the equation 4.6, together with relationship in 4.5, the power that an SPDB supplies defined in 4.4 can now be expressed as follows:

$$P_{SPDB} = n \cdot P_{cu} + 2 \cdot \sum_{i=1}^{n} \rho_i \cdot \frac{l_i}{A_i} \cdot I^2. \tag{4.8}$$

Assuming that all control units are connected with the same type of cable, the resistivity and cross section area of all cables remain constant. As a result, the expression in 4.8 can be written as follows:

$$P_{SPDB} = n \cdot P_{cu} + 2 \cdot \rho \cdot \frac{1}{A} \cdot I^2 \cdot \sum_{i=1}^{n} l_i. \tag{4.9}$$

The physical relationships explained above describe the electrical design model connecting the PSC to the SPDB. They serve as an input for the optimization use case that will be presented in section 7.

An overview of the system-of-interest and the main interacting systems is thus provided. The methodology described in chapter 3 demonstrated how the modeling language and tool are used to achieve the main goals of this work, whilst taking into account necessary principles and concepts, such as agility and knowledge management. It specified which information is extended through the system architecture development process and how its integration into the cabin design and visualization process enables new analyses and identifies new system synergies. However, the description of the methodology is abstract and based on theoretical methods (such as OOSEM and *Abulawi*'s approach). To test and validate the methodology, it is now used to analyze and design the selected SoI. The following chapter explains how the methodology is applied to model the system architecture.

5 The PSC's Model-based System Architecture

5.1 Model Organization and Structure

As stated in the methodology chapter, organizational tasks must be completed before proceeding with the actual system architecture modeling. Even though the main methodological scope and steps are already defined, each modeling use case and each considered system have their own unique properties, that can only be discovered during the modeling activities. For this reason, the model organization necessitates iterative and continuous adjustment throughout the modeling process.

The sections that follow describe these tasks, which include, on the one hand, the definition of a model structure - a specific schema to organize the model elements and must be uniformly respected throughout the model - on the other hand, the definition of a system ontology that extends the SysML language and is specific to this work's use case and SoI.

5.1.1 Definition of Model Structure

The model organization begins with a broad overview of its hierarchical structure. The model structure in this work is oriented to the aircraft systems categorization, based on the ATA chapters. Figure 5.1 shows the top-level package diagram including the system model as well as the imported and associated packages. First, the top-level package for the PSC is a model package (denoted by a triangle) that contains all of the model elements. It imports specific packages in order to use the required external elements. Modeling elements for the application of the FAS method, for example, are imported from the corresponding FAS package. Also files (e.g. CPACS files) that include data used in the system model are imported. Additionally, external model packages (e.g. aircraft model) including system elements that interact with the PSC internal elements are associated to the model package. Furthermore, model elements that are not used implicitly for system representation (e.g. external files or glossary) as well as global elements (global objects, constraints, etc.) are stored in the library and can be used multiple times in the model.

The system model package is organized in a specific way. It is made up of numerous packages for each aspect of system analysis or representation, such as requirements, structure, behavior, traceability and analysis. The structure package is the central package because it contains

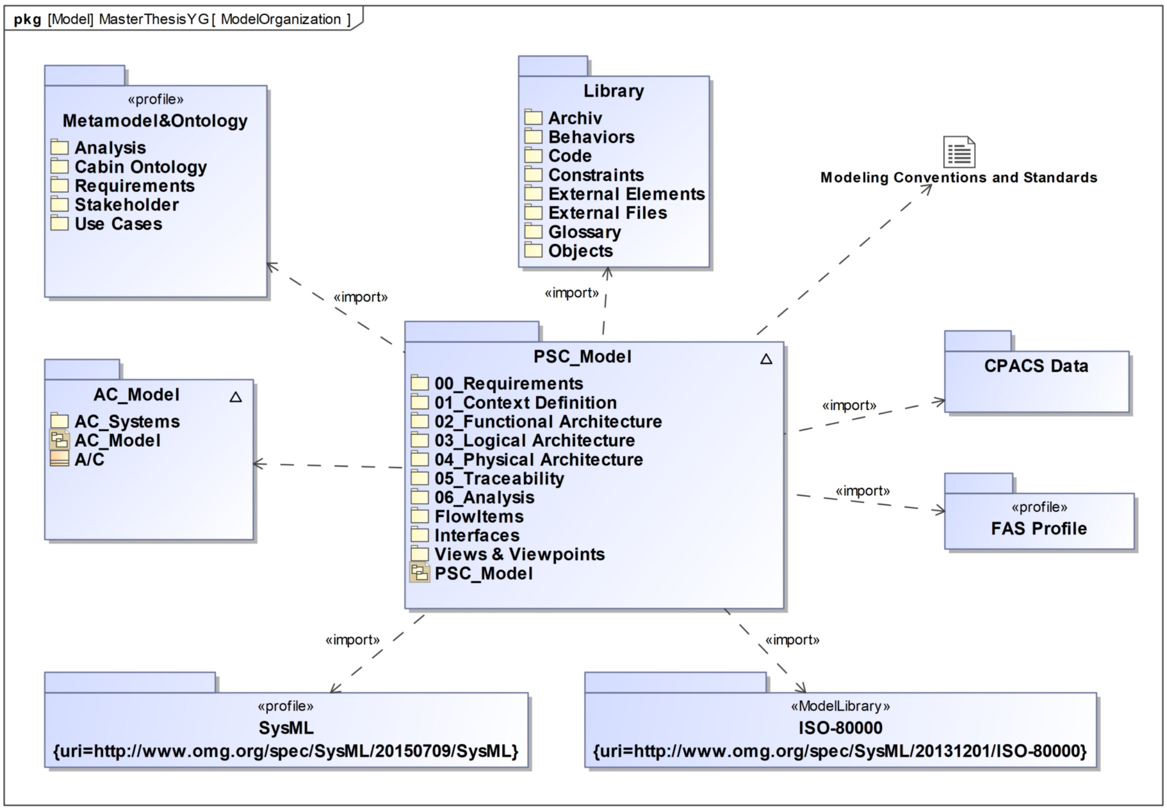


Figure 5.1: Top-level package diagram for PSC Model

the system block and its parts. These have their own package with identical structure as well. This package organization is recursively repeated for each system, subsystem, or component in the model. Hyperlinks connect the packages to the package diagrams. The packages can also be linked directly to other diagrams if one of the aspects represented by this diagram is the most important. This allows for user-friendly model navigation, in which a user can navigate through the system and its related elements with a double click (similar to HTML web navigation). The type of the hyperlinked model element is displayed on the package's bottom left side. As stated in section 2.3.2, all model elements, including packages and their described structure, can be found in the model repository.

In a subsequent step, the modeling standards and guidelines are defined. Therefore, the UML content diagram is used to define the main rules that must be followed while modeling (see fig. A.2. As previously stated, these conventions are based on the OOSEM methodology (cf. chapter 3) and also the standards used in the integrated external models (Matlab and CPACS). They state precisely the above-mentioned notation and syntax for modeling and are critical for proper data exchange and communication. No automated model validation has been implemented in this work because the effort required is beyond this work's scope. As a result, the validation is carried out manually. In addition, an aircraft-system-specific glossary has been created to define the model's acronyms (see fig. A.3).

In addition to the modeling notation described above, a system oriented notation is required to provide a modeling language extension, that considers the SoI, which is the PSC in the context of aircraft systems. This is accomplished through an ontology specification, which is described in the following section.

5.1.2 Definition of System Ontology

The first modeling step is the definition of the system ontology. This is due to the fact that syntax rules and constraints are defined at this stage and must be followed throughout the modeling process. However, this definition is a continuous and recursive activity that must be adjusted all across the analysis and modeling stages.

Since the CPACS representation has not (yet) defined any internal structure for aircraft systems, this work ontology considers it only for integration into the CPACS schema. This is the "genericSystems" node under "vehicles" > "aircraft" > "models" > "Systems." This is also the starting point for the modeled ontology depicted in Figure 5.2. Therefore, a UML profile diagram is used to extend SysML notations with stereotypes.

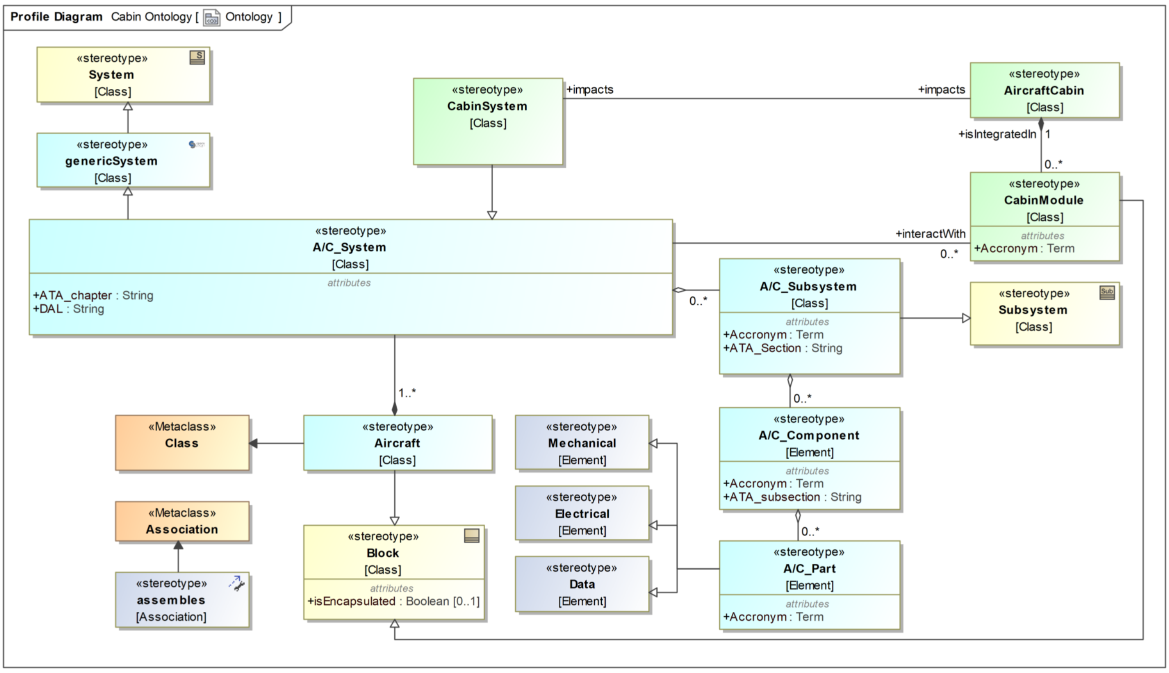

Profile Diagram Cabin Ontology [Ontology]
«stereotype» System [Class]
«stereotype» genericSystem [Class]
«stereotype» CabinSystem [Class]
+impacts
+impacts
«stereotype» AircraftCabin [Class]
+isIntegratedIn
1
0..*
«stereotype» CabinModule [Class]
attributes
+Accronym : Term
«stereotype» A/C_System [Class]
attributes
+ATA_chapter : String
+DAL : String
+interactWith
0..*
0..*
«stereotype» A/C_Subsystem [Class]
attributes
+Accronym : Term
+ATA_Section : String
«stereotype» Subsystem [Class]
1..*
0..*
«Metaclass» Class
«stereotype» Aircraft [Class]
«stereotype» Mechanical [Element]
«stereotype» A/C_Component [Element]
attributes
+Accronym : Term
+ATA_subsection : String
«Metaclass» Association
«stereotype» Electrical [Element]
0..*
«stereotype» Block [Class]
attributes
+isEncapsulated : Boolean [0..1]
«stereotype» Data [Element]
«stereotype» A/C_Part [Element]
attributes
+Accronym : Term
«stereotype» assembles [Association]

Figure 5.2: Aircraft system ontology

The SysML metaclass and some of its elements are used to model the attributes of new stereotypes (semantic) and their relationships with one another (syntax). The ontology's center element is the stereotype "A/C System", which is directly specialized from the CPACS "genericSystem" stereotype. The SysML generalization relationship is used to implement this specification (arrow showing in the direction of the general element). The ATA chapter and the DAL are two properties of the AC System stereotype. As a consequence, all model elements that are stereotyped by this (or inherited from an element stereotyped by it) have these attributes by default. According to the ATA specification (see section 4.2), the system hierarchy is represented by an aggregation relationship (white diamond on the including element) and the corresponding multiplicity, with the star marking the undefined upper boundary. "AC_-system", "AC_Subsystem", "AC_Component", and "AC_Part" (light blue element in fig. 5.2) are all part of this hierarchy.

To distinguish system parts regarding their physical type (electrical, mechanical, and data parts), a generalization relationship is used. This improves the clarity and understanding of modeled cyber-physical systems, where all three of these aspects are present. The "AC_-System" is one of the elements stereotyped as "Aircraft" because it constitutes itself a node belonging under the aircraft node in CPACS (e.g. engines, wings, landingGear, etc.). The composition connector (black diamond) defines this relationship, which differs from the aggregation relationship in that the general element consists of at least one or more of the associated parts. The SysML stereotype "Aircraft" is an extension (black arrow) of the SysML Metaclass "class" and a specialization of the SysML stereotype "Block." All SysML original stereotypes are yellow, while metaclass elements are orange.

This work focuses on specific aircraft cabin systems. For this reason, the general AC_-System is specialized in "CabinSystem" stereotypes. Not only systems, but also modules (e.g. the PSC) that group different parts from different cabin systems, are relevant. These cabin modules are parts of the aircraft cabin, as determined by the composition relationship between the stereotypes "AircaftCabin" and "CabinModule." All cabin related stereotypes are colored in green. Cabin modules "interact" with aircraft systems and are "integrated" into the cabin using specialized association relationships. The latter are also used to demonstrate how the cabin and cabin systems impact one another. Furthermore, the stereotype "assembles" has been specialized from association relationship to specify how the various parts are assembled inside the cabin modules.

The ontology depicted in figure 5.2 is more descriptive of the system's structural aspects. It is then expanded in stages to include analytical, functional, and requirements-related system aspects (see fig. A.6, A.5, A.4 and A.7). The analysis is based on the modeling's abstraction level. Therefore, the stereotypes "Black-Box", "Grey-Box", and "White-Box" are defined to specify the state of the analysis. Thus, the modeler can show which detail level in the modeling is targeted in each view. Moreover, a precise analytical categorization is possible through the usage of specific stereotypes. As a result, new stereotypes for the stakeholders such as "OEM", "Maintainer", "Supplier", "Operator", "Integrator", "Regulator", "Supplier", "Passenger", and "Crew" are defined in the context of aircraft. A three-classification is used to separate different types of system use cases. The "AC_SystemUseCase" stereotype is specified from the SysML use case element at the first level, and new attributes such as "status", "id", "story", "operational value" or "technical feasibility" are created. It is distinguished between system use misuse cases. The use cases are classified based on their deployment type, such as safety, security, performance, maintainability, and functionality. The latter is of major importance in this work since the aim is to depict and analyze the system's functional aspects. For this reason , the PSU-specific use cases are specified from this use case category. Similarly, the requirements are divided into stakeholder needs, which represent the requirements from the perspective of each stakeholder, and system requirements, which support the specification of the system and are defined at various stages of analysis. Attributes for system requirements are created and can be filled in for each requirement in the model, such as level, creation date, status, author, priority, verification method, or assumption. The system requirements are divided into functional and non-functional requirements, with non-functional requirements being mostly left out in this work.

As a result, an ontological framework for PSC modeling, that is oriented toward the context of this work, is established. This ontology is then applied to all modeling activities. These begin with a system analysis, which breaks down the context of the system at the outset. The following section addresses the analytical approach used and the modeling results obtained.

5.2 System Analysis and Functional Architecture

The system analysis is the main contribution of this work to the already existing design process. To create a consistent and traceable architecture, the elements of the emerging system architecture must be derived from different analytical abstraction levels, where root causes leading to each element are identified and analyzed. Furthermore, modeling a block-oriented functional architecture allows for the separation of abstract analysis aspects and first functional results.

The sections that follow describe how the analytical processes for generating the functional architecture are applied to the passenger service channel using the proposed methodology.

5.2.1 Stakeholders and System Context Analysis

It is critical to begin with a stakeholder analysis since it enables system design that is closely aligned with the needs of the concerned stakeholders. On the one hand, it demonstrates that subsequent analysis and design are motivated by their interests and requirements in such a way that the resulting architecture conforms to what the stakeholders truly expect. On the other hand, it is part of the validation process that assists designers and engineers in ensuring that the system is specified with the correct requirements by always tracing system elements to the top-level stakeholder needs.

Figure 5.3 shows the results of the stakeholder analysis of the PSC. A bdd is used to analyze and model these aspects. Initially, the stakeholders are identified by establishing a number of actors based on preset stakeholder stereotypes. This allows the modeler to ask the proper questions, such as who is the PSC's operator, regulator, developer and so on. These questions can then be explicitly addressed by naming the various actors.

The actors' needs are specified using the stereotype "StakeholderNeed". This allows to define in text form which aspects of the system are important from the standpoint of the stakeholders. Their needs are traced back to the actors. If different actors have the same point of interest, the same need may be traced back to them. For example, as an airline that offers a comfort

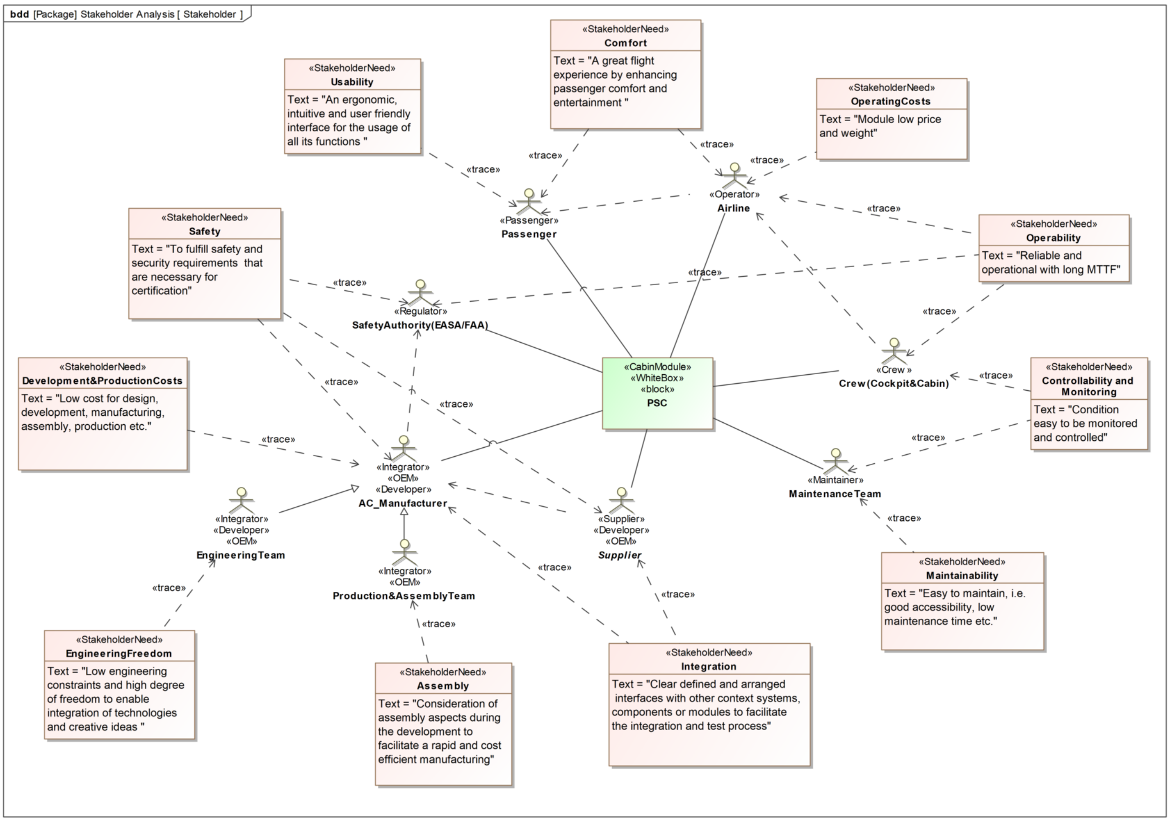

bdd [Package] Stakeholder Analysis [Stakeholder]
«StakeholderNeed»
Comfort
Text = "A great flight experience by enhancing passenger comfort and entertainment "
«StakeholderNeed»
Usability
Text = "An ergonomic, intuitive and user friendly interface for the usage of all its functions "
«StakeholderNeed»
OperatingCosts
Text = "Module low price and weight"
«StakeholderNeed»
Safety
Text = "To fulfill safety and security requirements that are necessary for certification"
«StakeholderNeed»
Operability
Text = "Reliable and operational with long MTTF"
«StakeholderNeed»
Development&ProductionCosts
Text = "Low cost for design, development, manufacturing, assembly, production etc."
«StakeholderNeed»
Controllability and Monitoring
Text = "Condition easy to be monitored and controlled"
«StakeholderNeed»
Maintainability
Text = "Easy to maintain, i.e. good accessibility, low maintenance time etc."
«StakeholderNeed»
EngineeringFreedom
Text = "Low engineering constraints and high degree of freedom to enable integration of technologies and creative ideas "
«StakeholderNeed»
Assembly
Text = "Consideration of assembly aspects during the development to facilitate a rapid and cost efficient manufacturing"
«StakeholderNeed»
Integration
Text = "Clear defined and arranged interfaces with other context systems, components or modules to facilitate the integration and test process"
«Passenger»
Passenger
«Operator»
Airline
«Regulator»
SafetyAuthority(EASA/FAA)
«CabinModule»
«WhiteBox»
«block»
PSC
«Crew »
Crew(Cockpit&Cabin)
«Integrator»
«OEM»
«Developer»
AC_Manufacturer
«Maintainer»
MaintenanceTeam
«Integrator»
«Developer»
«OEM»
EngineeringTeam
«Supplier»
«Developer»
«OEM»
Supplier
«Integrator»
«OEM»
Production&AssemblyTeam
«trace»

Figure 5.3: Stakeholders' identification and definition of needs

service and adapts the price of flight tickets accordingly, the aspect of comfort is crucial. Of course, this also applies to the passengers who are the PSC-users. The definition of needs also assists in determining the interdependence of the various stakeholders. This is represented using a dependency relationship to link requirements not just to the primary but also to related stakeholders during the traceability process. After identifying and associating stakeholder needs, PSC-stakeholder requirements can be used to refine each of these needs. However, since it is critical to involve stakeholders throughout the entire analysis and design process, these requirements must be defined and adjusted on a continuous basis.

Some of the identified stakeholders with an interface (of any type) with the PSC are selected for the context analysis that follows. The bdd in figure 5.4 depicts the modeled PSC context. Passengers, cabin and cockpit crew, as well as maintenance, production, and assembly teams, have been opted for as stakeholders. Furthermore, environmental elements that may have an impact on the PSC are identified and linked. Qualification standards documents (such as RTCA DO-168 for hardware qualification cf. [105]) are good references for analyzing which environmental elements are included and whose influence must be tested in a later qualification stage of the system life cycle. These elements are modeled using the SysML actor type "EnvironmentalEffect".

The following analysis stage focuses on physical interfaces. The SysML stereotypes "Boudary system" and "External system" are used to represent these. These elements are refined with the help of blocks that have been stereotyped by the developed ontology. Aircraft systems

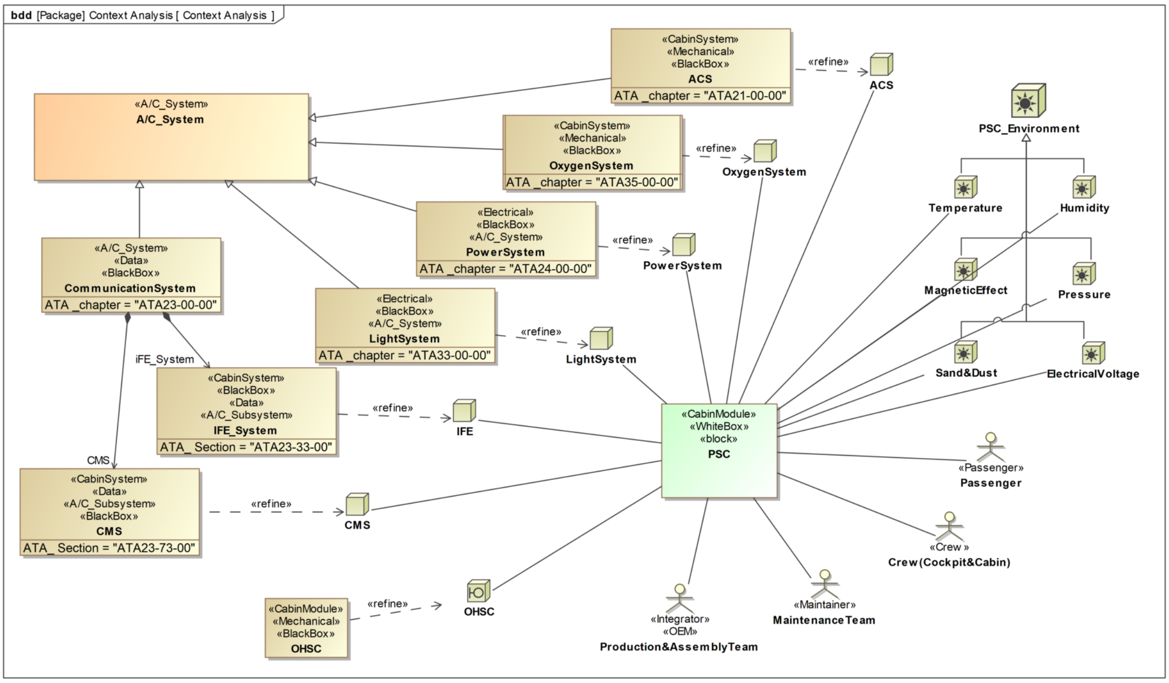


Figure 5.4: Context Analysis of the PSC

(e.g. oxygen or power) or subsystems (IFE, CMS) that have a mechanical, electrical, or data interface with the PSC are specified. At this stage, the ATA chapter attribute is crucial for analyzing the PSC's interaction with the various systems. The use of this attribute is especially important for an analysis of a cabin module like the PSC, where components and parts of different systems with separate ATA chapters are integrated together in the same module. The OHSC cabin module, into which the PSC is integrated, is also depicted as a boundary cabin module. The analysis stereotypes for each block are defined and continuously adapted to keep the model-user up to date on the representation details stage.

The analysis modeled in the bdd in figure 5.4 only describes the PSC context elements and their attributes. The interfaces with them have yet to be specified. Thus, an ibd is used to represent the PSC turnover and to depict its exchange of diverse types of flows (e.g. material, data or energy) with its context (see fig. 5.5). However, it is important to note that at this stage of the analysis, the PSC is still considered a black-box, which means that the internal aspects of the module have not been depicted yet.

The created actors and blocks for interacting contextual elements are imported into the ibd to accomplish this. The CSM tool generates part properties for each of these blocks automatically. Both the PSC's and external elements' interfaces are specified with SysML's "proxy ports" and typed by SysML's "interface blocks". A proxy port is a port that defines the characteristics of owning blocks or internal parts that are accessible to external blocks through external connectors to the ports. It does not define distinct components of owning blocks or internal parts. An interface block is the only way to type it [106]. The same interface block can be used to type various ports. The gas output interface, for example, is used by both the oxygen and the air conditioning systems. The flow direction is also determined by the interface block through the specification of a flow property. Because SysML does not allow actors (in this case, "passenger" and "PSC Environment") to have ports, interface blocks can only be applied to blocks. For this reason, these elements are directly linked to unspecified ports.

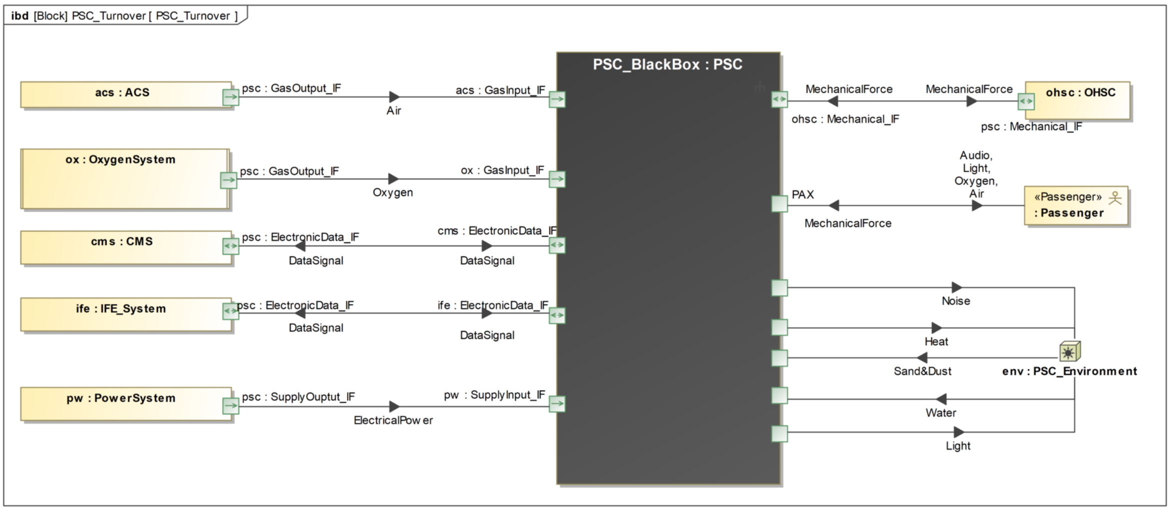


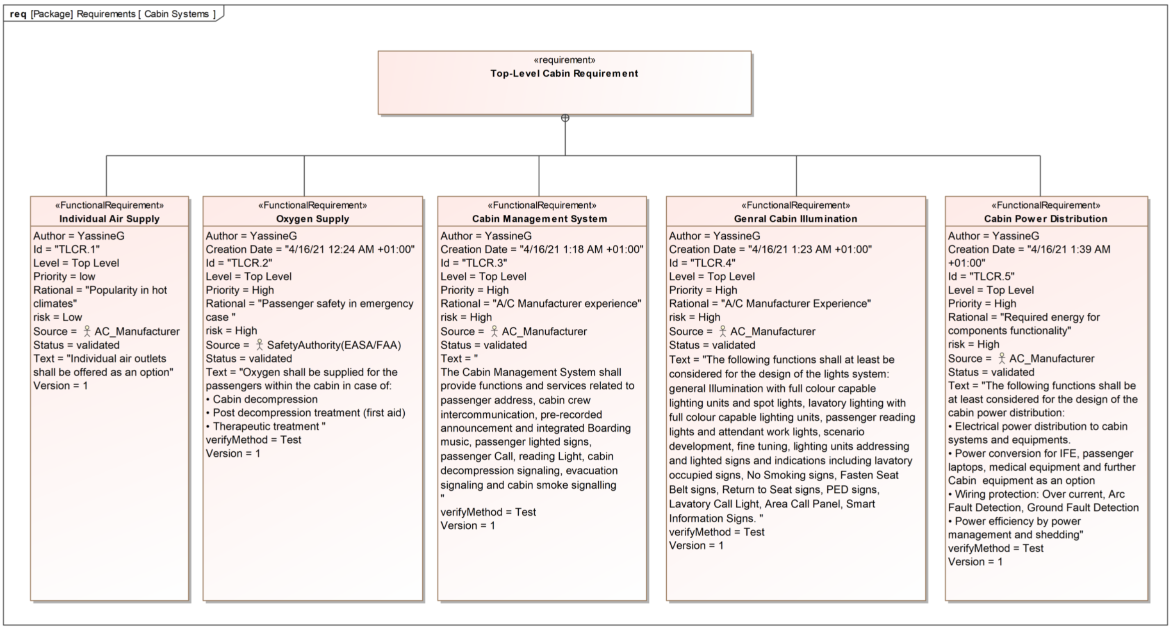
Figure 5.5: PSC's turnover and interface specification

The flow items (such as electrical power, data signal, or oxygen) are modeled using SysML's "flow specification" to specify the flow between the PSC and external elements. They are applied to the connectors in the corresponding flow directions. The CSM tool checks whether the flow type and direction match the previously defined interface blocks.

Similar to the stakeholder analysis, context requirements are specified based on the described analysis. Hence, each interaction with the external elements necessitates the use of a specific interface, which is picked out in the corresponding requirements. This describes how the PSC should be designed so that it interacts perfectly within its context. All gathered requirements serve as a starting point for the deviation of the system use cases intended in the next step.

5.2.2 Derivation of Use Cases from Top-Level Requirements

As previously stated in chapter 3, this work's methodology employs a top-down analysis approach, which means that the system's top-level requirements are defined first. The use cases are defined and refined based on the required functions to accomplish the system's goals. This results in the system's functional specification. However, in the case study presented in this work, the SoI, i.e. the PSC, is a module that groups components and parts from various aircraft systems. As a result, there are several systems whose use cases partly include the PSC. To determine which use cases are relevant for the PSC, top-level cabin requirements are considered, since all PSC functions are cabin functions. These requirements are made based on simple assumptions and are not derived from a real aircraft specification. They are shown in the requirements diagram in figure 5.6. Each of these requirements refer to the main functions of each involved system and only those use cases that pertain to the PSC being further analyzed and modeled.



Individual Air Supply
Author = YassineG
Id = "TLCR.1"
Level = Top Level
Priority = low
Rational = "Popularity in hot climates"
risk = Low
Source = AC_Manufacturer
Status = validated
Text = "Individual air outlets shall be offered as an option"
Version = 1

Oxygen Supply
Author = YassineG
Creation Date = "4/16/21 12:24 AM +01:00"
Id = "TLCR.2"
Level = Top Level
Priority = High
Rational = "Passenger safety in emergency case "
risk = High
Source = SafetyAuthority(EASA/FAA)
Status = validated
Text = "Oxygen shall be supplied for the passengers within the cabin in case of:
• Cabin decompression
• Post decompression treatment (first aid)
• Therapeutic treatment "
verifyMethod = Test
Version = 1

Cabin Management System
Author = YassineG
Creation Date = "4/16/21 1:18 AM +01:00"
Id = "TLCR.3"
Level = Top Level
Priority = High
Rational = "A/C Manufacturer experience"
risk = High
Source = AC_Manufacturer
Status = validated
Text = "
The Cabin Management System shall provide functions and services related to passenger address, cabin crew intercommunication, pre-recorded announcement and integrated Boarding music, passenger lighted signs, passenger Call, reading Light, cabin decompression signaling, evacuation signaling and cabin smoke signalling
"
verifyMethod = Test
Version = 1

Genral Cabin Illumination
Author = YassineG
Creation Date = "4/16/21 1:23 AM +01:00"
Id = "TLCR.4"
Level = Top Level
Priority = High
Rational = "A/C Manufacturer Experience"
risk = High
Source = AC_Manufacturer
Status = validated
Text = "The following functions shall at least be considered for the design of the lights system: general Illumination with full colour capable lighting units and spot lights, lavatory lighting with full colour capable lighting units, passenger reading lights and attendant work lights, scenario development, fine tuning, lighting units addressing and lighted signs and indications including lavatory occupied signs, No Smoking signs, Fasten Seat Belt signs, Return to Seat signs, PED signs, Lavatory Call Light, Area Call Panel, Smart Information Signs. "
verifyMethod = Test
Version = 1

Cabin Power Distribution
Author = YassineG
Creation Date = "4/16/21 1:39 AM +01:00"
Id = "TLCR.5"
Level = Top Level
Priority = High
Rational = "Required energy for components functionality"
risk = High
Source = AC_Manufacturer
Status = validated
Text = "The following functions shall be at least considered for the design of the cabin power distribution:
• Electrical power distribution to cabin systems and equipments.
• Power conversion for IFE, passenger laptops, medical equipment and further Cabin equipment as an option
• Wiring protection: Over current, Arc Fault Detection, Ground Fault Detection
• Power efficiency by power management and shedding"
verifyMethod = Test
Version = 1

Figure 5.6: Top-level cabin requirements as a starting point for use case analysis

The previously defined stakeholder and context requirements are important for determining which functions and thus use cases are relevant. Relating functions are identified and further developed in the external system's use cases. Included are both functions that contribute to the stakeholders' needs satisfaction and those that are satisfied via interaction with contextual systems. After analyzing these PSC's functions, it's clear that the majority is related to the cabin management system. Because *Fisher* thoroughly analyzed this system and elaborated on its use cases in his research (cf. [107]), several of them are incorporated into this work. Thus, the use cases that intersect with the PSC's functions are identified.

Use case diagrams with corresponding elements and relationships are used. For instance, figure 5.7 illustrates how cabin management system use cases (derived from Fischer's work) are analyzed in order to generate PSC use cases. The system use case "provide cabin services" is broken down into multiple use cases that must be combined to obtain the use case. This is depicted through "include" relationships. Use cases, that are not required for the parent use case's realization but add functionality to it (for example, the use case "provide comfort"), are modeled using the "extend" relationship and corresponding extension points. The actors who interact with the system in order to realize specific use cases are assigned to the corresponding use case using the "association" relationship.

Each use case is then refined in its own use case diagram, as illustrated in figure 5.7 for the use case "provide communication and connectivity". Furthermore, this one is divided into use cases until a PSC-related use case is identified. PSC use cases are highlighted in green to distinguish them from the rest, as demonstrated in this example for the use cases "provide recorded announcements" and "provide boarding music". Another way to differentiate be-

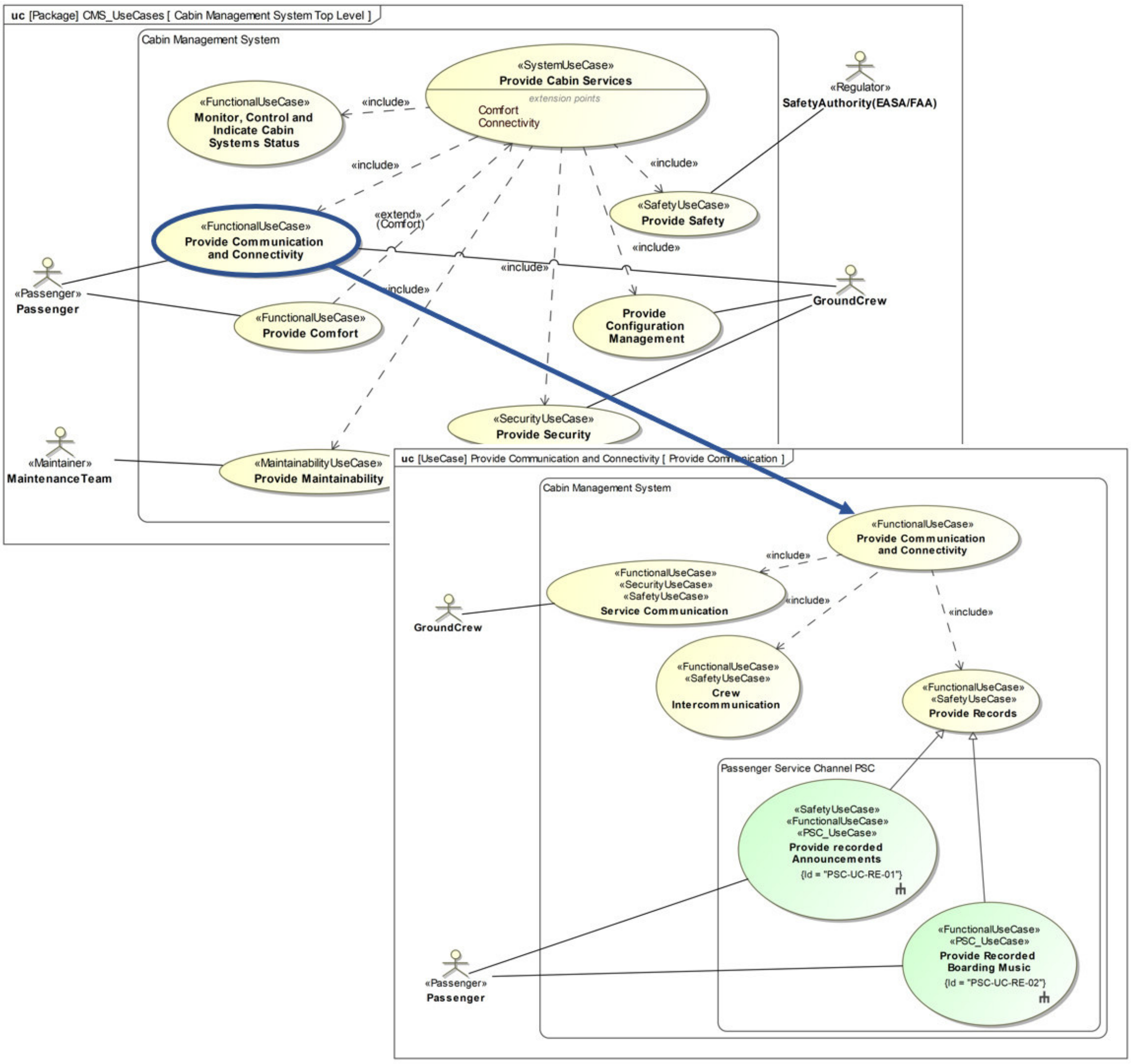


Figure 5.7: CMS Use Case analysis and derivation of PSC Use Cases

tween use cases is through the use of applied stereotypes defined in the use case ontology. As a result, these particular use cases are denoted by the stereotype "PSC UseCase". Moreover, using the generalization relationship, the PSC use cases are connected to the superordinated system use cases. As illustrated, the PSC use cases are included within the frame of both the PSC and the CMS. This is because they describe the functions of both of them, which demonstrates the functional interconnection between the cabin module and the external system. This approach is applied to all external systems and all PSC use cases are identified. Finally, these are organized in a separate diagram (see Appendix fig. A.8).

The use cases illustrate the various functional scenarios for the PSC's use. They encompass all functions that a PSC module should perform in conjunction with external systems and actors. Therefore, these use cases are conducive to refinement in accordance with functional requirements. It is essential to determine these requirements as part of the PSC's functional specification. The resulting functional architecture should then be traced back to these requirements in the following stages. The requirements are modeled using the stereotype

"FunctionalRequirement" in a requirements diagram (see Appendix fig. A.9). Moreover, the FAS method takes as input the modeled use cases and their associations with the various actors. The following section describes how these use cases are refined and how a functional architecture can be generated.

5.2.3 Application of FAS-Method and Definition of Functional Architecture

This work employs the FAS method to generate a functional architecture, that is independent of any technical solution or implementation. This makes PSC or similar systems difficult to implement, because a standard variant has been established over time. Therefore, it is even more intriguing to apply the method to such SoI in order to investigate the core functions of the system and enable new function-oriented visions for these systems.

First, an activity with the same name is created for each of the identified PSC use cases. All created activities are brought together in a bdd with the goal of decomposing PSC functionalities into required subordinated activities. At this stage, only the hierarchical definition of these activities is required, not how and in what sequence these activities interact with one another to realize the functions. Because there is a large number of resulting activities, these have been subdivided into numerous bdds to maintain model clarity. For example, the bdd in figure 5.8 displays the results of a functional analysis of audio use cases. The use cases "provide recorded announcements" and "provide recorded boarding music" (see fig. 5.7) are now subdivided into the same actvities "receive audio signal," "process audio signal," and "provide acoustic signal."

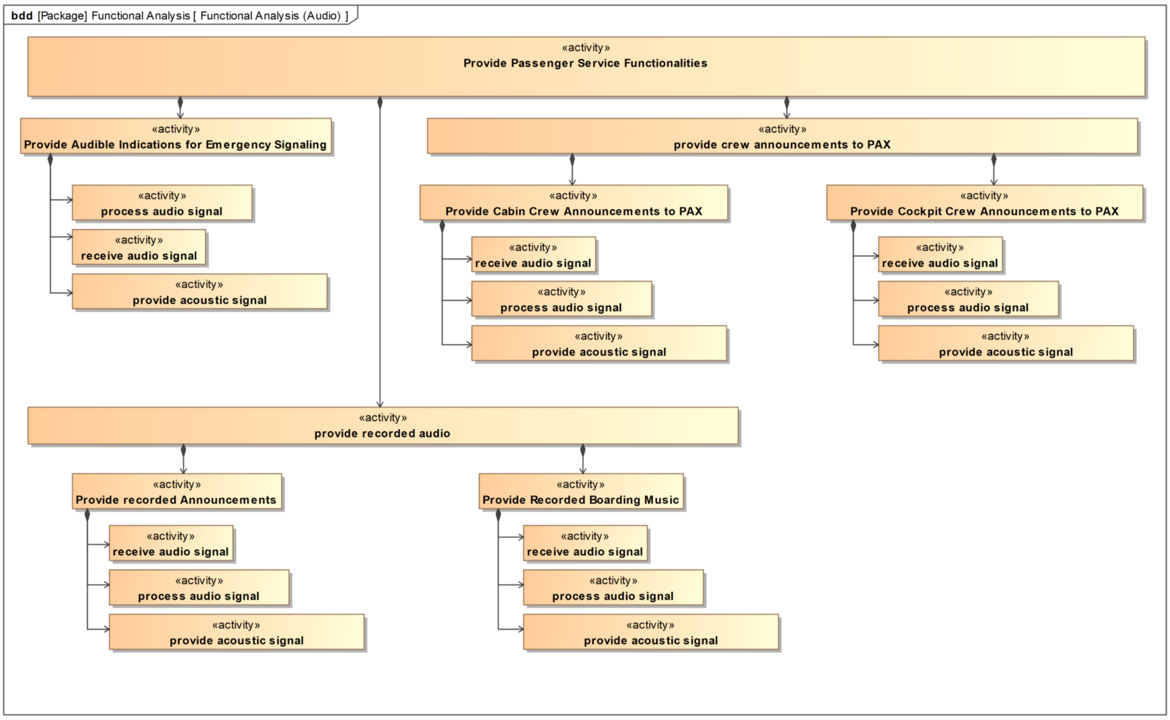

bdd [Package] Functional Analysis [Functional Analysis (Audio)]
«activity» Provide Passenger Service Functionalities
«activity» Provide Audible Indications for Emergency Signaling
«activity» process audio signal
«activity» receive audio signal
«activity» provide acoustic signal
«activity» provide crew announcements to PAX
«activity» Provide Cabin Crew Announcements to PAX
«activity» receive audio signal
«activity» process audio signal
«activity» provide acoustic signal
«activity» Provide Cockpit Crew Announcements to PAX
«activity» receive audio signal
«activity» process audio signal
«activity» provide acoustic signal
«activity» provide recorded audio
«activity» Provide recorded Announcements
«activity» receive audio signal
«activity» process audio signal
«activity» provide acoustic signal
«activity» Provide Recorded Boarding Music
«activity» receive audio signal
«activity» process audio signal
«activity» provide acoustic signal

Figure 5.8: Decomposition of PSC functionalities into separate activities (Audio)

In this work, the FAS plug-in for CSM is used to automate the application of the FAS method. In the first step, the plug-in assists in modeling the activities that have already been created in activity diagrams for each use case. It also creates swimlanes with FAS "I/O" partitions (cf. section 2.4) for each of the external actors or systems that interact with the PSC or are involved in the realization of each specific use case. This is identified by the automation plug-in through the use of created association relationships in the use case diagrams. The created activities are then allocated in different partitions. The I/O partitions are reserved for activities that serve as a means of interacting with the outside world. All remaining activities that describe system functions are assigned to the empty partition. Following that, the activities are linked using "object flows" and "activities pins" to describe the inputs and outputs of each function as well as their sequential interaction. The I/O partition activities have an empty input or output pin that describes the flow exchange with the corresponding external element. For example, the activity diagram created for the activity "provide recorded announcements" is shown in figure 5.9.

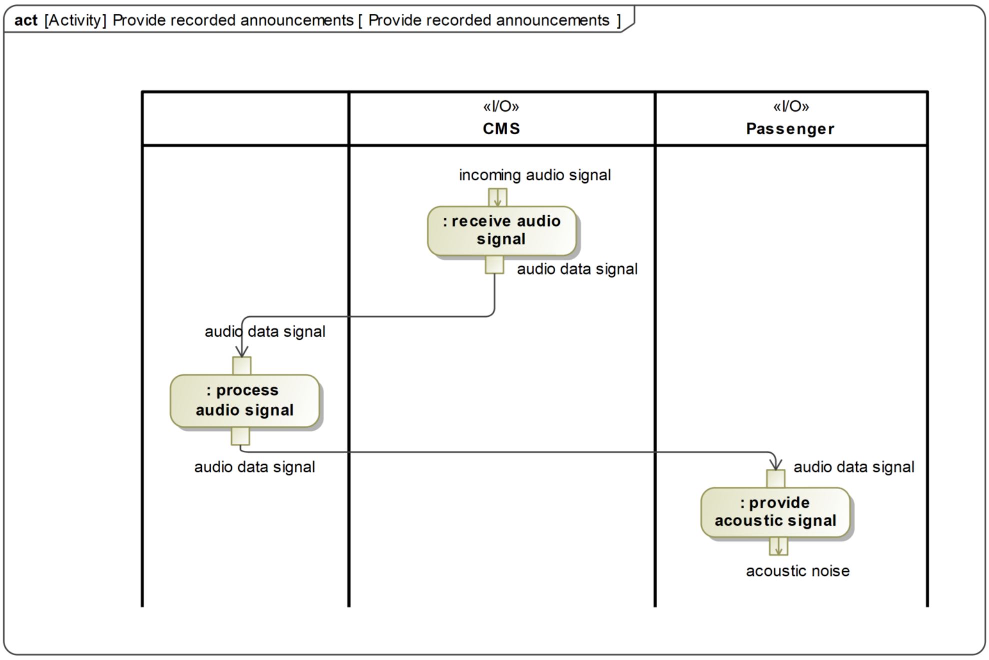
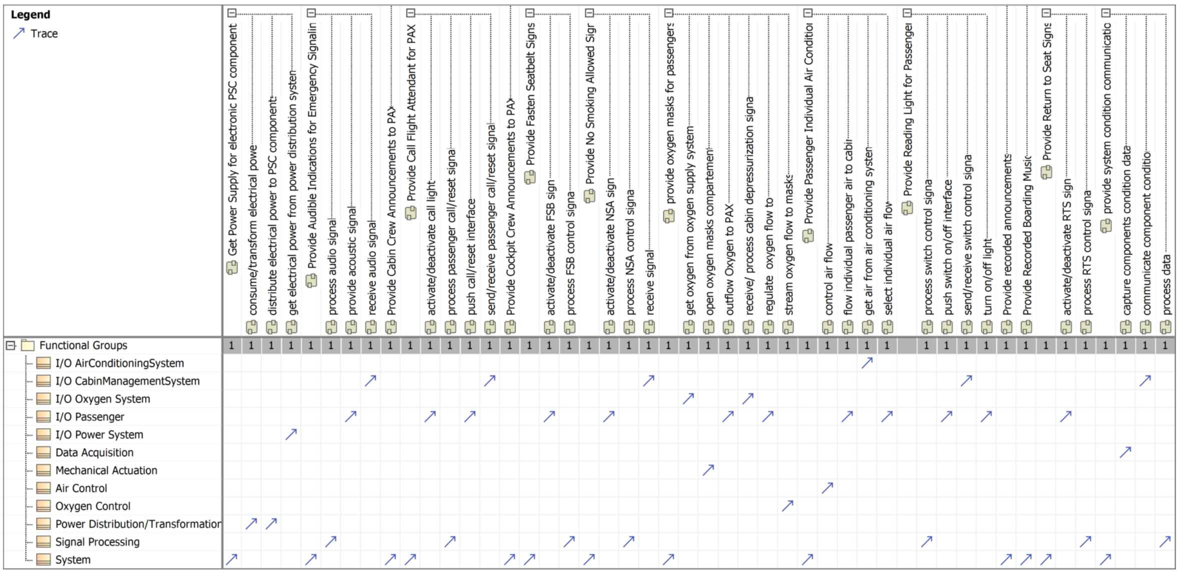


Figure 5.9: Functional behavior for the activity "provide recorded announcements"

After applying this to all functional activities, the FAS plug-in can be used to automatically initiate functional groups. It generates a "functional group" element for each of the I/O partitions and links it to the activities that are allocated in those partitions using the "trace" relationship. Thus, I/O functional groups for the PSC are "I/O AirConditioningSystem", "I/O CabinManagementSystem", "I/O Oxygen System", "I/O Power System", and "I/O Passenger". In addition to this, a "system" functional group is also created and traced to the superordinate activities.

However, no functional group has been assigned to the activities allocated in the empty partitions yet. As a result, these activities are examined in terms of their functional correlation. Based on this analysis and some of the heuristics (described in section 2.4), the new functional groups "Data acquisition", "Mechanical actuation", "Air control", "Oxygen control", "Power distribution/transformation", and "Signal processing" are defined and linked to the corresponding activities. The plug-in also automatically creates a dependency matrix to display and manage the relationships between functional groups and activities. The results of the PSC's functional matrix is shown in figure 5.10.



Legend
↗ Trace

Functional Groups	Get Power Supply for electronic PSC component	consume/transform electrical powe	distribute electrical power to PSC component	get electrical power from power distribution systen	Provide Audible Indications for Emergency Signalin	process audio signa	provide acoustic signal	receive audio signa	Provide Cabin Crew Announcements to PAX	Provide Call Flight Attendant for PAX	activate/deactivate call light	process passenger call/reset signa
Functional Groups	1	1	1	1	1	1	1	1	1	1	1	1
I/O AirConditioningSystem												
I/O CabinManagementSystem								↗				
I/O Oxygen System												
I/O Passenger							↗				↗	
I/O Power System				↗								
Data Acquisition												
Mechanical Actuation												
Air Control												
Oxygen Control												
Power Distribution/Transformatior		↗	↗									
Signal Processing						↗						↗
System	↗				↗				↗	↗		

Functional Groups	push call/reset interface	send/receive passenger call/reset signa	Provide Cockpit Crew Announcements to PAX	Provide Fasten Seatbelt Signs	activate/deactivate FSB sign	process FSB control signa	Provide No Smoking Allowed Sigr	activate/deactivate NSA sign	process NSA control signa	receive signal	provide oxygen masks for passengers	get oxygen from oxygen supply system
Functional Groups	1	1	1	1	1	1	1	1	1	1	1	1
I/O AirConditioningSystem												
I/O CabinManagementSystem		↗								↗		
I/O Oxygen System												↗
I/O Passenger	↗				↗			↗				
I/O Power System												
Data Acquisition												
Mechanical Actuation												
Air Control												
Oxygen Control												
Power Distribution/Transformatior												
Signal Processing						↗			↗			
System			↗	↗			↗				↗	

Functional Groups	open oxygen masks compartemen	outflow Oxygen to PAX	receive/ process cabin depressurization signa	regulate oxygen flow to	stream oxygen flow to masks	Provide Passenger Individual Air Conditior	control air flow	flow individual passenger air to cabir	get air from air conditioning systen	select individual air flow	Provide Reading Light for Passenger	process switch control signa
Functional Groups	1	1	1	1	1	1	1	1	1	1		1
I/O AirConditioningSystem									↗			
I/O CabinManagementSystem												
I/O Oxygen System			↗									
I/O Passenger		↗		↗				↗		↗		
I/O Power System												
Data Acquisition												
Mechanical Actuation	↗											
Air Control							↗					
Oxygen Control					↗							
Power Distribution/Transformatior												
Signal Processing												↗
System						↗						

Functional Groups	push switch on/off interface	send/receive switch control signa	turn on/off light	Provide recorded announcements	Provide Recorded Boarding Music	Provide Return to Seat Signs	activate/deactivate RTS sign	process RTS control signa	provide system condition communicatio	capture components condition data	communicate component conditio	process data
Functional Groups	1	1	1	1	1	1	1	1	1	1	1	1
I/O AirConditioningSystem												
I/O CabinManagementSystem		↗									↗	
I/O Oxygen System												
I/O Passenger	↗		↗				↗					
I/O Power System												
Data Acquisition										↗		
Mechanical Actuation												
Air Control												
Oxygen Control												
Power Distribution/Transformatior												
Signal Processing								↗				↗
System				↗	↗	↗			↗			

Figure 5.10: Dependency matrix between PSC's functional groups and corresponding activities

Following that, "functional blocks" are formed from each functional group, and "flow specifications" are assigned. To do this, the FAS plug-in finds all unconnected activity pins in the I/O partitions and assigns each one a flow property. Since the activities are linked to functional blocks (through the tracing mentioned above), the corresponding flow properties are automatically defined for the functional blocks' ports (as a flow specification).

The reason for this specification is because PSC's functional architecture is designed to be expressed in block form. To allow this depiction, a "functional system context" is also initiated. It encompasses both the "system" functional block (derived from the "system" function group) and the external actors, with both constituting its parts. Finally, by starting the creation of internal block diagrams, the functional architecture can be built. The plug-in generates part properties for the "system" functional block, each one corresponds to a function block. It also creates ports (depending on the selected SysML version: flow ports for v1.2 and proxy ports for v1.3) for each block that is specified by previously generated flow requirements. Figure 5.11 summarizes the repository representation of the FAS elements generated during the application of this method to provide a good understanding of the FAS elements.

The only manual task is connecting the ports to one another and to the external actors using SysML connectors. Figure 5.12 depicts the outcome for the PSC's block-oriented functional architecture.

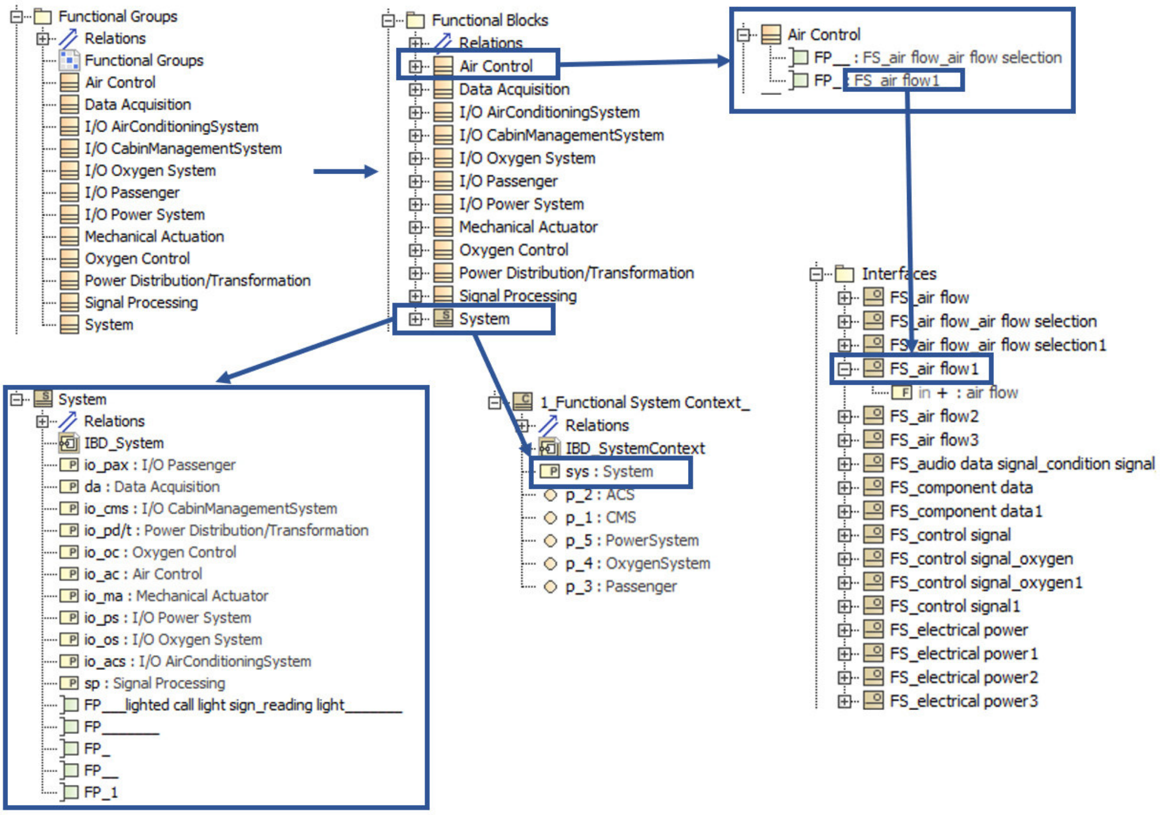


Figure 5.11: Repository representation of the FAS elements

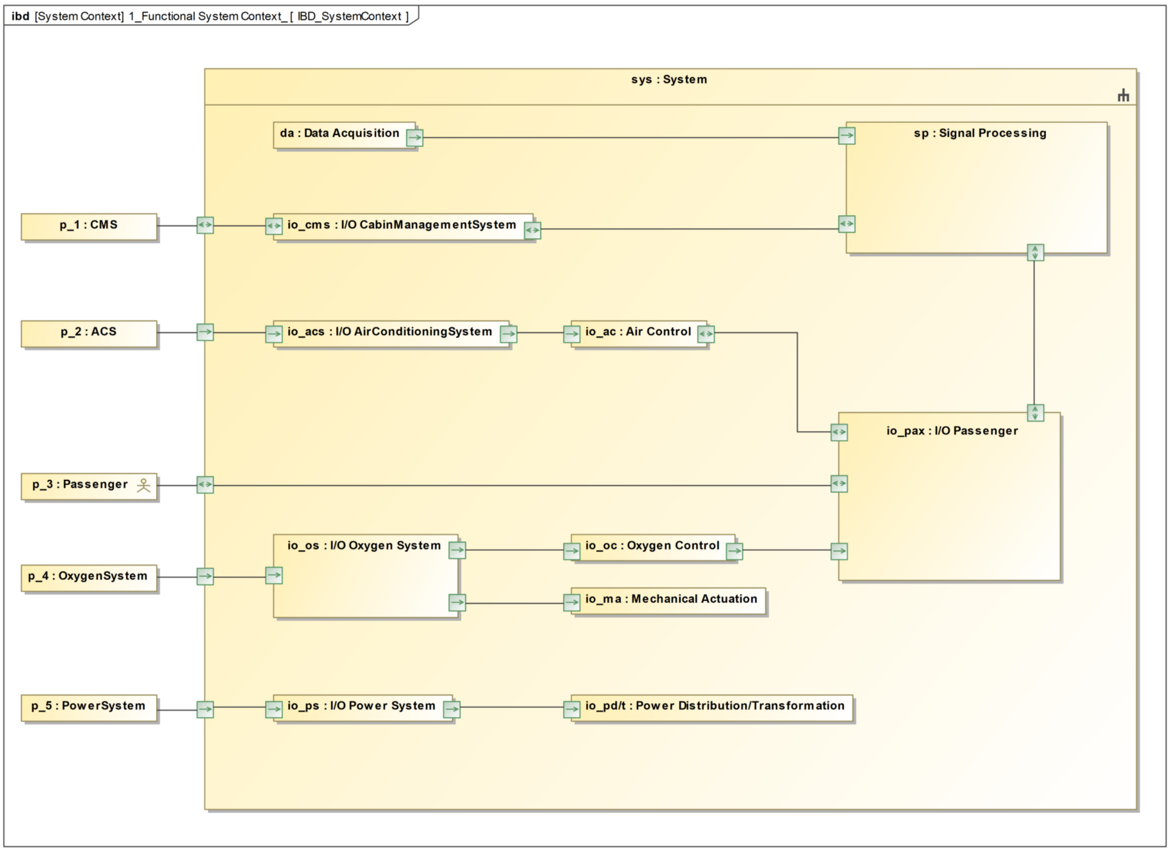

ibd [System Context] 1_Functional System Context_[IBD_SystemContext]
sys : System
da : Data Acquisition
sp : Signal Processing
p_1 : CMS
io_cms : I/O CabinManagementSystem
p_2 : ACS
io_acs : I/O AirConditioningSystem
io_ac : Air Control
io_pax : I/O Passenger
p_3 : Passenger
io_os : I/O Oxygen System
io_oc : Oxygen Control
p_4 : OxygenSystem
io_ma : Mechanical Actuation
p_5 : PowerSystem
io_ps : I/O Power System
io_pd/t : Power Distribution/Transformation

Figure 5.12: PSC's block-oriented functional architecture

The functional representation of the PSC does not consider any technical solution for implementing the functions. At this point, there is a complete separation of function and implementation. Making this the first "functional" solution to the requirements derived from the system analysis, as it already considers many architecting actions and decisions that have been made to create this solution. The next step involves pursuing system architecting actions in order to create a "logical" solution. The following section describes how the resulting logical architecture is specified and linked to the functional architecture that has already been obtained.

5.3 Logical Decomposition and Allocation

To meet many requirements identified during the system analysis, the functional architecture proposes an interaction of different functional blocks with corresponding activities. This is the first architectural task that has been completed in this work. To complement it, the system must now be logically decomposed into components capable of realizing these functions via logical interaction and specific behavior. The following sections present the main modeling activities that are carried out in order to create a logical architecture and enable the allocation of the functional system abstraction into the logical one.

5.3.1 Definition of Base Architecture

The structural context of the PSC is prepared before starting the modeling of the PSC's logical architecture. A bdd is used to depict the hierarchy of the PSC's context at an aircraft level and is shown in figure 5.13. An aircraft block is created and decomposed into an airframe, a power plant, systems, and a cabin. The external systems of interest are represented as stereotyped blocks with relevant properties using the generalization relationship. The remaining aircraft parts will not be modeled further because this work only expands on the systems that interact with the PSC. The depicted external systems are the same model elements that were used in the context analysis (see fig. 5.4), but in a different view (following the segregation between view and model repository, cf. section 2.3.2). The interacting systems are organized in separate hierarchical levels. For example, light or power systems are situated on A/C system level whereas CMS and IFE on A/C subsystem level. The specialization of some systems from the "CabinSystem" block creates a further distinction between cabin systems and other external systems. The property "Interact With" can be used to detect cabin systems that have an interface with the PSC (e.g. Water/Waste has no interface to the PSC). In addition, the cabin is made up of various cabin modules such as the PSC itself (in green) but also galleys, lavatories, seats, and an OHSCs. This representation allows for cabin's reconfigurability by allowing the addition or removal of modules to create new cabin layouts. Furthermore, the (*) multiplicity used has no effect on the number of these modules and doesn't set any constraints. Thus enabling modeling flexibility. Hence, a good platform is created as an important foundation for future logical architecture modeling activities.

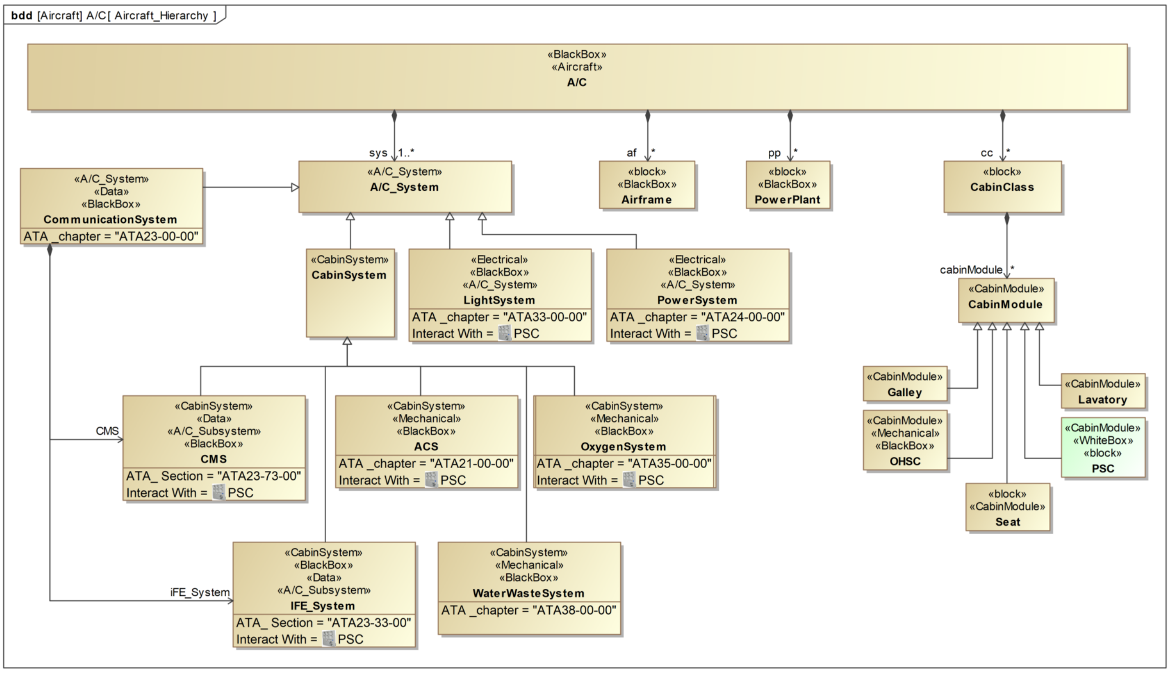


Figure 5.13: Aircraft hierarchical structure as a context of PSC's logical architecture

A base architecture is used as a starting point for the logical analysis, as previously described in the methodology section. The previously presented PSC and external cabin system architectures (see section 4) are used for this purpose. A hierarchical representation of the PSC structure is generated using a bdd to model it as shown in figure 5.14.

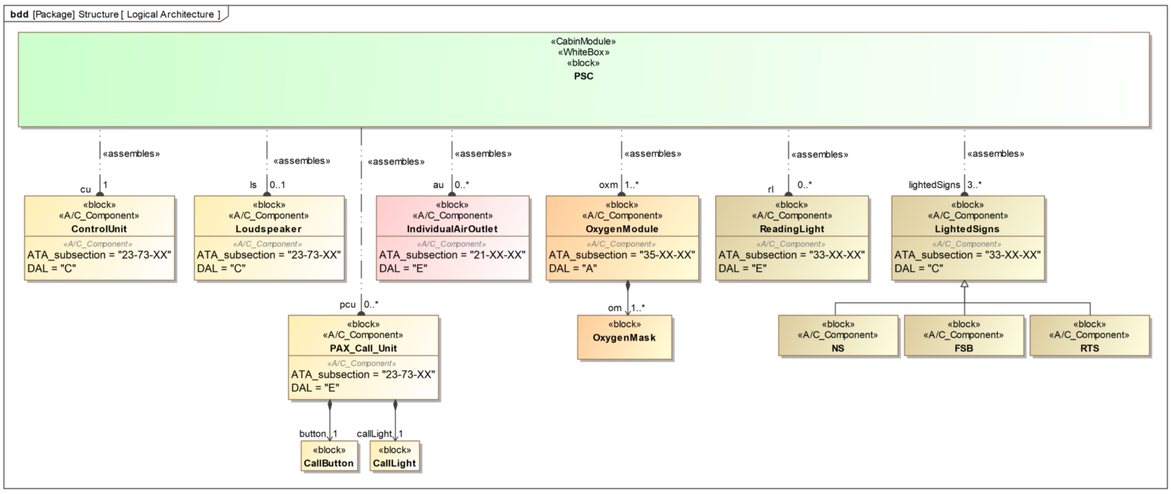


Figure 5.14: PSC's hierarchical logical structure

The PSC's pecularity as a cabin module is that it combines components from various aircraft systems. Each of these systems is represented by a different set of blocks, colored depending on the ATA subsection they correspond to. The relationship "assemble" created in the ontology specification is used to depict this special decomposition of the PSC. Components that occur in the majority of PSC configurations and variants (cf. section 4.1) are obviously represented and the significance of each component is defined by the multiplicity. On the one hand, components that are safety relevant and thus required to be existent in any PSC's configuration, such as the oxygen module and the lighted signs, are specified with a "1 to *" multiplicity. A digital control unit is also required for data communication with external systems such as the cabin management system or IFE. It is also required to provide electrical power to electronic components. The "PISA" component presented in section 4.2 is a specific airbus configuration of a control unit. On the other hand, components that can be configured based on the cabin layout, the position of the PSC (which cabin class) or the wishes of an airline are specified with a "0 to *" multiplicity. These are the individual air outlet, passenger call unit and reading lights.

The specification of each component's ATA-attribute (via the applied stereotype) enables a good understanding of the cabin module's complexity. Since the precise data regarding the affiliation of each component is unavailable, a "XX" has been used to specify the ATA-attribute. Thus, the specification demonstrates that the PSC not only has several interfaces to external systems, as stated in the context analyses, but also integrates many components from these systems as part of it.

Furthermore, determining the DAL level demonstrates the PSC module's heterogeneity. Normally, the DAL is determined via an elaborate fault analysis. However, because this analysis information is not available in this work, the specified values for the PSC component are estimated based on their criticality and are not taken from specification documents. The DAL has been defined for each component using the DAL attribute defined in the ontology. As shown in figure 5.14, the DAL of PSC components decreases in criticality from the greatest (Catastrophic DAL A) to the lowest (No safety effect DAL E). This is because the safety criticality of the many functions integrated into each component varies. For instance, the oxygen module on the one hand, has a direct impact on the passenger's chances of life in the event of a failure. On the other hand, components that provide passenger comfort, such as individual air outlets or the call passenger unit, have no bearing on passenger or aircraft safety in the event of a malfunction. The integration of both types of components into a single module demonstrates the PSC's complexity, and its heterogeneity must be considered throughout the technical design and integration stages.

The bdd representation does not show how these components interact with one another. An ibd is created to model the PSC inter-components flow to complement the logical architecture (see fig. 5.15). The ports that have already been created in the context analysis (PSC turnover in fig. 5.5) are shown as bordery ports in the diagram. The oxygen and air interfaces communicate with the oxygen module and individual air outlets. The control unit is connected to remaining electronic components. It distributes energy and transmits data signals to the other components (signs, loudspeakers, PAX call, and reading lights). The corresponding flow specifications are then assigned to the connectors that connect the components to each other.

Thus, the model-based logical base architecture references described in the theoretical chapter is set. The different logical components must be now traced back to the results of the

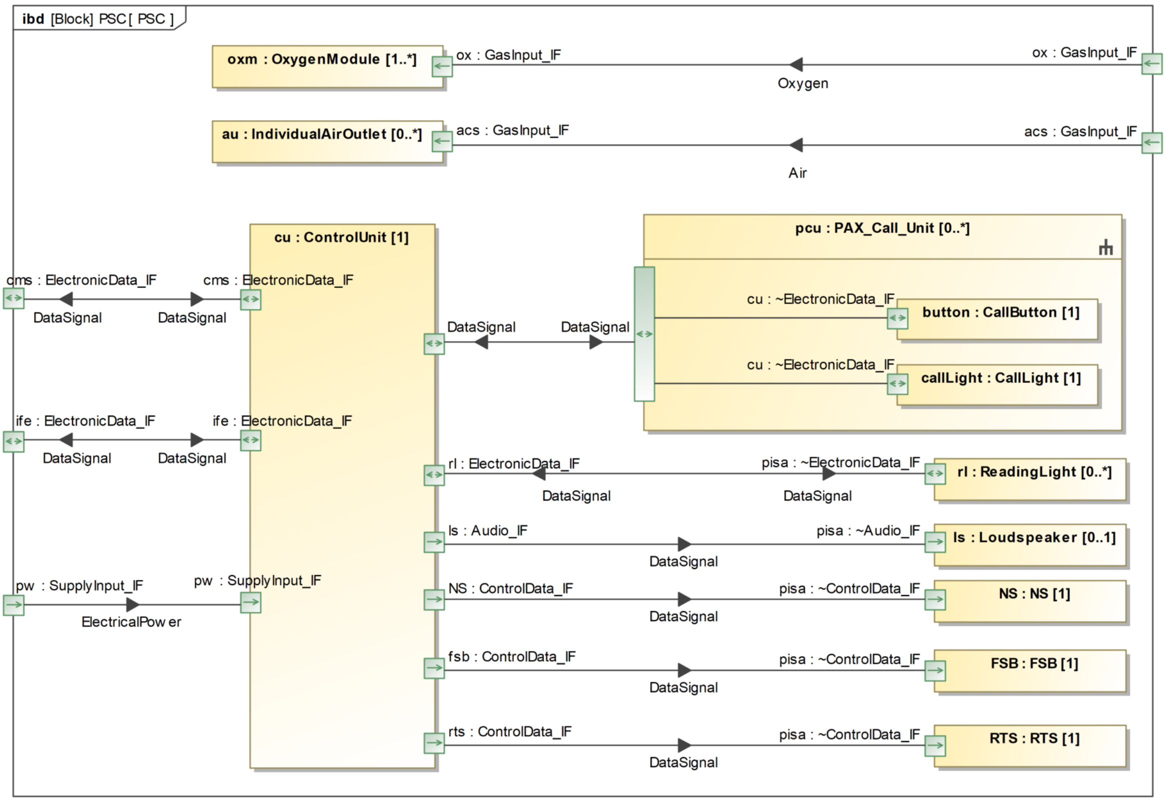


Figure 5.15: PSC's logical base architecture

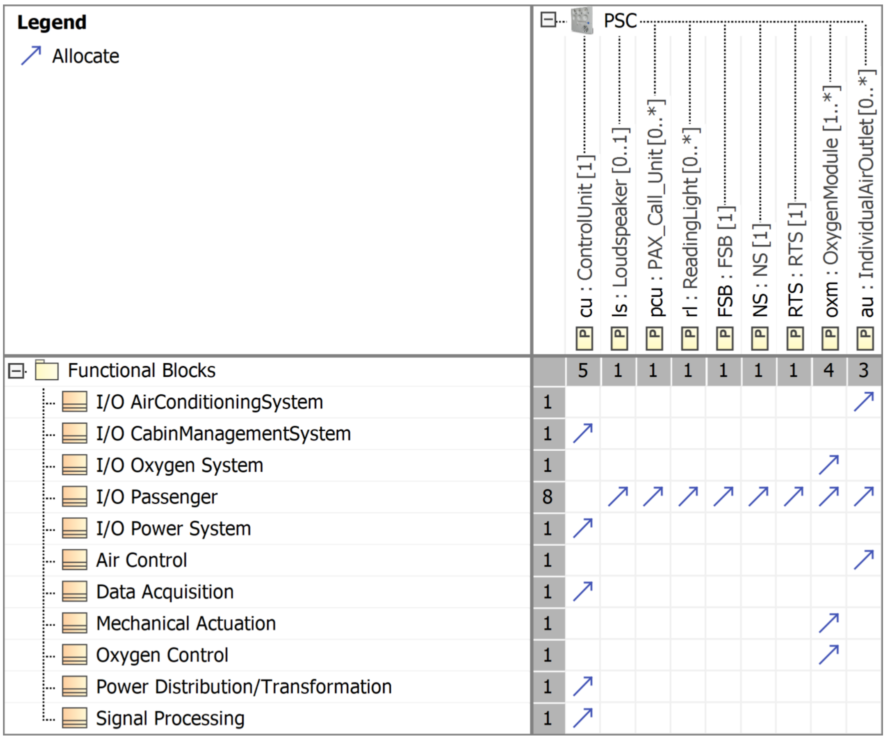
functional analysis. This means the functional architecture components must be allocated in the logical architecture.

5.3.2 Logical to Functional Allocation

The digital thread and traceability throughout the modeling of system abstraction levels is one of the major theoretical advantages of MBSE. At the same time, the developed methodology places a high value on the agility principle and, as a result, fully separates functional and logical system analysis and modeling. Thus, allocating functional architecture to the logical one is required to achieve both the goals of traceability and agility simultaneously. It enables defining the relationships between the two abstraction levels and ensuring that the functional and logical results fit and cohere.

Figure 5.16 depicts an allocation matrix that is used to create the allocation relationships. Each of the I/O functional blocks, as shown in the matrix, is assigned to the corresponding logical component that is linked to this external system. The functional block "I/O AirConditiongSystem", is, for example, assigned to the individual air outlet, "I/O OxygenSystem" to the oxygen module, and so on. The passenger is the only actor who has a functional interface with the PSC. He or she interacts with the majority of the logical components because they are mostly interface components (input of passenger command or output of component reaction). Only internal components, which in this case are represented by the control unit, do not interface with the passenger.

Other (non-I/O) functional blocks can also be intuitively classified based on the thematic correspondence between function and component. So, the "air control" is assigned to the individual air outlet, the "oxygen control" and "mechanical actuation" to the oxygen module, and the electrical and electronic functions, such as "signal processing", "data acquisition", and "power distribution/transformation", are assigned to the control unit. Hence, all functional blocks are assigned to logical components to carry out these functions.



Legend
↗ Allocate

		PSC								
		cu : ControlUnit [1]	ls : Loudspeaker [0..1]	pcu : PAX_Call_Unit [0..*]	rl : ReadingLight [0..*]	FSB : FSB [1]	NS : NS [1]	RTS : RTS [1]	oxm : OxygenModule [1..*]	au : IndividualAirOutlet [0..*]
Functional Blocks		5	1	1	1	1	1	1	4	3
I/O AirConditioningSystem	1									↗
I/O CabinManagementSystem	1	↗								
I/O Oxygen System	1								↗	
I/O Passenger	8		↗	↗	↗	↗	↗	↗	↗	↗
I/O Power System	1	↗								
Air Control	1									↗
Data Acquisition	1	↗								
Mechanical Actuation	1								↗	
Oxygen Control	1								↗	
Power Distribution/Transformation	1	↗								
Signal Processing	1	↗								

Figure 5.16: Allocation of functional block into logical components

Following that, once the allocation is determined, the functional and logical architectures can be compared in terms of this relationship. In an ideal case, the functional flow in the ibd connecting the functional blocks (see fig. 5.12) is the same as the functional flow between the logical components attributed to them (see fig. 5.15). This means that the logical implementation corresponds to the functional model. This case applies to the PSC's architectures developed in this work. If this is not the case, the logical components can be changed or modified to achieve a optimal fit. Furthermore, the activities associated with each functional block are now used as a starting point for the behavioral refinement of the assigned logical components. This is accomplished by developing new state machines, activity diagrams, and sequence diagrams to specify the detailed logical behavior of each component. This step exceeds the scope of this work and has therefore not been considered.

Finally, to get a sense of the main outcomes of the system architectures generated and presented in this chapter, the following section describes how these results are related to one another. All at the same, it illustrates which elements and tools are used to enable model-based traceability across system architecture.

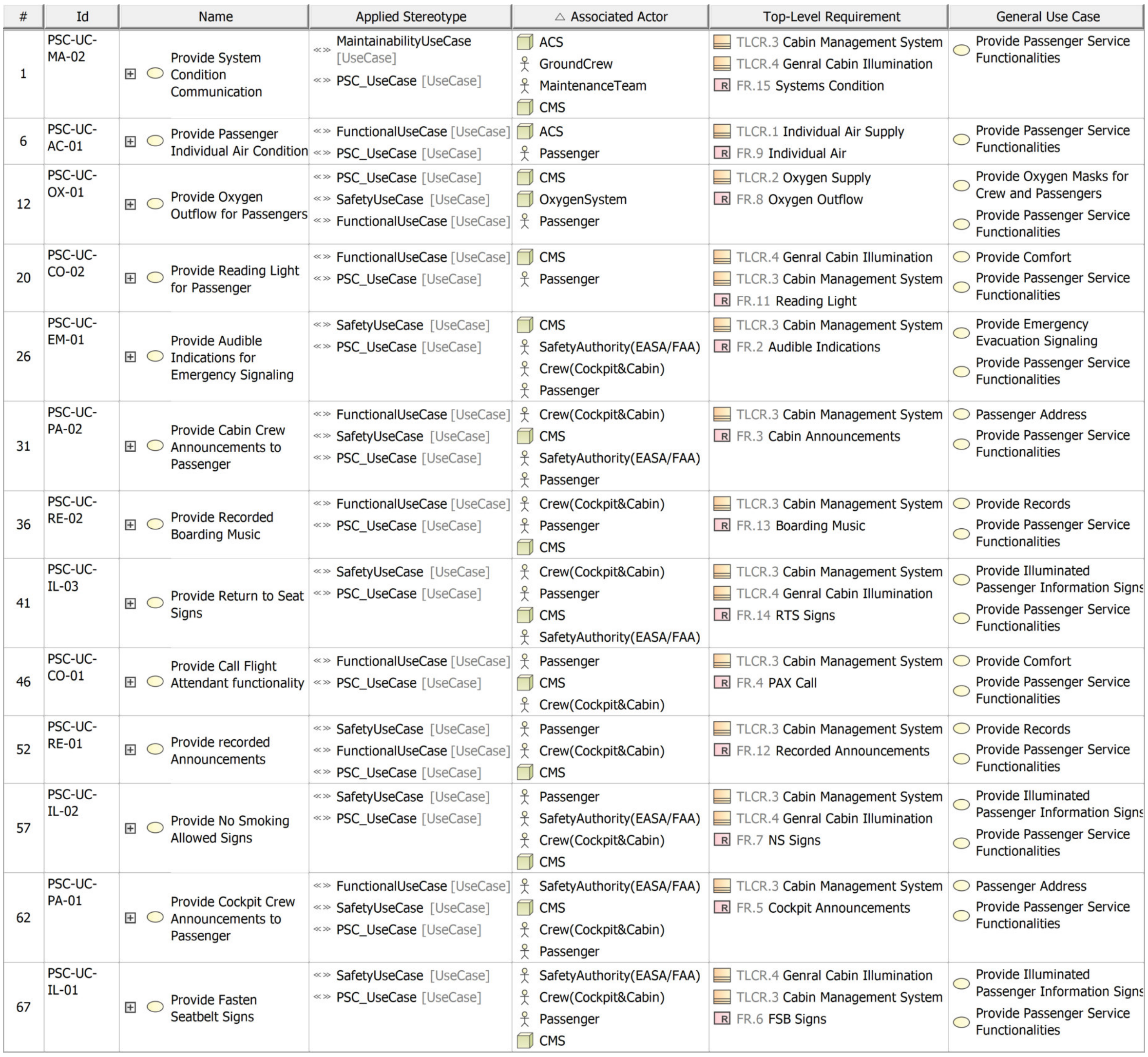
5.4 Towards System Traceability

Traceability begins during the modeling process. Many of the described modeling activities, such as creating "trace" relationships between functional groups and FAS-activities or allocating functional components into logical components, are the basis for system traceability. However, some additional modeling tasks are required to supplement these and connect all of the system elements in the model so that a model user can automatically interpret how the analysis results evolved.

This process starts with a stakeholder analysis, in which the defined "stakeholder requirements" are traced back to the relevant actors. Likewise, context requirements are traced back to context elements such as external, environmental, and boundary actors and systems. After that, a "derive relationship" should be established between these and the functional requirements defined based on the same foundation. The "refine" relationship is then used to link the use cases that model the behaviors of these functional requirements to them. The appendix shows the generated dependency matrices that were used to support the creation of the relationships (see fig. A.10, A.11 and A.12).

At this point, there is a clear connection between use cases, requirements, and stakeholders. The CSM supports the automated creation of generic tables to visualize these relationships. It will automatically detect and display all linked elements. Figure 5.17 depicts this for the PSC use cases. In this case, all model use cases are selected as an analysis target and filtered based on the stereotypes they employ. Then, only those with the "PCS UseCase" stereotype are displayed. Furthermore, the attributes "Id," "Associated Actor," "Top-level Requirement," and "General use Case" (which refers to the external system use case from which the PSC's was derived) are chosen to be showcased. The types of relationships are specified so that the tool can automatically find all of these related elements. If there is no direct link between the use case and the model elements, another alternative is to specify these as attributes of the PSC use case when creating it (see fig. A.6). Thus, the model user can trace these resulting use cases and gain a thorough understanding of the key factors that led to them.

Moreover, traceability activities have already been implemented as a result of the FAS plug-in automated modeling. Thus, functional blocks are traced to function groups, which in turn are traced to the activities that comprise the various use cases. A "thread" in the model is now set up through the already established allocation relationships, allowing a tracing from the logical architecture elements, through the functional architecture, and up to the context analysis and requirements. This is an important goal of this work, which is realized through the SysML tool's automation support, which enables overall system traceability. CSM provides additional tools for visualizing and navigating the model using a relation map diagram. Essentially, the modeler can choose the context that defines the starting model element and the relation criterion that should be detected, such as "derive," "refine," "trace," "assemble," or "allocate." The modeler also specifies the type of model elements (for example, blocks, part properties, activities, use cases, etc.) that will be displayed in the relation map. The tool generates a map automatically that can be customized based on the request.



#	Id	Name	Applied Stereotype	Associated Actor	Top-Level Requirement	General Use Case
1	PSC-UC-MA-02	Provide System Condition Communication	«» MaintainabilityUseCase [UseCase] «» PSC_UseCase [UseCase]	ACS GroundCrew MaintenanceTeam CMS	TLCR.3 Cabin Management System TLCR.4 Genral Cabin Illumination FR.15 Systems Condition	Provide Passenger Service Functionalities
6	PSC-UC-AC-01	Provide Passenger Individual Air Condition	«» FunctionalUseCase [UseCase] «» PSC_UseCase [UseCase]	ACS Passenger	TLCR.1 Individual Air Supply FR.9 Individual Air	Provide Passenger Service Functionalities
12	PSC-UC-OX-01	Provide Oxygen Outflow for Passengers	«» PSC_UseCase [UseCase] «» SafetyUseCase [UseCase] «» FunctionalUseCase [UseCase]	CMS OxygenSystem Passenger	TLCR.2 Oxygen Supply FR.8 Oxygen Outflow	Provide Oxygen Masks for Crew and Passengers Provide Passenger Service Functionalities
20	PSC-UC-CO-02	Provide Reading Light for Passenger	«» FunctionalUseCase [UseCase] «» PSC_UseCase [UseCase]	CMS Passenger	TLCR.4 Genral Cabin Illumination TLCR.3 Cabin Management System FR.11 Reading Light	Provide Comfort Provide Passenger Service Functionalities

26	PSC-UC-EM-01	Provide Audible Indications for Emergency Signaling	«» SafetyUseCase [UseCase]
			«» PSC_UseCase [UseCase]
				CMS
				SafetyAuthority(EASA/FAA)
				Crew(Cockpit&Cabin)
				Passenger
					TLCR.3 Cabin Management System
					FR.2 Audible Indications
						Provide Emergency Evacuation Signaling
						Provide Passenger Service Functionalities

31	PSC-UC-PA-02	Provide Cabin Crew Announcements to Passenger	«» FunctionalUseCase [UseCase]
			«» SafetyUseCase [UseCase]
			«» PSC_UseCase [UseCase]
				Crew(Cockpit&Cabin)
				CMS
				SafetyAuthority(EASA/FAA)
				Passenger
					TLCR.3 Cabin Management System
					FR.3 Cabin Announcements
						Passenger Address
						Provide Passenger Service Functionalities

36	PSC-UC-RE-02	Provide Recorded Boarding Music	«» FunctionalUseCase [UseCase] «» PSC_UseCase [UseCase]	Crew(Cockpit&Cabin) Passenger CMS	TLCR.3 Cabin Management System FR.13 Boarding Music	Provide Records Provide Passenger Service Functionalities

41	PSC-UC-IL-03	Provide Return to Seat Signs	«» SafetyUseCase [UseCase]
			«» PSC_UseCase [UseCase]
				Crew(Cockpit&Cabin)
				Passenger
				CMS
				SafetyAuthority(EASA/FAA)
					TLCR.3 Cabin Management System
					TLCR.4 Genral Cabin Illumination
					FR.14 RTS Signs
						Provide Illuminated Passenger Information Signs
						Provide Passenger Service Functionalities

46	PSC-UC-CO-01	Provide Call Flight Attendant functionality	«» FunctionalUseCase [UseCase] «» PSC_UseCase [UseCase]	Passenger CMS Crew(Cockpit&Cabin)	TLCR.3 Cabin Management System FR.4 PAX Call	Provide Comfort Provide Passenger Service Functionalities
52	PSC-UC-RE-01	Provide recorded Announcements	«» SafetyUseCase [UseCase] «» FunctionalUseCase [UseCase] «» PSC_UseCase [UseCase]	Passenger Crew(Cockpit&Cabin) CMS	TLCR.3 Cabin Management System FR.12 Recorded Announcements	Provide Records Provide Passenger Service Functionalities

57	PSC-UC-IL-02	Provide No Smoking Allowed Signs	«» SafetyUseCase [UseCase]
			«» PSC_UseCase [UseCase]
				Passenger
				SafetyAuthority(EASA/FAA)
				Crew(Cockpit&Cabin)
				CMS
					TLCR.3 Cabin Management System
					TLCR.4 Genral Cabin Illumination
					FR.7 NS Signs
						Provide Illuminated Passenger Information Signs
						Provide Passenger Service Functionalities

62	PSC-UC-PA-01	Provide Cockpit Crew Announcements to Passenger	«» FunctionalUseCase [UseCase]
			«» SafetyUseCase [UseCase]
			«» PSC_UseCase [UseCase]
				SafetyAuthority(EASA/FAA)
				CMS
				Crew(Cockpit&Cabin)
				Passenger
					TLCR.3 Cabin Management System
					FR.5 Cockpit Announcements
						Passenger Address
						Provide Passenger Service Functionalities

67	PSC-UC-IL-01	Provide Fasten Seatbelt Signs	«» SafetyUseCase [UseCase]
			«» PSC_UseCase [UseCase]
				SafetyAuthority(EASA/FAA)
				Crew(Cockpit&Cabin)
				Passenger
				CMS
					TLCR.4 Genral Cabin Illumination
					TLCR.3 Cabin Management System
					FR.6 FSB Signs
						Provide Illuminated Passenger Information Signs
						Provide Passenger Service Functionalities

Figure 5.17: Use case traceability using a generic table

Figure 5.18 shows the outcome of the PSC cabin module's generated relation map. Because of the numerous linked elements, only a part of these are depicted in the figure. The logical component "individual air outlet," which is a property of the PSC block, is traced up to the top level requirements as an example. The map uses different colors to distinguish between the various types of relationships. By clicking (+), all related elements that meet the predefined criteria are displayed. The criteria also specify the direction of the relationship. Only the existence of a relationship is considered relevant, when both directions are selected, such as the present case.

This completes the system architecture's creation and traceability. It was possible to model the various abstraction levels in a very compact and linked manner. The architecture must then be refined and extended in the following step. Model interfaces are processed and prepared for this purpose in order to integrate the architecture in the digitization process described in section 2.1. The following chapter describes how this is implemented and what benefits and refinements can be obtained through integration.

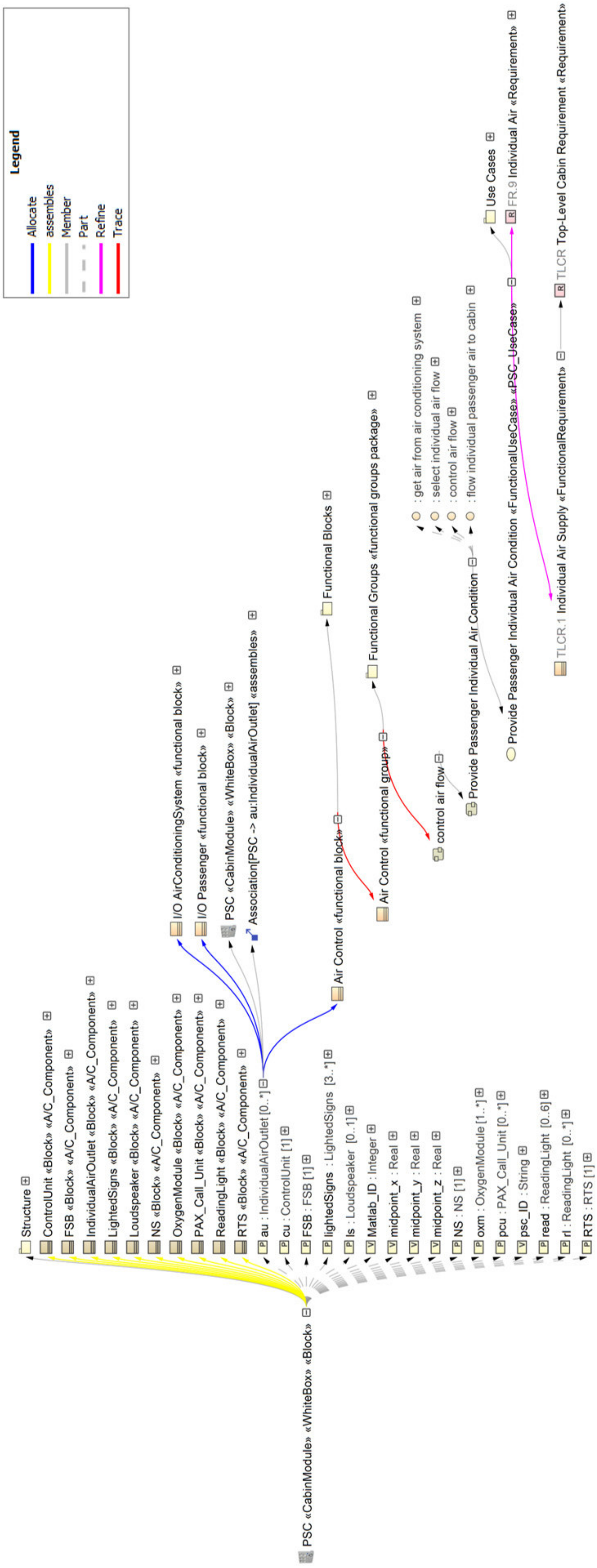

Legend
Allocate
assembles
Member
Part
Refine
Trace
PSC «CabinModule» «WhiteBox» «Block»
Structure
ControlUnit «Block» «A/C_Component»
FSB «Block» «A/C_Component»
IndividualAirOutlet «Block» «A/C_Component»
LightedSigns «Block» «A/C_Component»
Loudspeaker «Block» «A/C_Component»
NS «Block» «A/C_Component»
OxygenModule «Block» «A/C_Component»
PAX_Call_Unit «Block» «A/C_Component»
ReadingLight «Block» «A/C_Component»
RTS «Block» «A/C_Component»
au : IndividualAirOutlet [0..*]
cu : ControlUnit [1]
FSB : FSB [1]
lightedSigns : LightedSigns [3..*]
ls : Loudspeaker [0..1]
Matlab_ID : Integer
midpoint_x : Real
midpoint_y : Real
midpoint_z : Real
NS : NS [1]
oxm : OxygenModule [1..*]
pcu : PAX_Call_Unit [0..*]
psc_ID : String
read : ReadingLight [0..6]
rl : ReadingLight [0..*]
RTS : RTS [1]
I/O AirConditioningSystem «functional block»
I/O Passenger «functional block»
PSC «CabinModule» «WhiteBox» «Block»
Association[PSC -> au:IndividualAirOutlet] «assembles»
Air Control «functional block»
Functional Blocks
Air Control «functional group»
Functional Groups «functional groups package»
control air flow
Provide Passenger Individual Air Condition
: get air from air conditioning system
: select individual air flow
: control air flow
: flow individual passenger air to cabin
Provide Passenger Individual Air Condition «FunctionalUseCase» «PSC_UseCase»
Use Cases
FR.9 Individual Air «Requirement»
TLCR.1 Individual Air Supply «FunctionalRequirement»
TLCR Top-Level Cabin Requirement «Requirement»

Figure 5.18: Relation map diagram for system model traceability

6 Integration of PSC's Architecture in Cabin Design Process

The integration of the system architecture into the cabin design process is intended to achieve several goals on different levels. Obviously, the main goal is to connect this process to the resulting architecture described in Chapter 5. The architecture was methodologically derived from system analysis. Its elements are linked to each other and traced back to the requirements. The systems that are designed and installed in the aircraft must now be imported from the system architecture model. However, there are several other reasons for the integration. First, the use of CPACS design data as a knowledge exchange platform is critical for a consistent overall aircraft and system design. This interface should also allow for the reconfigurability of the evaluation of new cabin layouts. Furthermore, not only can data from the system architecture be used as an input in the design algorithm, but reversing it's flow also allows new analysis, design assessment, and optimization.

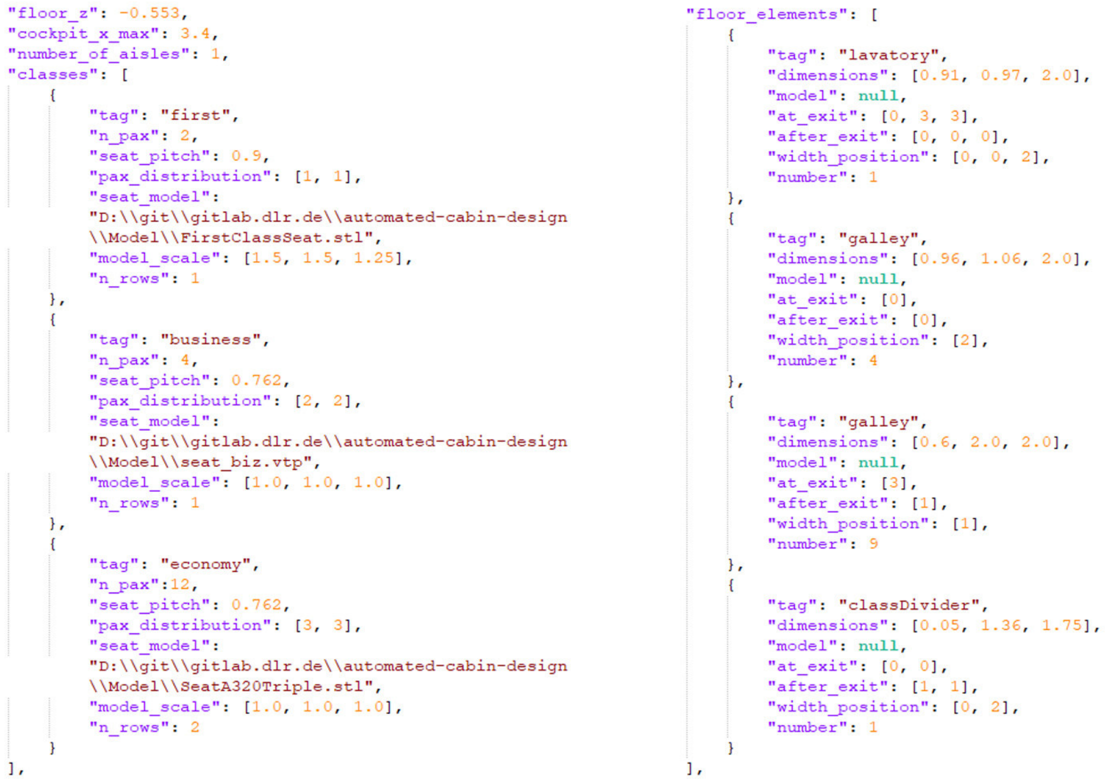
Therefore, the following sections will focus on the main integration aspects of the system architecture. The approach used to implement the interfaces and data exchange is described.

6.1 Design Data Import from CPACS

In CPACS, there is a large amount of design data available. Since only cabin data is required for the PSC system architecture, the relevant data is formatted to a simplified JavaScript Object Notation (JSON)[1] file. Figure 6.1 shows an extract of some cabin data.

This data contains two major parts. The first one on the left side depicts the cabin layout. It contains information about the various cabin classes (first, business, and economy) for a specific design configuration considered in this CPACS file. These information are directly related to the PSC architecture and provide new input that was not available through the system analysis. Thus, the aisle number, the number of seats in each class, and the passenger distribution ([1,1] in first, [2,2] in business, and [3,3] in economy) are identified. Because the PSC performs functions for each passenger, this data can be linked to the PSC distribution inside the cabin. Furthermore, the instantiation of the internal structure of each PSC can be based on row specifications that differ from one class to another.

[1] JSON is an open standard file format and data interchange format that stores and transmits data objects made up of attribute value pairs and arrays using human-readable text (or other serializable values). It is a common data format with a wide range of data interchange functionality, including communication between web applications and servers [108].



```
"floor_z": -0.553,
"cockpit_x_max": 3.4,
"number_of_aisles": 1,
"classes": [
    {
        "tag": "first",
        "n_pax": 2,
        "seat_pitch": 0.9,
        "pax_distribution": [1, 1],
        "seat_model":
        "D:\\git\\gitlab.dlr.de\\automated-cabin-design
        \\Model\\FirstClassSeat.stl",
        "model_scale": [1.5, 1.5, 1.25],
        "n_rows": 1
    },
    {
        "tag": "business",
        "n_pax": 4,
        "seat_pitch": 0.762,
        "pax_distribution": [2, 2],
        "seat_model":
        "D:\\git\\gitlab.dlr.de\\automated-cabin-design
        \\Model\\seat_biz.vtp",
        "model_scale": [1.0, 1.0, 1.0],
        "n_rows": 1
    },
    {
        "tag": "economy",
        "n_pax":12,
        "seat_pitch": 0.762,
        "pax_distribution": [3, 3],
        "seat_model":
        "D:\\git\\gitlab.dlr.de\\automated-cabin-design
        \\Model\\SeatA320Triple.stl",
        "model_scale": [1.0, 1.0, 1.0],
        "n_rows": 2
    }
],
```

```
"floor_elements": [
    {
        "tag": "lavatory",
        "dimensions": [0.91, 0.97, 2.0],
        "model": null,
        "at_exit": [0, 3, 3],
        "after_exit": [0, 0, 0],
        "width_position": [0, 0, 2],
        "number": 1
    },
    {
        "tag": "galley",
        "dimensions": [0.96, 1.06, 2.0],
        "model": null,
        "at_exit": [0],
        "after_exit": [0],
        "width_position": [2],
        "number": 4
    },
    {
        "tag": "galley",
        "dimensions": [0.6, 2.0, 2.0],
        "model": null,
        "at_exit": [3],
        "after_exit": [1],
        "width_position": [1],
        "number": 9
    },
    {
        "tag": "classDivider",
        "dimensions": [0.05, 1.36, 1.75],
        "model": null,
        "at_exit": [0, 0],
        "after_exit": [1, 1],
        "width_position": [0, 2],
        "number": 1
    }
],
```

Figure 6.1: Extract of cabin data from formatted JSON file

On the right side of the figure, there is information about the floor elements, which are the modules that are installed on the cabin floor. According to this data, there are two galleys, one lavatory, and a class divider in this CPACS cabin configuration. The main information pertains to the positioning and size of each of these elements. However, the "number" value indicates the element's type. For example, different galley types exist according to different standards (ATLAS[2], KSSU[3], ARINC[4], etc.) that specify the galley and its inserts (GAINs) such as trolleys or ovens. The "number" thus points the types out, which are defined in a different part of the same CPACS file. Initially, this information is not directly related to the PSC architecture. However, it can be used for an MDAO. That would consider both modules for a superordinate optimization use case. Section 7 contains more information on how this data is used and how the MDAO is applied.

In a further step, importing this data from the JSON file to the system model is required. The CSM tool provides numerous methods for importing external files in various formats. The chosen method allows the use of executable behavior and run-time objects in activities during the model execution. This enables an automated import, that can be easily restarted for new data files. To accomplish the latter, the execution specification must begin at the aircraft level. Therefore, an activity diagram "initializeData" is assigned to the "A/C" state machine in the "A/C" block (see fig. 6.2). It consists of three activities that allow for the import

[2] Acronym for Air France (**AF**), Air Portugal TAP (**TP**), Lufthansa (**LH**), Alitalia (**AZ**), Sabena (**SN**)

[3] Acronym for KLM (**KL**), SAS (**SK**), Swissair (**SR**), Union Transport Arienne (**UT**)

[4] ARINC Standards describe avionics, cabin systems, protocols, and interfaces used by more than 10,000 air transport and business aircraft worldwide [109].

of cabin layout information as well as the use of the imported data to initiate architecture objects. Only the import of layout data from a JSON file is covered in this section. Other steps are detailed in the following section.

Figure 6.2 shows that the first activity "importCabinLayout" includes an activity diagram with the same name. The goal of this activity is to access specific data values and specify them in the system model as new run-time object values. External CPACS access is accomplished through the use of a so-called *opaque behavior* known as "import CPACS". These are model elements that are implementation-specific and can be included in CSM activity diagrams. They use SysML to specify executable behaviors and support languages such as BeanShell, Groovy, JRuby, JavaScript, Jython, and StructuredExpression. All opaque behaviors are saved in the library under the package "Code" and can be used at various stages of the model (see fig. 6.2). The javascript language is used in this work for the opaque behaviors because its simple syntax is suitable for evaluating user interactions and for modifying, reloading, or generating content. This opaque behavior's programming code can be found in the list of code (see code A.1).

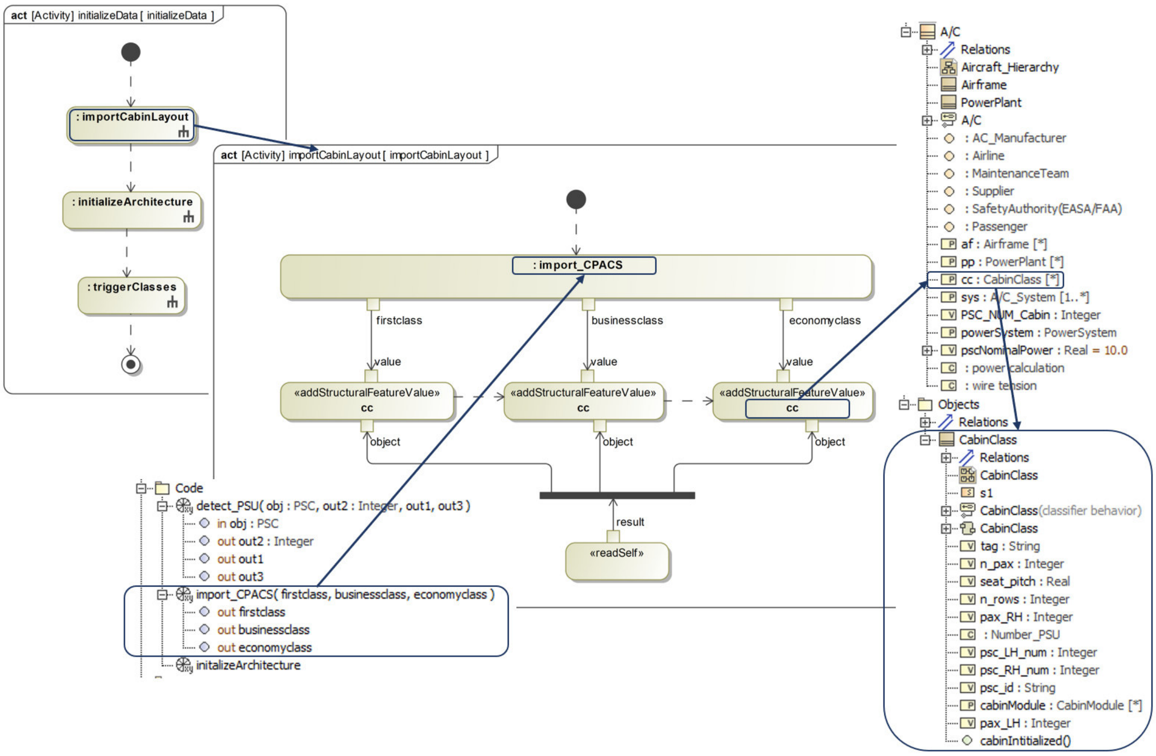


Figure 6.2: Specified activities and objects for the automated CPACS data import

The opaque behavior begins by specifying the JSON file name and location, followed by the creation of a new content and completing it with the JSON file data. Afterwards, some Java libraries are imported in order for a JSON parser[5] to be used during the behavior. This behavior starts by generating three run-time objects of the type "CabinClass," which is a model block (see fig. 6.2). This block represents each of the cabin classes (First Class

[5] A JSON Parser is used to format the JSON data into a properly readable JSON format, that can easily view and identify its key and value [110].

(FC),Business Class (**BC!**), Economy Class (EC)) and includes value properties that specify these classes' properties. "Tag" represents the class type, "n_pax" represents the number of passengers per class, "seat pitch" represents the distance between two rows, "n_rows" represents the number of rows, and "pax_RH" as well as "pax_LH" represent the passenger distribution on each aisle side. At this point, it is important to emphasize that the run-time objects can only be created, modified, or deleted when the simulation is still running. This means, for each new execution, completely new objects are initialized with corresponding data from the latest file versions. Therefore, the JSON parser extracts this information from the specified location in the file. These values are then assigned to the three newly created run-time objects as value properties. To accomplish this, CSM provides the "Action Language Helper (ALH)" API, which includes numerous commands for manipulating block properties, such as getting and specifying a structural feature value. No Magic provides an overview as well as applications for these ALH commands [111].

The three run-time objects are the output of the "importCPACS" opaque behavior. These are used to specify new properties of the A/C block. To implement this, the "addStructuralFeatureValue" action is used. It accesses the local block "A/C", creates new "cc" part properties and specifies them with the run times objects. Thus, the aircraft cabin classes are now initialized and specified with the layout configuration data from CPACS. The PSCs in each of these cabin classes must then be created as the next step. Their corresponding internal architecture must be initialized and passed to Matlab, based on the imported CPACS data. The next section explains how the interface between the system and design models is realized in order to attain these goals.

6.2 Data Exchange with the Cabin Design Algorithm

The interface between the system and the design models is implemented in three steps. First, the PSCs are distributed in each cabin class created in the previous step, creating the internal architecture of each PSC. Second, the internal architecture is transferred to the design model in Matlab. Third, Matlab is executed, and the design results are displayed. Each of these steps require an interaction between Matlab and CSM. The following sections explain how these steps are carried out.

6.2.1 Initializing the PSC's Distribution in the Cabin Classes

Following the initiation and specification of the cabin classes with CPACS data within the activity "importCabinLayout," each of these classes is now inducted with a distribution of the PSC's service functions. This occurs within the activity "initializeArchitecture," which consists solely of an opaque behavior of the same name. Its coding content is shown in the code list (see code A.2). The primary goal of this behavior is to analyze the cabin class specifications and create PSC objects in each class as a result. To do so, a parametric diagram must first be evaluated (see fig. 6.3). The constraint specified there allows adherence to the requirement that each seat row has one a set of PSC's service functions on each aisle side. So, the value properties "psc_LH_num" and "psc_RH_num" of the cabin classes are set to the number of rows. Furthermore, this parametric diagram begins the identification of the PSC based on the cabin class tag (PSC-FC if it is the first class). CPACS is responsible for both the rows and tag properties in each cabin class object.

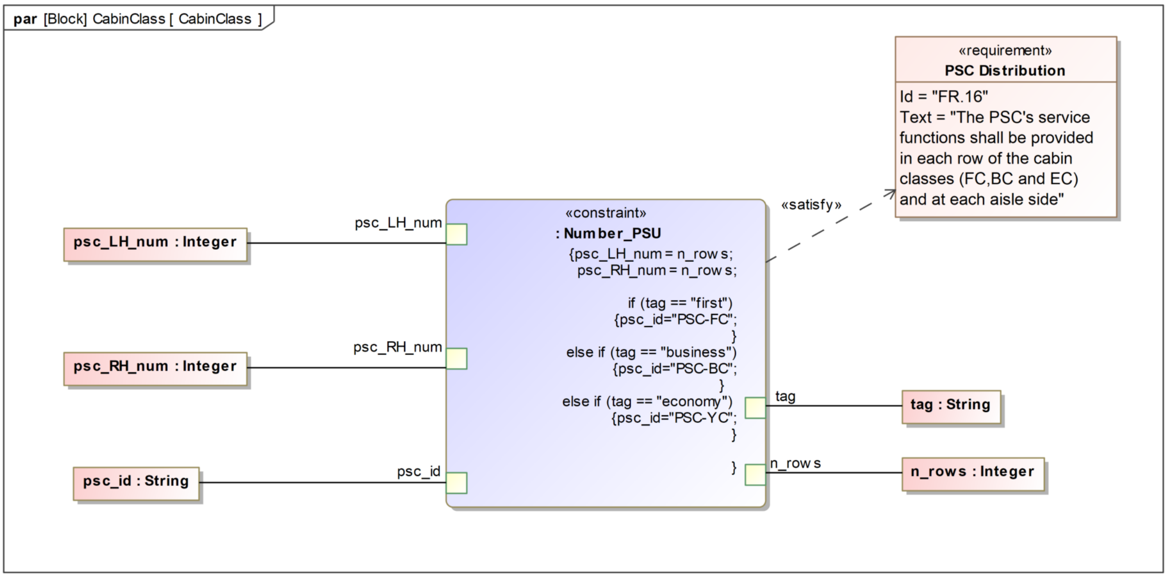


Figure 6.3: Parametric diagram to constraint the PSCs based on a distribution requirement

Based on this, the opaque behavior knows how many PSC service function objects to create and how to identify them. It also accesses the number of passengers seated in each row using ALH commands, depending on the cabin class. The algorithm in the opaque behavior defines the internal architecture of the PSC based on this information. As a result, the number of reading lights, individual air outlets, and oxygen masks is linked to the number of passengers. Other logical components, such as the control unit, signs, and loudspeakers, which are specified by their unalterable multiplicity, are created automatically as part of the architecture. Finally, the resulting PSC object is added as a part property called "cabinModule" to each class.

The PSC distribution must then be initialized in the design model. The reason for implementing this step at this stage, is that information about the distribution of PSCs is available at the cabin class level (in the "CabinClass" block) rather than at the PSC level. First, the so-called *classifier behavior* of each of the CabinClass objects (specified as part of the A/C block's properties) must be triggered. A classifier behavior is a block property that specifies which behavior (state machine, activity, sequence diagram, or opaque behavior) is the defining one and will thus be executed first once this block exists. The CabinClass object's classifier behavior is the state machine "CabinClass" (see fig. 6.4). It first remains in the "idle" state until all PSC objects of each class have been created at the A/C level (in the opaque behavior "initializeArchitecture"). The final activity in the A/C block is "triggerClasses" (see fig. 6.2). The content of this activity is depicted in the activity diagram on the left side of figure 6.4. There, it reads each of the existing clabin classes and triggers them using the *Call Operation Action* called "cabinInitialized." This is a SysML element that causes the state machine to transition from "idle" to "initializeMatlab." The use of such *Call Operation Actions* is a simple way to enable communication between different blocks during execution.

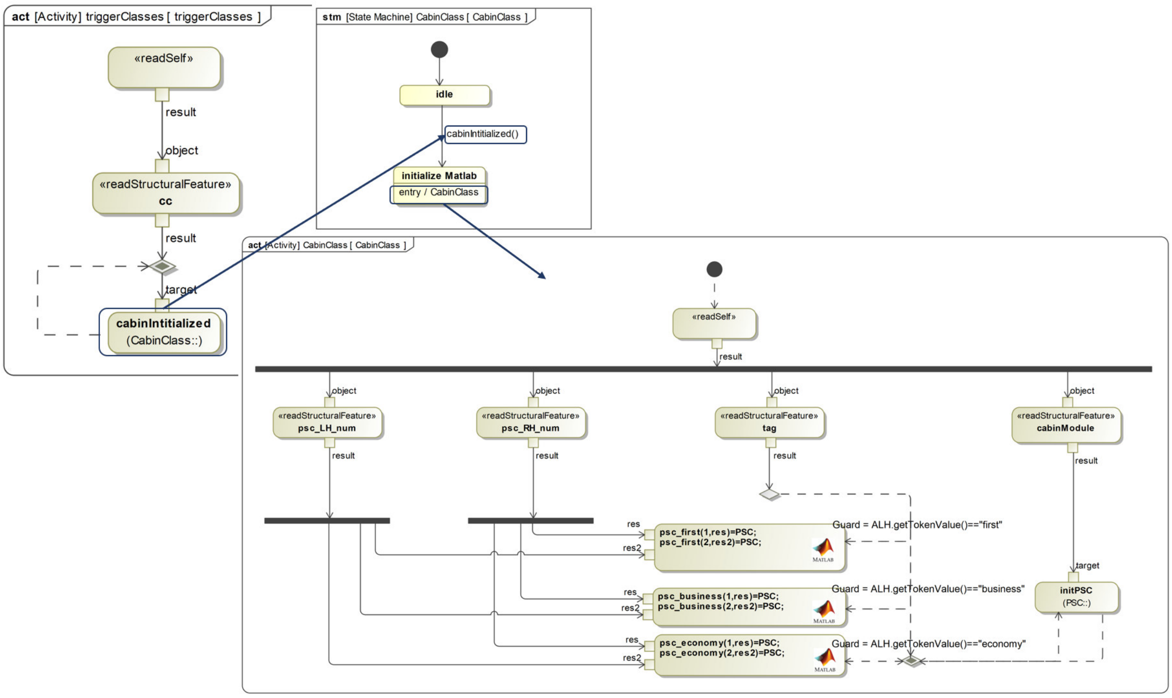


Figure 6.4: Behavior of "CabinClass" to initialize the PSC objects creation in Matlab

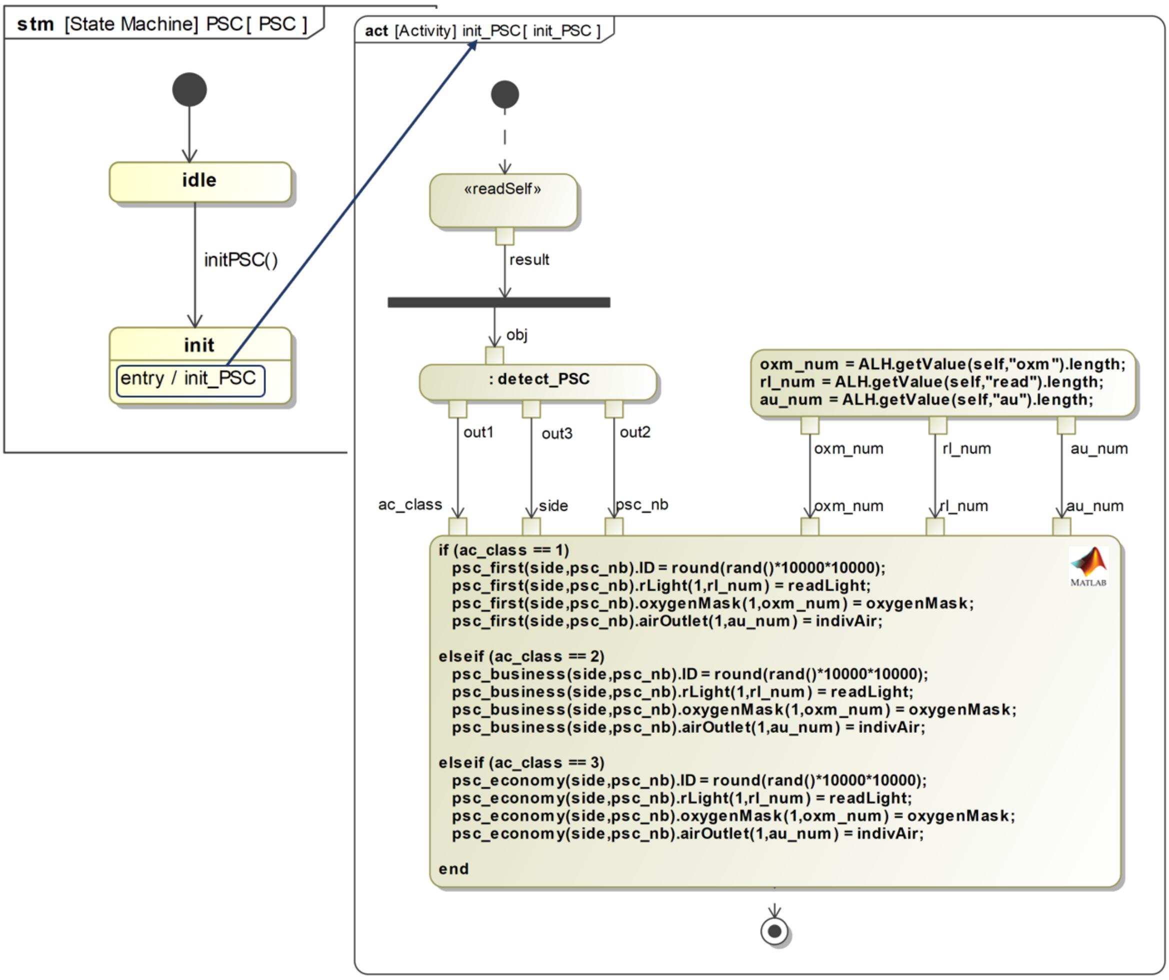
The activity "CabinClass" shown in figure 6.4 is specified as an *entry*-behavior and is automatically triggered when the state "initialize Matlab" is active. Its goal is to read the cabin class's properties ("psc_LH_num", "psc_RH_num", and "tag") and generate PSC objects in Matlab. The CSM tool provides an integration interface to Matlab, allowing the modeler to use Matlab to evaluate expressions written in Matlab syntax in Cameo Simulation Toolkit. No Magic's manual contains detailed documentation on the Matlab integration in CSM [112].

Once the integration is established, the Matlab engine must be shared by the two tools. Therefore, the command "matlab.engine.shareEngine" is used. Thus, CSM has access to and can edit all variables in the Matlab workspace. In the figure 6.4, the three "opaque actions" (with a Matlab logo) are used for this purpose. In this element, the specification language "Matlab" can be chosen, and the Matlab syntax can be used. Based on the information imported as described above, these actions generate three arrays of a specified number of objects in the Matlab workspace (depending on the configuration of each class). These arrays are referred to as "psc_first," "psc_business," and "psc_economy." The opaque action also specifies the type "PSC" of the objects created, which is defined in a Matlab object script (see code A.3).

Hence, the design model incorporates the architecture of the classes in terms of their PSC composition. It is important to note at this point that this work only considers the PSC (since it is the selected SoI). This integration, however, can be applied to all other cabin modules in the same way (seats, galleys, lavatories etc.) to depict a realistic cabin architecture. The integration then proceeds to the PSC level, where the internal structure of each PSC is transferred to the Design Model in Matlab. This is explained in more detail in the next section.

6.2.2 Transferring the PSC's internal Architecture to the Design Model

The final integration level, the PSC level, is essential for the transfer of each PSC's internal architecture to the design model. Figure 6.5 depicts the state machine that represents the PSC classifier behavior. This behavior is executed for each PSC and can access the information stored in it. The transition to the state "init" is triggered by the *Call Operation Action* "cabinInitialized" (in the activity in fig. 6.4). Thus, the activity "initPSC" can be triggered to initiate the internal structure in Matlab. Each PSC was specified with specific data when it was created (see section 6.2.1). This data is first read using the "readSelf" action and contains information about the identification and logical components of each PSC. Using the opaque behavior "detect_PSC," (see code A.4) the class, aisle side, and PSC number are detected from the ID (e.g. PSC-EC-13). This information is used in an opaque action (in Matlab syntax) to specify the internal structure of each specific PSC in Matlab.



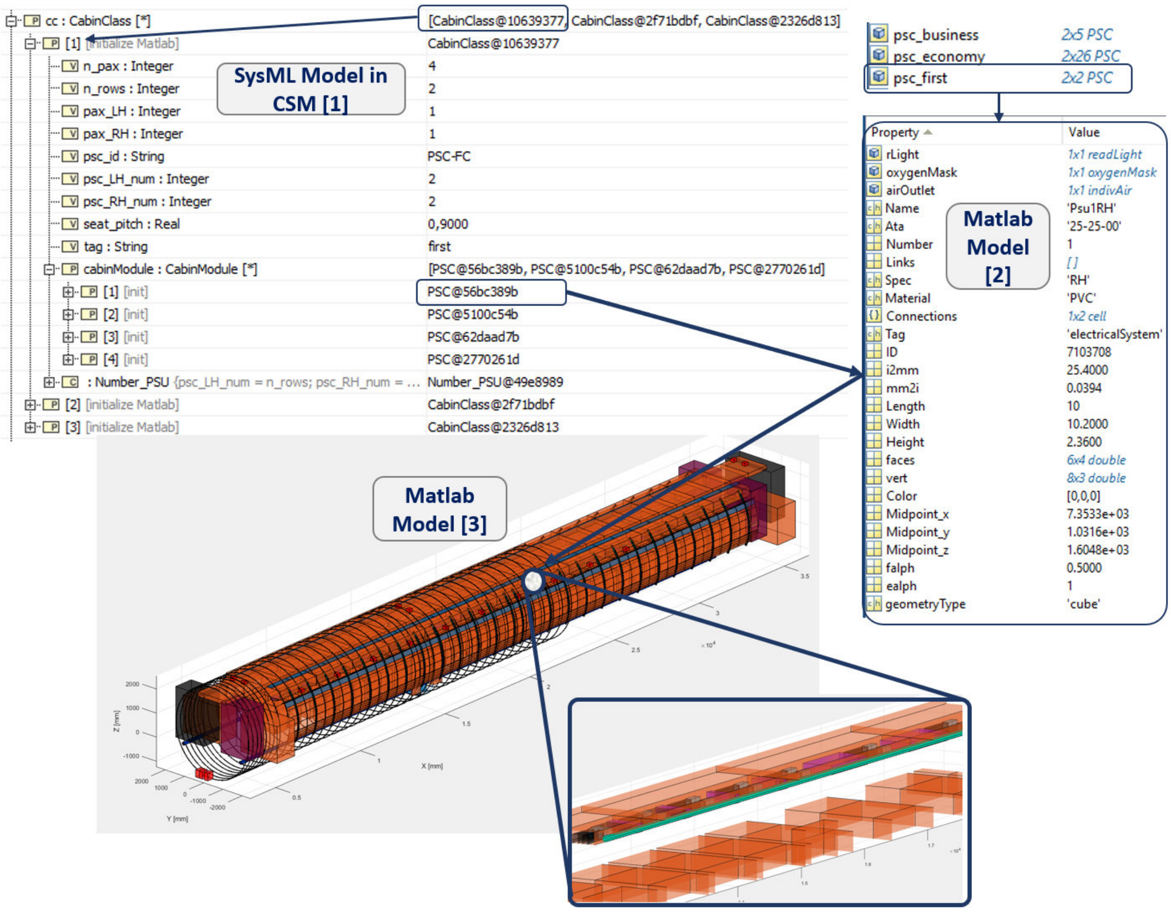
```
oxm_num = ALH.getValue(self,"oxm").length;
rl_num = ALH.getValue(self,"read").length;
au_num = ALH.getValue(self,"au").length;
```

```
if (ac_class == 1)
  psc_first(side,psc_nb).ID = round(rand()*10000*10000);
  psc_first(side,psc_nb).rLight(1,rl_num) = readLight;
  psc_first(side,psc_nb).oxygenMask(1,oxm_num) = oxygenMask;
  psc_first(side,psc_nb).airOutlet(1,au_num) = indivAir;

elseif (ac_class == 2)
  psc_business(side,psc_nb).ID = round(rand()*10000*10000);
  psc_business(side,psc_nb).rLight(1,rl_num) = readLight;
  psc_business(side,psc_nb).oxygenMask(1,oxm_num) = oxygenMask;
  psc_business(side,psc_nb).airOutlet(1,au_num) = indivAir;

elseif (ac_class == 3)
  psc_economy(side,psc_nb).ID = round(rand()*10000*10000);
  psc_economy(side,psc_nb).rLight(1,rl_num) = readLight;
  psc_economy(side,psc_nb).oxygenMask(1,oxm_num) = oxygenMask;
  psc_economy(side,psc_nb).airOutlet(1,au_num) = indivAir;

end
```

Figure 6.5: Initialization of the PSC's internal Architecture in Matlab

After executing the system model in CSM and transferring data to the matlab model, the latter contains all relevant architecture information. These represent the input for the design algorithm. The design execution provides results for the entire cabin. Only the PSC architecture is imported from the system model in this case, whereas other systems and modules are specified as algorithm inputs. Their properties (e.g. Size of galleys and OHSCs) are entered manually in Matlab.

The data exchange between the two models is summarized in figure 6.6. It displays the SysML model's instantiated PSC objects in the CSM's output window (1). The objects can be geometrically placed in the cabin using Matlab's predefined rules and algorithms. The additional property parameters can be stored as attributes in the object-oriented model (2). The graphical geometric representation of the corresponding objects in the cabin is used to visualize the placement result in Matlab (3).



cc : CabinClass [*]	[CabinClass@10639377, CabinClass@2f71bdbf, CabinClass@2326d813]
[1] [initialize Matlab]	CabinClass@10639377
n_pax : Integer	4
n_rows : Integer	2
pax_LH : Integer	1
pax_RH : Integer	1
psc_id : String	PSC-FC
psc_LH_num : Integer	2
psc_RH_num : Integer	2
seat_pitch : Real	0,9000
tag : String	first
cabinModule : CabinModule [*]	[PSC@56bc389b, PSC@5100c54b, PSC@62daad7b, PSC@2770261d]
[1] [init]	PSC@56bc389b
[2] [init]	PSC@5100c54b
[3] [init]	PSC@62daad7b
[4] [init]	PSC@2770261d
: Number_PSU {psc_LH_num = n_rows; psc_RH_num = …	Number_PSU@49e8989
[2] [initialize Matlab]	CabinClass@2f71bdbf
[3] [initialize Matlab]	CabinClass@2326d813

psc_business	2x5 PSC
psc_economy	2x26 PSC
psc_first	2x2 PSC

Property	Value
rLight	1x1 readLight
oxygenMask	1x1 oxygenMask
airOutlet	1x1 indivAir
Name	'Psu1RH'
Ata	'25-25-00'
Number	1
Links	[]
Spec	'RH'
Material	'PVC'
Connections	1x2 cell
Tag	'electricalSystem'
ID	7103708
i2mm	25.4000
mm2i	0.0394
Length	10
Width	10.2000
Height	2.3600
faces	6x4 double
vert	8x3 double
Color	[0,0,0]
Midpoint_x	7.3533e+03
Midpoint_y	1.0316e+03
Midpoint_z	1.6048e+03
falph	0.5000
ealph	1
geometryType	'cube'

Figure 6.6: Summary of the data exchange between the system and design model and graphical representation of the results

Once the PSC's system architecture has been fully integrated into the cabin design process, the CPACS and Matlab interfaces are implemented, allowing for the necessary data exchange with the system model in CSM. However, in order to evaluate this integration, the interaction must be extended. The goal is to demonstrate how data exchange can provide relevant and measurable benefits for system analysis and development. Furthermore, it should be possible to implement and execute an MDO at this stage. The following chapter explains how these goals are accomplished.

7 Trade-off Study: PSC's Architecture Assessment and Optimization for Electrical Design

Although the primary goals for modeling and integrating the system architecture have been met, they must still be completed in order to concretize the integration's significance. The true benefits of modeling and integration can only be quantified with an assessment use case, that demonstrates how model-based integration can provide a significant analytical contribution, that would not be possible with single models.

Upon examining the models in terms of their disciplinary affiliation in the context of aircraft design, three main domains come into sight. The first one being the aircraft preliminary design domain. The cabin layout parameters extracted from CPACS, which is used as a knowledge exchange platform in this work, result from this design domain. The optimization model (in Matlab), when combined with the 3D-Models in Blender and the visualization models in VR, represent the geometrical cabin design and visualization as a part of traditional mechanical design, defining the second domain. Finally, the system architecture model created in SysML represents the domain of system design and analysis.

So, the following step is to find a use case, which combines data from all three domains into an integrative analysis. This assessment can be supplemented with an optimization case, in which parameters from each discipline model are varied to optimize overriding design goals. The following section presents the suggested assessment and optimization use case as well as the physical fundamentals required for analysis and optimization.

7.1 Definition of Optimization Use Case

When it comes to trade-off analyses, the integration and installation of cabin modules in the aircraft cabin is a very important design aspect. The integration affects not only the component itself, but also its external systems and their physical properties as well as the aircraft structure. Therefore, affecting the entire aircraft. These integration effects are significant because they can have a direct impact on various types of mass and costs. Thus, many integration parameters must be considered and assessed in order to obtain the best solution at the aircraft level.

A use case involving the PCS's interface with the power system is developed to elucidate and exemplify these ideas. The consideration of this interface enables the investigation of the PSC's multiphysicality as well as the interaction between the three domains with the corresponding models as explained above. So, in this use case, the goal is to identify the parameters from different domain models that impact one another.

The cable material and cross section are considered configuration parameters and are used to instantiate the logical architecture. The cable length is a property that is specified based on the PSC's design results in Matlab. Thus, the geometrical design outcome is used as an input for the electrical power calculation in expression 4.8 (cf. 4.3).

This equation expresses the main idea and goal of the given use case. The electrical aspect, represented by the power to be supplied from the SPDB, is brought into relationship with the geometrical design aspect, represented by the cable length, which varies depending on the PSC's location in the cabin. Given that the PSC's architecture imported from the system model was generated in accordance with the CPACS layout data, the relationship in 4.8 provides a clear indication of the interaction between the three domains (represented by each of the models).

On the one hand, the power required from each SPDB is directly related to the fuel consumption and thus to the aircraft operating costs. So the goal is to minimize the power consumption on the loads side. On the other hand, longer cable means more cable mass. Meaning, the integration of additional cables into the aircraft, consume more space, assembly effort, and maintenance. This leads to high production and operation costs.

The influence of the interfaces between the SPDBs and the PSCs on decreasing both the power required and the cable length, is examined in this use case. The purpose is to define an input variable that describes this interface and evaluates its influence on the mentioned parameters. This use case represents a trade-off analysis, in which two objectives, cable length and power distribution, are affected by a single input parameter. Therefore, multi-objective optimization[1] is used to solve this issue.

The power distribution between the different SPDBs in the aircraft must be uniform in order to minimize power. This means that no SPDB must supply more power than the other, so that the power supply's design point can be reduced. The standard deviation is used as a function to represent this objective mathematically, and it is defined as follows:

$$y_1 = \sqrt{\frac{1}{k-1} \cdot \sum_{i=1}^{k} (P_{SPDB\ i} - \bar{P}_{SPDB\ i})^2}, \tag{7.1}$$

with k representing the number of SPDBs installed in the aircraft.

[1] Multi-objective optimization is a branch of multiple criterion decision making. It deals with mathematical optimization issues, involving more than one objective function that must be optimized at the same time [113].

$\bar{P}_{SPDB}$ represents the arithmetic mean value of the SPDB power and is calculated as follows:

$$\bar{P}_{SPDB} = \frac{\sum_{i=1}^{k} P_{SPDB\ i}}{k}. \tag{7.2}$$

This first objective function considers the power supply of each SPDB and the mean value for all SPDBs. Since only the power supply is used as a parameter and all SPDBs are treated equally, the function does not include any weighting or prioritization. The standard deviation was chosen as an objective function for the power distribution because its a measure for the variation and dispersion of the power values between the installed SPDB. If the required power values for all SPDBs are close to one another, the standard deviation's value is small. Meaning, the objective function minima are sought in order to obtain the optimal power distribution.

The cable length parameter is defined by the second objective function that considers all cables connecting the PSCs to the SPDBs. The following equation is the corresponding objective function:

$$y_2 = \sum_{i=1}^{k} l_{cable\ i}. \tag{7.3}$$

This objective function considers only the cable length as a parameter. As with the first objective function, ther is no weighting or prioritization included because all cables connecting the PSCs to their corresponding SPDBs are treated equally. The sum of cable lengths was chosen as an objective function because of the direct relationship between the cable length and weight as well as integration complexity. This simplifies the objective function definition by considering exclusively the cable length parameter.

The investigation of the influence of the SPDB-PSC interfaces is mathematically expressed as an input variable for the optimization case. Therefore, the model depicted in figure 7.1 is considered. The cabin is separated into many sections. Due to the symmetric SPDB distribution, the number of these zones corresponds to half of the total number of SPDBs installed in the aircraft. The concept behind this division is that each PSC in this area is

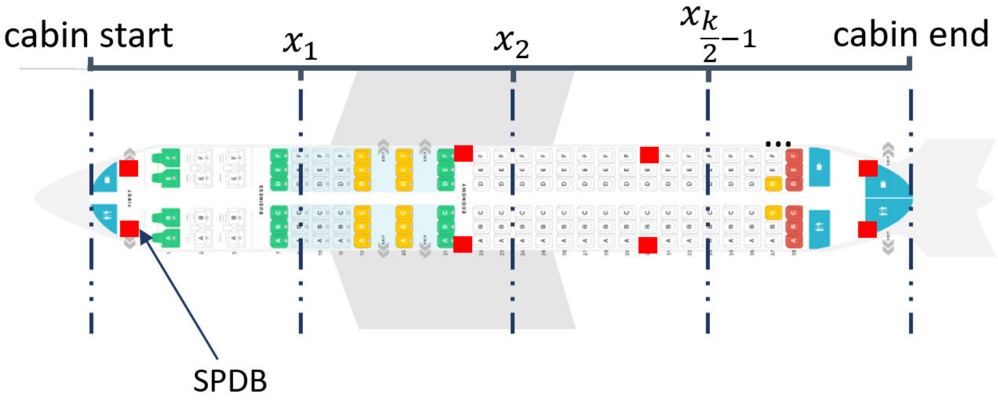


Figure 7.1: Definition of input variable X of optimization case (Cabin layout modified from [114])

linked to the SPDB in that same area. The coordinates of the start of each new area are specified as $x_1, x_2, ..., x_{\frac{k}{2}-1}$. These are the variables that allow the adjustment of the width of each section. When the area changes, the SPDB interfaces to the PSCs change since the PSCs are now in a different area. Analysis of all conceivable variants of these locations enables the consideration of all interface possibilities. As a result, the SPDB-PSC interface is represented by the input variable, which is stated as follows:

$$X = (x_1, x_2, ..., x_{\frac{k}{2}-1}), \tag{7.4}$$

$$with \begin{cases} k : \text{number of SPDBs} \\ x_1 > \text{cabin start} \\ x_{i+1} > x_i \\ x_{\frac{k}{2}-1} < \text{cabin end} \end{cases}. \tag{7.5}$$

The variation of X yields to a different cable output for connecting the two components, as well as a varied cable length. Thus, it has a direct impact on the results of the two objective functions in 7.1 and 7.3.

The optimization problem has now been outlined and is summarized as follows. In equation 7.4, the input variable X specifies the interfaces between the SPDBs and PSCs in the cabin. The two optimization objectives y_1 and y_2 in 7.1 and 7.3 have a distinct outcome for each variant of X. The power distribution between the various SPDBs and the total cable length for all PSCs are specified there.

An algorithm is used to generate the optimization result. It facilitates the selection of X input values that produce optimal results for y_1 and y_2. The following section describes how this implementation was accomplished.

7.2 Assessment and Optimization Algorithm

Matlab is used to implement the assessment and optimization algorithm. To begin, the electrical components are prepared. These are the SPDB and cables that are created as new objects initially. As previously indicated, the SPDB position remains constant throughout the assessment and optimization process. Therefore, the design model's parameters are imported, and SPDB objects are initialized and placed using the SPDB configuration. The configuration is defined in the system model in CSM, and parameters are associated with the Matlab model. On the one hand, the cabin length is evenly divided and the SPDBs are distributed according to the configuration's specified number of SPDBs. On the other hand, the cable objects' parameters specify the identification numbers of the SPDB and PSC that are to be connected. The cable object coordinates are not mentioned since they are variable and are attached to different components in each scenario (for a specific X value).

After that, the algorithm generates the input variable X. It specifies a matrix containing all potential values for the cabin division. To do so, it first imports the parameters for the cabin's start and finish locations from the design model. These are always the matrix row's first and last values. Then, using the binomial coefficient, it generates all possible combinations of $x_1, x_2, ..., x_{\frac{k}{2}-1}$. This one is defined mathematically in equation 7.6.

$$C_n^k = {}^nC_k = \binom{n}{k} = \frac{n!}{k!(n-k)!}. \tag{7.6}$$

The variable n denotes the number of items, here the cabin length, which must be divided. For the binomial coefficient, a step of one meter is utilized to reduce the number of possible values. Thus, if the cabin length is ten meters and a configuration of six SPDBs is examined (i.e. three sections), the matrix contains $10^3 = 1000$ possible combinations.

Following that, each row is filled with a single combination. However, the constraints associated with these variables mentioned in 7.4 are not considered. Therefore, the algorithm specifies and checks these constraints for each row of the matrix. When the criterion is not met, the matrix's associated row is deleted. Finally, the matrix has only valid combinations and the input parameter is ready.

The algorithm then provides an iterative function with the SPDB and cable objects, as well as the PSC's from the cabin design, as inputs. Each iteration considers a single row of the input matrix X (i.e. one cabin division variant). The function must then return the power distribution and total cable length as an output.

To accomplish this, the algorithm evaluates each PSC's position within the cabin and determines, which cabin part it is located in. It sets the parameters of a related cable that connects this PSC to the SPDB in that cabin portion based on this information. To simplify the algorithm's representation of cable length, only the x-coordinates of the PSC and SPDBs are used to define cable length; 3D cable routing within the cabin is not considered. Finally, all PSCs are connected via a cable object to one of the SPDBs. Each of these cables has a specific length, which is employed to calculate the power output of the SPDBs using the equation in 4.9. Following that, the power distribution and total cable length outputs can be calculated according to equations 7.2 and 7.3 and assigned to the input row of X.

After completing this assessment for each iteration, each variant of the cabin division has its own set of two objective values. The results for the objective functions are normalized in order to provide simpler dimensionless results and to increase comparability. These are then plotted along the x and y axes of the two goal variables, with each point representing one cabin division input. The optimization process seeks to reduce both of these objectives. Therefore, a pareto front[2] is produced to determine which X inputs result in the two objective functions' minima.

[2] The Pareto front is the collection of all Pareto-efficient solutions in multi-objective optimization. It enables the optimizer to focus on a subset of efficient solutions and make trade-offs within that subset rather than examining the entire range of all parameters [113].

The list of codes includes an implementation of the algorithm as Matlab code (see code A.5). The algorithm was developed leveraging the integration results from the PSC described in 6.2.2 and various SPDB configurations. The next section summarizes the outcomes of each configuration.

7.3 Assessment and Optimization Results

The use case provided in the preceding sections is separated into three scenarios. Prior to discussing the outcomes of each scenario, the input parameters stated in each related configuration are described. The configuration is generated by instantiating SPDB blocks into the SysML model. Each instance of the electrical system is executed for each configuration. These configuration specifies the number of SPDBs, the cable parameters such as cross section and resistivity, as well as the PSC's rating power. A parametric diagram is used to convey the configuration parameters to Matlab (see fig. A.13).

The cable parameters and the rating power of the PSC's control unit are constant for all configurations. The aircraft is equipped with six SPDBs in the first configuration. The second configuration evaluates the effect of a higher number of SPDBs (eight) on power distribution and cable length. The bdd in figure 7.2 shows the parameters used in this configuration. The third scenario is more realistic, as it incorporates other SPDB-supplied components (Galleys). To accomplish this, the system model makes use of the configuration data stored in CPACS (see fig. 6.1). Thus, each module is connected to the SPDB in accordance with its indicated position in the cabin in the CPACS data. By specifying the type and size of the modules installed (cf. section 6.1), the quantity of power that must be supplied can be determined. Hence, the SPDBs attached to each of these modules have an initial power rating that is evaluated in a parametric diagram (see fig. A.14) and added to the PSC's computed power rating in the assessment algorithm.

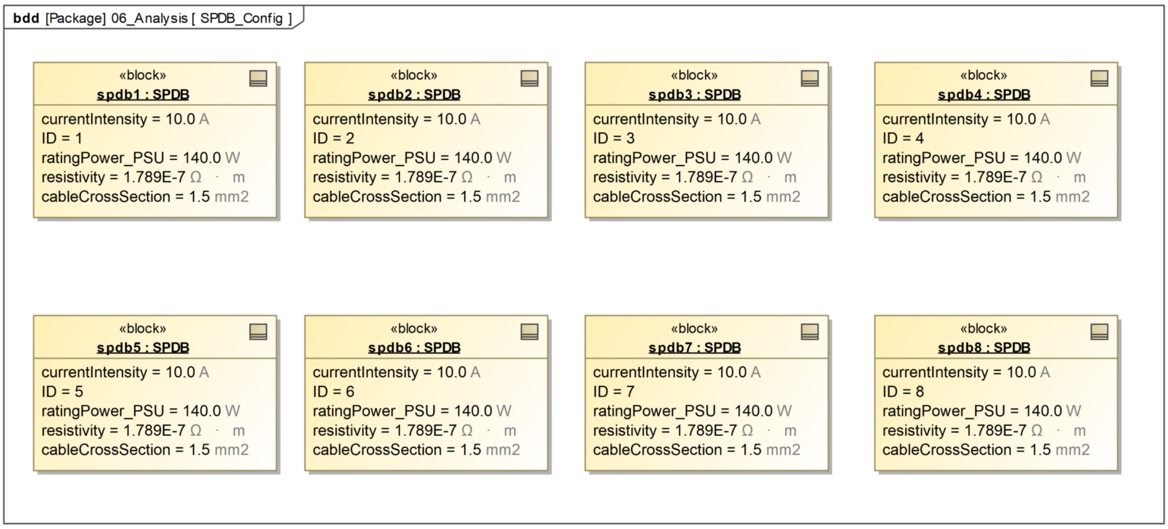

bdd [Package] 06_Analysis [SPDB_Config]
«block»
spdb1 : SPDB
currentIntensity = 10.0 A
ID = 1
ratingPower_PSU = 140.0 W
resistivity = 1.789E-7 Ω · m
cableCrossSection = 1.5 mm2
«block»
spdb2 : SPDB
currentIntensity = 10.0 A
ID = 2
ratingPower_PSU = 140.0 W
resistivity = 1.789E-7 Ω · m
cableCrossSection = 1.5 mm2
«block»
spdb3 : SPDB
currentIntensity = 10.0 A
ID = 3
ratingPower_PSU = 140.0 W
resistivity = 1.789E-7 Ω · m
cableCrossSection = 1.5 mm2
«block»
spdb4 : SPDB
currentIntensity = 10.0 A
ID = 4
ratingPower_PSU = 140.0 W
resistivity = 1.789E-7 Ω · m
cableCrossSection = 1.5 mm2
«block»
spdb5 : SPDB
currentIntensity = 10.0 A
ID = 5
ratingPower_PSU = 140.0 W
resistivity = 1.789E-7 Ω · m
cableCrossSection = 1.5 mm2
«block»
spdb6 : SPDB
currentIntensity = 10.0 A
ID = 6
ratingPower_PSU = 140.0 W
resistivity = 1.789E-7 Ω · m
cableCrossSection = 1.5 mm2
«block»
spdb7 : SPDB
currentIntensity = 10.0 A
ID = 7
ratingPower_PSU = 140.0 W
resistivity = 1.789E-7 Ω · m
cableCrossSection = 1.5 mm2
«block»
spdb8 : SPDB
currentIntensity = 10.0 A
ID = 8
ratingPower_PSU = 140.0 W
resistivity = 1.789E-7 Ω · m
cableCrossSection = 1.5 mm2

Figure 7.2: Electrical assessment configuration of SPDBs

Table 7.1 summarizes the main assessment and optimization results of the three considered configurations. For each of these configuration, two outcomes are noteworthy: the corresponding cabin division that results in the lowest possible cable length value (solution 1) and the lowest power distribution value (solution 2). The corresponding normalized values for both cable length and power distribution are shown for each solution. The value in bold represents the lowest normalized value, being the optimal one.

Table 7.1: Summary of assessment and optimization results

Configuration	**Cabin division (X) in mm**	**Cable index**	**Power distribution**
6 SPDBs	S1:(4,000|20,000|31,000|36,000)	**-1.276**	-1.196
	S2:(4,000|17,000|27,000|36,000)	-1.0242	**-1.878**
8 SPDBs	S1:(4,000|16,000|24,000|31,000|36,000)	**-1.423**	-1.412
	S2:(4,000|15,000|24,000|29,000|36,000)	-1.283	**-1.996**
8 SPDBS and 2 Galleys	S1:(4,000|16,000|24,000|31,000|36,000)	**-1.423**	-0.290
	S2:(4,000| 7,000|23,000|33,000|36,000)	0.385	**-1.303**

Figure 7.3 depicts the result of the optimization plot for the first configuration. The x axis represents the cable length (here referred to as the cable index), while the y axis represents the power distribution. Both values are normalized as stated in section 7.2. This is implemented by dividing each value by the norm of a vector that includes all the configuration values. This enables a comparison between different configurations. Each point (blue circle) represents a distinct cabin division, denoted by an X value.

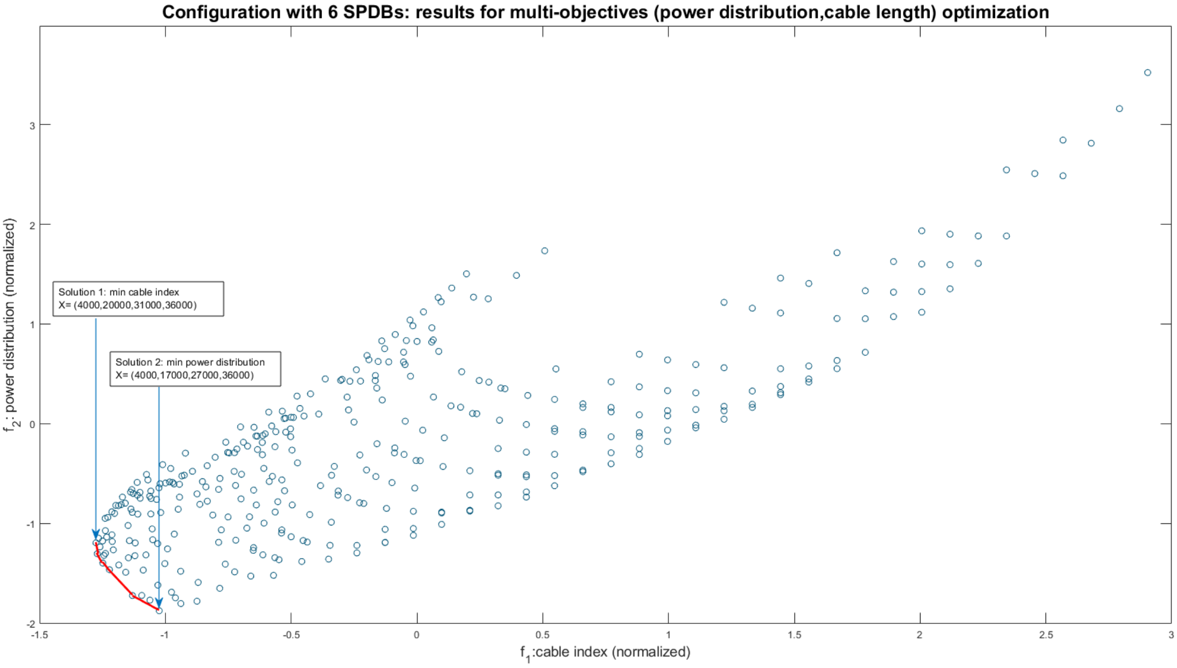


Figure 7.3: Cable index and power distribution results for the various cabin divisions (configuration with 6 SPDBs)

To help visualize the difference between the two results, figure 7.4 depicts the cabin division graphically. The cabin arrangement depicted in the illustration, which is being used as an example here, does not represent the one imported from CPACS. It merely facilitates a more accurate visualization of the cabin divisions. As can be seen, the optimal cable length (solution 1) is defined by the SPDB being nearly in the center of each section. This is because the algorithm makes an effort to minimize the distance between each PSC and the SPDB. On the other hand, the best power distribution (solution 2) is shifted leftwards. This is because the emphasis is mostly on connecting the same number of PSCs to every SPDB, which results in an optimal power distribution. In both situations, the rear section is the shortest, owing to the high level of PSC condensation in that region. The front part is the widest, due to the fact that first and business class have a lower amount of fitted PSCs.

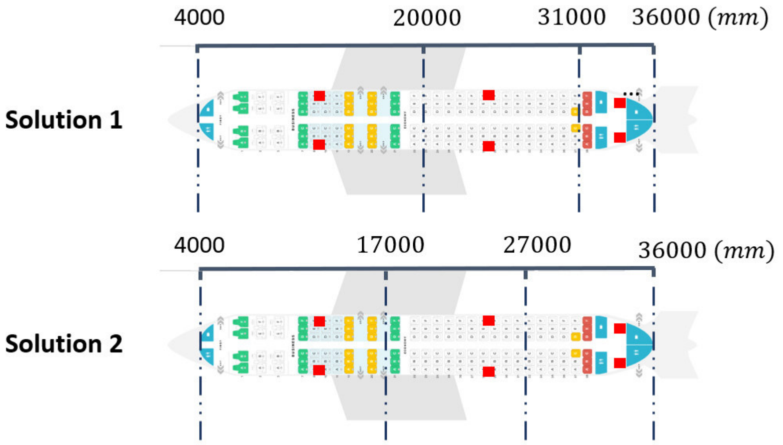


Figure 7.4: Cabin division of optimal solutions for power distribution and cable index (configuration with 6 SPDBs)

Figure 7.5 illustrates the optimization result for the second configuration. Due to the addition of two SPDBs to the previous configuration, the number of possible cabin divisions has increased.

As can be seen, both values of cabin length and power distribution are reduced. The normalized cable index was -1.276 in configuration 1 and decreased to -1.423 in configuration 2, whereas the normalized power distribution was -1.878 in configuration 1 and decreased to -1.996 in configuration 2. This means that by adding two SPDBs, the power distribution is improved and the cable length is reduced. Despite this expansion of the optimization objectives, the costs associated with extra SPDBs are not evaluated, and the benefits of saving two components may be greater than the improvement gained in this investigation.

The cabin division outcomes are depicted in figure 7.6. The comparative remarks for solutions 1 and 2 remain identical to those of the first configuration.

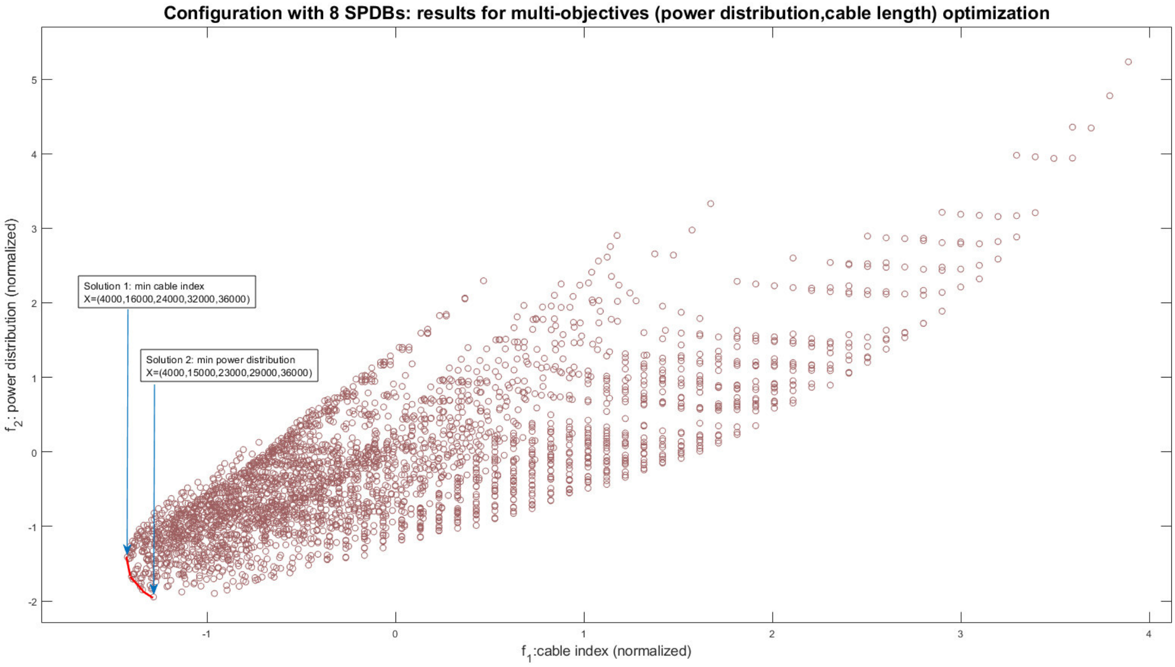


Figure 7.5: Cable index and power distribution results for the various cabin divisions (configuration with 8 SPDBs)

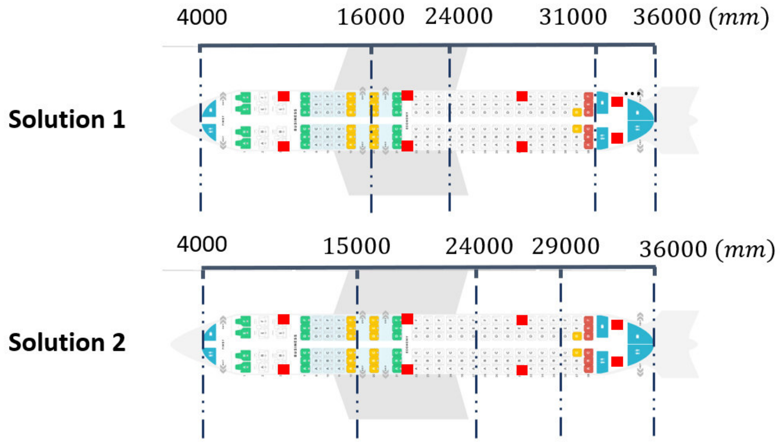


Figure 7.6: Cabin division of optimal solutions for power distribution and cable index (configuration with 8 SPDBs)

The final configuration incorporates the installation of two galleys within the cabin. These are connected to the aircraft's front and rear SPDBs. The front galley is rated at 25 kW, while the rear one is rated at 50 kW. These values are based on the power consumption of the galley in an A320 [115]. As illustrated in Figure 7.7, the cable index remains unchanged. This is a natural consequence, as the location of both SPDBs and PSCs are identical to the previous configuration. As a result, the algorithm's produced cable object remains constant. However, a deterioration in the power distribution can be observed with a normalized value of -1.303. This is because galleys require far more energy than PSCs. Thus, the SPBDs supplying the two galleys have a higher rating power than other SPDBs, resulting in an asymmetrical power distribution between them.

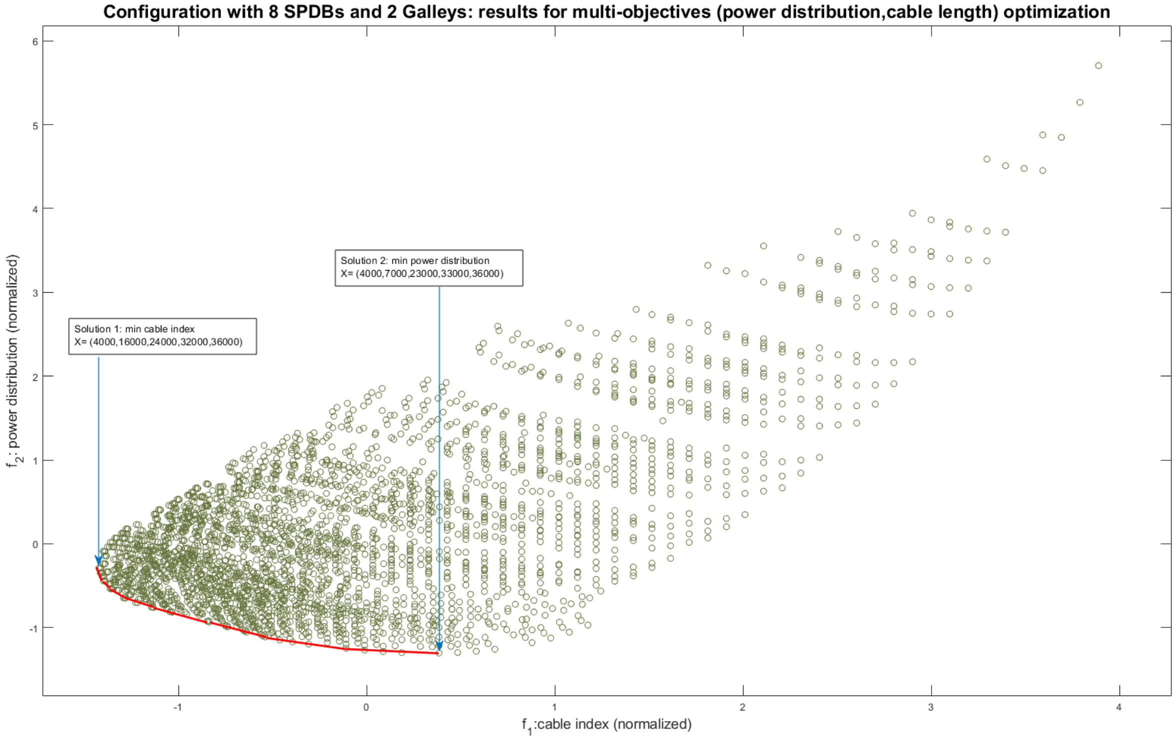


Figure 7.7: Cable index and power distribution results for the various cabin divisions (configuration with 8 SPDBs and 2 Galleys)

An interesting aspect of the data can be seen in figure 7.8, where the cabin division is depicted. For a minimal power distribution (solution 2), the cabin's front and back sections are narrower than the cabin's middle section. Examining the placements of the PSCs in the cabin, one can notice that neither of those two sections contains a component (the frontest PSC located at 7,353 mm and the rearest at 33,503 mm). This means that the power demands for the front and back galleys are greater than those for the PSCs. Thus, the algorithm determines that the optimal power distribution occurs when no PSC is linked to the SPDBs connected to the galleys. Additionally, the two middle parts illustrate how the PSCs are distributed across the four SPDBs in the center. Due to the fact that the first and business classes have less PSCs, the corresponding section is larger than the one that supplies the majority of the economy class's PSCs.

When all three configurations are considered, it is obvious that the total rating power supply of all PSCs connected to the same SPDB correlates with the loss of power induced by the connecting cables. Thus, the effect of the loss of power on the power distribution is discernible.

The analysis results for the examined use case suggest additional pathways for meaningful consideration. Addition of a new cable index objective that considers not only the cable

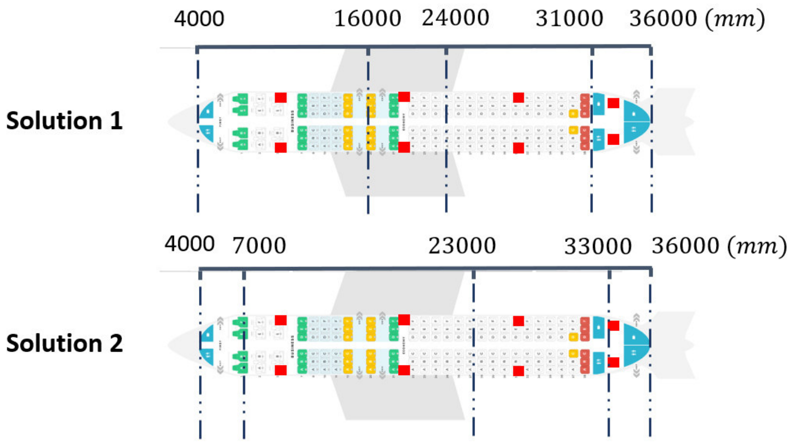


Figure 7.8: Cabin division of optimal solutions for power distribution and cable index (configuration with 8 SPDBs and 2 Galleys)

weight generated by the cable length, but also the integration and assembly in the cabin. The aircraft's front and rear sections incorporate more components (e.g. avionic) than the aircraft's center sections. Furthermore, the fuselage cross section is smaller, resulting in less area for the installed components to be integrated. An extension of this use case's trade-off analysis is considering these integration features and evaluating the architecture configurations regarding them.

Moreover, in this use case, the cable parameters remained unchanged. Given the inability to discern the effect of the power loss on the power distribution, varying specific characteristics (such as cable material and cross section) can focus attention on the cable power loss.

Chapters 5, 6, and 7 demonstrated how the PSC's analysis, architecture and integration are approached using the model-based methodology presented in this work. The following chapter will discuss relevant outcomes from the PSC's architecture definition and its integration into the cabin design process. Additionally, it contrasts the methods utilized in this work with the ones used in the other studies mentioned in Chapter 2's literature review.

8 Discussion

This chapter is divided into three sections, that discuss the main outcomes of this work. First, the methodology for system analysis and architecture modeling is examined. The knowledge gained from the PSC modeling is then presented. The second section discusses the integration methodology used to connect the resulting system architecture to the cabin design process. It is contrasted with the integration approaches described in the chapter on literature review. The model execution and simulation results are discussed. The final section focuses on the evaluation and optimization results obtained throughout the selected use case. Characteristics of the optimization are presented, as well as previously unconsidered aspects of cabin design and system integration. Additionally, an overview of additional optimization possibilities is given.

8.1 System Analysis and Architecture Modeling

It is difficult to evaluate or validate the approach of this work by comparing it to that of others. Numerous aspects influence the methodological decisions. These are related to the modeling objectives, the tool employed, the technical scope of the application use case, or the desired level of modeling abstraction and detail. However, the PSC's modeling results can be examined in light of observations and assertions made during the modeling process.

The methodology developed leverages the strengths of each method. The extent, to which OOSEM's analytical and development approaches (such as enterprise model or test) are detailed, is not necessary in this work. Due to the work's emphasis on system analysis, architecture, and traceability, all of which match to the V-Model's link tight, only OOSEM's parts pertaining to these elements were considered. This allowed for a reduction in methodological complexity without compromising critical steps.

Obviously, referring to OOSEM and *Abulawi*'s method, results in a top-down approach being considered in this work's methodology. Its application to the modeling of PSC demonstrated both the benefits and drawbacks of this approach. Beginning the analysis at the highest level of abstraction allowed for a separation of the SoI context, its functional analysis, and the logical synthesis from its components. Since each of these abstraction layers is independent, it is simple to generate new architectures. For instance, if a better solution (i.e. an alternative logical component) to implement some of the identified PSC's functions exists, the logical architecture can be easily exchanged, and the required allocation (of logical component to functional block) can then, thanks to the realized traceability, be directly adapted. Or, if stakeholders' needs evolve over time, necessitating the addition of new system functions, the architecture on a functional layer can be changed and linked to the new requirements, allowing necessary changes on the logical layer to simultaneously be identified and adapted. Moreover, requirement specifications can be easily adapted to new system architectures due to the traceability effort applied throughout the modeling process, which results in a causal connection between all architecture aspects.

Another advantage of the top-down approach is a more complete awareness of the systems value among its context. By beginning with an abstraction of the SoI as a black box, it is easy to focus on all external elements that influence or are influenced by it. Thus, no interaction between the system and its environment is overlooked, and all critical elements and contextual requirements can be discovered and considered during the modeling of lower abstraction levels. Moreover, once the system's structure and behavior are specified as a white box, the association of stakeholder needs with the resulting system design elucidates the rationale for each architecture element. Since a base architecture is regarded as a starting point for further refinement, many of its components may be unnecessary and incompatible with the stakeholders' and contextual requirements. This way, system architects can ensure that their designs are responsive to actual needs and expectations.

However, during the modeling process, some shortcomings of the top-down approach were uncovered. This is primarily due to the SoI type considered in this work, the PSC. Its uniqueness as a cabin module renders it impossible to be characterized as a system in the conventional sense, as it combines components from many aircraft systems. As a result, it cannot be decomposed and examined hierarchically within its scope as a single system. This created immense difficulties, particularly during the functional analysis. Many of the functions integrated into these modules were specified as part of subordinate system functions during the use case specification. To determine the role of the PSC or its integrated components in relation to other external system components, an examination of each of these systems required to begin at the top level. The top-down method then led to lower abstraction levels at which the functionality of PSCs could be discovered. This was achievable solely in this work for two reasons. Firstly, the functions of certain external systems, such as the oxygen or air conditioning systems, were considered rather simplistically, implying that their required full analysis was not performed. Secondly, *Fischer*'s work has been used to import the simplified results for the cabin management system's use case study. However, in a true architectural design project, requiring thorough and precise analysis, this could represent an obstacle. Due to a lack of information on the involved systems, obtaining the necessary information requires considerable time and effort. Even if information were available, the module architect lacks the expertise of all the various complex systems necessary to identify and comprehend the module's involvement and role within them. In this instance, a bottom-up approach may be appropriate, as it enables the module architect to initiate functional analysis and provide functional requirements for the module's interface with external systems.

Both the benefits and drawbacks discussed previously highlight an essential fact about the reproducibility of modeling other SoIs. Because this work's SoI is an integrative module composed of elements from various aircraft systems, the architecture is heterogeneous. Internal components interact with one another infrequently, as they are primarily involved in interactions with their corresponding systems. This has an effect on the modeling methodology, as the emphasis is shifted toward the analysis and comprehension of the module's heterogeneity. The model ontology defined in this work has been adapted to identify specific component attributes, such as ATA or DAL, and to comprehend the role and impact of each external system within the PSC. This approach, however, is not as useful when it comes to analyze a single system or subsystem (e.g. CIDS or CIDS director). In this case, a more sophisticated behavioral understanding of the components' internal interactions is critical. Additional modeling activities, such as defining the system's states, modeling functional scenarios with an emphasis on each component's role, as well as detailed exchange and communication via activities and sequence diagrams, are then required to accomplish this.

Moreover, this work demonstrates that methodological reproducibility is highly dependent on the MBSE tool used. In numerous steps taken to model the PSC's architecture, the tool played a critical role in achieving modeling benefits. For instance, the automated application of the FAS method and the block-oriented synthesis of the functional architecture of the PSC are primarily due to the CSM tool's support. The effort required to implement the FAS methods' various steps and failure detection manually cannot be quantified in this work. This effort might well be greater than the resulting benefits if a different tool were to be used. Additionally, the model's traceability is largely due to the tool support. The ability to use SysML relationships and visualize them in a variety of formats is not supported by all SysML tools. Manual creation of traceability diagrams and views can be counterproductive if the modeling effort required and the affiliated error probability are too excessive.

Besides, the modeling performed in this work sheds light on new possibilities for system design. The integration of disparate system components corresponding to diverse ATA chapters within a single module provides an opportunity to identify synergies and intersections between these distinct systems. The module is then the point of intersection from which this heterogeneous interaction can be exploited. To gain a better understanding of these effects, consider the PSC's logical architecture. Due to the fact that the oxygen system, air conditioning system, and cabin communication system are all integrated into the PSC, the latter can be used to enable new decentralized interactions (e.g. for data communication or material flow) between these components. For instance, if the CIDS director requires information about passenger oxygen, the PSC can facilitate this exchange because it integrates both the control unit and the oxygen module. These considerations can be extended to all cabin modules, assisting in the optimization of both the cabin and system designs, as well as the reduction of installation space and material demands.

8.2 Integration in Cabin Design Process

The methodology used in this work to integrate the SoI's architecture into the cabin design process enabled to accomplish the integration objectives. When communicating with both the aircraft design exchange platform (CPACS) and the Matlab-based cabin design and optimization model, the SysML system model serves as the connecting link between systems analysis and design data. The automated use of CPACS data in the system model enabled the emergence of a configurable system design with aircraft and cabin design parameters. Thus, the intended objective of configuring and instantiating the system architecture using aircraft design data, via a knowledge exchange platform has been accomplished. However, the architecture results could not be transferred back to the CPACS platform due to the CPACS schema's lack of aircraft systems consideration. For example, quantifiable functional system specifications and system properties, such as PSC rating power or cable properties, are important information that should be shared via CPACS with involved aircraft designers and engineers. This is unquestionably a meaningful extension of this work's methodology, once the CPACS schema has also been appropriately extended.

Furthermore, by transferring the architecture results to the Matlab design model, it is possible to trace the cabin objects back to a detailed analysis. Both the cabin objects' composition and internal structure were imported from the system model. The realized integration extended the causal relationship between the analytical derivation of the system architecture and the cabin geometrical design that was missing in the initial process. Additionally, by transferring cabin design data back to the system model, synergies between geometrical and system-analytical parameters could be identified and exploited. This enabled the implementation of a multi-domain optimization, using both models and a trade-off analysis, that considers the objectives of different disciplines.

Converting relevant data to JSON-format simplified the interface's implementation and data exchange. The support for external languages in opaque behaviors, as well as the ability to import required objects, such as the Json-parser, facilitated the rapid processing and import of cabin data from CPACS. However, it is important to state at this stage that a direct interaction with XML-based CPACS-files without intermediary is feasible. While processing JSON-files in Javascript code specifications is unquestionably simpler, direct access is also possible using the same approach with opaque behaviors. It simply requires more detailed coding and effort to identify all pertinent cabin data. Furthermore, the Matlab interface was implemented in stages throughout the model by sharing the Matlab workspace. Due to the ease with which Matlab could be integrated into CSM, specifying and reading cabin objects and their properties could be done directly from the system model. In parametric diagrams, where the constraints are specified in the Matlab language, simple binding of values between the two models was possible.

The Matlab interface could not be fully automated in the same way that CPACS is. This is because certain elements of the system model require the cabin objects to be initialized before the architecture emergence process can begin. Moreover, the Matlab script part, which is responsible for optimizing the placement of cabin objects, requires input from CSM prior to running. However, this communication automation between the two models is possible by implementing synchronization signals that trigger the execution of the external model. This is an extension possibility to automate models integration in future work.

In contrast, a synchronized evaluation based on parametric diagrams and Matlab values is not possible. This is because the SysML model's associated parametrics for each block are only executed at the start of the block's execution. The issue arises when a new evaluation must be performed following a parameter change caused by an action in the external model, in this case in Matlab. The system model is unable to check for new values in this case because its parametrics have already been executed via the parametric constraint. Throughout the execution, only continuous value binding is possible. When continuous synchronization under the usage of constraints is required, it is necessary to improve the parametric evaluation. For this purpose, the integration of Simulink blocks into the SysML model is useful. CSM enables easy integration of Simulink models, allowing for the implementation of this synchronization, with the help of its various synchronization options/tools.

Just like the system analysis and architecture modeling, the tool used has a huge impact on the approach taken for model integration. The interfaces to CPACS and Matlab implemented in opaque behaviors or parametric constraints were possible for two reasons. First, the CSM tool facilitates this by supporting external programming languages. This approach would make it tremendously more challenging to access data from external models otherwise. The second reason is the model execution capability, which is necessary for specifying run-time objects, that are processed across multiple models. This approach is not possible if the tool being used does not support model simulation. Thus, the integration capabilities of the tool are the primary factor in determining the integration approach.

As stated in the literature review (cf. section 2.3.4), *Vanderperren et al.* state that SysML and Matlab can be combined via either co-simulation or the use of an executable language. In this work, both the tool and the application use case resulted in the latter approach. In comparison to *Chabibi et al.*, who take a similar approach but develop a separate DSL to facilitate communication, this work requires significantly less effort. It was therefore possible to avoid the specification of a DSL due to the integrated scripts in the opaque behaviors. Thus, data could be exchanged without errors only by using the same concrete syntax between Matlab and SysML, which is guided by the ontology and modeling guidelines developed here. Additionally, this work did not necessitate detailed synchronization between the two models. However, if this were the case, due to the identified inability to use parametric diagrams, a co-simulation would be meaningful and is comparable to the use case presented by *Johnson et al.* (cf. section 2.3.4).

Furthermore, the abstraction level of the system modeling has also an effect on the integration approach. The SysML and Matlab models in this work can be classified as belonging to an intermediate abstraction level. On the one hand, more detailed system development typically involves the use of higher-fidelity models for specific domains. On the other hand, other models, platforms, and tools are used to consider the system's complete life cycle at a higher abstraction level. In this case, the OSLC exchange format presented in the section 2.3.4 or semantic web technologies (SWT)[1] are more appropriate means of facilitating information

[1] Semantic Web technologies are intended to describe and link data in a manner comparable to how traditional web technologies define and connect web pages [116].

exchange across different system life cycle models. This is an important extension of the integration use case discussed here, which is essential for the development of model- and knowledge-based system life cycle analysis and design.

Finally, it is critical to discuss the execution time of the models as an integrating factor. The models were able to emerge and communicate with one another in a very short period of time (less than one minute). This is advantageous, as it enables system architects and engineers to rapidly configure, design, and evaluate systems based on different models. However, it is crucial to emphasize at this point that the use cases considered in this work, as well as the selected SoI, are significantly less complex than those for large aircraft systems. The time required to complete the system model and integration can be significantly longer. This effect will need to be evaluated in future work. Additionally, the model scope used in the execution includes all data generated during the system analysis and specification. This can be minimized because not all of this data is required for external communication. This has a sizable effect on the simulation time as well. Therefore, a method for reducing this data is critical to consider in future research.

8.3 Architecture's Assessment, Optimization and Trade-off Analysis

The objective of the architectural assessment study presented in chapter 7 was to demonstrate the benefits of emerging system architecture and its integration into the cabin design process. To accomplish this, several facets of the practical usefulness of the system architecture and its interaction with the design process models were depicted using the chosen use case. The latter aimed to evaluate interdisciplinary interactions by utilizing the information obtained from architecture integration into the design process. Therefore, the cable length objective function was chosen because it incorporates information about the geometric placement of the PSCs and SPDBs. It also enables the architectural interfaces between two distinct systems, namely PSC equipment and the power system, to be evaluated. The objective function of power distribution was chosen to determine the effect of equipment location on the required power supply. It also considers the power loss which is directly related to the interface linking the SPDBs to the PSCs.

One usefulness aspect is the realization of a multidisciplinary design optimization, that treats each model as a representation of a distinct domain. The fact that the various models integrated into the cabin design process are affiliated to distinct disciplines, enables to combine the system analysis according to systems engineering activities with other disciplinary domains. The disciplines considered in this work include the system analysis and design (using SysML and CSM), the geometrical cabin design (using Matlab), and the preliminary aircraft design (extracted from CPACS). The optimization was carried out using the relationships between the parameters in each domain. The PSC's system architecture was instantiated using several of the cabin design parameters imported from CPACS, such as seat distribution and cabin class configuration. Other parameters, such as galley type and location, were associated with power system instantiation, as this had an effect on the SPDB power supply needed for the galleys in the third configuration. System architecture data, such as the PSC distribution within each cabin class, has an effect on the geometrical cabin design result in Matlab. Additionally, the location of each PSC created by this cabin design is related to the cable length, which affects the cable's power loss. This demonstrates that the three models' parameters are closely linked and have an effect on one another. This impact is evidently included in the optimization process, as these parameters are either considered directly (e.g. cable length, galley type and location) or indirectly (e.g. PSC distribution and position in the cabin) when calculating the optimization objective functions. Given that the system architecture was developed using the MBSE approach, the interaction of the multidisciplinary parameters can be viewed as a bridge between MBSE and MDO.

It is critical to emphasize that the optimization study that was conducted is extremely simplified. Numerous aspects and methods for defining and solving optimization problems that occur during an MDO are not considered in this work. The algorithm employed is not a mathematical optimization algorithm; rather, it was developed and programmed specifically for this optimization case. It defined the problem by specifying the range, step, and constraints of the input variable X, and then solved it by computing the two objective functions for all possible input values. Due to the fact that all results are considered, it was possible to avoid local minima and validate the optimization directly. After that, a pareto front was generated as a starting point for trade-off analysis. The two input values of X that result in the minima of each objective function were highlighted, allowing for direct comparison. As a result, the intended identification of interfaces between the PSCs and SPDBs that result in the optimal power distribution and cable index was accomplished in a simple manner. However, considering a larger number of input values and optimization objectives expands the design space and necessitates an appropriate optimization definition and solution. In this case, the consideration of the optimization problem formulation and strategic organization is very important and also referred to as *MDO architecture*. *Martins* elaborates on these considerations in his survey of MDO architectures [117]. Additionally, there is a diverse range of optimization algorithms that are suitable for specific MDO architectures, making it critical to select the most appropriate one. *Martins et al.* discuss and explain various optimization methods and algorithms in their book [118]. Numerous types of numerical solver are available and required to solve the optimization problem using the selected algorithm. An overview of the numerical solver for MDO is provided in [119].

Another benefit of the implemented optimization is the opportunity revealed by the resulting trade-off analyses. By implementing a multi-objective optimization that incorporate input parameters from various systems and modules (e.g. power system and equipment), designers and engineers can make decisions based on direct relationships between these parameters. By normalizing the objectives' results, it was possible to compare different design configurations and to identify the effects of input parameters with the assistance of the optimization algorithm. The three configurations were chosen to emphasize the relationships and interactions between these design parameters. The first configuration considered is a power system architecture that includes six SPDBs and connects them to the PSC architecture developed in this work. The second configuration was designed to examine the effect of changing the power system architecture on power distribution and cable length. Additional weight, procurement costs, installation space, and maintenance are associated with the addition of two SPDB components. However, when compared to the first configuration, the optimization results for both power distribution and cable length objectives were improved. The architecture change altered the position of the SPDBs, allowing for a more accurate cable index. Simultaneously, the difference between SPDBs' power supply is smaller, allowing it to reduce its electrical performance requirements and thus associated costs. This also shows the correlation between the cable length and power distribution. The shorter the cable length is, the less loss power resulted in the electrical design. This reduces the power supply to each of the PSCs. Thus, the differences in power supply required from the various SPDBs is also reduced, resulting in a more efficient power distribution. Both configurations have their advantages and disadvantages, as demonstrated by the results. To facilitate decision-making, the scope of this work's investigation must be expanded. A trade-off analysis must be conducted to compare the total costs of each configuration, considering all of the factors mentioned above. Furthermore, the third configuration sought to determine the impact of electrical equipment on the optimization objectives. Consideration of the galley as an electrical consumer significantly changed the SPDB-PSC interfaces, which result in optimal power distribution. Besides this, equipment considerations deteriorated the power distribution objective value. This means that an electrical power design that considers only PSCs and galleys as electrical equipment is not beneficial. This information is critical when designing and installing additional electrical equipment, such as seats. This consideration will assist in compensating for the differences in power distribution between the SPDBs that supply the galleys and those that supply other equipment.

Moreover, analyzing these interactive effects, between various aircraft parameters early in the design process, is extremely beneficial. It enables to determine the design modification's effect of a single aircraft system on external systems and thus to account for the aircraft's global effect. This reduces the number of unnecessary and cost-ineffective changes in the aircraft design. In the aircraft industry, a practical example is the use of carbon fiber reinforced polymer in the aircraft structure. Motivated by its superior mechanical properties and low density, aircraft designers and engineers overlooked the impact of the material's insufficient electrical conductivity for certain applications (e.g. lightning protection). The architectural modifications required additional implementing of electrical functions [120]. This demonstrated design flaws. Aircraft designers are therefore re-investigating the possibility of reusing aluminum, as a structural material in future designs. By implementing trade-off analyses in the demonstrated manner and analyzing the interdisciplinary effects, a detailed assessment of design modification at all aircraft levels can be accomplished, avoiding extra costs.

The optimization implemented in this research serves as a proof-of-concept (PoC). It demonstrated the feasibility of a multidisciplinary design optimization and trade-off analysis, that consider systems from various domains and make use of data from various models. However, as with any PoC, this work's use case is incomplete and predicated on numerous assumptions. For example, treating consumed power as the rating power for all PSCs in the cabin during all flight phases, ignoring 3D cable routing in the cabin, and disregarding numerous electrical loads such as seats or IFE components.

Additionally, the optimization results obtained, demonstrated that considering more optimization's boundary conditions and requirements enables the analysis of more design effects. As can be seen in this study, the addition of the galleys to the PSCs resulted in completely different interfaces between the power supply components (SPDBs) and the power consumer (PSCs). As a result, it is necessary to expand this study in future work and consider more systems and modules in the cabin configuration. Moreover, incorporating additional optimization objectives and variable input parameters into a trade-off analysis, provides a holistic view of the effects of various system and module parameters on multiple design objectives.

This study identified potential for future expansion of the selected use case. As demonstrated, the integration and installation of components in the aircraft enables it to combine geometrical design and analytical system properties. The optimization process was carried out using very simple parameters, such as cable length, but it could easily be extended to include cost-related factors, such as assembly, operability, maintainability, safety, reliability, and availability. Consideration of each of these in combination with system architecture using MBSE broadens the scope of assessment and optimization and enables a better understanding of design relationships and synergies.

This work demonstrated several benefits in terms of system traceability, reconfigurability of knowledge-based architectures, and multidisciplinary design optimization. However, as INCOSE states in its Vision 2025, in order to overcome challenges not only in the development of aeronautical systems but also in the development of other complex systems in a variety of industries, a need to extend the vision of systems modeling is necessary [56]. A transition from understanding it as a way to overcome the limitations of document-based approaches to viewing it as a standard practice, fully integrated with other engineering modeling, and utilizing internet-based knowledge representation to share human underlying knowledge, is crucial. The next chapter summarizes this work's findings and describes important expansion that would contribute to achieving these objectives.

9 Conclusion and Outlook

The aviation sector relies on the integration of new and innovative technologies and concepts to meet environmental and socio-economic challenges. Model-based Systems Engineering represents an appropriate design methodology in the field of aircraft design and supports the development of new innovative concepts and configurations. The German Aerospace Center is involved in achieving aviation goals and advancing the industry's digitalization. Its efforts have resulted in the development of a process, that generates cabin system designs automatically from conceptual aircraft design parameters. Cabin design outcomes can be evaluated and validated in the virtual environment by utilizing a virtual reality platform in order to enhance the immersion in design. This work's primary objective was to extend this process through the use of a model-based function-oriented system architecture. To accomplish this, a methodology for system analysis, architecture development, and integration, had to be developed. Due to its complexity and multidisciplinary nature, the passenger service channel was chosen to apply and evaluate the methodology.

The architecture of the PSC was modeled using the Systems Modeling Language and the Cameo Systems Modeler software. To begin, the model was structured and a model ontology, which has been related to both the CPACS structure and the ATA definition, was developed. The system analysis followed a top-down approach, beginning with the identification of stakeholder needs and context modeling. This initially permitted a black-box representation of the PSC. Following that, PSC's use cases were deduced from high-level functional requirements. The use cases were used as inputs for the FAS-method that was applied to generate a block-oriented functional architecture. This enabled the creation of a functional representation of the PSC that was not dependent on any technical solution. Additionally, a logical base architecture was defined and allocated to the functional architecture, which includes the PSC's internal composition as well as the interaction between its components. All relationships between different model elements leading up to the architecture results were depicted and highlighted using maps, matrices, and tables. The agile top-down approach and linking of model elements enabled requirement-based system analysis and the causal derivation of system architecture. The modeling activities reduced the system's complexity and resulted in a high degree of traceability, which facilitates the validation of stakeholders' needs and system requirements.

Subsequently, the system model was integrated into the cabin design process to refine the logical architecture. To begin, an interface to the conceptual design parameters in CPACS was implemented. A Json-file containing relevant cabin parameters extracted from CPACS was used as an intermediary for cabin data exchange. In CSM, the system model accessed cabin data via opaque behavior and run-time objects that specified system block properties during model execution. The parameters were used internally to instantiate the PSC distribution in the cabin, as well as the configuration of its internal components. The resulting architectural instance was then transferred to Matlab's cabin design model. The CSM tool enabled the sharing of Matlab's workspace as well as the integration of Matlab scripts within opaque behaviors. This provided direct access to the cabin objects and parameters in Matlab's design model for the purpose of creating, modifying, and reading cabin design data. The PSC objects and system architecture properties were used as inputs to the Matlab design algorithm. Along with the imported architecture data, a set of geometric parameters for cabin objects, as well as design and certification requirements were defined in Matlab. The design algorithm was then executed and resulted in an optimized geometrical distribution of the PSC and other cabin systems and modules. The knowledge-based and automated configuration of the system architecture using preliminary design parameters from CPACS enabled the consideration of new cabin layouts in the system architecture instantiation. Furthermore, the interface to Matlab and the corresponding architectural data exchange bridged functional analysis with traceable geometrical cabin design.

To demonstrate some of the benefits of developing and integrating the PSC's architecture into the design process, a trade-off study was conducted using data from multiple models. Therefore, an optimization use case has been defined and considered the objectives of various design disciplines and optimizes the PSC architecture accordingly. It investigated the PSC's interface with the power supply elements (SPDBs). It treated power distribution between the various SPDBs as an objective for the electrical design domain, and the length of the cable connecting the two components as an index, affecting cable weight and integration in the aircraft. Three configurations were evaluated to determine the effect of the electrical power system architecture by varying the SPDBs number and the effect of including additional cabin equipment parameters such as galley power consumption. The results were comparable and demonstrated, that the two optimal objectives were achieved through the use of distinct PSC-SPDB interfaces, allowing for a trade-off between the two design domains. The advantages and disadvantages of each configuration were discussed in terms of optimization objectives as well as design parameters such as weight, maintenance, and installation. Furthermore, when additional power consumers, such as galleys, were considered, notable differences in the power distribution were observed, motivating future research to consider as many cabin parameters as possible to enable a broader and thorough design assessment.

Outlook

To expand the scope and benefits of this work, more aircraft design parameters in combination with model-based system architectures are needed. When data from different life cycle phases are integrated, the system can be viewed holistically and its design's effects and interactions can be evaluated throughout its lifecycle, not just during the system's development stage. OSLC or semantic web technologies can be used to achieve this at high abstraction levels. Even so, specific data from high-fidelity analysis models must be incorporated into the digital representation of the system. The communication between system models and engineering domain models can be implemented by using the Modelcenter platform. Furthermore, the integration of more multidisciplinary data will necessitate the use of appropriate optimization algorithms and solvers, which will enable MDO studies to provide highly reliable results, that can be considered at the early stages of aircraft development.

Additionally, the existing VR platform should be directly connected to the system model during the design process, to enable visualization of system behavior in the virtual environment. Through the use of a real-time interface, the model will be able to modify system parameters and validate requirements simultaneously while a VR-user interacts with the cabin design.

Finally, due to the considerable relevance of the electrical design use case investigated in this work, an expansion thereof is important. Communication data and power-related properties of each cabin payload are crucial in a more electric aircraft because they affect critical aircraft design parameters such as mass, fuel consumption, and aircraft data-bases. Therefore, it is necessary to consider in subsequent work new or enhanced electrical system architectures, that are applicable to futuristic aircraft programs.

Bibliography

[1] Steffen Wischmann, Leo Wangler, and Alfons Botthof. Industrie 4.0 volks- und betriebswirtschaftliche faktoren für den standort deutschland. *Autonomik für Industrie 4.0*, 2015.

[2] Deloitte. Industry 4.0 and the digital twin. https://www2.deloitte.com/us/en/insights/focus/industry-4-0/digital-twin-technology-smart-factory.html, 2017. urldate: 08.11.2021.

[3] Michael Grieves. Digital twin: manufacturing excellence through virtual factory replication. *White paper*, 1:1–7, 2014.

[4] Azad M. Madni, Carla C. Madni, and Scott D. Lucero. Leveraging digital twin technology in model-based systems engineering. *Systems*, 7(1):7, 2019.

[5] Malcolm L. Nicolai. *Fundamentals of aircraft design*. Nicolai, 1975.

[6] Ajoy K. Kundu. *Aircraft design*, volume 27. Cambridge University Press, 2010.

[7] Dieter Scholz. Aircraft design. *Lecture script*, Department of Automotive and Aeronautical Engineering Hamburg University of Applied Sciences, 25.08.2021.

[8] William H. Mason. Modern aircraft design techniques. *Handbook of Transportation Engineering*, pages 26–1, 2003.

[9] Joaquim Martins and Andrew B. Lambe. Multidisciplinary design optimization: a survey of architectures. *AIAA journal*, 51(9):2049–2075, 2013.

[10] Luiz F. T. Fernandez. Applying mdo to classical conceptual aircraft design nmethodologies. Master's thesis, Instituto Tecnológico de Aeronáutica, 2019.

[11] Pier D. Ciampa and Björn Nagel. Agile the next generation of collaborative mdo: Achievements and open challenges. In *2018 Multidisciplinary Analysis and Optimization Conference*, page 3249, 2018.

[12] Pier D. Ciampa and Björn Nagel. Agile paradigm: the next generation collaborative mdo for the development of aeronautical systems. *Progress in Aerospace Sciences*, 119:100643, 2020.

[13] Pier D. Ciampa and Björn Nagel. Accelerating the development of complex systems in aeronautics via mbse and mdao: a roadmap to agility. In *AIAA AVIATION 2021 FORUM*, page 3056, 2021.

[14] Olivier de Weck and Karen Willcox. Multidisciplinary system design optimization. 2010.

[15] Craig B. Chapman and Martyn Pinfold. Design engineering a need to rethink the solution using knowledge based engineering. *Knowledge-based systems*, 12(5-6):257–267, 1999.

[16] Josip Stjepandić, Wim JC. Verhagen, Harald Liese, and Pablo Bermell-Garcia. Knowledge-based engineering. In *Concurrent Engineering in the 21st Century*, pages 255–286. Springer, 2015.

[17] Gianfranco La Rocca. *Knowledge based Engineering techniques to support aircraft design and optimization*. PhD thesis, Faculty of Aerospace Engineering, TU Delft, Delft, 2011.

[18] John McCarthy. What is artificial intelligence? *Computer Science Department Stanford University*, 2004. http://www-formal.stanford.edu/jmc/.

[19] Mohammed M. M. Sarcar, Mallikarjuna K. Rao, and Lalit K. Narayan. *Computer aided design and manufacturing*. PHI Learning Pvt. Ltd., 2008.

[20] Marko Alder, Erwin Moerland, Jonas Jepsen, and Björn Nagel. Recent advances in establishing a common language for aircraft design with cpacs. *Aerospace Europe Conference 2020*, 2020.

[21] Deutsches Zentrum für Luft-und Raumfahrt e.V. cpacs homepage. https://cpacs.de/, note = urldate: 27.08.2021,, 2021.

[22] Helmut Vonhoegen. *Einstieg in XML: Grundlagen, Praxis, Referenz*. Rheinwerk Computing. Rheinwerk Verlag GmbH, Bonn, 8., 2015.

[23] World Wide Web Consortium. Extensible markup language (xml) 1.0 (fifth edition). https://www.w3.org/TR/REC-xml/, 2008. urldate: 27.08.2021.

[24] William J. Rapaport. Syntax, semantics, and computer programs. *Philosophy & Technology*, 33(2):309–321, 2020.

[25] Florian Beckert. Automatisierte kabinendarstellung der cpacs-flugzeugdefinition in der virtuellen realität. Master's thesis, Technische Universität Hamburg, 2020.

[26] Jan-Niclas Walther, Bahadir Kocacan, Christian Hesse, Alex Gindorf, and Björn Nagel. Automatisierte kabinenvirtualisierung auf basis von flugzeugvorentwurfsdaten. In *Deutscher Luft- und Raumfahrtkongress 2020*, September 2020.

[27] Mara K. Fuchs, Florian Beckert, Mitul Gopani, Alex Gindorf, and Björn Nagel. Virtuelle realität im digitalen designprozess von flugzeugkabinensystemen. In *Deutscher Luft- und Raumfahrtkongress 2020*, 2020.

[28] Mara K. Fuchs, Florian Beckert, Jörn Biedermann, and Björn Nagel. Experience of conceptual designs and system interactions for the aircraft cabin in virtual reality. In *AIAA AVIATION 2021 FORUM*, page 2773, 2021.

[29] Julian Scherer and Dieter Kohlgrüber. Fuselage structures within the cpacs data format. *Aircraft Engineering and Aerospace Technology: An International Journal*, 2016.

[30] Mathworks Inc. Matlab. https://de.mathworks.com/products/matlab.html, 2021. urldate:20.09.2021.

[31] Python Software Foundation. What is python? executive summary. https://www.python.org/doc/essays/blurb/, 2021. urldate:30.08.2021.

[32] Mathworks Inc. 7 reasons matlab is the easiest and most productive environment for engineers and scientists. https://de.mathworks.com/products/matlab/why-matlab.html, 2021. urldate:30.08.2021.

[33] Mara K. Fuchs, Jörg Fuchte, and Jörn Biedermann. Ein mbse-ansatz für die auslegung von flugzeugkabinen am beispiel eines passenger supply channel. In *Deutscher Luft- und Raumfahrtkongress 2019*, 2019.

[34] Mara K. Fuchs, Christian Hesse, Jörn Biedermann, and Jörg Fuchte. A multidisciplinary design optimization of the passenger supply channel in the aircraft cabin. In *9th EASN International Conference on Innovation in Aviation & Space*, October 2019.

[35] Blender. Blender - about. https://www.blender.org/, 2021. urldate:30.08.2021.

[36] Fiete Rauscher, Jörn Biedermann, Alex Gindorf, Frank Meller, and Björn Nagel. Permanente geometrische digitalisierung der flugzeugkabine zur änderungsnachverfolgung. In *Deutscher Luft- und Raumfahrtkongress 2020*, 2020.

[37] Unity Technologies. Unity manual. https://docs.unity3d.com/Manual/index.html, 2021. urldate:30.08.2021.

[38] Sonja Eilmann, Frank Behrend, Raimo Hübner, and Erwin Weitlander. Interessengruppen/interessierte parteien. *Kompetenzbasiertes Projektmanagement*, 4:71, 2011.

[39] ISO/IEC. Iso/iec/ieee international standard - systems and software engineering – system life cycle processes. *ISO/IEC/IEEE 15288 First edition 2015-05-15*, pages 1–118, 2015.

[40] INCOSE. *INCOSE Systems Engineering Handbook: A Guide for System Life Cycle Processes and Activities*, 2021. https://www.incose.org/products-and-publications/se-handbook.

[41] Elyse Fosse. Model-based systems engineering (mbse) 101. In *Board Meeting*, pages 1–82, 2012.

[42] INCOSE - International Council on Systems Engineering. About incose. https://www.incose.org/about-incose, 2021.

[43] Charles Dickerson and Dimitri N. Mavris. *Architecture and principles of systems engineering*. CRC Press, 2016.

[44] Tim Weilkiens, Jesko G. Lamm, Stephan Roth, and Markus Walker. *Model-based System Architecture*. Wiley, Hoboken,NJ, 2016.

[45] Oliver Alt. *Modellbasierte Systementwicklung mit SysML*. Carl Hanser Verlag, München, 2012.

[46] Ann M. Garrison Darrin and William S. Devereux. The agile manifesto, design thinking and systems engineering. In *2017 Annual IEEE International Systems Conference (SysCon)*, pages 1–5. IEEE, 2017.

[47] Iris Graessler and Julian Hentze. The new v-model of vdi 2206 and its validation. *at-Automatisierungstechnik*, 68(5):312–324, 2020.

[48] Stephan Finkel, Sebastian Märkl, and Claudia Kessler. Zurück in die zukunft: Systems engineering! In *Deutscher Luft- und Raumfahrtkongress 2020*, 2020.

[49] Reinhard Haberfellner and Olivier De Weck. 10.1. 3 agile systems engineering versus agile systems engineering. In *INCOSE International Symposium*, volume 15, pages 1449–1465. Wiley Online Library, 2005.

[50] Mike Russell. Using mbse to enhance system design decision making. *Procedia Computer Science*, 8:188–193, 2012.

[51] Sanford Friedenthal, Alan Moore, and Rick Steiner. *A practical guide to SysML: the systems modeling language*. Morgan Kaufmann, 2014.

[52] INCOSE - International Council on Systems Engineering. *SYSTEMS ENGINEERING VISION 2020*, 2.03 edition, 2007.

[53] Herbert Stachowiak. *Allgemeine modelltheorie*. Springer, 1973.

[54] Stephen J. Topper and Nathaniel C. Horner. Model-based systems engineering in support of complex systems development. *Johns Hopkins APL technical digest*, 32(1), 2013.

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

[56] International Council on Systems Engineering (INCOSE). Systems engineering vision 2025. https://www.incose.org/products-and-publications/se-vision-2025, 2014. urldate:30.08.2021.

[57] John R. Palmer, Michael E. Crow, and Ronald S. Carson. Model-based systems engineering without sysml. In *Boeing presentation to National Defense Industrial Association Systems Engineering Conference, San Diego*, pages 24–27, 2011.

[58] Mohammad Chami and Jean-Michel Bruel. A survey on mbse adoption challenges. Wiley Interscience Publications, 2018.

[59] Object Management Group (OMG). About us. https://www.omg.org/about/index.htm, 2021. urldate:02.09.2021.

[60] Dominic Lehle. Quantitative safety analysis of sysml models. Master's thesis, University of Konstanz Department of Computer and Information Science, 2011.

[61] Thorsten Kiehl and Ralf God. An mbse-approach for using near field communication in the aircraft cabin. In *Workshop on Aircraft System Technology, AST 2013*, pages 1–10. von Estorff, Otto and Thielecke, Frank, 2013.

[62] Jon Holt and Simon A. Perry. *SysML for Systems Engineering*. The Institution of Engineering and Technology, London, United Kingdom, 2008.

[63] David R. Coelho. *The VHDL handbook*. Springer Science & Business Media, 2012.

[64] Systems engineering Trends. Was bringt uns sysml v2? interview mit uwe kaufmann. https://www.se-trends.de/sysml-v2/, October 2019. urldate:03.09.2021.

[65] Sanford Friedenthal. Requirements for the next generation systems modeling language (sysml® v2). *INSIGHT*, 21(1):21–25, 2018.

[66] Dot Net Tutorials. Object-oriented programming in java. https://dotnettutorials.net/lesson/object-oriented-programming-in-java/, 2021. urldate:03.09.2021.

[67] Sebastien Gerard. Papyrus user guide series about uml profiling, version 1.0. 0. https://www.eclipse.org/papyrus/resources/PapyrusUserGuideSeries_AboutUMLProfile_v1.0.0_d20120606.pdf, 2011. urldate:06.09.2021.

[68] Sparx Systems Ltd and SparxSystems Software GmbH. The token-concept for activity-diagrams. https://www.sparxsystems.eu/resources/project-development-with-uml-and-ea/activity-diagram/, 2021. urldate:06.09.2021.

[69] Alexander Blumör, Gerhard Pregitzer, and Martin Bothen. Werkzeuge für die entwicklung mechatronischer systeme mit methoden des mbse. *Tag des Systems Engineering*, pages 193–202, 2017.

[70] Phoenix Integration. Modelcenter enables model based systems engineering (mbse). https://www.phoenix-int.com/application/mbse-model-based-systems-engineering/, 2021. urldate:06.09.2021.

[71] Hannes Rosenow. Trade off bewertungsmethodik für tool-und methodenentscheidungen zur virtualisierung und modellbasierung in der entwicklung. Master's thesis, 2018.

[72] Henric Andersson. *Aircraft systems modeling: Model based systems engineering in avionics design and aircraft simulation*. PhD thesis, Linköping University Electronic Press, 2009.

[73] The Object Management Group (OMG). What is sysml? https://www.omgsysml.org/what-is-sysml.htm, 2021. urldate:08.09.2021.

[74] Open Services for Lifecycle Collaboration OSLC. About the oslc open project. https://open-services.net/about/, 2021. urldate:15.09.2021.

[75] IP Insider. Was ist http (hypertext transfer protocol)? https://www.ip-insider.de/was-ist-http-hypertext-transfer-protocol-a-691181/, 2018. urldate:16.09.2021.

[76] Digital Guide IONOS. Uri: The uniform resource identifier explained. https://www.ionos.com/digitalguide/websites/web-development/uniform-resource-identifier-uri/, 2020. urldate:16.09.2021.

[77] Mehrdad Saadatmand and Alessio Bucaioni. Oslc tool integration and systems engineering–the relationship between the two worlds. In *2014 40th EUROMICRO Conference on Software Engineering and Advanced Applications*, pages 93–101. IEEE, 2014.

[78] Uri Shani, Shmuela Jacobs, Niva Wengrowicz, and Dov Dori. Engaging ontologies to break mbse tools boundaries through semantic mediation. In *2016 conference on systems engineering research*, 2016.

[79] Yves Vanderperren and Wim Dehaene. From uml/sysml to matlab/simulink: current state and future perspectives. In *Proceedings of the Design Automation & Test in Europe Conference*, volume 1, pages 1–1. IEEE, 2006.

[80] Bassim Chabibi, Adil Anwar, Mahmoud Nassar, Labo Admir, and RI Center. Model integration approach from sysml to matlab/simulink. *Journal of Digital Information Management*, 16(6):289–307, 2018.

[81] Mohd A. A. Rahman and Makoto Mizukawa. Model-based development and simulation for robotic systems with sysml, simulink and simscape profiles. *International Journal of Advanced Robotic Systems*, 10(2):112, 2013.

[82] Thomas Johnson, Aleksandr Kerzhner, Christiaan J. J. Paredis, and Roger Burkhart. Integrating models and simulations of continuous dynamics into sysml. *Journal of Computing and Information Science in Engineering*, 12(1), 2012.

[83] James N. Martin. *Systems engineering guidebook: A process for developing systems and products*. CRC press, 2020.

[84] Object Management Group(OMG). Methodology and metrics. https://www.omgwiki.org/MBSE/doku.php?id=mbse:methodology, April 2018. urldate:15.09.2021.

[85] Jeff A. Estefan. Survey of model-based systems engineering (mbse) methodologies. *Incose MBSE Focus Group*, 25(8):1–12, 2007.

[86] Jutta Abulawi. A sysml-based approach to exploring innovative system ideas for aeronautical applications. In *31st Congress of the International Council of the Aeronautical Sciences (ICAS 9-14 September 2018)*, pages 817–826, 2018.

[87] Jesko G. Lamm and Tim Weilkiens. Functional architectures in sysml. *Proceedings of the TdSE*, 2010.

[88] Jesko G. Lamm and Tim Weilkiens. Method for deriving functional architectures from use cases. *Systems Engineering*, 17(2):225–236, 2014.

[89] FAS working group of Gesellschaft für Systems Engineering The German chapter of INCOSE. Functional architectures for systems: Fas plugins. http://fas-method.org/content/fas-plugins/, 2021. urldate:08.09.2021.

[90] Tim Weilkiens. *Systems Engineering mit SysML/UML: Anforderungen, Analyse, Architektur. Mit einem Geleitwort von Richard Mark Soley*. dpunkt. verlag, 2014.

[91] Tim Weilkiens, Axel Scheithauer, Marco Di Maio, and Niklas Klusmann. Evaluating and comparing mbse methodologies for practitioners. In *2016 IEEE International Symposium on Systems Engineering (ISSE)*, pages 1–8. IEEE, 2016.

[92] European Aviation Safety Agency (EASA). *Certification Specifications for Large Aeroplanes CS-25, Amendment 3*, 2007 edition, 2018.

[93] Wiegmann Marc. Elektronische kabinensysteme. *Lecture script*, Department of Automotive and Aeronautical Engineering Hamburg University of Applied Sciences, 25.08.2021.

[94] Mara K. Fuchs. Ein mbse-ansatz für die auslegung von flugzeugkabinen am beispiel eines passenger service channel. Master's thesis, Technische Universität Hamburg, December 2018. Supervisor: Dr.-Ing. Jörn Biedermann (DLR-SL-KNS).

[95] Dieter Scholz. Aircraft systems. *Lecture script*, Department of Automotive and Aeronautical Engineering Hamburg University of Applied Sciences, 11.09.2021.

[96] Air Transport Association. Ata ispec 2200. http://www.ataspec.com/index.php?main_page=index&cPath=6, 2000. urldate:11.10.2021.

[97] Stefan Burger, Oliver Hummel, and Matthias Heinisch. Airbus cabin software. *IEEE software*, 30(1):21–25, 2013.

[98] SAE International. *Guidelines for Development of Civil Aircraft and Systems ARP4754A*, 2010.

[99] Frank M. Leipold. *Wireless UWB aircraft cabin communication system*. PhD thesis, Technische Universität München, 2011.

[100] Marc Riedlinger, Oliver Marquardt, Reza Ahmadi, and Reinhard Reichel. An adaptive self-managing platform for cabin management systems. *CEAS Aeronautical Journal*, 7(3):483–498, 2016.

[101] Torben Schröter, Torsten Benstem, and Detlef Schulz. Aircraft availability and the optimized electrical system. In *3rd International Workshop on Aircraft System Technologies*, pages 3–12, 2011.

[102] Brice Nya, Johannes Brombach, Torben Schröter, and Detlef Schulz. Weight evaluation of cabin power architecture on smaller civil aircraft. In *3rd International Workshop on Aircraft System Technologies AST 2011*, 2011.

[103] Hartmut Hintze, Andreas Tolksdorf, and Ralf God. Cabin core system-a next generation platform for combined electrical power and data services. In *Tagungsband zum 3rd International Workshop on Aircraft System Technologies-AST*, pages 221–231, 2011.

[104] Rajendra Prasad. *Fundamentals of electrical engineering*. PHI Learning Pvt. Ltd., 2014.

[105] Incorporated RTCA. Rtca do-160f environmental conditions and test procedures for airborne equipment. https://do160.org/rtca-do-160g/, 2007. urldate:30.08.2021.

[106] Judita Andrianova No Magic Inc. No magic documentation: Proxy port. https://docs.nomagic.com/display/SYSMLP184/Proxy+Port, 2016. urldate:04.10.2021.

[107] Nils Fischer. *Automated Validation of Minimum Risk Model-Based System Designs of Complex Avionics Systems*. PhD thesis, Fakultaet fur Informatik und Automatisierung der Technischen Universitaet Ilmenau, 2016.

[108] ECMA international. The json data interchange syntax - 2nd edition december 2017. https://www.ecma-international.org/publications-and-standards/standards/ecma-404/. urldate:11.10.2021.

[109] SAE Industry Technologies Consortia (SAE ITC). Arinc standards. https://www.aviation-ia.com/content/arinc-standards, 2021. urldate:02.11.2021.

[110] Json Parser. Json parser. https://jsonparser.org/. urldate:11.10.2021.

[111] No Magic Inc. Alh apis. https://docs.nomagic.com/display/CST2021x/ALH+APIs. urldate:11.10.2021.

[112] No Magic Inc. Integration with matlab. https://docs.nomagic.com/display/CST185/Integration+with+MATLAB. urldate:14.10.2021.

[113] Kalyanmoy Deb. Multi-objective optimization. In *Search methodologies*, pages 403–449. Springer, 2014.

[114] SeatGuru by Tripadvisor. Lufthansa seat maps airbus a320 layout 1. https://www.seatguru.com/airlines/Lufthansa/Lufthansa_Airbus_A320-200_NEK.php, 2021. urldate:02.11.2021.

[115] Johannes Brombach. Methoden zur gewichtsreduktion in elektrischen flugzeugkabinennetzen. 2014.

[116] Wendy Evans and David Baker. *Trends, discovery, and people in the digital age*. Elsevier, 2013.

[117] Joaquim R. R. A. Martins and Andrew B. Lambe. Multidisciplinary design optimization: a survey of architectures. *AIAA journal*, 51(9):2049–2075, 2013.

[118] Joaquim R. R. A. Martins and Andrew Ning. *Engineering design optimization*. Cambridge University Press, 2021.

[119] The OpenMDAO Development Team. Openmdao solvers. https://openmdao.org/newdocs/versions/latest/theory_manual/solver_api.html, 2021. urldate:12.09.2021.

[120] Air and Space Magazine. How things work: Lightning protection. https://www.airspacemag.com/flight-today/how-things-work-lightning-protection-161993347/, 2011. urldate:13.11.2021.

A Appendix

All of the figures in this appendix have been referenced to in the upper parts of this work. These are presented here to provide more model details to help improve the understanding of this work. A brief introduction to the figures is given as follows.

The Structure of the PSC package diagram showing the hierarchical structure:

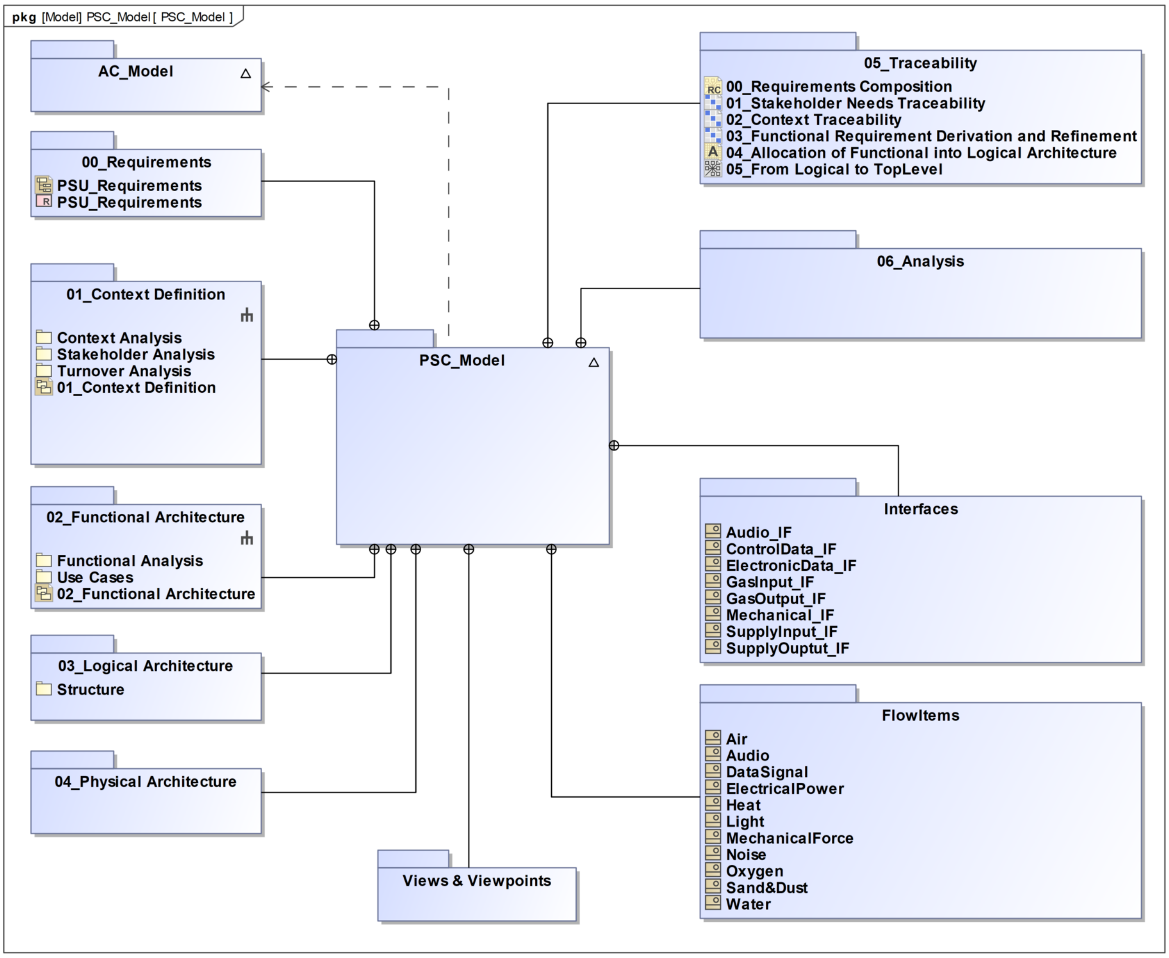
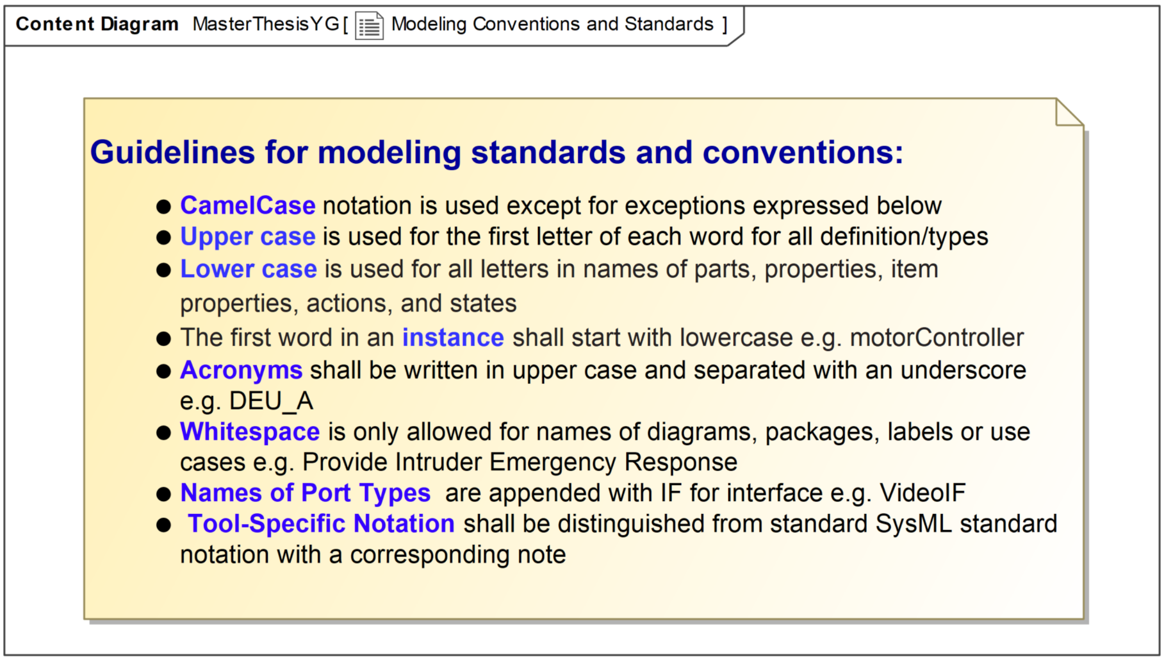


Figure A.1: Internal structure of PSC Model

The modeling guidelines defined and used in the model:



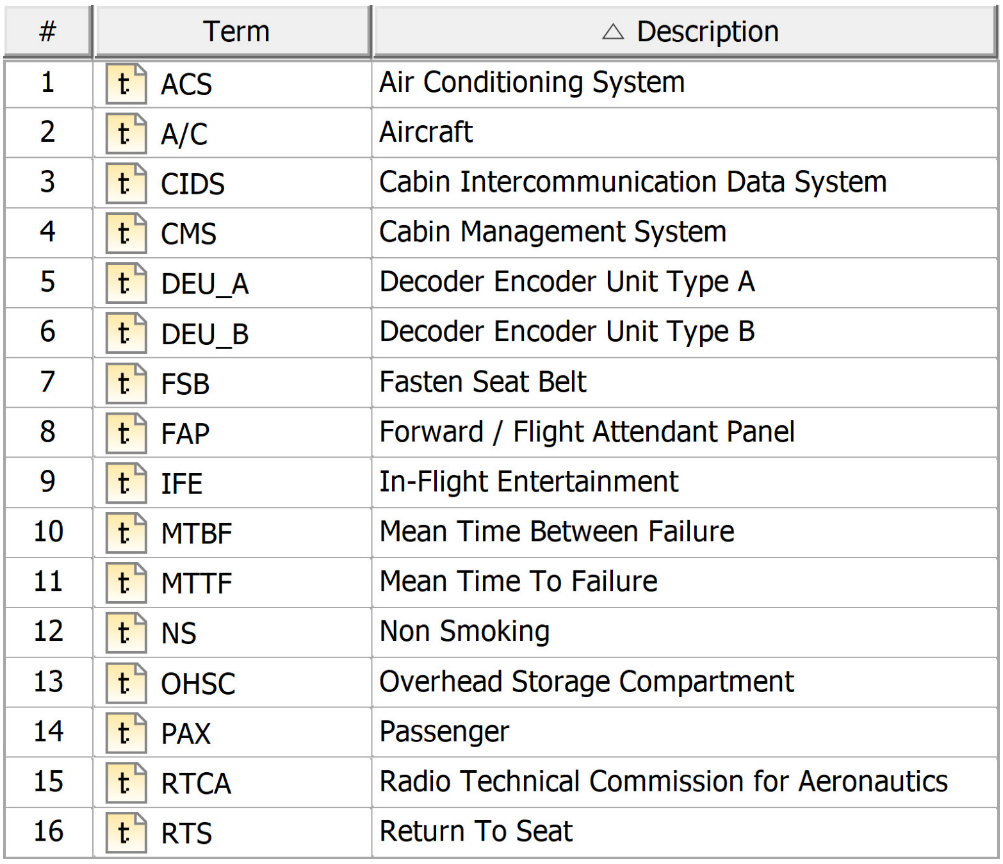
Content Diagram MasterThesisYG [Modeling Conventions and Standards]

Guidelines for modeling standards and conventions:

- **CamelCase** notation is used except for exceptions expressed below
- **Upper case** is used for the first letter of each word for all definition/types
- **Lower case** is used for all letters in names of parts, properties, item properties, actions, and states
- The first word in an **instance** shall start with lowercase e.g. motorController
- **Acronyms** shall be written in upper case and separated with an underscore e.g. DEU_A
- **Whitespace** is only allowed for names of diagrams, packages, labels or use cases e.g. Provide Intruder Emergency Response
- **Names of Port Types** are appended with IF for interface e.g. VideoIF
- **Tool-Specific Notation** shall be distinguished from standard SysML standard notation with a corresponding note

Figure A.2: Convention and standards for model notation

The model glossary that includes the abbreviations used in the model:



#	Term	△ Description
1	ACS	Air Conditioning System
2	A/C	Aircraft
3	CIDS	Cabin Intercommunication Data System
4	CMS	Cabin Management System
5	DEU_A	Decoder Encoder Unit Type A
6	DEU_B	Decoder Encoder Unit Type B
7	FSB	Fasten Seat Belt
8	FAP	Forward / Flight Attendant Panel
9	IFE	In-Flight Entertainment
10	MTBF	Mean Time Between Failure
11	MTTF	Mean Time To Failure
12	NS	Non Smoking
13	OHSC	Overhead Storage Compartment
14	PAX	Passenger
15	RTCA	Radio Technical Commission for Aeronautics
16	RTS	Return To Seat

Figure A.3: Specific glossary of aircraft systems

The following pictures illustrate the model ontology defined in this work. They specify it for stakeholder consultations, analysis, use cases, and requirement modeling:

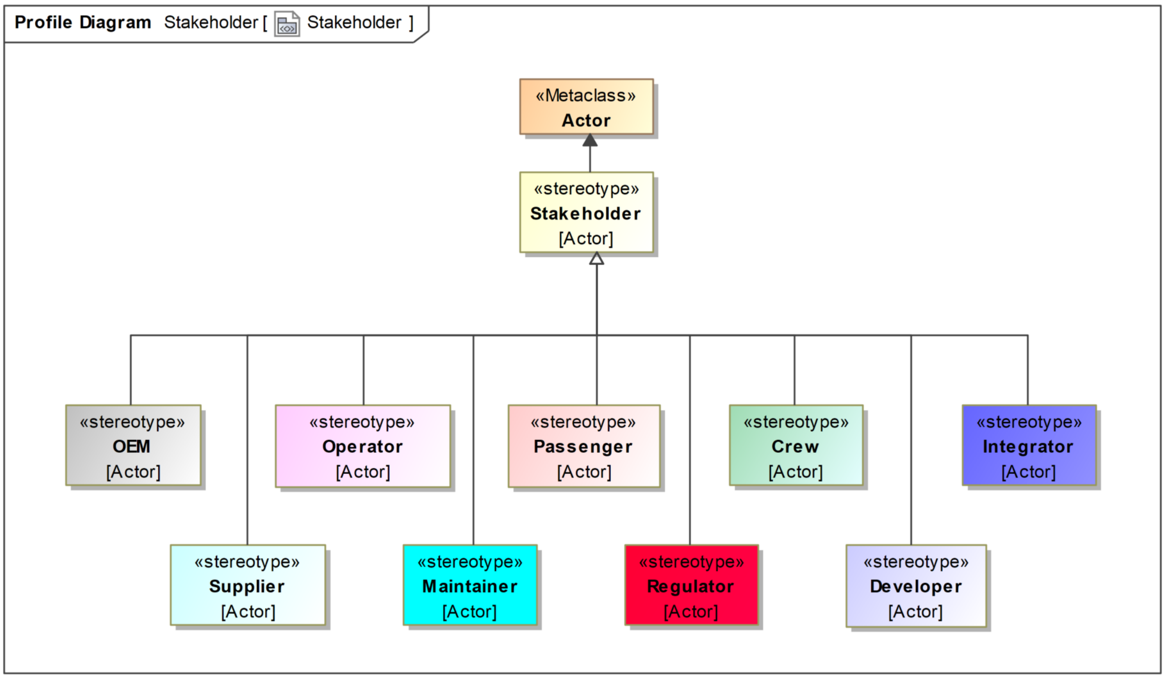

Profile Diagram Stakeholder [Stakeholder]
«Metaclass» Actor
«stereotype» Stakeholder [Actor]
«stereotype» OEM [Actor]
«stereotype» Operator [Actor]
«stereotype» Passenger [Actor]
«stereotype» Crew [Actor]
«stereotype» Integrator [Actor]
«stereotype» Supplier [Actor]
«stereotype» Maintainer [Actor]
«stereotype» Regulator [Actor]
«stereotype» Developer [Actor]

Figure A.4: Definition of stakeholder ontology

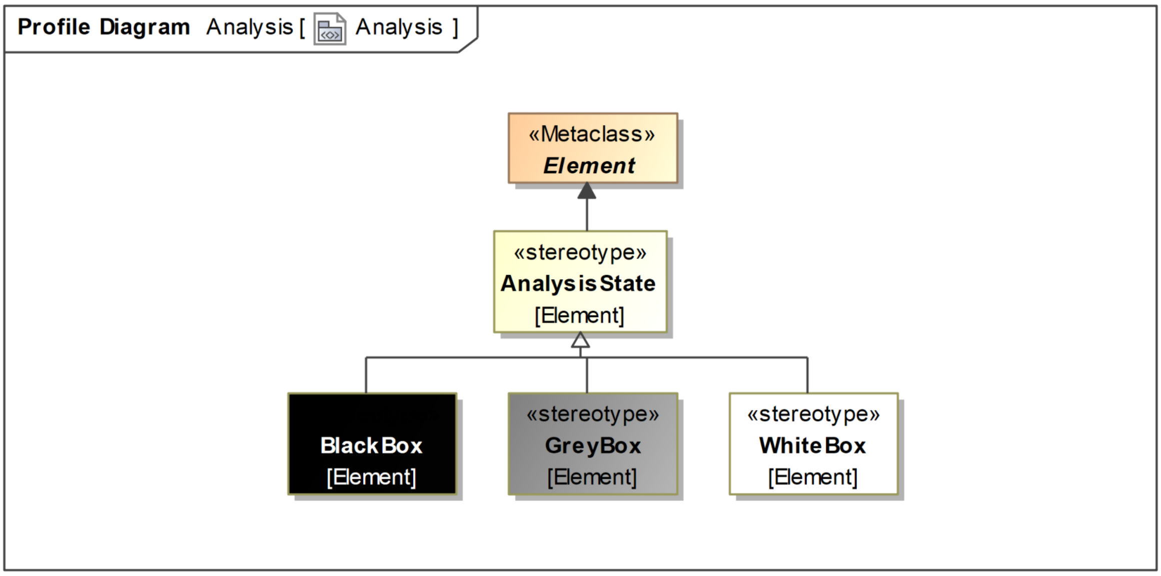

Profile Diagram Analysis [Analysis]
«Metaclass» Element
«stereotype» AnalysisState [Element]
BlackBox [Element]
«stereotype» GreyBox [Element]
«stereotype» WhiteBox [Element]

Figure A.5: Definition of analysis ontology

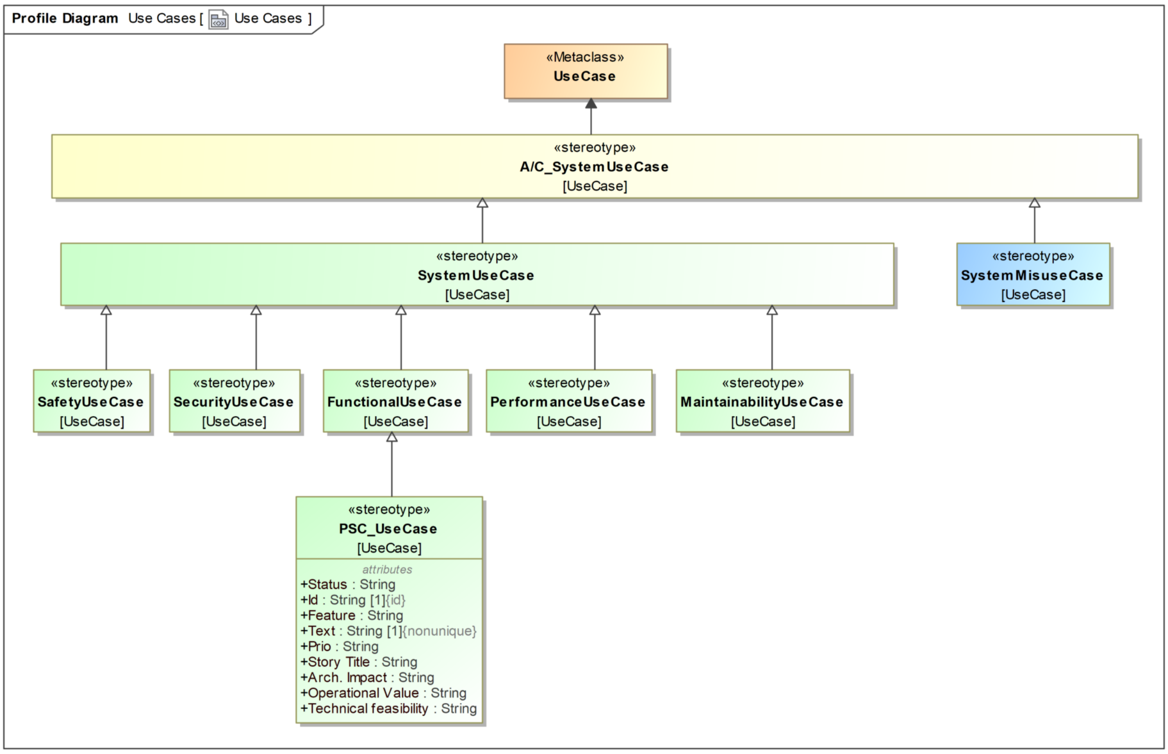


Figure A.6: Definition of use case ontology

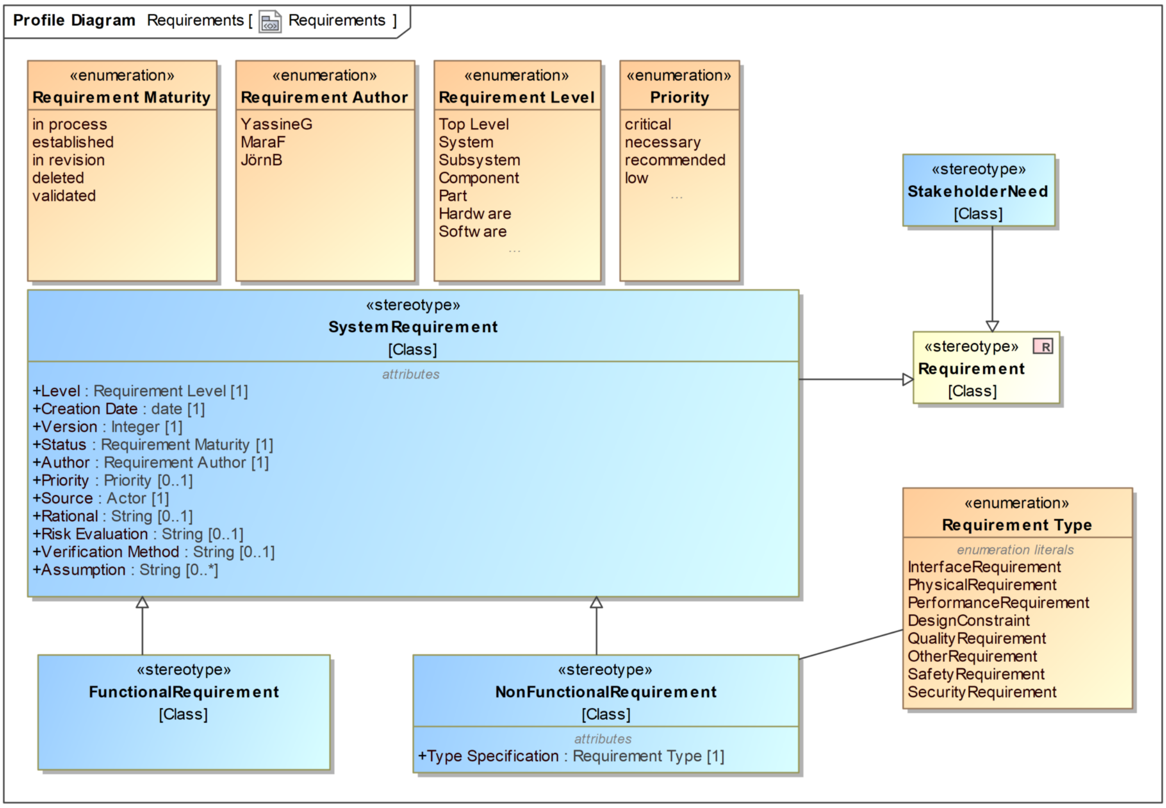


Figure A.7: Definition of requirements ontology

The use cases identified during this work are bundled into a use case diagram that depicts their relationships to the external actors:

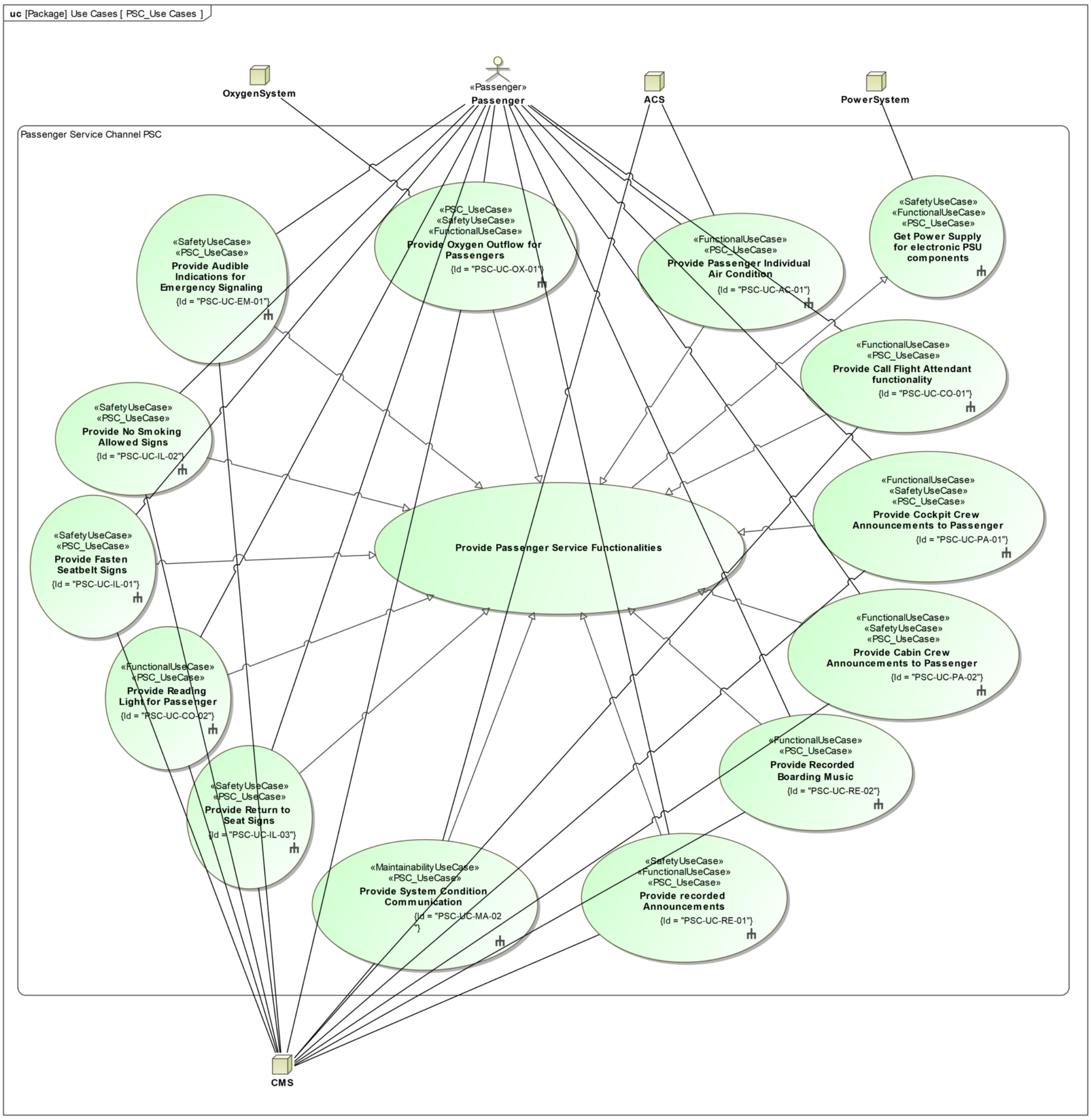
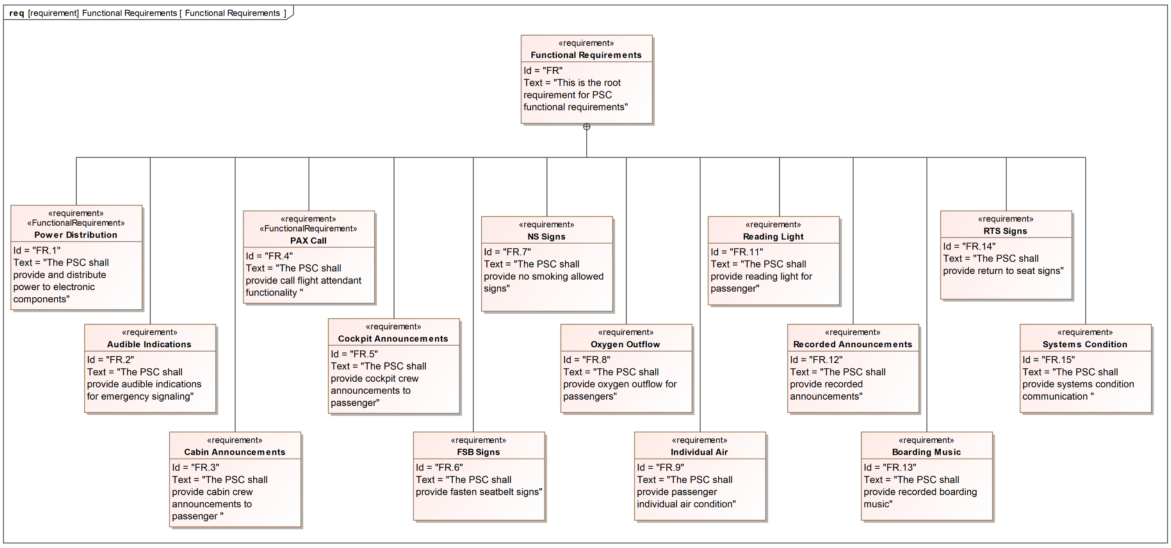


Figure A.8: Use case diagram with identified and derived PSC use cases

The functional requirements resulting from the PSC's use cases are depicted in the requirement diagram below. The following traceability matrix shows how the relationships between these requirements and the use cases:



req [requirement] Functional Requirements [Functional Requirements]

«requirement»
Functional Requirements
Id = "FR"
Text = "This is the root requirement for PSC functional requirements"

«requirement»
«FunctionalRequirement»
Power Distribution
Id = "FR.1"
Text = "The PSC shall provide and distribute power to electronic components"

«requirement»
Audible Indications
Id = "FR.2"
Text = "The PSC shall provide audible indications for emergency signaling"

«requirement»
Cabin Announcements
Id = "FR.3"
Text = "The PSC shall provide cabin crew announcements to passenger "

«requirement»
«FunctionalRequirement»
PAX Call
Id = "FR.4"
Text = "The PSC shall provide call flight attendant functionality "

«requirement»
Cockpit Announcements
Id = "FR.5"
Text = "The PSC shall provide cockpit crew announcements to passenger"

«requirement»
FSB Signs
Id = "FR.6"
Text = "The PSC shall provide fasten seatbelt signs"

«requirement»
NS Signs
Id = "FR.7"
Text = "The PSC shall provide no smoking allowed signs"

«requirement»
Oxygen Outflow
Id = "FR.8"
Text = "The PSC shall provide oxygen outflow for passengers"

«requirement»
Individual Air
Id = "FR.9"
Text = "The PSC shall provide passenger individual air condition"

«requirement»
Reading Light
Id = "FR.11"
Text = "The PSC shall provide reading light for passenger"

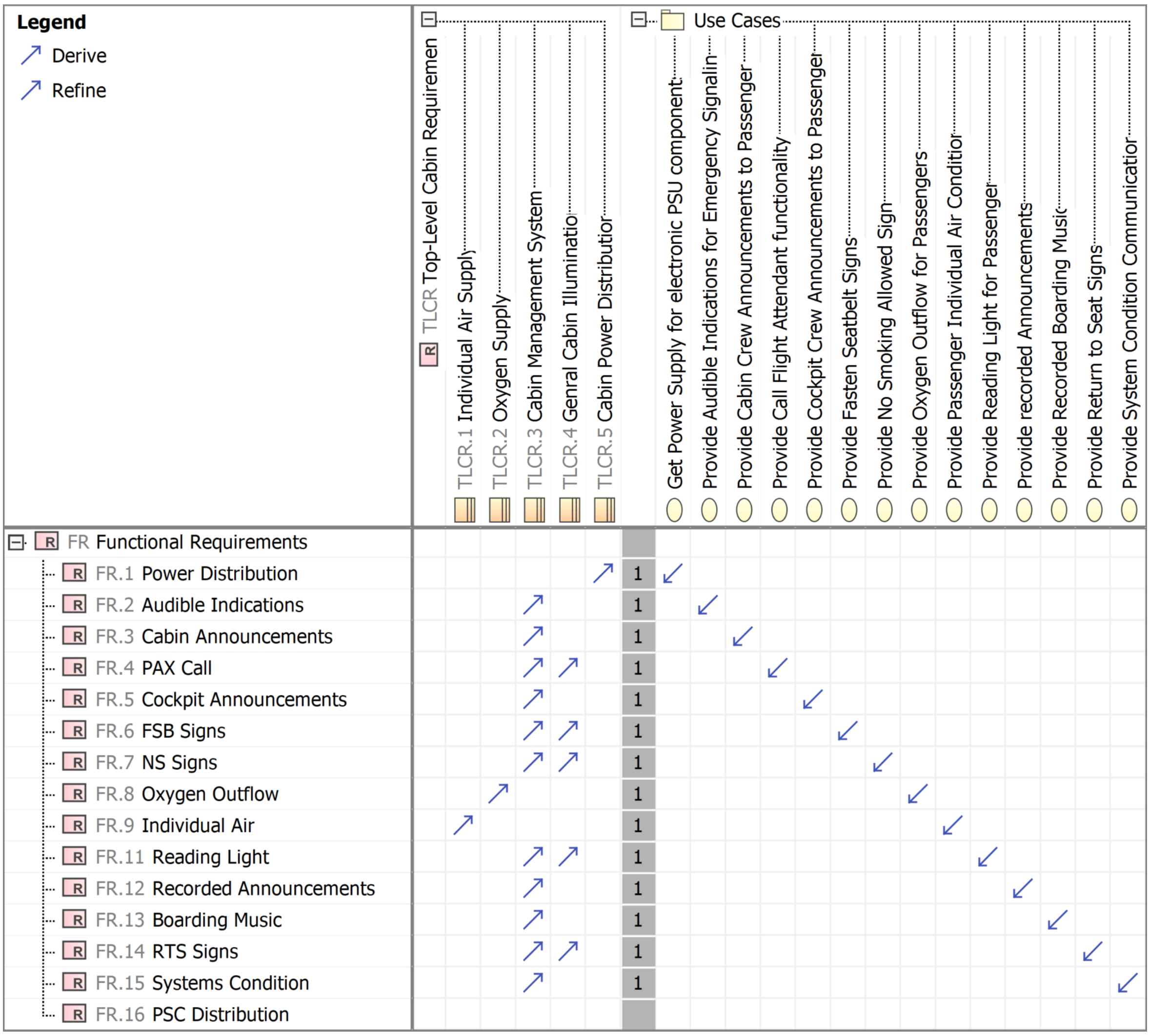
«requirement»
Recorded Announcements
Id = "FR.12"
Text = "The PSC shall provide recorded announcements"

«requirement»
Boarding Music
Id = "FR.13"
Text = "The PSC shall provide recorded boarding music"

«requirement»
RTS Signs
Id = "FR.14"
Text = "The PSC shall provide return to seat signs"

«requirement»
Systems Condition
Id = "FR.15"
Text = "The PSC shall provide systems condition communication "

Figure A.9: PSC functional requirements derived from PSC's use cases

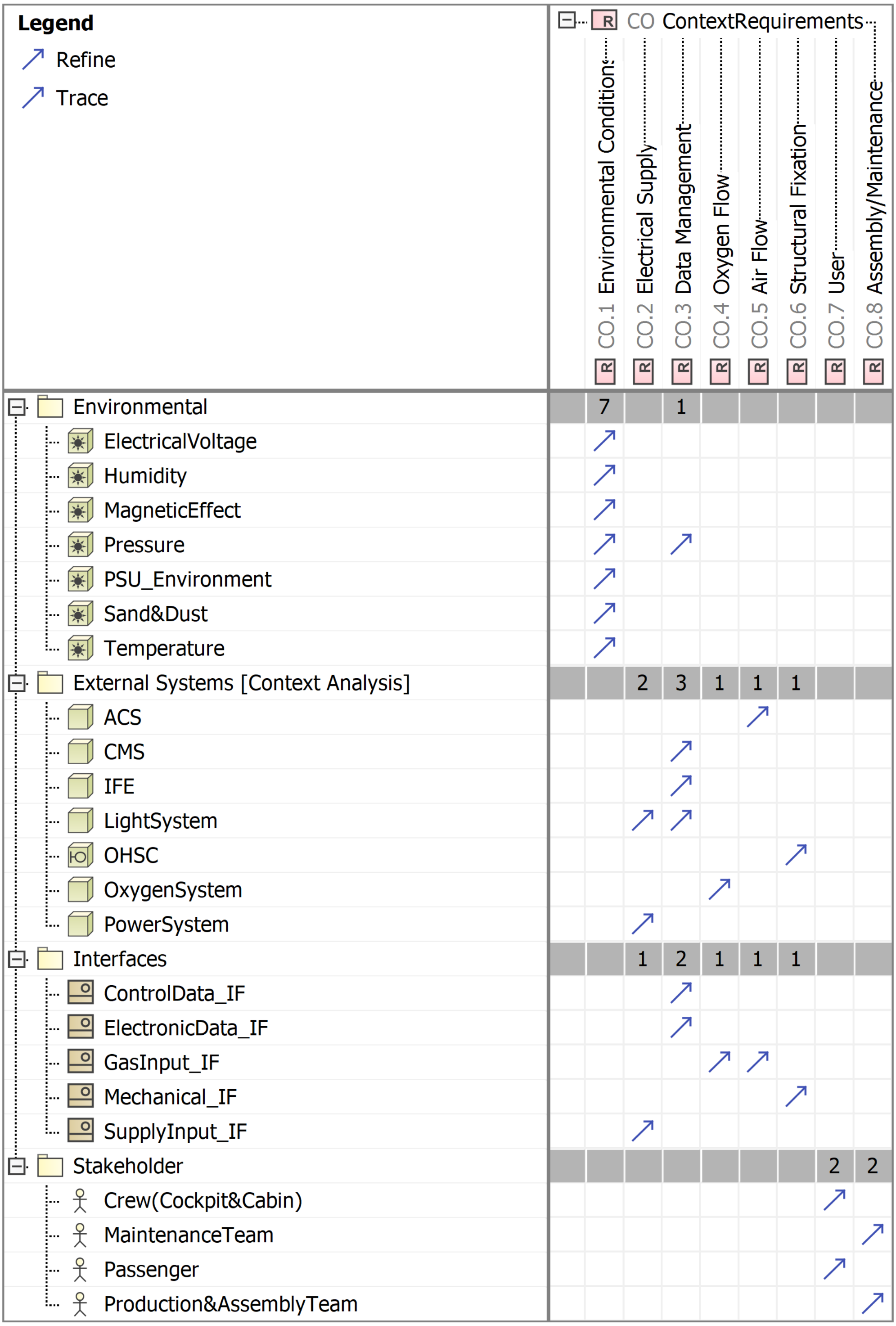


Legend
↗ Derive
↗ Refine

	TLCR Top-Level Cabin Requiremen	TLCR.1 Individual Air Supply	TLCR.2 Oxygen Supply	TLCR.3 Cabin Management System	TLCR.4 Genral Cabin Illuminatio	TLCR.5 Cabin Power Distributior		Get Power Supply for electronic PSU component	Provide Audible Indications for Emergency Signalin	Provide Cabin Crew Announcements to Passenger	Provide Call Flight Attendant functionality	Provide Cockpit Crew Announcements to Passenger	Provide Fasten Seatbelt Signs	Provide No Smoking Allowed Sign	Provide Oxygen Outflow for Passengers	Provide Passenger Individual Air Conditior	Provide Reading Light for Passenger	Provide recorded Announcements	Provide Recorded Boarding Music	Provide Return to Seat Signs	Provide System Condition Communicatior
FR Functional Requirements																					
FR.1 Power Distribution						↗	1	↙													
FR.2 Audible Indications				↗			1		↙												
FR.3 Cabin Announcements				↗			1			↙											
FR.4 PAX Call				↗	↗		1				↙										
FR.5 Cockpit Announcements				↗			1					↙									
FR.6 FSB Signs				↗	↗		1						↙								
FR.7 NS Signs				↗	↗		1							↙							
FR.8 Oxygen Outflow			↗				1								↙						
FR.9 Individual Air		↗					1									↙					
FR.11 Reading Light				↗	↗		1										↙				
FR.12 Recorded Announcements				↗			1											↙			
FR.13 Boarding Music				↗			1												↙		
FR.14 RTS Signs				↗	↗		1													↙	
FR.15 Systems Condition				↗			1														↙
FR.16 PSC Distribution																					

Figure A.10: Derivation of functional requirements and refinement through use cases

The two traceability matrix below depicts the relationships between stakeholders and context elements, as well as the established context requirement and stakeholders' needs:

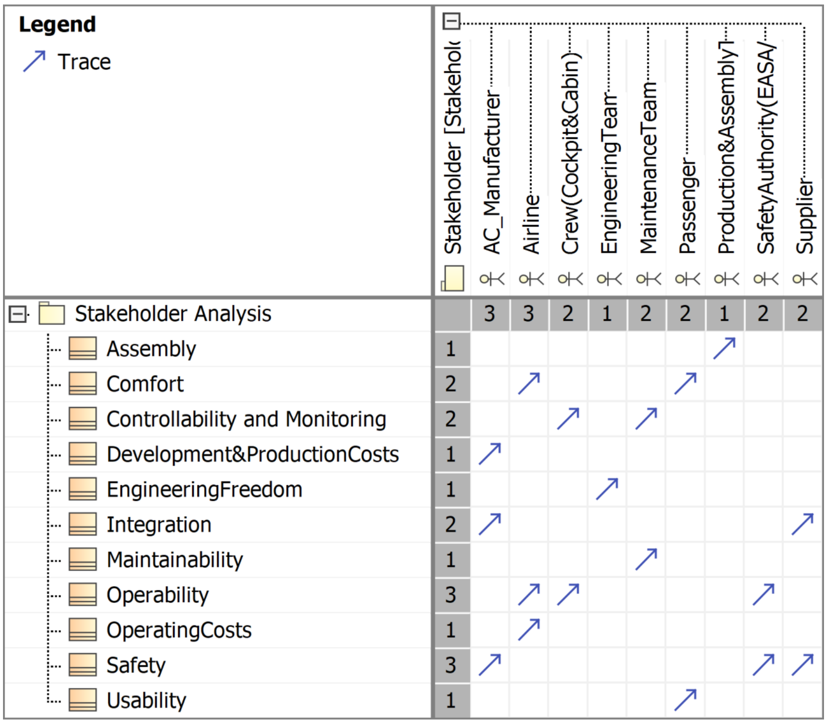


Legend
↗ Refine
↗ Trace

CO ContextRequirements

	CO.1 Environmental Conditions	CO.2 Electrical Supply	CO.3 Data Management	CO.4 Oxygen Flow	CO.5 Air Flow	CO.6 Structural Fixation	CO.7 User	CO.8 Assembly/Maintenance
Environmental	7		1					
ElectricalVoltage	↗							
Humidity	↗							
MagneticEffect	↗							
Pressure	↗		↗					
PSU_Environment	↗							
Sand&Dust	↗							
Temperature	↗							
External Systems [Context Analysis]		2	3	1	1	1		
ACS					↗			
CMS			↗					
IFE			↗					
LightSystem		↗	↗					
OHSC						↗		
OxygenSystem				↗				
PowerSystem		↗						
Interfaces		1	2	1	1	1		
ControlData_IF			↗					
ElectronicData_IF			↗					
GasInput_IF				↗	↗			
Mechanical_IF						↗		
SupplyInput_IF		↗						
Stakeholder							2	2
Crew(Cockpit&Cabin)							↗	
MaintenanceTeam								↗
Passenger							↗	
Production&AssemblyTeam								↗

Figure A.11: Traceability and refinement of context requirements



Legend

↗ Trace

Stakeholder Analysis		AC_Manufacturer	Airline	Crew(Cockpit&Cabin)	EngineeringTear	MaintenanceTeam	Passenger	Production&Assembly	SafetyAuthority(EASA/	Supplier
Stakeholder [Stakeholc		3	3	2	1	2	2	1	2	2
Assembly	1							↗		
Comfort	2		↗				↗			
Controllability and Monitoring	2			↗		↗				
Development&ProductionCosts	1	↗								
EngineeringFreedom	1				↗					
Integration	2	↗								↗
Maintainability	1					↗				
Operability	3		↗	↗					↗	
OperatingCosts	1		↗							
Safety	3	↗							↗	↗
Usability	1						↗			

Figure A.12: Traceability of needs to the corresponding stakeholders

The first parametric diagram demonstrates how the electrical parameters in the system model are bound with the values in matlab. The second diagram enables for the evaluation of electrical initial parameters for a two-galley configuration. It adjusts power values and transmits them to the Matlab model.

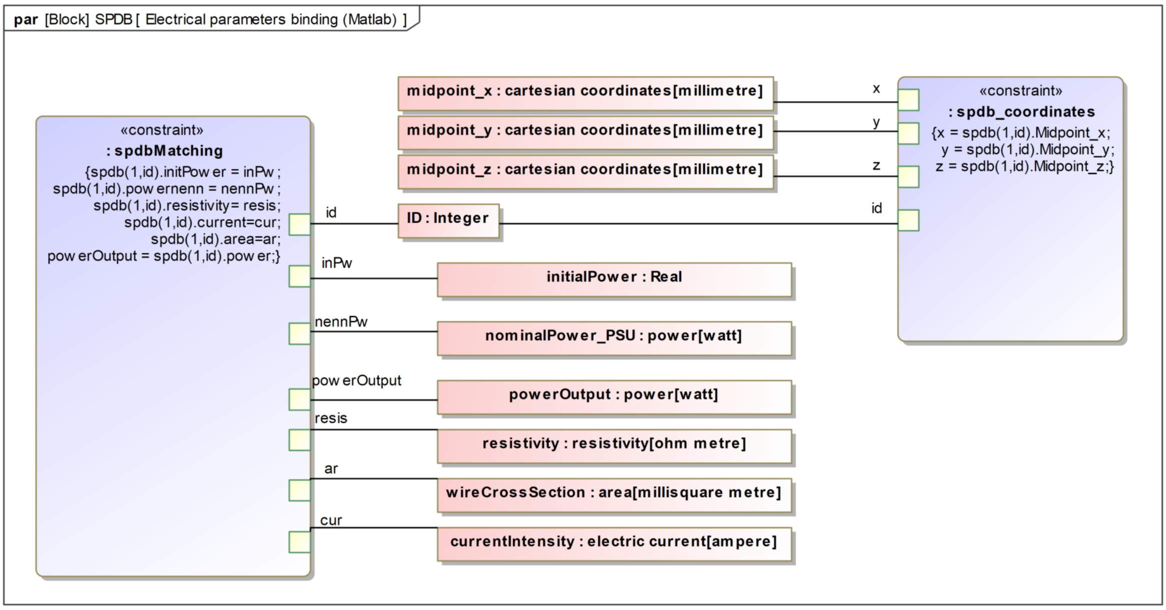
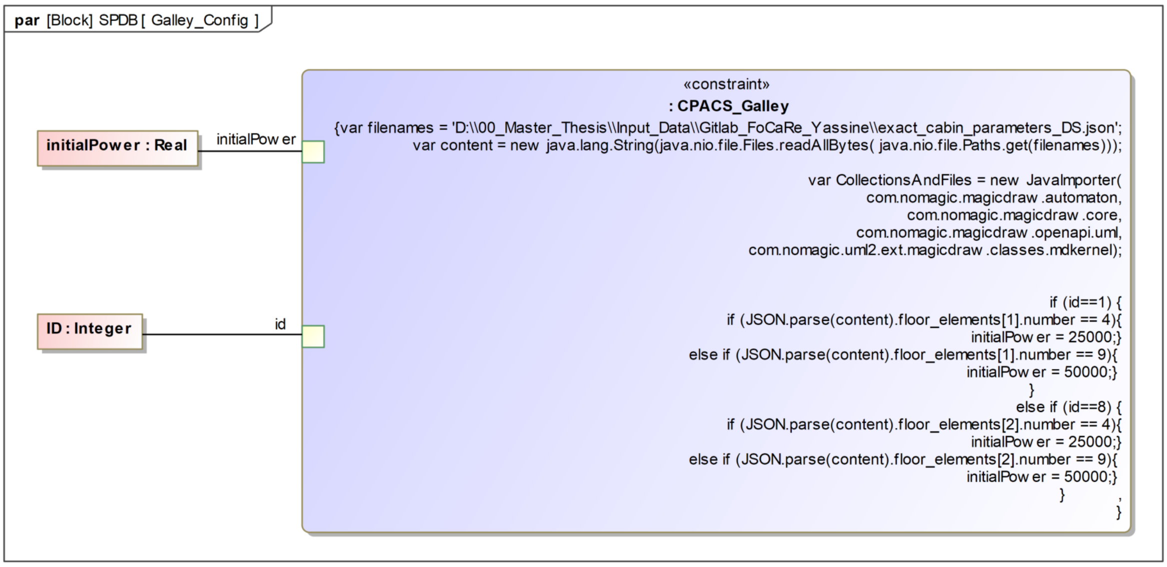

par [Block] SPDB [Electrical parameters binding (Matlab)]
«constraint»
: spdbMatching
{spdb(1,id).initPow er = inPw ;
spdb(1,id).pow ernenn = nennPw ;
spdb(1,id).resistivity= resis;
spdb(1,id).current=cur;
spdb(1,id).area=ar;
pow erOutput = spdb(1,id).pow er;}
id
inPw
nennPw
pow erOutput
resis
ar
cur
midpoint_x : cartesian coordinates[millimetre]
midpoint_y : cartesian coordinates[millimetre]
midpoint_z : cartesian coordinates[millimetre]
ID : Integer
initialPower : Real
nominalPower_PSU : power[watt]
powerOutput : power[watt]
resistivity : resistivity[ohm metre]
wireCrossSection : area[millisquare metre]
currentIntensity : electric current[ampere]
x
y
z
id
«constraint»
: spdb_coordinates
{x = spdb(1,id).Midpoint_x;
y = spdb(1,id).Midpoint_y;
z = spdb(1,id).Midpoint_z;}

Figure A.13: Configuration parameters binding with Matlab



```
{var filenames = 'D:\\00_Master_Thesis\\Input_Data\\Gitlab_FoCaRe_Yassine\\exact_cabin_parameters_DS.json';
var content = new java.lang.String(java.nio.file.Files.readAllBytes( java.nio.file.Paths.get(filenames)));

var CollectionsAndFiles = new JavaImporter(
com.nomagic.magicdraw.automaton,
com.nomagic.magicdraw.core,
com.nomagic.magicdraw.openapi.uml,
com.nomagic.uml2.ext.magicdraw.classes.mdkernel);


if (id==1) {
if (JSON.parse(content).floor_elements[1].number == 4){
initialPower = 25000;}
else if (JSON.parse(content).floor_elements[1].number == 9){
initialPower = 50000;}
}
else if (id==8) {
if (JSON.parse(content).floor_elements[2].number == 4){
initialPower = 25000;}
else if (JSON.parse(content).floor_elements[2].number == 9){
initialPower = 50000;}
}       ,
}
```

Figure A.14: Configuration evaluation parametric diagram for Galleys

List of Code

Code A.1: Javascript source code for the opaque behavior "importCPACS"

```javascript
var filenames = ’D:\\00_Master_Thesis\\Input_Data\\Gitlab_FoCaRe_Yassine\\exact_cabin_parameters_DS.json’;
var content = new java.lang.String(java.nio.file.Files.readAllBytes( java.nio.file.Paths.get(filenames)));

var CollectionsAndFiles = new JavaImporter(
    com.nomagic.magicdraw.automaton,
    com.nomagic.magicdraw.core,
    com.nomagic.magicdraw.openapi.uml,
    com.nomagic.uml2.ext.magicdraw.classes.mdkernel);

firstclass = ALH.createObject("CabinClass")
ALH.setValue(firstclass,"tag",JSON.parse(content).classes[0].tag)
ALH.setValue(firstclass,"n_pax",JSON.parse(content).classes[0].n_pax)
ALH.setValue(firstclass,"seat_pitch",JSON.parse(content).classes[0].seat_pitch)
ALH.setValue(firstclass,"n_rows",JSON.parse(content).classes[0].n_rows)
ALH.setValue(firstclass,"pax_RH",JSON.parse(content).classes[0].pax_distribution[0])
ALH.setValue(firstclass,"pax_LH",JSON.parse(content).classes[0].pax_distribution[1])

businessclass = ALH.createObject("CabinClass")
ALH.setValue(businessclass,"tag",JSON.parse(content).classes[1].tag)
ALH.setValue(businessclass,"n_pax",JSON.parse(content).classes[1].n_pax)
ALH.setValue(businessclass,"seat_pitch",JSON.parse(content).classes[1].seat_pitch)
ALH.setValue(businessclass,"n_rows",JSON.parse(content).classes[1].n_rows)
ALH.setValue(businessclass,"pax_RH",JSON.parse(content).classes[1].pax_distribution[0])
ALH.setValue(businessclass,"pax_LH",JSON.parse(content).classes[1].pax_distribution[1])

economyclass = ALH.createObject("CabinClass")
ALH.setValue(economyclass,"tag",JSON.parse(content).classes[2].tag)
ALH.setValue(economyclass,"n_pax",JSON.parse(content).classes[2].n_pax)
ALH.setValue(economyclass,"seat_pitch",JSON.parse(content).classes[2].seat_pitch)
ALH.setValue(economyclass,"n_rows",JSON.parse(content).classes[2].n_rows)
ALH.setValue(economyclass,"pax_RH",JSON.parse(content).classes[2].pax_distribution[0])
ALH.setValue(economyclass,"pax_LH",JSON.parse(content).classes[2].pax_distribution[1])
```

Code A.2: Javascript source code for the opaque behavior "initializeArchitecture"

```
temp = 0;
for (var j= 0; j<3; j++)
{
obj =cc.get(j);
temp= temp + ALH.getValue(obj,"psc_LH_num") + ALH.getValue(obj,"psc_RH_num") ;
id= ALH.getValue(obj,"psc_id");
limit_LH= ALH.getValue(obj,"psc_LH_num");
limit_RH= ALH.getValue(obj,"psc_RH_num");
pax_LH= ALH.getValue(obj,"pax_LH");
pax_RH= ALH.getValue(obj,"pax_RH");


for (var i=1; i<=limit_LH;i++)
{
 psc_object = ALH.createObject("PSC");
 var id_psc= id + '-LH-' + i;
 ALH.setValue(psc_object,"psc_ID",id_psc);
 for (var k=1;k <= pax_LH ;k++)
 {psc_rl=ALH.createObject("ReadingLight");
  psc_oxm=ALH.createObject("OxygenModule");
  psc_au=ALH.createObject("IndividualAirOutlet");
  ALH.addValue(psc_object,"read",psc_rl);
  ALH.addValue(psc_object,"oxm",psc_oxm);
  ALH.addValue(psc_object,"au",psc_au);
 }
 ALH.addValue(obj,"cabinModule",psc_object)
}

for (var i=1; i<=limit_RH;i++)
{
 psc_object = ALH.createObject("PSC");
 var id_psc= id + '-RH-' + i;
 ALH.setValue(psc_object,"psc_ID",id_psc);
  for (var k=1;k <= pax_LH ;k++)
 {psc_rl=ALH.createObject("ReadingLight");
  psc_oxm=ALH.createObject("OxygenModule");
  psc_au=ALH.createObject("IndividualAirOutlet");
  ALH.addValue(psc_object,"read",psc_rl);
  ALH.addValue(psc_object,"oxm",psc_oxm);
  ALH.addValue(psc_object,"au",psc_au);
 }
 ALH.addValue(obj,"cabinModule",psc_object);
}
}
ALH.setValue(self,"PSC_NUM_Cabin",temp);
```

Code A.3: Matlab source code for "PSC" object

```matlab
classdef Psu < Component & geometryObject
    %PSU will be defined
    properties
        rLight readLight;
        oxygenMask oxygenMask;
        airOutlet indivAir;

    end

    methods

        function obj = initPsu(obj, Length,Width,Height,Midpointx,
           Midpointy,Midpointz,Number,Spec)
                obj.Height = Height;
                obj.Width = Width;
                obj.Length = Length;
                obj.Color = [0, 0, 0];
                obj.Tag = 'electricalSystem';
                obj.Spec = Spec;
                obj.Number = Number;
                obj.Midpoint_x = Midpointx;
                obj.Midpoint_y = Midpointy;
                obj.Midpoint_z = Midpointz;
                obj.Ata = '25-25-00';
                obj.Material = 'PVC';
                obj.Connections = {'Psc',1};
                obj.Name = strcat('Psu',num2str(Number),Spec);
        end

    end
end
```

Code A.4: Javascript source code for the opaque behavior "detectPSC"

```
temp_id= ALH.getValue(obj,"psc_ID");
ac_class= temp_id.slice(4,6);
out2 = temp_id.slice(10);
side = temp_id.slice(7,9);

if (ac_class == "FC")
   {out1=1;}

else if (ac_class == "BC")
   {out1=2;}

else if (ac_class == "YC")
   {out1=3;}

if (side =="RH")
   {out3 = 1;}

else if (side =="LH")
   {out3=2;}
```

Code A.5: Matlab source code for assessment and optimization algorithm

```
%% Configure SPDB
spdb = initSPDB(8,ceilings);

%% Power Calculation and Optimization
[input]= generateInput(params.cabin.length); %Coordinates for spdb areas
result=zeros(length(input),5);
power=zeros(length(input),8);
for i=1:length(input)
    X = [0,input(i,1),input(i,2),input(i,3);input(i,1),input(i,2),input(i,3),params.cabin.length];
    X= X + params.cabin.xStart;
    [wire, wireIndex, powerDistribution,spdbpower] = startPowerCalculation (psc,X,spdb);
    power(i,:)= spdbpower;
    result(i,1)= wireIndex;
    result(i,2)= powerDistribution;
    result(i,3)= X(1,2);
    result(i,4)= X(1,3);
    result(i,5)= X(1,4);
end

%% Nomalize inputs
f1= normalize(result(:,1));
f2= normalize(result(:,2));
%% Plot pareto front
figure
plot(f1,f2,’ko’);
xlabel(’f_1:cable index (normalized)’)
ylabel(’f_2: power distribution (normalized)’)
title(’Configuration with 8 SPDBs and 2 Galleys: results for multi-objectives (power distribution,cable length) optimization’)


Solution1_cableLength= result(find(result(:,1)== min(result(:,1))),:);
Solution2_powerDistribution= result(find(result(:,2)== min(result(:,2))),:);

%% Function initSPDB
function [spdb] = initSPDB(spdbNumber,ceilings)


global params

k=1;
for i = 1:2:spdbNumber-1
    spdb(1,i)= SPDB;
    spdb(1,i).Midpoint_x = params.cabin.xStart + (k* (params.cabin.length
       ))/(0.5 * spdbNumber);
    spdb(1,i+1).Midpoint_x = spdb(1,i).Midpoint_x;

    spdb(1,i).Midpoint_y = params.cabin.radius / 2;
    spdb(1,i+1).Midpoint_y = -1* spdb(1,i).Midpoint_y;

    spdb(1,i).Midpoint_z = ceilings(1,1).Midpoint_z;
    spdb(1,i+1).Midpoint_z = spdb(1,i).Midpoint_z;

    spdb(1,i).ID = i;
    spdb(1,i+1).ID = i+1;

    spdb(1,i).Name = strcat('SPDB',num2str(i),'RH');
    spdb(1,i+1).Name = strcat('SPDB',num2str(i+1),'LH');
    k = k+1;
end
end


%% Function generateInput
function [input] = generateInput(cabinLength)


n= ceil(cabinLength / 1000);                       %round the x coordinate in
    meter
input =  1000 * nchoosek([1:n 1:n 1:n],3);        %generate matrix with all
   combinations in mm
input= unique(input(:,1:3),'rows');               %delete identical rows

i=1;
while i <= length(input)
   if input(i,2)<= input(i,1) || input(i,3)<= input(i,2) || input(i,3)<=
      input(i,1)
       input(i,:)=[];
   else
       i= i+1;
   end
end
end

%% Function startPowerCalculation
function [wire, wireIndex, powerDistribution,spdbpower] =
   startPowerCalculation(psu,X,spdb)

[spdb.power]=deal(0);
[spdb.length]=deal(0);
[spdb.numPSU]=deal(0);

```

```

for j = 1:2:length(spdb)
    for i=1:2:length(psu)

        if (psu(1,i).Midpoint_x <  X(2,((j+1)/2))) && (psu(1,i).Midpoint_x >= X(1,((j+1)/2)))
         wire(1,i)= PowerWire;
         wire(1,i+1) = PowerWire;

         wire(1,i).spdb = spdb(1,j).ID;
         wire(1,i+1).spdb = spdb(1,j+1).ID;

         wire(1,i).linkedPsu = psu(1,i).ID;
         wire(1,i+1).linkedPsu = psu(1,i+1).ID;

         wire(1,i).wireLength = abs(psu(1,i).Midpoint_x - spdb(1,j).Midpoint_x);
         wire(1,i+1).wireLength = wire(1,i).wireLength;

       end
    end
end


%%Calculate wire length as an index for cable integration
for i = 1:length(spdb)
  for j = 1:length(wire)
      if (spdb(1,i).ID == wire(1,j).spdb)
          spdb(1,i).length = spdb(1,i).length + wire(1,j).wireLength;
          spdb(1,i).numPSU = spdb(1,i).numPSU +1;
      end
  end
end

wireIndex = 0;
for i= 1:length(spdb)
    wireIndex = wireIndex + spdb(1,i).length ;
end

%%Calculate mean power of total SPDBs considering required PSU power
mean_power = 0;
for i=1:length(spdb)
spdb(1,i).power = spdb(1,i).initPower + 2*(spdb(1,i).length * 1000 * spdb(1,i).resistivity * spdb(1,i).current * spdb(1,i).current / spdb(1,i).area) + (spdb(1,i).numPSU * spdb(1,i).powernenn);
mean_power = mean_power + spdb(1,i).power;
end
mean_power = mean_power / length(spdb);
```

```

powerDistribution =0;
%%Calculate standard deviation as an index for power distribution in
    cabin
for i= 1:length(spdb)
    spdbpower(1,i)=  spdb(1,i).power;
    powerDistribution =   powerDistribution + (spdb(1,i).power -
        mean_power)* (spdb(1,i).power - mean_power);
end
powerDistribution = sqrt((1/(length(spdb)-1))* powerDistribution);
end
```

Erklärung zur selbstständigen Bearbeitung der Arbeit

Gemäß der Allgemeinen Prüfungs- und Studienordnung ist zusammen mit der Abschlussarbeit eine schriftliche Erklärung abzugeben, in der der Studierende bestätigt, dass „die Abschlussarbeit bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit [(§18 Abs. 1 APSO-TI-BM bzw. §21 Abs. 1 APSO-INGI)] - ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt wurden. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich zu machen.“

Quelle: §16 Abs. 5 APSO-TI-BM bzw. §15 Abs. 6 APSO-INGI

Erklärung zur selbstständigen Bearbeitung der Arbeit

Hiermit versichere ich,

Name: ______________________________

Vorname: ______________________________

dass ich das vorliegende Master Thesis – bzw. bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit – mit dem Thema:

A Model-Based Methodology for the Integration of a System Architecture in a Digital Aircraft Design Process

ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

________________ ________________ ____________________________

Ort Datum Unterschrift im Original