Model-Based System Engineering Methodology for Implementing Networked Aircraft Control System on Integrated Modular Avionics - Environmental Control System Case Study

Prince George Mathew

A Thesis

in

The Department

of

Mechanical, Industrial and Aerospace Engineering

Presented in Partial Fulfillment of the Requirements

for the Degree of

Master of Applied Science (Mechanical Engineering) at

Concordia University

Montréal, Québec, Canada

March 2019

Concordia University
School of Graduate Studies

This is to certify that the thesis prepared

By: **Prince George Mathew**

Entitled: **Model-Based System Engineering Methodology for Implementing Networked Aircraft Control System on Integrated Modular Avionics - Environmental Control System Case Study**

and submitted in partial fulfillment of the requirements for the degree of

Master of Applied Science (Mechanical Engineering)

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the Final Examining Committee:

_____ Chair
Dr. Ivan Contreras

_____ External Examiner
Dr. Ferhat Khendek

_____ Examiner
Dr. Youmin Zhang

_____ Supervisor
Dr. Susan Liscouët-Hanke

Approved by _____
Martin D. Pugh, Chair
Department of Mechanical, Industrial and Aerospace Engineering

_____ 2019

_____
Dr. Amir Asif, Dean
Gina Cody School of Engineering and Computer Science

Abstract

Model-Based System Engineering Methodology for Implementing Networked Aircraft Control System on Integrated Modular Avionics - Environmental Control System Case Study

Prince George Mathew

Integrated Modular Avionics (IMA) architecture host multiple federated avionics applications into a single platform and provides benefits in terms of Size, Weight and Power (SWaP), nonetheless brings a high level of complexity to aircraft control systems. The thesis presents Model-Based System Engineering a novel, structured development methodology to cope efficiently with increased complexity due to IMA. Using ARCADIA methodology and the open source Capella tool, the developed methodology is implemented for a complete design cycle: starting with capturing requirements from the aircraft level to streamlining the development, integration of avionics application in an ARINC 653 platform. The proposed methodology provides effective traceability and management of specification artifacts from aircraft to system to item-level adhering to SAE ARP4754A guideline. Further, the thesis presents the capability of the MBSE framework to effectively address a few technological variants through the proposed methodology. To illustrate the efficiency of the methodology and MBSE approach an Environmental Control System (ECS) case study is presented. The case study focuses on implementing ECS in an IMA architecture using MBSE framework and proposed methodology. However, the derived methodology is also applicable to other systems. Further, the case study also presents a demonstration of integrating Cabin Pressure Control Sub-system (CPCS) into a real-time IMA platform for validation of MBSE approach. In addition, the thesis provides important insights in challenges and advantages of the MBSE process in contrast to the traditional paper-based specification process.

Acknowledgments

Today marks the day, after two years of hard work I am completing my thesis with this thanking note. These two years helped me grow both academically and personally. I want to take this opportunity to express my gratitude to those wonderful people who supported me directly or indirectly throughout this journey.

Dr. Susan Liscouët-Hanke, my supervisor, gave me this research opportunity and provided me with an excellent work environment and the necessary industrial contacts needed for my research. I am very grateful for the unparallel support, understanding and useful guidance she has provided me with, in these two years.

I am very grateful to Yann Le Masson and Jean-Richard Coté from Bombardier Aerospace for providing me with the opportunity to conduct this project and support my visions. I thank Jean-Richard who provided me with resources on Avionics and further with enthusiastic encouragement to think out of the box. My special thanks to Yann Le Masson for his patient guidance and valuable suggestions that mould this project to great success.

I also extend my thanks to Sandro Fagundes, Issa Ibrahim, Vincent Saluzzi and Gustave Nfonguem from Bombardier for their timely support and valuable advice. Their willingness to provide their time so generously has been much appreciated.

A special thanks go to my admirable friends in the lab Andrew Jeyaraj, Ezhil Shakti, Hasti Jahanara, Florian Sanchez, Noah Sadaka, Abdul Malik and many others who were patient enough to bear my long presentations during lab meetings and support me with constructive suggestions and ideas throughout the period of work. I also wish to thank Seyyede Shahrzad, Mahdi Riazat for the moral support they gave me to finish writing this thesis.

I wish to offer special thanks to my friends: Ajay Prasad, Amith Krishnan, Prabjout Kaur, Angitha Maria Thomas, Linda Kevin, Aneesh Raj, Alen Davis, Lalet Scaria, Jibin John Joseph, Denny Mathew and others for your understanding and endless support.I thank my friends from XDIMA team: Khadietou Ndiya, Mohammedreza Sadri for making my eight months at Bombardier memorable.

I wish to express my deepest gratitude to my friend: Dony Joseph Augustine, who supported and guided me from my university application till this thesis completion.

Finally, I thank my parents Mathew and Mary, my brother Blessen and my sister Angel for believing in me and supporting me unconditionally since childhood. I am very grateful to my parents for helping me study thousands of miles away from home and chasing my dreams. Therefore, I dedicate this thesis to my dearest family.

Contents

List of Figures

List of Tables

Nomenclature

ACS	Air Conditioning System
ARCADIA	Architecture Analysis & Design Integrated Approach
ARINC	Aeronautical Radio, Incorporated
ARP	Aerospace Recommend Practices
BITE	Built In Test Equipment
CDS	Cockpit Display System
CPCS	Cabin Pressure Control System
CSA	System Contextual Actor
DARPA	Defense Advanced Research Projects Agency
DIMA	Distributed Integrated Modular Avionics
ECS	Environmental Control System
ECUs	Electronic Control Units
EICAS	Engine Indication and Crew Alerting System
ES	Exchange Scenario
FAA	Federal Aviation Administration
FCC	Flight Control Computer
FCS	Flight Control System.
FMS	Flight Management System
FS	Functional Scenario
HLR	High-level Requirements
HMI	Human-Machine Interface
HTML	Hypertext Markup Language

HX	Heat Exchanger
IC	Integrated Circuit
IDE	Integrated Development Environment
INCOSE	International Council on Systems Engineering
ISA	International Standard Atmosphere
LA	Logical Architecture
LCBD	Logical Component Breakdown Diagram
LDFB	Logical Dataflow Flow Blank
LFBD	Logical Functional Breakdown Diagram
LRM	Line Replacement Module
LRU	Line Replacement Unit
MARTE	Modeling and. Analysis of Real-Time Embedded systems
MBSE	Model-based System Engineering
MB	Mission Blank
MCB	Mission Capabilities Blank
MDE	Model Driven Engineering
MSM	Mode State Machine
MTBF	Mean Time Between Failures
NextGen	Next Generation
NIST	National Institute of Standards and Technology
OABD	Operational Activity Breakdown Diagram
OAB	Operational Architecture Blank
OAIB	Operational Activity Interaction Blank
OAS	Operational Activity Scenario
OA	Operational Analysis

OCB	Operational Capabilities Blank
OEBD	Operational Entity Breakdown Diagram
OES	Operational Entity Scenario
PAB	Physical Architecture Blank
PA	Physical Architecture
PCBD	Physical Component Breakdown Diagram
PDFB	Physical Functional Dataflow Blank
PFBD	Physical Functional Breakdown Diagram
R&D	Research and Development
RDC	Remote Data Concentrator
SA^{2}GE	Smart Affordable Green Efficient Phase 2
SAB	System Architecture Blank
SAE	Society of Automotive Engineers
SA	System Analysis
SDFB	System Functional Dataflow Blank
SE	Systems Engineering
SFBD	System Functional Breakdown Diagram
SRATH	System Requirements Allocated to Hardware
SRATS	System Requirements Allocated to Software
SWaP	Size, Weight, and Power
SysML	Systems Modeling Language
TRD	Technical Requirement Document
TTETHERNET	Time Triggered Ethernet
UA	User Application
UML	Unified Modeling Language

VLSI	Very-large-Scale Integration
VVI	Integration, Verification and Validation
WCET	Worst-Case Execution Time

Chapter 1

Introduction

Challenging industry targets for reduction in fuel burn and emission drive the aircraft manufacturers to provide better performance with reduced environmental impact. Therefore, they develop new technologies such as advanced avionics systems, high-efficiency engines, composite, or advanced materials to reduce the overall Size, Weight and Power (SWaP). To tackle the competition in the market, aircraft manufactures try to come up with a breakthrough in technologies within a short development period. However, the system engineering process for complex system-of-systems[1], like aircraft, is not well developed and is one of the reasons why aircraft development programs are time-consuming and resulting in high product development cost and period. Wherefore, manufactures research on methodologies and tools to improve the efficiency and effectiveness of their development methodologies processes, to address the new complex technologies.

1.1 Background and motivation

An average period to develop and deploy a new aircraft into the market can range from 7 to 10 years. For instance, from the feasibility study to end product took approximately ten years for the Airbus 220 (former CSeries) and Boeing 787 Dreamliner [1–3].The urge to provide a more capable and efficient product lead to high complexity in aerospace systems and the development process. The term complexity is broad and lacks a single definition. Reference [4]provides a few examples of complexity that apply to the aerospace industry. The degree

[1] A set of systems assigned with a task that none of the systems can accomplish on its own.

of interaction between systems in aerospace exhibits the *coupling complexity*. The higher the degree of complexity, the higher the difficulty in predicting the interaction behaviour. For example, modern aircraft have a high degree of coupling complexity due to a highly coupled large-scale system-of-systems with distributed software sub-systems. Moreover, a rapid growth in complexity can be seen in the beginning of millennium where more than 1000 interconnected Electronic Control Units[2] (ECUs) are integrated into the civil aircraft [5]. Further, the complex distributed international supply chains in aerospace create *business system complexity*. The A380, largest of all the passenger aircraft have more than 120 sub-systems which account for almost 5000 ECUs supplied from over 20 countries [6] is an example of business system complexity. Moreover, there exist *cognitive complexity* because of the depth of details and the complex interfaces in aerospace systems. For instance, A350 and B787 two of the most modern aircraft have approximately 4000 input/outputs (I/O) units integrated into the avionics systems [7]. In summary, the different kinds of complexity tend to extend the development duration in the aerospace industry.

Figure 1.1 shows historical trends of product development time as a function of measured "complexity" for three key industries namely aerospace systems, automobile and integrated circuits (IC) industries. The aerospace industry shows a linear increase in the development period with an increase in complexity. The aerospace industry requires innovation in design, integration, and testing to reduce the development period and catch up with the automobile industry. One of the main reasons for the long development time is the strict certification process that requires the aircraft manufacturer to show compliance with regulations and standards while meeting the customer requirements. That is, the authorities need to be satisfied that each element is validated and verified against its intended function and should be fault proof and deterministic [8–10].

[2]ECU is an embedded system that controls electrical systems or sub-systems

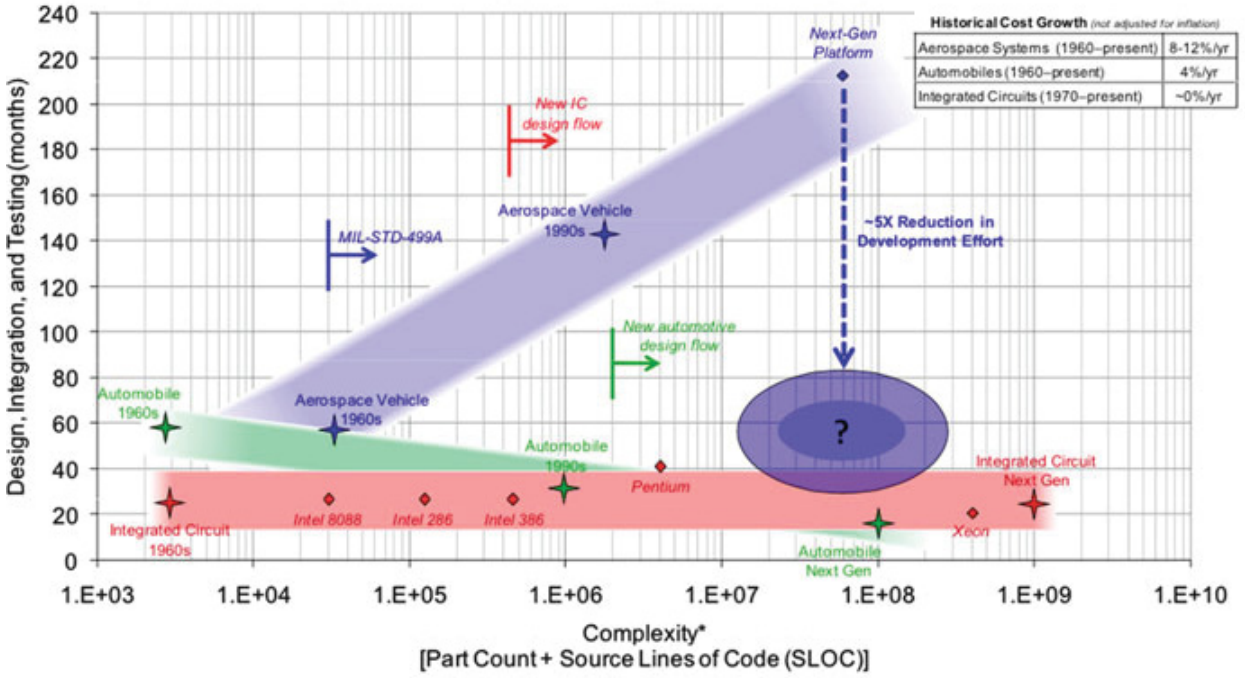

Figure 1.1: Historical schedule trends with complexity by DARPA [11]

The SAE APR4754A, "Guidelines For Development Of Civil Aircraft and Systems" is the governing guideline for aerospace to cope with the complexity [12]. The development process is graphically represented by the so called V-cycle or V-model [13]. The early phase of the process consists of design and specification and validation of the artifacts. Validation is the determination that the requirements for a product are correct and complete [12].That is, Validation ensures that the design and specification are for developing the right product. The later phases consist of development, and implementation, integration and testing with verification. Verification is the evaluation of an implementation of requirements to determine that they have been met [12]. That is, verification establishes whether the product is built right. Faults can be introduced at any stage of the aircraft program. Figure 1.2 depicts the cost of the late detection of faults during software development has been characterized by Lewis and Feiler [14, 15]. The later the faults are discovered in the development process, the costlier they are to correct, and the more significant is their impact on the schedule of the development program. Further, the V-model shows how a wrong decision made in system design affects the software development, thus the whole system.

However, the majority (70%) of the faults are introduced in the early design and analysis phase and are caused by errors in requirements and system interaction or interface definitions. Moreover, 80 % of these errors are detected in the verification phase which requires fifty-five times more nominal cost for fault removal than in the design phase. The reason for these costly errors is mainly because of challenges in the validation process as the aerospace industry has failed to adequately address the increase in complexity during design and early analysis.

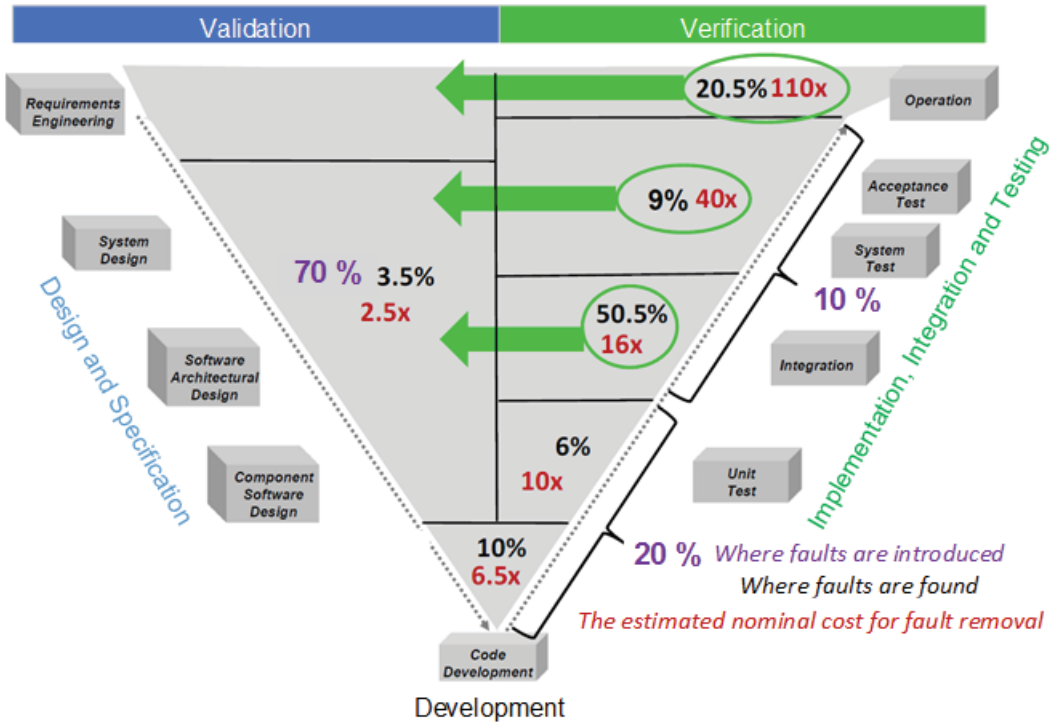

Figure 1.2: The traditional 'V- model' for aircraft development showing an estimation of the introduction, detection, and cost of removal of faults adapted from ref [16–18]

From the previous paragraph, it is clear that the faults increase the development time. Further, the faults are more likely to occur in complex system design due to the coupling and cognitive complexity. A fair share of complexity is due to the integration and optimization efforts known as "IMA (Integrated Modular Avionics) or open architecture concepts," in aircraft control systems to make the aircraft more efficient.The following section presents the IMA architecture.

1.2 Integrated modular avionics systems

The Federal Aviation Administration (FAA) definition of IMA is "a shared set of flexible, reusable, and interoperable hardware and software resources that create a platform that provides services designed and verified to a defined set of safety and performance requirements to host applications that perform aircraft functions" [19]. Boeing 777, one of the early user of IMA spend forty-seven percent of it is development cost for systems only; out of which 30% was used for hardware development, Validation, Verification, and Integration (VVI) and seventy percent for software development and VVI [20, 21]. That is, 33% of overall development cost was spend on IMA software. Hence, it is clear that the 8 to 12 % increase in development cost and five times increase in the development period of aerospace systems as depicted in Figure 1.1 is also because of complexity introduced by IMA. Compared to the traditional federated avionics systems, the IMA promise to increase the overall functionality of the aircraft by increasing integration as shown in Figure 1.3.

In a federated avionics architecture, each aircraft function has a standalone controller unit, called Line Replacement Unit (LRU) composing of application software, hardware board, and operating system and dedicated wiring for each connection. That is, from the landing gear to the Environmental Control System (ECS), from flight controls to the electric system, every aircraft system is equipped with its own control logic (function) and physical boxes. The impact during the loss of an aircraft function determines the criticality. Higher the impact hazard higher will be criticality. The benefit of the traditional system is that each aircraft function has assured access to the processors. Further, the federated system provides critical functional separation, that is a low-critical aircraft function cannot corrupt critical functions as they are separate LRUs that are loosely coupled. However traditional architecture cannot optimize the use of resources. Present processing units have advanced capability than LRU needs and limit many processors with lower utilization. Furthermore, each LRU requires multiple power supplies, networks, and I/O connectors. The rapid increase in the number of subsystems designed to provide a specific solution; this means more LRUs and increase in total weight, power, and wiring. As a result, the cost of operation for an airline to support

modifications and maintenance logistics is more significant in federated systems.

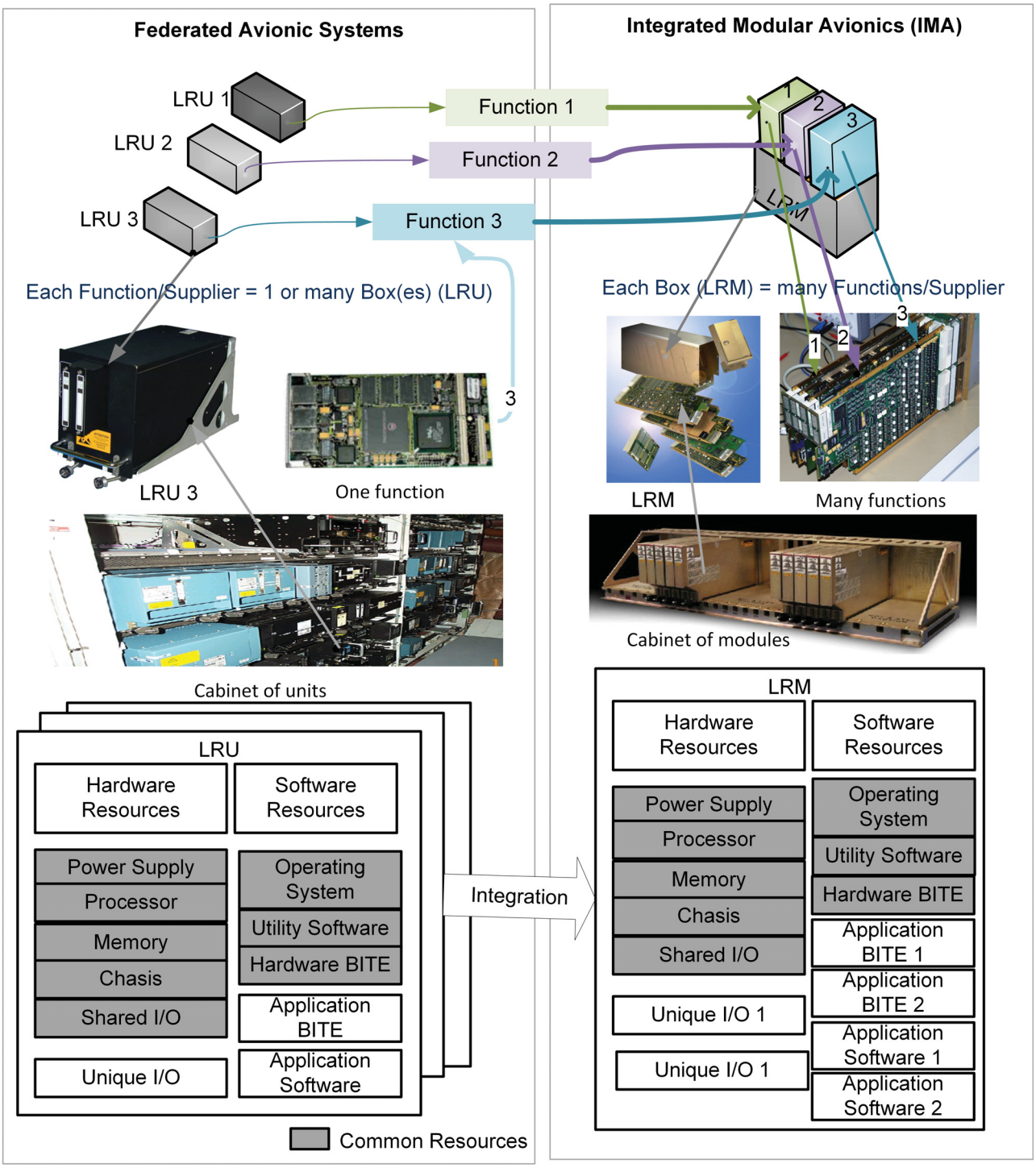

Figure 1.3: Federated avionics systems vs integrated avionics architecture concept, adapted from ref [22, 23]

These control systems, as well as their size, weight, and cost could be significantly optimized

if they shared a common hardware platform called Line Replacement Module (LRM). An IMA host multiple mixed critical aircraft functions in a single module by effective sharing of resources as shown in Figure 1.3.

The IMA implementation provides benefits in Mean Time Between Failures (MTBF)[3], the number of electronic systems reduced, percentage volume reduction, percentage power reduction, and weight reduction as shown in Table 1.1.

Table 1.1: Benefits of IMA implementation over federated system [24]

Impact factor	**Benefits compared to federated architecture**
MTBF	1.67 times to 4 times increase
Reduction in the number of electronic systems	14 to 145 electronic systems eliminated
Impact on avionics weight	Up to 350 pounds weight saving (2000 pounds for Boeing 787 using Common Core System)
Impact on space utilization	Up to 50 percent volume reduction
Impact on power consumption	25 to 50 % power savings

Further, Boeing observed the effect of integrating the additional software applications[4] in B777 compared to former federated architecture as shown in Table 1.2.

Table 1.2: Comparison of the effect of integrating additional application [25, 26]

	Related hardware cost (%)		Related software complexity (%)	
	IMA enclosure + 1^{st} application	Each additional application	IMA enclosure + 1^{st} application	Each additional application
Federated	100 %	100 %	100 %	100%
Integrated	155 %	25 %	110 %	60 %

[3]MTBF is a quantitative measure of hardware or component reliability.

[4]In the assumption that integration of related functions of equal size & complexity is carried out; 25% error margin

An IMA is characterized by three key attributes:

1. Integration: Integration of information leads to high operational effectiveness. Moreover, system reliability is increased through the integration of resources (shared resources) reducing duplication.

2. Modularity: Modularity replaces Line Replacement Unit (LRU) with Line Replacement Module (LRM). LRM decreases the Size, Weight and Power (SWaP) through hosting multiple LRU applications in a single hardware. These multiple applications share common resources through space and time partitioning. While the space partitioning guarantees the integrity of the allocated program & data memory space and registers, the time partitioning guarantees timely access to processing and communication resources. Further, robust partition provides benefits to host multiple applications with mixed criticality. Modularity also brings flexibility through the addition or removal of functions or hardware without affecting the functionality of other systems. Furthermore, LRMs can be reconfigured or changed in architecture and expanded to accommodate future requirements by adding more modules. Thus, modularity also supports incremental certification process.

3. Standardization: Standardization of IMA implementation brings a generic architecture that can be adapted for various applications or functions. Also, standardization brings less unique designs and parts that help logistics with increased supportability and reduced documentation, test equipment and configuration management.

The DIMA (Distributed Integrated Modular Avionics) concept an advanced IMA prototype that defines a networked and real-time computing architecture for airborne systems, aiming to streamline the development, integration, and maintenance processes of avionics software and hardware. With DIMA, application modules share computing and hardware resources, a communication network, and lower-level software layers, which allows system suppliers to focus on the application layer, enables reconfiguration in the presence of faults and significantly decreases the weight of avionics systems compared to traditional federated

architectures. Shared computing resources have so far been mainly used for primary avionics functions in modern aircraft. As part of the Smart Affordable Green Efficient phase 2 (SA^2GE) consortium project, a concept of highly integrated control systems using IMA is being studied at Bombardier Product Development Engineering, Aerospace [27]. DIMA reflects a highly complex system with a considerable number of interfaces and inter-dependancies. Integration issues and defects realized during downstream lifecycle phases are some of the challenges faced by system engineering process when implementing complex systems avionics systems [28]. The following section presents the systems engineering process in aerospace industry.

1.3 Towards model-based systems engineering

A broad definition of systems engineering (SE) by INCOSE is as follows. *"Systems engineering is an interdisciplinary approach and means to enable the realization of successful systems. It focuses on defining customer needs and required functionality early in the development cycle, documenting requirements, and then proceeding with design synthesis and system validation while considering the complete problem: operations, cost and schedule, performance, training and support, test, manufacturing, and disposal. SE considers both business and technical needs of all customers with the goal of providing a quality product that meets the user needs"* [29]. Therefore, systems engineering methodologies are considered appropriate to deal with the complexity and to design specification from stakeholders[5] needs. NASA has been using and developing SE methods since 1995 [30]. For instance, NASA applied systems engineering (SE) approach for the design, construction, and validation process of a lunar excavation prototype (Lunabot) and the project was able to achieve all the stakeholder and subsequent derived requirements [31]. SE has been used in the software development process prior to its application in aerospace systems development [32, 33].

[5]Stakeholder can include users, operators, acquirers, owners, suppliers, developers, builders, and maintainers of a system

The artifacts of systems engineering are information such as stakeholder requirements definitions, requirement analysis, architectures, interface definition documents, test cases, scenarios and more. Although the system engineers deploy various models to deal with the aspects of functional, behavioural, safety, cost, mass and power, the majority relies on a paper-based approach to capture the artifacts. In a paper-based approach, information is stored in multiple documents as shown in Figure 1.4 (a).

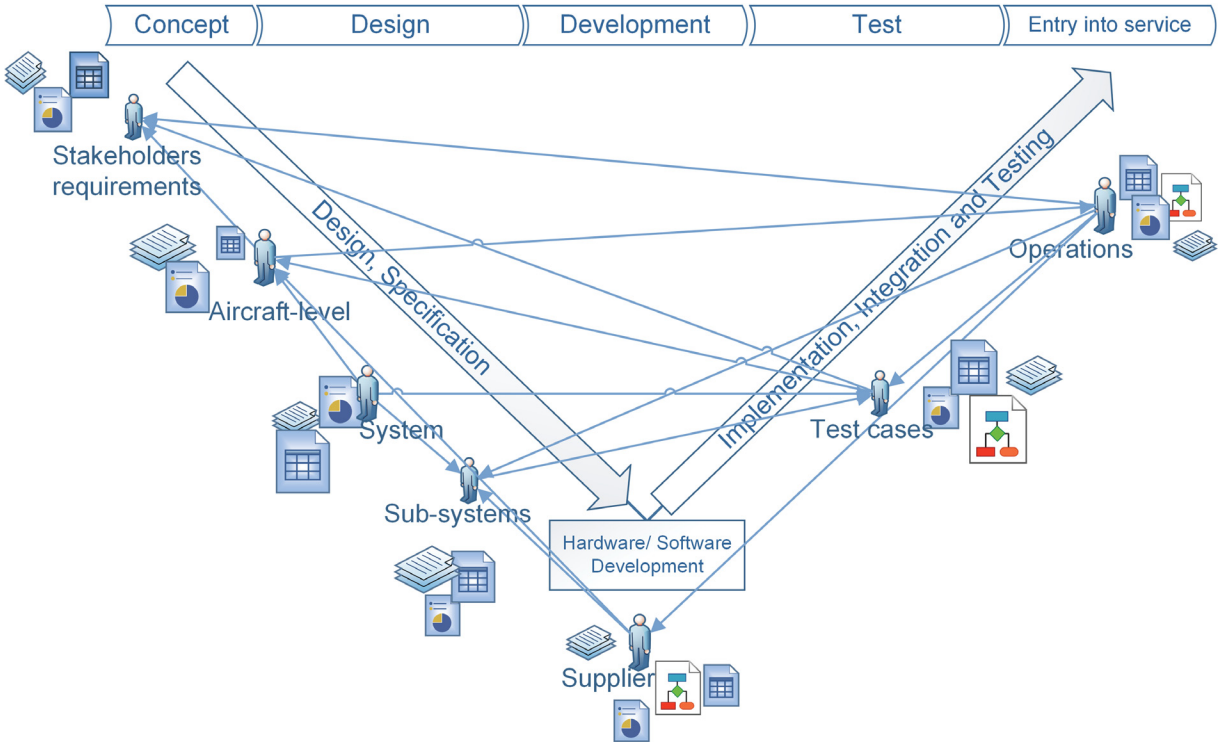

Figure 1.4: An example of the paper-based systems engineering process

The fact that in DIMA platforms resources are shared, makes the development and certification of individual real-time applications much more challenging. The initial engineering phases of such systems are critical. These early phases condition the aptitude of the architecture used to answer the needs of the clients, as well as the proper distribution of the requirements toward the components. The process is iterative between stakeholders and engineers in different disciplines until the design specification is matured. However, when information is spread across different documents in a paper-based approach, engineers face the following challenges such as [33]:

1. *Requirements consistency:* Systems engineering is a multi-level, multi-stage process dealing with the various degree of information. Consistency is the uniformity or conformity of requirements throughout the design process. Although consistency in requirements is necessary to develop a quality product that meets user needs, it is often difficult to maintain in document-based process. For instance, the system engineer could interpret stakeholder's requirements differently due to ambiguity. This ambiguity may introduce unintended capability and as a result diversion from stakeholders needs leading to high production cost.

2. *End-to-end traceability:* The by-product information at each level or stages should be maintained without losing up-, down- and horizontal traceability. In the document-based process, it is time-consuming to keep up the end-to-end traceability.

3. *Changes:* Additional resources and the workforce is required to track the changes in multiple separate documents at a different level or stages. Each change should be processed manually to account for end-to-end traceability and consistency.

4. *Integration:* When dealing with a system of systems or a system with multiple subsystems, integration of information as a whole system is difficult

For the aerospace industry, where systems are highly complex and critical, the maturity or completeness of the specifications or artifacts are essential. Moreover, being an iterative process, document-based systems engineering for complex aircraft systems makes it challenging to determine the requirements consistency, changes, end-to-end traceability, validation and integration than usual systems. Because of the lack of tools to analyze and design complex and highly integrated aircraft systems, engineers must resort to time-consuming simulations, and multiple redesigns and testing phases before the performance of a system can be judged as adequate. Moreover, simulation-based methods do not typically provide rigorous performance or stability guarantees, and the result is that critical functions are usually implemented on a separate dedicated network, increasing the platform complexity, cost, and weight. Further, the current aircraft are developed by distributed suppliers spread across the world, and it is challenging to determine consistency and traceability. For instance, the design,

engineering, manufacturing of Boeing 787 Dreamliner distributes to a global network of 700 suppliers [34] and came across several delays as aircraft failed to meet stakeholders needs [34]. The paper-based approach hardly supports the current distributed and multi-disciplinary system engineering. Therefore, a novel system-engineering framework needs to be defined and evaluated to advance state of the art in system engineering practices and methodology and to cope with the increasing complexity. The model-centric Model-Based System Engineering (MBSE) seems promising to address the challenges named above [35, 36].

A model in an MBSE is a self-explanatory representation that is used to explain the process behind a real-world system or event. In other words, an abstraction of a system, aimed at understanding, communicating, define or design some aspects of this system [37]. A model to be used for engineering purpose should have the characteristics expressed in Table 1.3.

Table 1.3: Characteristics of a model

Abstract	Highlight relevant aspects
Understandable	Represented in a way that is readily understood by the viewer
Accurate	Reliably represents the actual or modelled system
Predictive	Can be used to answer questions about modelled systems
Inexpensive	Must be low-cost to construct and study

MBSE is a formalized application of modelling 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 [38]. Characteristics of the model described in Table 1.3 helps to provide an overview of the whole system, its interfaces and eliminates ambiguity to the highest level. Modelling has been used to support certain aspects of early analysis and design phases such as safety and performance. However, MBSE introduces standardized modelling languages that can be utilized to represent a system through unified information models. Moreover, MBSE tools provide the capability to integrate different models and maintain a centralized architecture definition. Figure 1.5 shows the MBSE approach.

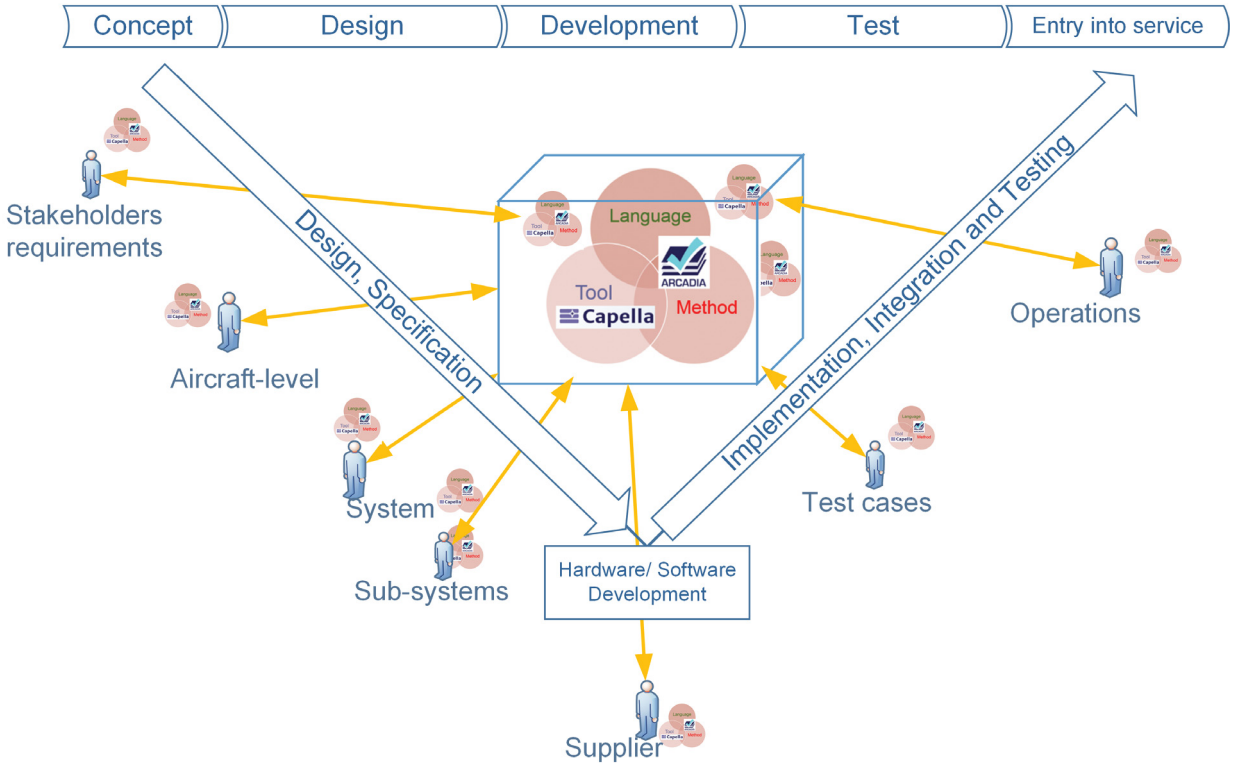

Figure 1.5: Model-based process

To conclude, aircraft development is one of the most complex and challenging processes due to the overall design complexity and because of the lack of proper tools to support the early design and analysis phases. However, MBSE can address the challenges faced during the early design and analysis of highly complex aircraft systems.

This thesis is a part of a larger research and development (RD) activity at Bombardier Aerospace called XDIMA [39]. The overall objective of the RD activity is to increase the maturity level in several facets of the management of the complexity induced by the architectures and development through MBSE. However, transitioning from traditional SE to MBSE environment comes with the following challenges:

- The modelling language should be able to express the complex domain-specific models or should be able to provide an extension.

- The tools to build and manipulate models should be efficient for use and ease for

learning.

- The MBSE environment should be able to support the methodology in the domain specific guidelines and standards.
- The framework should be compatible with existing design support tools. However, sometimes the organization needs to spend resources in developing the bridge between tools or to enhance the modelling capability.
- The widespread adoption of MBSE should be implemented to facilitate a unified representation of artifacts.
- The organization should be developing internal manuals or guidelines to avoid any ambiguity between the MBSE expression or terminologies with the former process. Moreover, these manuals should also include how the model should be interpreted depending upon the scope of the stakeholder.
- The environment should support real-time collaboration and track all the projects, versions, libraries, and dependence.

Therefore, to address some of the above challenges, this thesis provides a case study on implementing the complete design cycle of a complex system in the DIMA platform using an MBSE framework.

1.4 ECS case study overview and thesis objectives

The Environmental Control System (ECS) is selected as the ideal candidate for the case study. As presented in Figure 1.6 ECS is networked aircraft control system with complex interfaces and interdependencies.

Aircraft encounter an extensive range of flight and ground conditions, and it is essential that crew and passengers be kept in safe and comfortable conditions with the equipment in an

operating enclosure. Within the operational limit, aircraft experiences extreme conditions [40,41]. For instance, even though outside temperature is sub-zero (up to -80^0 C) the internal temperature should be maintained at 21^0 C - 25^0 C. At the same time, the outside pressure decreases with increasing flight altitude. For example, the outside pressure at 41000 ft is approximately 2.59 psi which would cause hypoxia. The maximum cabin altitude[6] allowed without causing distress to occupants is 8000 ft. In other words, the differential pressure must be controlled.

The three main functions of ECS are as follows:

1. Provide and control ventilation (implemented by the air-conditioning sub-system)
2. Control pressurization (implemented by the pressurization sub-system)
3. Provide equipment cooling (implemented by the equipment cooling sub-system)

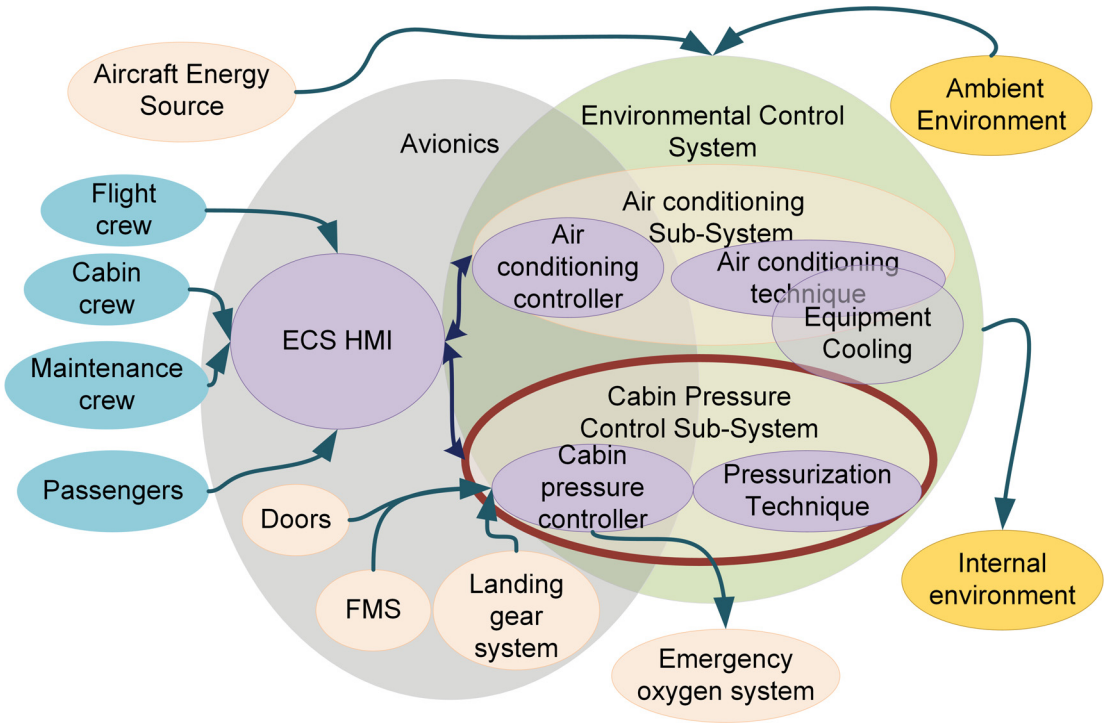

Figure 1.6: Environmental Control System (ECS) overview

[6]Pressurization inside the aircraft is defined either in cabin altitude or in terms of the differential pressure between the atmosphere and the cabin pressure.

An overview of ECS and its interface with other aircraft system is depicted in Figure 1.6. The primary users of ECS are flight crew, cabin crew, passengers and maintenance crew. For the ECS to perform its intended functions effectively, it needs to interact with other aircraft systems such as: doors, Flight Management System (FMS), landing gear system and emergency oxygen system. For efficient communication between the flight crew and ECS, Human-Machine Interface (HMI) requirements are essential. The ECS case-study focuses on the air conditioning controller and a cabin pressure controller.

Generally, an ECS includes an air-conditioning subsystem, a cabin pressure control subsystem, and equipment-cooling systems. There are two main technological categories for ECS as shown in Figure 1.7: the so-called conventional bleed air-driven ECS and a more-electric, so-called bleedless ECS (such as in the Boeing 787 [42]). In bleed ECS, the bleed air from the engine is the source of cabin air. Whereas, in bleedless ECS, ram air will be the only source. The ram air is compressed through a compressors. The compressors are rotated using motors, which is electrically powered by generator.

Electro-pneumatic and electric technologies for pressure control are variants in pressurization control. Electro-pneumatic pressurization technology uses both electric and pneumatic power to control the outflow valves. Electric pressurization operates entirely on electrical resources, and as a result, an aircraft with electric pressurization needs to include a safety pressure-relief function or redundancy to cope with electrical malfunctions. The equipment cooling system is required to provide suitable operating conditions for the safety-critical equipment. Generally, an ECS features two types of equipment cooling systems: (a) Equipment bay cooling provides cooling for avionics systems and (b)In-system cooling provides cooling for the ECS subsystem.

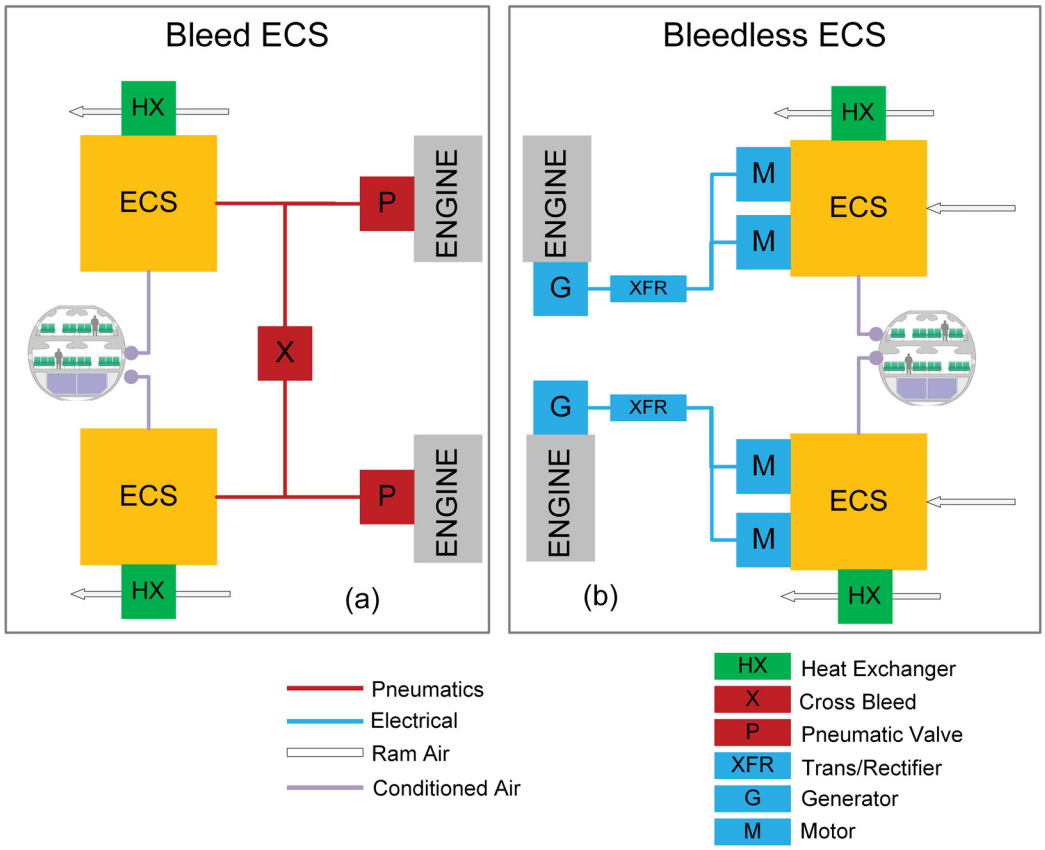

Figure 1.7: (a) Bleed ECS and (b) bleedless ECS architectures inspired from [42]

The "aircraft energy source" (in Figure 1.6) is a container for any system providing electrical or pneumatic power to the ECS. For the case study, the aircraft engines, the auxiliary power unit (APU), batteries, and airport facilities are considered. The only exception is for bleedless implementation, wherein pneumatic power generation is part of the ECS.

To summarise, the ECS is a complex networked aircraft control system with a higher number of interfaces and interdependencies. Therefore, for the case study, the ECS will be implemented on an IMA platform using SA2GE consortium project resources under the XDIMA project. The case study will investigate and evaluate the capability of MBSE framework through the following objectives.

1. Develop a practical, reusable MBSE methodology for the aerospace industry for IMA implementation of networked aircraft control systems.

2. Develop an ECS case study to evaluate the developed methodology

3. Integrate the cabin pressure controller into an IMA demonstrator for the validation and verification of the pressure controller specification.

1.5 Organization of the thesis

In order to achieve the objectives, the presented work is organized in the following way.

The state of the art of MBSE is presented in chapter 2. The early design and analysis phase problems for implementing ECS in an IMA configuration are outlined. Next, the selected MBSE framework to address the challenges faced when implementing the case study is presented.

In Chapter 3 the methodology to address the challenges is outlined in chapter 2. The methodology considers the multi-level system engineering process and the aircraft system development process described by guidelines in the aerospace industry.

Chapter 4 includes the specification models developed using the defined methodology. Various viewpoints that enhance the model properties are checked to analyze the potential on tool infrastructure. Chapter 5 presents the integration into the XDIMA demonstration platform.

The conclusion to the research work is presented in Chapter 6 along with future work and improvements. Additionally, Appendix A provides an overview of Capella (MBSE framework) diagrams. Appendix B, C, D, E provide examples of specifications defined in Capella and Appendix F gives a short description of an electric air conditioning system modelled in Capella.

Chapter 2

Model-Based System Engineering: State of the Art

Implementing an IMA configuration for networked control system would increase the system reliability and fault tolerance. Further, adopting a controlled, disciplined, and consistent system engineering process like MBSE would enhance fault avoidance. In this section, the state of the art of model-based system engineering and IMA case study is discussed.

2.1 MBSE approach and ARCADIA methodology

The MBSE method helps to eliminate challenges faced by document-based systems engineering. At present, there are methods available that can be divided into three catalogues [43, 44]:

1. Modeling Language and Frameworks: These standards are used to express and communicate models. The models provide better understanding and interpretation by users and can be used for analysis and processing by programs. Some of the examples for this category are modelling language (SysML - Systems Modeling Language [45], UML-Unified Modelling Language [46]), model exchange format (XMI), representation model (diagram definition, documents).

2. Mapping Specifications: These standards are used for the integration of models across multiple domains and communities. That is, these specifications helps to map, integrate

and provide interoperability across multiple sources and forms of models. For instance, the SysML-Modelica transformation specification provides a bi-directional mapping between SysML and Modelica [47].

3. Problem-specific frameworks, models and reference data: These are generated and shared for a particular system, and problem types. An example would be Architecture Modeling Language (UPDM - Unified Profile for DoDAF/MODAF [48]) or Hardware/Software Systems (MARTE -Modeling and Analysis of Real-time and Embedded systems [49])

The UML is a collection of diagrams that depict software structures graphically. UML is a modelling language that helps to cope with complex structures by specifying, designing and documenting the artifacts graphically. Further, UML is object-oriented and focus mainly on data, interaction and evolution. Data is modelled using class diagrams while the interactions are depicted through a collaboration or sequence diagram. Evolution focuses more on the states of the system and their transition. However, each diagram is developed as separate entities[1], and UML does not fully define a relationship between them. Moreover, from a software point of view, the same impact of UML can be obtained through informal, box and line diagrams drawn in standard drawing or diagram software. One reason is that software engineers do not prefer complexity or formality at an architectural level.UML is specifically used for software analysis. However, the UML profile is suitable for system engineering of complex systems if additional concerns of stakeholders can be addressed. From this idea, SysML was created.

SysML is a modelling language and an extension of UML 2 that supports the specification, analysis, design, verification, and validation of systems that include hardware, software, data, personnel, procedures, and facilities [45]. SysML provides two new diagram types called requirements diagram and parametric diagram. The requirements diagram provided the capability to support requirements and traceability. Furthemore, the parametric diagram provides the capability to define the mathematical relationship between software and

[1]Entities are something capable of an independent existence that can be uniquely identified

environment for verification. For example, parametric diagrams can be used check if the software can control the environment or the required parameters. However, the parametric diagram analysis is suitable for the only the ideal model, and when transformed into real executable software the validity of the analysis is threatened. To overcome this problem specific framework MARTE was developed.

MARTE is also an extension of UML. MARTE provides modelling capabilities to verify the schedulability, performance and time. The profile supports the modelling of three aspects in the real-time embedded systems. They are a software resource model, hardware resource model and the allocation of the software model to the hardware model [50]. There exist several MBSE tools that facilitate systems engineering using modelling mentioned above standards. Some of them are Rational Rhapsody, Papyrus, Microsoft Visio, MagicDraw, MARTE Profile for Rational Software Architect and MagicDraw.

The evolution of UML to MARTE was to address specific concerns set by the stakeholders. The segregation of these concerns is called viewpoint in systems engineering. Viewpoint " *is a systems engineering concept that describes a partitioning of concerns in the characterization of a system* [51]". The viewpoints help to identify the design concerns and problems in a particular aspect. Functional viewpoint, physical viewpoint, information viewpoint, technology viewpoint, enterprise viewpoint and engineering viewpoint are some of the concerns specified in ref [52]. The functional viewpoint focus on functions, interaction, behaviour and interfaces. Further, the physical viewpoint focuses on the link or physical connection, components and connectivity to functional aspects. The information viewpoint deals with the flow of information and how information is managed. For instance, a sequence or collaboration diagram is a subset to this viewpoint. Further, the enterprise viewpoint deals with requirements management and organizational aspects. Finally, the engineering viewpoint deals with the design, allocation of functions, validation and verification of system [53]. Each viewpoint contributes an augment to capture such as performance, structure, mass, thermal and more. The combination of these viewpoints provide a complete architecture description of a system [54]. One of the primary objectives of an MBSE framework is to support the designing of architectural models that meet the stakeholder's concerns.

Although there exist several modelling standards, for implementing an MBSE framework in the aerospace industry, a significant effort is required to establish a proper model-based approach, requiring resources, considering a learning curve. The industry has addressed the complexification of aircraft systems through the establishment of an industry-wide design guideline ARP4754A [12], a system engineering guide for aerospace systems. However, the ARP4754A states only the principles and not the practical "how-to," and it is generally implemented via a paper-based approach, using requirements documents rather than models to define, evaluate, validate, and verify the architecture. The main factors of choosing a useful MBSE framework will depend on the capability to allows engineers to implement principles specified in ARP4754A, follow other relevant standards, and adequately address the complexity and configuration of the system under consideration. Airbus Operations have implemented a model-based approach for the A350 program with great success [55]. However, being a new approach model-based was limited as certification requested for a specification understandable for all and the final format was new and time consuming for system engineers to adapt.

ARCADIA (Architecture Analysis & Design Integrated Approach) methodology [56] developed by Thales and open source Capella tool was chosen to implement the MBSE framework in this thesis. One of the reasons was that the de facto modelling language SysML and MARTE are object-oriented and often there is difficulty regarding comprehension and use due to lack of appropriate trained highly qualified personnel. Further, instead of focusing on modelling languages like SysML or UML, ARCADIA focuses on the method and as a result systems engineers are not required to be a modelling expert. ARCADIA is a structured engineering method for defining and verifying the architecture of complex systems. Capella is the tool that implements the ARCADIA methodology. Unlike SysML or UML, ARCADIA/Capella supports functional analysis. That is, ARCADIA represents functional requirements in terms of system functions with well-defined inputs and outputs [57]. The ARP4754A requires functional analysis for complex system development and to determine the safety aspects. However, SysML does not strictly support concepts of functions and functional hierarchy but instead implement a function-oriented approach through activity

diagrams. Further, SysML also uses blocks to represent functions, and the conceptual difference between structural element and functions is lost in this process. Furthermore, in SysML information flow is restricted between activities at the same level as shown in Figure 2.1. Therefore, it is difficult to realize the information flow at multiple levels of decomposition. On the contrary, ARCADIA/Capella realize only exchange between leaf functions and only leaf functions be allocated to the structure. Moreover, Capella recommends to first create functional flows independently from components architecture, then to allocate functions to components, and finally to deduce components interfaces from functional exchanges and their contents [58].

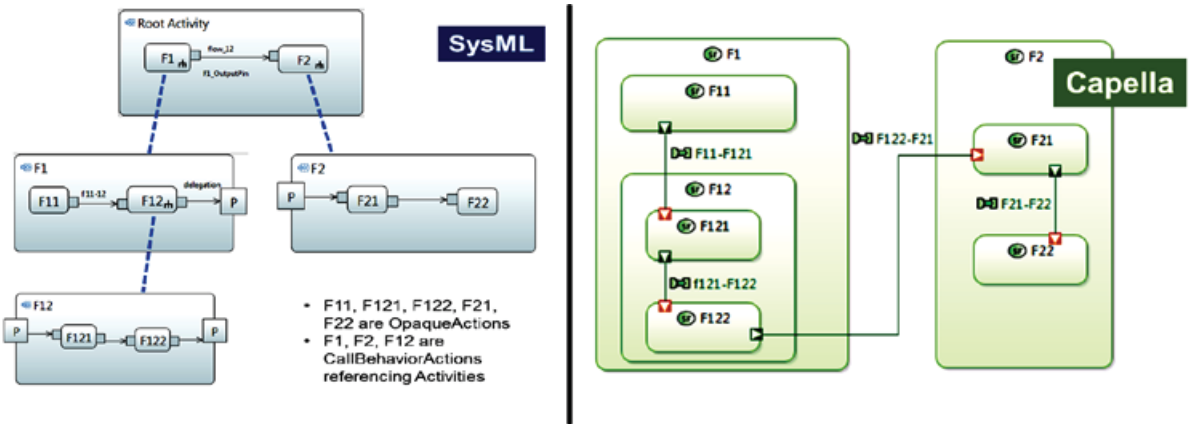

Figure 2.1: Equivalent functional decompositions in Capella and SysML [57]

As illustrated in Figure 2.2, the ARCADIA method defines four different working levels for the architecting process:

1. *Operational Analysis (OA)* defines what the customer and users of the system need to accomplish. The primary objective is to identify the so-called "actors" that interact with the system, and their associated needs towards the system (e.g., the flight crew, or cabin crew, performs the activities to regulate the cabin temperature). OA defines use cases or needs through operational capability[2].

2. *System Analysis (SA)* defines what the system has to do for the user/actor[3]. This

[2]Capability of an organization to provide a high-level service leading to an operational objective being reached.

[3]Any user or entities that is external to the system.

process helps to determine the functions that are needed by the system, such as control pressurization. SA also perform functional analysis by developing information flow between functions. The system starts to appear at this working level, and requirements are consolidated and formalized. SA defines the use cases or needs through system capability[4].

3. *Logical Architecture (LA)* defines how the system works to achieve the required performance. Along with developing functions, this process also identifies the components that perform these functions. The resultant is a logical architecture. Here, component exchanges are also justified by allocating functional exchanges. The LA provides a logical solution to the needs specified in SA and OA.

4. *Physical Architecture (PA)* defines how the system will be developed and built. This process sheds light on the architecture's physical components, such as turbines or compressors, along with the logical functions they perform

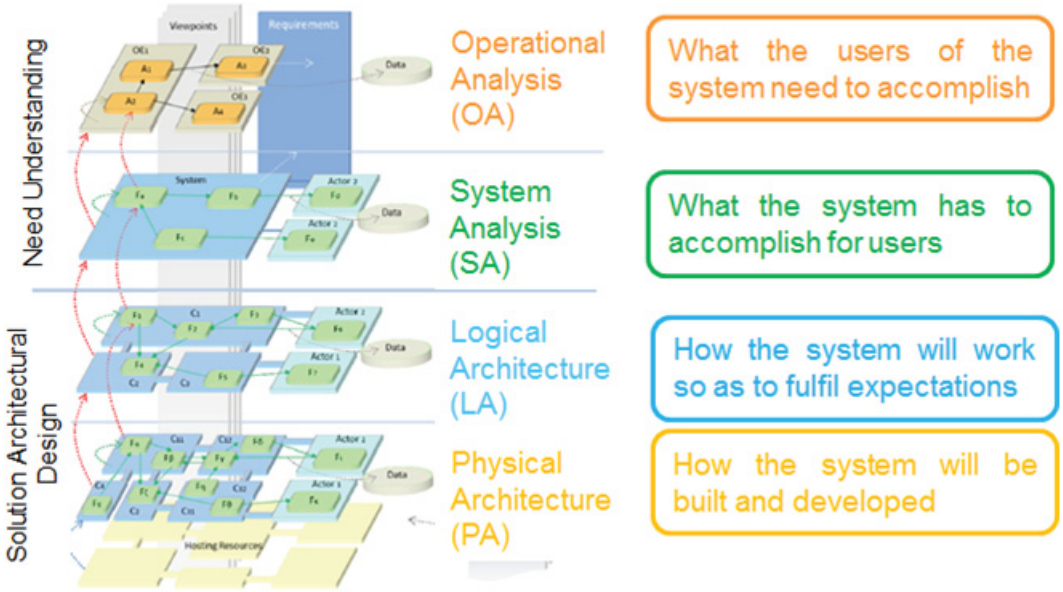

Figure 2.2: Overview of ARCADIA methodology adapted from [56]

[4]Capability of the System to provide a high-level service allowing it to carry out an operational objective.

This 4-level process makes sure that the needs of the customer are respected, and requirements are accurately allocated towards the components. As a result, the system specification should be consistent with customer needs. Further, ARCADIA/Capella provides most of the viewpoint specified in ref [52] through the four working levels. For instance, LA specifies functional viewpoint and PA specifies physical viewpoint. The framework also supports information and engineering viewpoints. All levels support the enterprise viewpoint through requirements management. In ARCADIA/Capella each level transition is an automatic process with traceability possible at any level.

First, in the OA the operational capabilities are defined for the actors[5] or entities[6]. For instance, the pilot, or the cabin crew should have the capability to control or regulate internal ambient temperature. Then activities and interaction of activities and actors/entities are specified to meet the operational capability. The second working level is the System Analysis or the Functional Analysis. Like in the OA, first the system capabilities are defined. For example, "provide acceptable thermal comfort" will be the system capability to fulfil the pilot's operational capability of regulating internal temperature. All such system capabilities together form the mission of the system. The SA also defines the functional need requirements as functions. Further, SA also supports functional analysis by developing information flow between the functions. The system starts to conceptualize at this working level, and requirements are consolidated and formalized. The first two levels provide the need for the system. SA also perform functional analysis by developing information flow between functions.

The next working level is the Logical Architecture which specifies how the system must work to fulfil expectations. At this level, new developed logical functions are allocated to the realized logical components. The outcome of logical architecture contains all the logical solution possible for the needs specified in OA and SA. Then the additional analysis is performed to select a suitable solution.

[5]Actors are particular case of a (human) non-decomposable operational entity.

[6]Entity belonging to the real world (organization, existing system, etc.) whose role is to interact with the system being studied or with its users.

Table 2.1: List of Capella diagrams [59]

Diagram	Description
Breakdown diagram	The component/function/users/activity hierarchy through a graphical tree.
Capability diagram	Provides the needs or use-case of the system and aids in organizing the functional analysis.
Dataflow diagram	Provides information flow between functions or activities. Capabilities can be highlighted using process chains or functional chains. The chains are sub-routes of dataflow diagram.
Architecture diagrams	This diagram shows the allocation of activities to actors in OA and allocation of functions to components or system. The diagram also shows the allocation of a dataflow diagram to interfaces and within the component.
Scenarios	Provides the process chains or functional chains that execute sequentially to fulfil a specific capability. The scenario can have actors, system and component interaction.
Modes and State Machine	Provides the working type of a function or actor or system. For instance, a process or task in an application can have *Dormant*, Waiting, running or terminating modes. Alternatively, application management can have a cold start, warm start, normal or idle. (Flight phases are also an example). It is used for representing behaviour as the state history of an object in terms of its transitions and states.
Class diagrams	Often, data-class diagrams compress of exchange items or data parameters utilized in a system

The last working level, the Physical Architecture shows how the system will be developed

and built. All the transitions to the next working level are an automated process. Several diagrams are available in Capella as listed in Table 5 that help to create a centralized architecture definition. These diagrams are available to use in all working level. Appendix A presents the diagrams available at each working level.

As discussed in section 1.3, the specification of aircraft systems is conducted via a paper-based process. This paper-based process uses aircraft-level requirements, typically called aircraft-level requirements documents (ARD) and derived system-level requirements, also called technical requirements documents (TRD). The ARCADIA/Capella provides an enhanced view of the specification through OA, SA, LA and PA as shown in Figure 2.3.

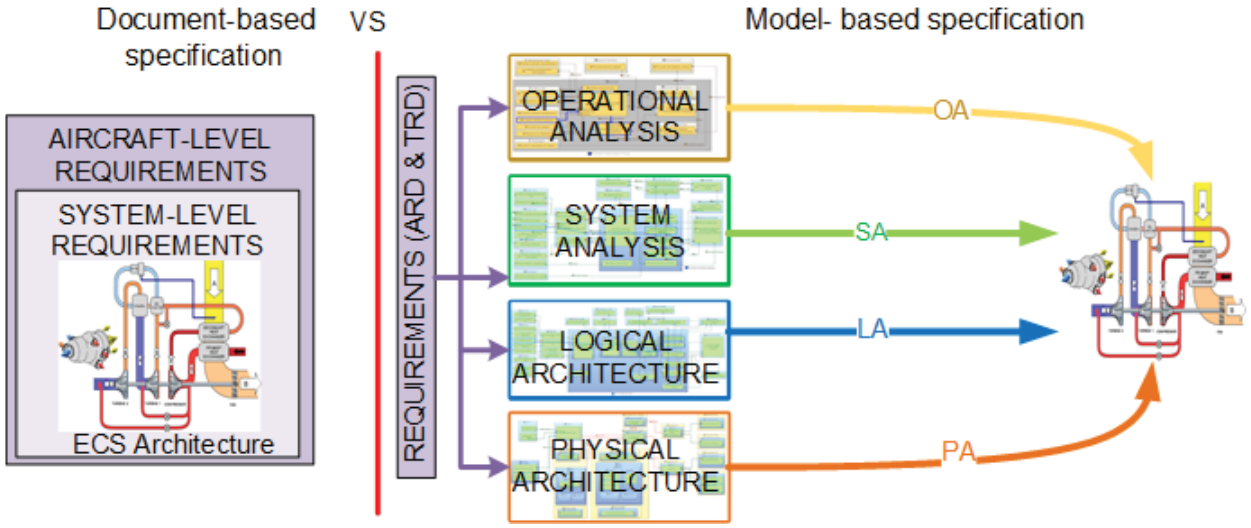

Figure 2.3: Architecting process for an aircraft system: traditional document-based method compared to an MBSE approach using the ARCADIA method

Capella has been tested and proven for aerospace systems engineering. For instance, the ARCADIA method and the Capella tool have been used for the optimization of a DIMA architecture in ref [60]. Here, Capella was used to defining functional requirements and systems constraints and automatically extract for optimization techniques. However, this prior work was focused mainly on software functions, not the overall system. Therefore, ARCADIA helps to address the key challenges in the early analysis and design phase for DIMA architecture.

There are commercial tools for variants management, such as pure::variants [61] available and can be used in Capella. Thales has used pure::variants to deal with large scale variability management for flight control computer test means [62]. However, The work shows that pure::variants provide the system variant[7] by fading the design elements that are not part of the variant. The main problem is that all those faded elements still exist in the model and is visible to anyone authorized to view only a particular variant. Moreover, the commercial tools are expensive and therefore effective ways to manage a few variants in Capella should be defined. Further, ref [63] explored the feasibility of using Capella for flight control system architecture exploration in conceptual design. The work also explored development of reusable catalog of actuator architecture for use and synthesis of FCS architecture.

Bombardier Aerospace has been collaborating with universities to conduct a feasibility study on ARCADIA/Capella for aerospace systems. One of the prior work is about MBSE approach for the conceptual design of aircraft high-lift system architectures [64]. This feasibility study focused on both top-down and bottom-up approaches. The top-down approach was used to develop a generic high-lift system at OA, SA, LA, and PA levels. Several iterations were carried out to determine what level of details is sufficient to represent the system. The bottom-up approach was used to represent an existing architecture graphically. Both approaches were carried out separately and then compared to demonstrate the effectiveness of an MBSE approach using the ARCADIA/Capella framework in conceptual design. Another prior work included a top-down approach for developing landing gear system specification in a DIMA architecture [65]. The work focused on the detailed design of the physical system interfaced to a DIMA controller. However, the prior works focused on less complex systems and parts of the development process. To understand completely how ARCADIA/Capella can be implemented in an effective manner, a complete development cycle needs to be carried out. The here presented ECS DIMA implementation case study covers this gap. The next section presents an overview of the implementation.

[7]System variant is a variant of a system derived from possible set of solutions.

2.2 ECS DIMA implementation

The case study of this thesis mainly focuses on the MBSE approach for effective early analysis and design phase for implementing ECS in a DIMA architecture. However, it is required to strictly follow the standards and guidelines to develop an airworthy IMA implementation. Figure 2.4 presents the DIMA implementation of ECS.

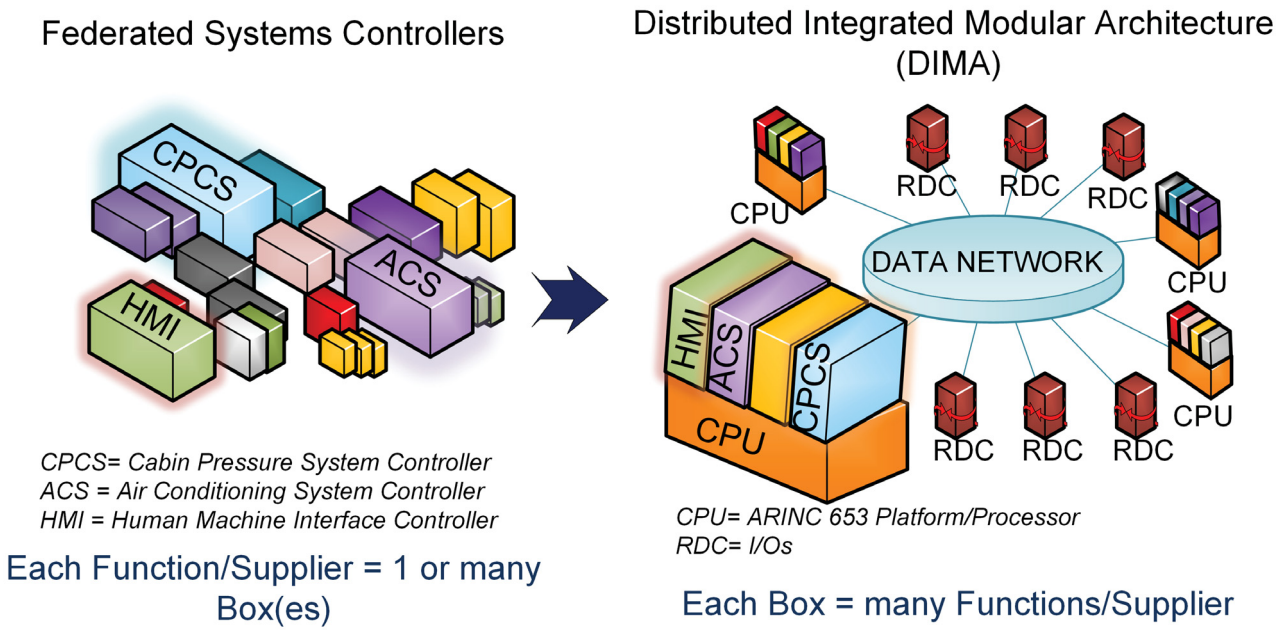

Figure 2.4: DIMA concept [27]

1. *SAE ARP4754A:* Aerospace Recommended Practices provides systems engineering guideline for the development of complex aircraft systems [12]. By adopting a suitable MBSE approach, it is possible to adapt and tailor guideline implementation suitable to complex architecture like DIMA.

2. *RTCA DO - 297:* Integrated Modular Avionics (IMA) development and guidance and certification considerations provides objectives, processes and activities for those involved in the development and integration of IMA modules, applications, and systems to incrementally accumulate design assurance toward the installation and approval of IMA system on an approved aviation product [66].

3. *DO – 178C*: Software considerations in airborne systems and equipment certification standard provides required software development plans and process, and verification process for approval of software-based aerospace systems. Philosophy is such that standard will result in fewer errors and equal importance is given to requirement-based software testing and analyses such as safety and software [67].

4. *DO – 331:* Model-based development and verification supplement to DO-178C and DO-278A is supplement that contains modifications and additions to DO-178C and DO-278A objectives, activities, explanatory text and software life cycle data that should be addressed when model-based development and verification are used as part of the software life cycle. Therefore, this supplement also applies to the models developed in the system process that define software requirements or software architecture [68].

5. *ARINC 651:* The design guidance for IMA by Aeronautical Radio Inc (ARINC) is for the definition of the generic hardware architecture with the design philosophy and recommended practices concerning the design of IMA [69].

6. *ARINC 653:* Avionics Application Software Standard Interface defines a general-purpose APEX (APplication/EXecutive) interface between the Operating System (O/S) of an avionics computer resource and the application software. Included within this specification, are the interface requirements between the application software and the O/S and the list of services which allow the application software to control the scheduling, communication, and status information of its internal processing elements [70]. ARINC standards define the central idea of IMA as effective resource sharing and robust partitioning.

7. *ARINC 661:* Cockpit display system interfaces to user systems describes the concept of operation for the standard protocol used between avionic-equipment user applications (UAs) and the cockpit display system (CDS) [71]. A display unit consists of a window managed by the CDS. Each window can have multiple layers owned by the UAs, and each layer can have multiple widgets defined with regard to the UAs.

8. *SAE AS6802*: Time-Triggered Ethernet (TTETHERNET) standard defines a fault-tolerant synchronization strategy for building and maintaining synchronized time in Ethernet networks, for critical integrated applications, IMA and integrated modular architectures [72]. This standard helps to develop a deterministic Ethernet suitable for time-critical applications creating a unified data network.

9. *ASHRAE Standard 161-2013*: Air quality within commercial aircraft looks at the factors affecting air quality, ANSI/ASHRAE Standard 161-2013 addresses guidance for temperature, moisture, pressure, and ventilation. It also greatly takes into account contaminants, including de-icing fluid, exhaust fumes, fuel, ozone, and bacteria [73].

To transition from federated to a DIMA configuration with effective early analysis and design phase using MBSE, the following key challenges are addressed in this thesis:

1. **Generic architecture:** A generic architecture will need to be developed to comprise all the logical solution possible for an ECS. The question is how to develop a generic, reusable system architecture, valid for the implementation of various technologies, including generic controllers.

2. **Variant technological implementations:** The case study will efficiently derive variants of physical systems from this generic system specification without losing upward, downward, or horizontal traceability.

3. **IMA controller architecture:** The case study will efficiently develop controller architecture for a subsystem. This architecture includes the application or software architecture and hardware architecture.

4. **ECS HMI architecture:** The ECS HMI architecture will be developed for a generic flight deck specification.

The state-of-the-art shows that a methodology needs to be defined for the model-based systems engineering process to be reliable and re-useable for multiple aerospace projects. Therefore, the necessary step is to understand how ARCADIA methodology along with Capella

tool can be utilized with aerospace guidelines and standards to define a methodology for aerospace use-case. Next chapter focuses on aerospace guideline and standards compatibility with Arcadia/Capella to define the methodology.

Chapter 3

Methodology

The thesis aims to develop an MBSE methodology for the aerospace industry for IMA implementation of complex aircraft control systems. In this chapter, the developed methodology for ARCADIA/Capella framework is presented.

3.1 Methodology overview

The methodology for the model-based development process of any control system that will be integrated into an IMA platform needs to comply with the aerospace standards. As introduced in section 2.2, the following two guidelines define the development process needed for IMA.

1. SAE ARP4754A: Guidelines for Development of Civil Aircraft
2. DO – 178C: Software Considerations in Airborne Systems and Equipment Certification

The herein proposed methodology addresses three aspects:

1. Mapping between the MBSE steps and the development steps in the ARP4754A and the DO-178C
2. Implementation of the multi-level approach in the MBSE framework, including a bridge between the ARP4754A and DO–178C
3. Management of the architecture variants and reusability of the specification model

The scope of the methodology is presented in Figure 3.1. The methodology covers the complete development cycle from aircraft to airborne software.

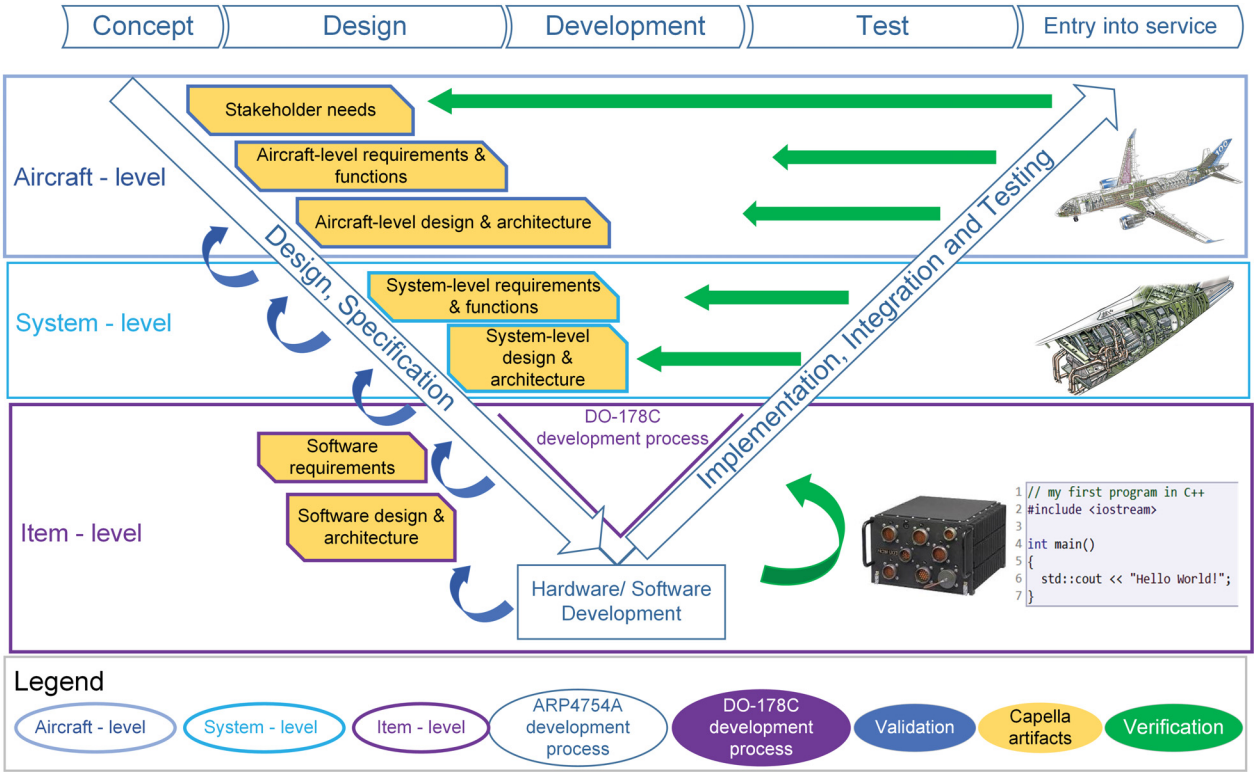

Figure 3.1: Multi-level engineering specification

The multi-level development process mapped to ARCADIA/Capella is presented in the following section.

3.2 MBSE methodology aligned with ARP4754A and DO-178C

This section presents the aerospace guideline and how the guideline can be implemented using proposed methodology.

3.2.1 MBSE for ARP4754A

ARP4754A

As its name says, the ARP4754A is the recommended practice to develop complex aircraft systems. This document is intended to be a guide for both the certification authorities and applicants for certification of highly-integrated or complex systems. As mentioned in Section 1.1, the ARP4754A is based on the V-model of the development process and identifies three main engineering levels of specification namely aircraft-level, system-level and item-level as shown in Figure 3.1.

Aircraft-level: The aircraft-level is the first level, where high-level functions form the idea of the aircraft. Here, each system is a high-level block to meet aircraft requirements. For instance, the system such as the ECS, landing gear, and the primary flight-control system contains only aircraft-level functions without any subsystems. For instance, ECS aircraft-level function can be "Provide a controlled environment," and that of landing gear can be "Provide landing solution and Provide ground maneuvering solution." The aircraft-level model contains all the systems required to meet stakeholders' needs and aircraft-level requirements.

System-level: The second level gives a top-down view with subsystems. It includes interfaces and functions necessary for subsystems to achieve their potential. Here, the ECS subsystems, such as the air-conditioning system, the pressurization system is defined.

Item-level: The third level gives a detailed view of the software and hardware components. Here, ECS subsystem components, IMA hardware components and applications or software are defined. The lowest system-level requirements become (a) system requirements allocated[1] to software (SRATS) and (b) system requirements allocated to hardware (SRATH) at the item-level. For the scope of the thesis, the item-level corresponds to the software application level (ARINC 653 software-based), where the emphasis is on the required controller software code or application. At item-level, the development is handed over to software developers.

[1]Allocation is the process of linking or assigning a detail such as functions or requirements to enable traceability

The software developers will develop the application following the DO-178C.

For each level, one should follow the development process as specified in the ARP4754A guideline and depicted in Figure 3.2

The development process should start with *function development (step 4.2).* In this process step, the functions for the engineering level in question are developed. In the aircraft-level, the aircraft-level functions are developed by aircraft-level engineers. These functions give the overall capability of an aircraft. In the system-level, firstly the predefined aircraft-level functions for a system are acquired, and then the required functions for conceptual design are developed. In item-level, first item or component functions will be acquired, and then required functions are developed for the application to perform fault free.

The next step in the process is the *allocation of functions (step 4.3).* In this step, the developed functions from step 4.2 in ARP4754A are allocated to the system or subsystems. It needs to be identified which system performs which functions.

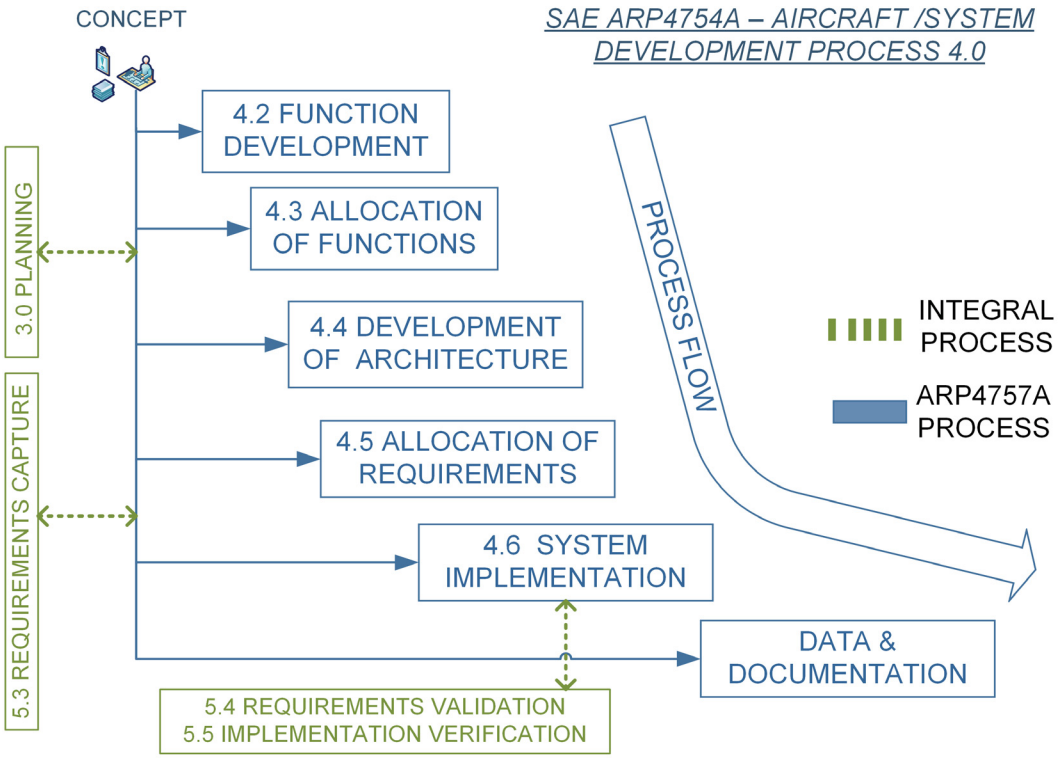

Figure 3.2: SAE ARP4754A aircraft/system development process adapted from [12]

With the functions developed and allocated to systems, the *development of architecture (step 4.4)* is carried out. The definition of architecture establishes the interfaces between systems or components. Therefore, by revealing the interdependencies, early validation of the system architecture can be obtained. Furthermore, the optimization of the architecture, as well as the validation, verification and integration (VVI) is possible.

Allocation of requirements (step 4.5) is an essential step of the process in the early design and analysis phase. Throughout the development, process requirements are captured and allocated to the functions or component that are developed. This capturing and allocation will provide end-to-end traceability and consistency in requirements. The resulting architecture together with allocated requirements will constitute the specification. In addition to the functional requirements, this specification also contains requirements regarding safety, performance, mass, cost etc. The analysis of artifacts such as functions, failure and safety information might result in need of a new functions and requirements to make the system fault tolerant. In such, situations the new requirements are called derived requirements and the development process is repeated until the system complies with the safety requirements.

The next step in the process is *system implementation (step 4.6)*. This step deals with three aspects: (1) the information flow from higher engineering level to lower engineering level and vice versa, (2) system design and built, and (3) system integration. The information flow process includes the exchange of artifacts such as requirements, functions, constraints, descriptions, interface definitions, safety analysis reports and anything that is particular to the system or component. The artifacts transferred through information flow from higher level support the development of lower engineering level specification. If a derived requirement is formed at the item level, then the new requirement is shared with the system-level team for impact analysis and eventually an update of the associated requirements. The final design and build of the system and its components are a part of system implementation. Further, with the integration of the components and systems, a verified integrated system with acceptance[2] is the end product of system implementation.

[2] Acknowledgement by certification authority that module, application, or system complies with its defined requirements

Data & Documentation. The ARP4754A also recommends that every step in the development process along with the data should be documented. The documents carry artifacts, and as mentioned in the 1.1, the documentation process mostly is paper-based. As a result, during system implementation, it is challenging to determine the traceability and consistency of artifacts. Therefore, documentation should be an integral process and support multi-level engineering.

ARCADIA/Capella mapping to ARP4754A

This section presents the proposed MBSE methodology for developing complex aircraft systems with DIMA architecture with ARP4754A.

The first step of the proposed methodology consists of mapping the ARCADIA/Capella against the ARP process shown in Figure 3.2. Each working level can implement the ARP4754A process 4.2 to 4.6 as shown in Figure 3.3.

The first phase 4.2 is *function development* where the functions for the working level in question are developed. This phase is defined as "*Refine System/logical/physical functions, describe functional exchanges*" in Capella activity browser. First, the functions are developed from the requirements. Then a breakdown of the function is carried out. The breakdown helps to realize the leaf functions and parent functions. Next, the dataflow between functions is defined. The flow definition helps to identify the relationship between functions and the exchange happening between them. With the dataflow realized a functional scenario can be created to check the completeness of functions. Any missing function can be developed and update the dataflow diagram.

The next phase 4.3 is the *allocation of functions.* At this step the process is to define the actors that are interacting with the system of interest. The actors can be other systems, or users and are defined as per specified in requirements. Additionally, in LA and PA components are also defined, and a breakdown of components is carried out to realize any leaf component that exists. Finally, the functions are allocated to the components

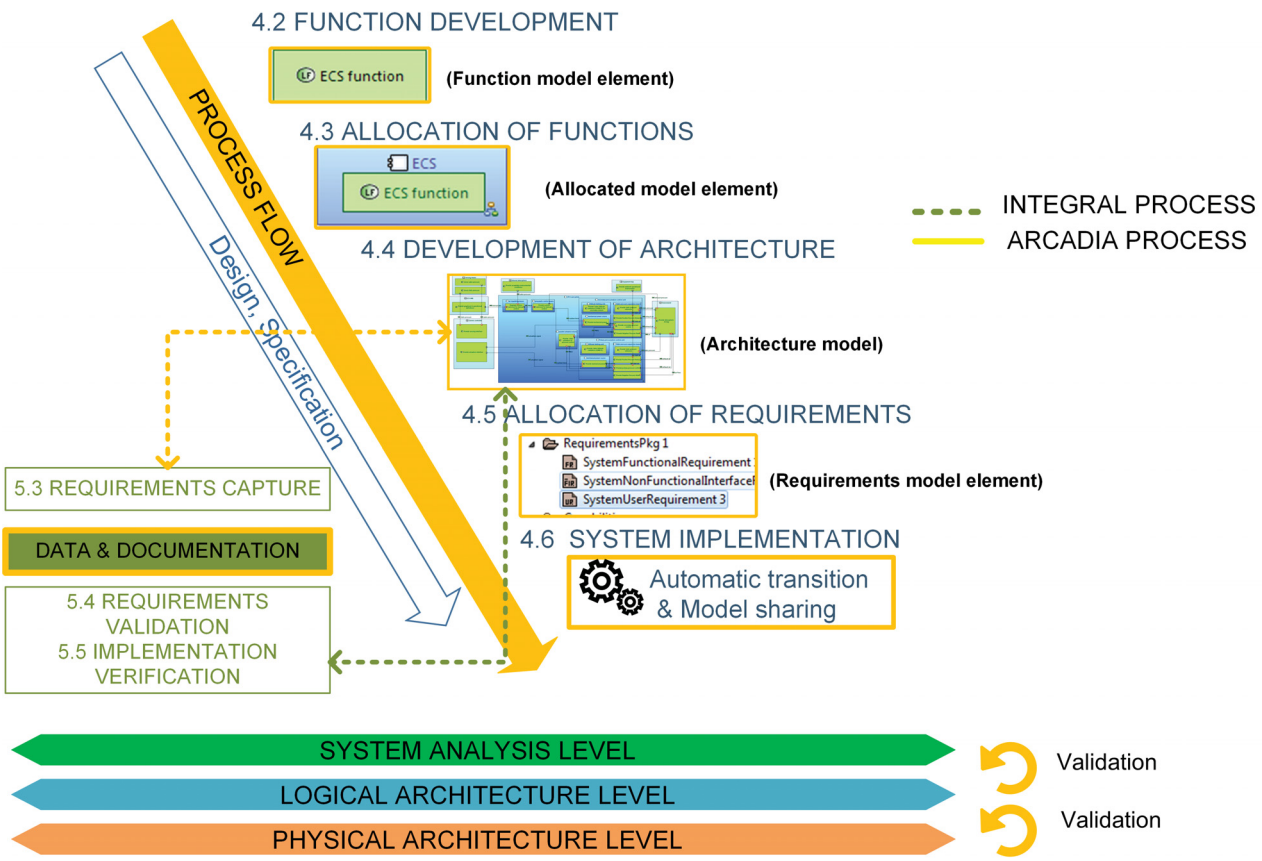

Figure 3.3: ARCADIA implementation of SAE ARP4754A

Next, the *development of architecture* is carried out by defining the interface between the functions and components. While architecture in OA, SA gives the need for the system, architecture in LA and PA provides the logical solution and final implementation of the system. Once the architecture is created an "*exchange scenarios*" can be defined to validate specific use cases. The scenarios help to identify any missing components or actors or functionality. Also, scenarios can realize any element that will not contribute to the needs specified in requirements. Further, Capella also provides architecture validation that ensures every element has up and down traceability in all working levels and follows the architecture specification rules. The validation rules are a set of rules that can include strict naming conventions, interface definitions or user-specified rules.

Capella provides the capability to import requirements files as in step 5.3 from reference-management software such as IBM Doors and also to feed the requirements manually. The

latter manual feeding helps the user to enter requirements in each working level as text manually. This is much needed when a derived requirement is created. Further, the requirements are allocated to the specific functions, components, and interfaces in each working level. This validation ensures that every element is justified to the requirements. The requirements that are not allocated can be realized and also any requirements specified at the wrong level can be identified.

The third phase 4.6 is *system implementation.* System implementation process deals with information flow from higher engineering level to lower engineering level and vice-versa, system design and built, and system integration. However, system implementation in Capella is carried out after each working class. The process is automatic and thus maintain the traceability and consistency in upper and lower working levels. Moreover, the PA working level provides the final system design and built specification.

Further, ARP4754A recommends parallel documentation and data handling, and there are several ways documentation can be integrated into Capella:

1. Using Capella add-ons, which can generate documents of the specification [74] "HTML Document Generation," creating an HTML document of the whole model with each working level and a simple description or "M2doc," [75] which generates documents using a Microsoft Word template defined by the engineer.

2. Using Capella's integrated summary and simple description features to attach as much as information as possible to a function or component or interface, allowing the model to pack any level of description and thereby enriching the architecture.

3. Data-class diagrams are a data structure that stores the information needed in SA. For example, a temperature-data class can have the unit "Degree Celsius," the data parameter "temperature," and the range "[-2,36]." These diagrams help keep track of the data parameters or exchange items.

Each engineering level (aircraft-level, system-level, item-level) has four working levels of ARCADIA (OA, SA, LA, PA). Therefore, ARCADIA/Capella implementation creates triple

times more traceability, consistency and validation for aircraft-level, system-level and item-level specifications. Moreover, during *system implementation,* the artifacts including requirements are passed on through each level and become system requirements allocated to software (SRATS) and system requirements allocated to hardware (SRATH) at the item level. Aircraft-level forms the base for system-level, and the system-level forms the base for item-level. Transition to the next engineering level is called a subsystem transition for future reference in the thesis. Thus, the development process is adapted to each ARCADIA working level in every engineering level.

The ARP4754A does not include specific coverage of detailed software or electronic hardware development/design or safety assessment processes, which is further detailed in the DO-178C.

3.2.2 MBSE for DO-178C

DO-178C

DO-178C covers the development process within the lower level or item-level design and implementation of software. Modern airborne software provides capabilities that reduce the flight crew effort to a minimum. As a result, failures are undesirable in airborne software, and a proper engineering process should be applied in the software development cycle. Figure 3.4 depicts the software development process defined in DO-178C and recommended by the certification authorities.

The system specification drives software development. Hence, the first step in the software development process is to acquire the system requirements that are allocated to software (SRATS). SRATS are then analyzed, and High-Level Requirements (HLR) are developed. HLR specifies what the software must do for the system. The HLRs shows the major functionalities that the software must perform. The HLR includes data flow, software components and their decomposition, behaviour, and physical components such as hardware, network,

and interfaces. The next step is the software design based on Low-Level Requirements (LLR) and software architecture definition. LLRs are developed from HLR, derived requirements and other safety analysis. The LLRs specify how the software works to fulfil its expectation and how to carry out the implementation. For instance, the LLR contains operational scenarios, I/O data, a data dictionary, algorithms, health monitoring, configuration details, communication links and many more. In short, the documentation of the LLRs provides all the necessary information to proceed to the software coding.

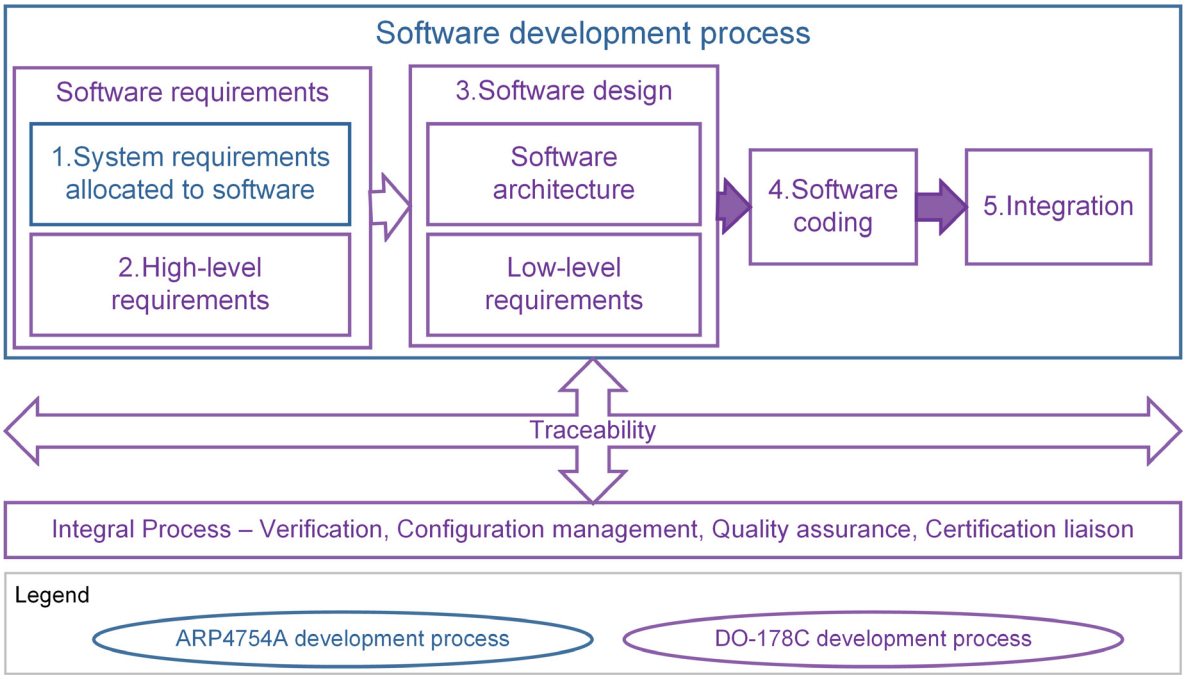

Figure 3.4: Software development process defined by DO-178C

The software architecture provides "the structure of the software selected to implement the software requirements" [67]. The architecture must be clear, consistent, and compatible with requirements, ensuring traceability. The software design step contains all the information needed for software coding. Next, the software is coded according to design specification and integrated into the hardware.

The DO-331 is the Model-based development supplement guideline which is a supplement to DO-178C for implementing MBD. The guideline defines a model as: "*An abstract representation of a set of software aspects of a system that is used to support the software development*

process or the software verification process" [68]. In a model-based approach, software requirements are called specification model and software design artifacts are called design model. The guidelines strictly prohibit the existence of specification and design in the same model as there should be a clear differentiation between "what" (requirements) and "how" (design) aspects. The specification model should only point out the functionality of the software, not its implementation. Whereas, the design model should point out the software components, data structure, data, and control flow. The guideline states that the model should have the capability to be validated through coverage analysis for correctness and completeness. The coverage analysis will help to find out if any requirements or design elements are left out. Moreover, the analysis also helps to identify those model elements that do not contribute to requirements or implementation [68].

The ANNEX A in DO-178C gives a set of tables that point to the process objectives and outputs by software level. Table 3.1 shows an overview of verification of outputs of software design in Table A-4 in ANNEX. To proceed to the software coding phase, the objectives should be satisfied with or without independence depending upon the critical nature of the software.

The aerospace industry usually follows two design approaches for software design: structured-based and object-oriented. Structure-based is the traditional approach and is flow-oriented. The flow-oriented diagrams show the flow of information or data through the system and how the information is transformed from input to output when moved through the system. In short, structure-based focus on the action and logic required to manipulate data. As a result, programs are a long piece of code containing the logic and data.

On the other hand, the Object-Oriented (OO) approach gives importance to the object and data that needed to be manipulated, not to the logic required to manipulate. Objects are a group of variables and functions that are related to a unit or class. For instance, a class called "*aircraft characteristics*" will have variables such as "*length, height, speed*" and function may be "*calculate take off length.*" Then an object can be different models of Boeing 787 such as "*787-8, 787-9, 787-10*". Further, each model will have variables defined in *aircraft*

characteristics class, and each model can use the function *calculate take off the length* to determine the required runway length. In OO, the program is split into objects, and each object represents a part of the application and contains its own data and logic. As a result, OO programs are usually modular and easy to analyze.

Table 3.1: Verification of outputs of the software design process (adapted from DO-178C) [67]

Objective	**Description**
A-4.1 Compliance	Low-level requirements comply with high-level requirements
A-4.2 Accuracy & consistency	Low-level requirements are accurate and consistent
A-4.3 Hardware compatibility	Low-level requirements are compatible with the target computer
A-4.4 Verifiability	Low-level requirements are verifiable
A-4.5 Conformance	Low-level requirements conform to standard
A-4.6 Traceability	Low-level requirements are traceable to high-level requirements
A-4.7 Algorithm accuracy	Algorithms are accurate
A-4. 8 Architecture compatibility	Software architecture is compatible with high-level requirements
A-4.9 Consistency	software architecture is consistent
A-4.10 Hardware compatibility	Software architecture is compatible with the target computer
A-4.11 Verifiability	Software architecture is verifiable
A-4.12 Conformance	Software architecture conforms to standards
A-4.13 Partition integrity	Software partitioning integrity is confirmed

ARCADIA/Capella mapping to DO-178C

For a DIMA architecture, the item-level (the application/software/controller level) specification is essential. For the scope of the thesis, more focus is given on the software aspects than hardware. The working levels also match with the software development process defined in DO-178C guideline. When using ARCADIA/Capella for software specification, SA deals with the first phase of process, software requirements. In SA, SRATS will be imported, and HLRs will be developed along with high-level functions and architecture. Next, an automatic transition is performed to LA to define the software design. First LLRs are defined, and then software components and functions are developed. Next, the software architecture is created, and LLRs are allocated to elements. The LA can contain a set of logical solutions, and after analysis, the solution is forwarded to PA to define the software design implementation. Moreover, Capella supports the process objectives and outputs by software level as shown in Figure 3.5 through its features and additional tools.

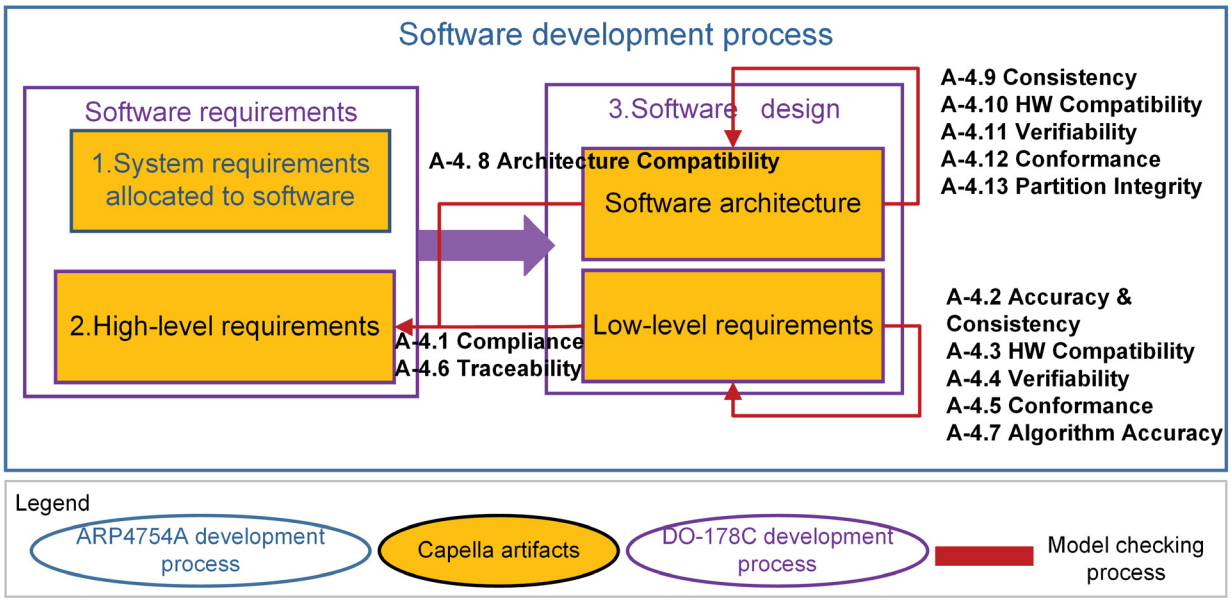

Figure 3.5: Model-Based checking process inspired by [76]

Thus, Capella supports software development process through DO-178C implementation and providing object-oriented design and structural design diagrams.

3.2.3 Multi-level approach

As mentioned in shown in Figure 3.1, function development needs to be performed at multiple engineering levels. The organization of the aircraft development process has three levels as mentioned in 3.2.1. The ARCADIA method focuses on the relationship between the various levels in the system architecting process[3]. However, the complexity of the aircraft development necessitates a common aircraft reference (such as functions, requirements, etc.), which becomes the starting point for (sub-)system architecting. Figure 3.6 gives a detailed view of how it is proposed to implement a multi-level engineering process with Capella for the example of the ECS system development. However, this method is also applicable to any other aircraft system.

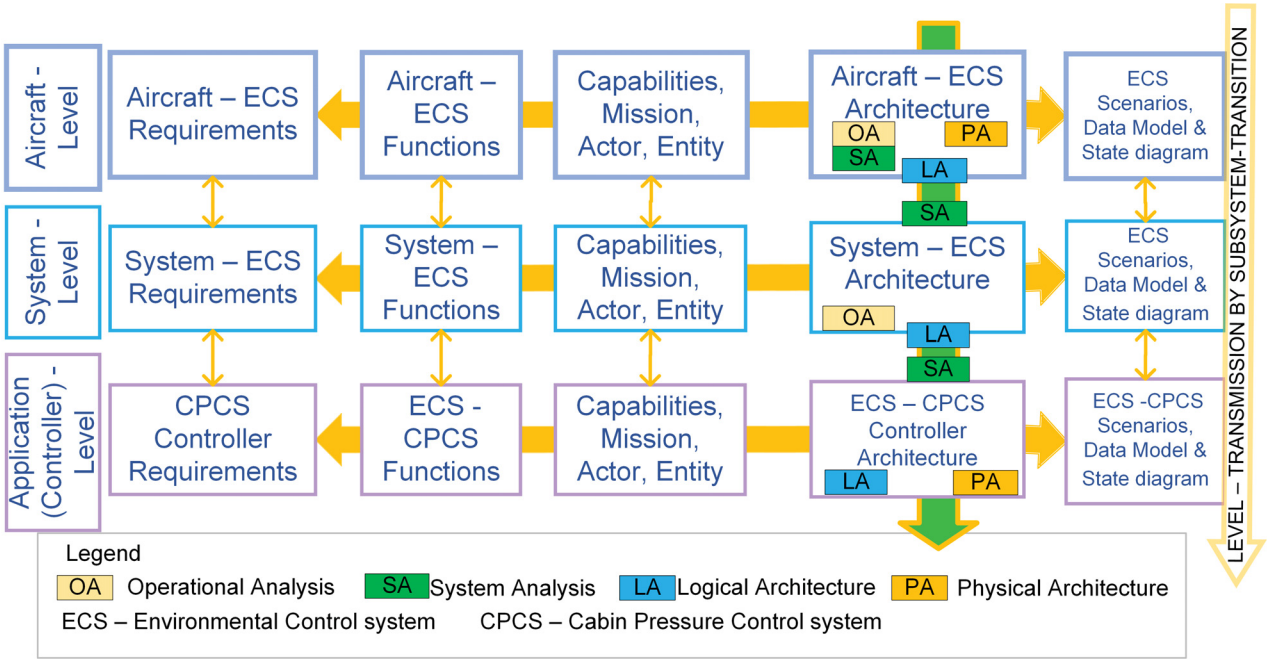

Figure 3.6: A multi-level process for MBSE implementation

The green arrow shows the level transition, and the yellow ones show the traceability. Capella can also define scenarios and data structures needed for early validation and verification. To inherit effectively the interfaces, requirements, and interdependencies, a subsystem transition is carried out from LA to SA. This method chooses a component in LA and makes it the

[3]System architecting is the specification process to derive a solution at each engineering level

system of interest at SA in a new model. For the case study, the ECS SA for the system-level is obtained by performing a subsystem transition from the aircraft-level LA. Then, the ECS functions, capabilities, actors, and entities are defined, requirements are allocated, and the architecture is specified. Similarly, at the item level (software/hardware), the SA is generated from the system-level LA. The inherited requirements are then divided into requirements allocated to software or hardware, which are high-level requirements (HLRs) specifying what is to be implemented. The HLRs are then used to develop low-level requirements (LLR) and, in turn, the software/hardware architecture, according to the Radio Technical Commission for Aeronautics (RTCA).

3.2.4 Management of variants

The case study investigates how few variants can be efficiently managed within the MBSE environment using Capella. In the herein presented work, a generic ECS architecture is presented, and the creation of two architecture variants are investigated: a conventional bleed-driven ECS and a bleedless ECS variant as presented in Figure 1.7. Two approaches are investigated: the so-called *horizontal adaptation* and *vertical transition*.

Figure 3.7 presents a horizontal adaptation method. This adaption is performed at LA level to define variants of a specific aircraft-level function. Using the principle of horizontal adaptation, multiple technological implementations can be derived. For example, Figure 19 presents a horizontal adaption of aircraft-level function “Provide fresh air” to derive two different technological implementations of ECS namely bleed ECS and bleedless ECS. The implementations are defined as leaf dataflow diagrams. As a result, no subcomponents are present. Here, the focus is on the air-conditioning subsystem and not on the subsystem components. Thus, both diagrams co-exist at LA level.

This method provides insights into solutions without losing sight of the overall picture. That is, a conventional ECS and a bleedless ECS air-conditioning process can be defined in a single parent aircraft-level function with two distinct, “unsynchronized” data-flow models.

Capella facilitates the creation of new diagrams within the same working level by preventing synchronization. However, when synchronized, the leaf data-flow diagrams merge and result in unrealized interactions for a specific implementation.

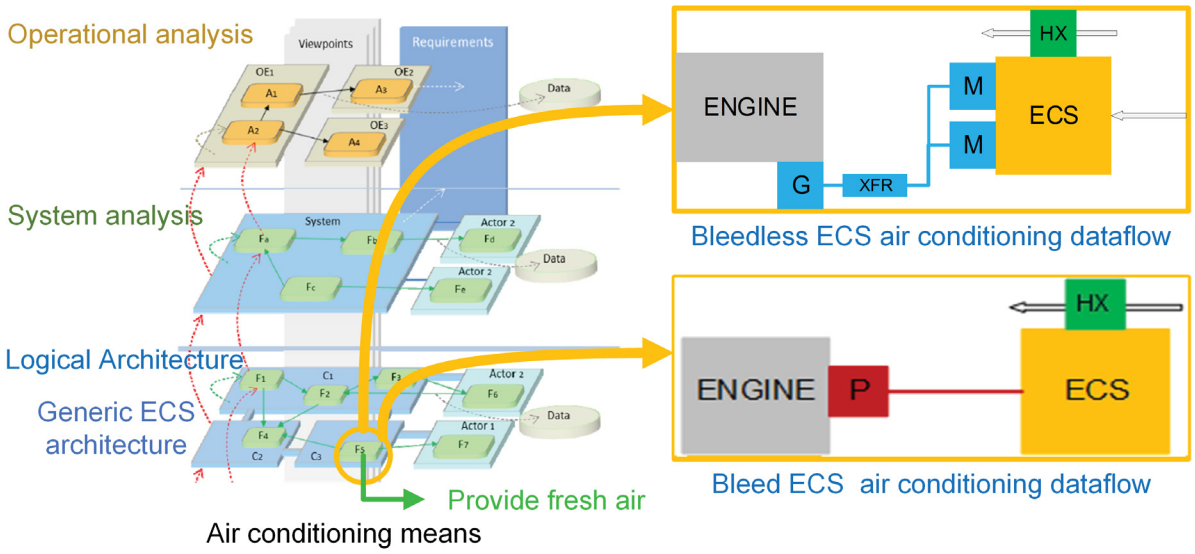

Figure 3.7: Horizontal adaptation of generic architecture

To conclude, *Horizontal adaptation* is the process of using generic architecture for system specifications to maintain abstraction while defining the process of subsystems through encapsulation. That is, without emphasizing subcomponents, the functions of the subsystem are encapsulated in the form of leaf data-flow diagrams into the parent functions of the main components. For effective system engineering, each subsystem requires a separate model. Therefore, horizontal adaptation is not suitable for sub-system specifications. Horizontal adaptation is only used when the specification of subcomponents or behaviour is not necessary. In the ECS case study, this method is applied only for the controller and aircraft-energy specifications.

The next, and more suitable, method is *vertical transition.* Figure 3.8 shows the vertical transition method. As the name suggests, the transition enable a top-down approach to move from higher engineering level to lower engineering level. Using the vertical transition method, a detailed specification of a sub-system can be derived. For instance, the "Air-conditioning means" component along with the function "Provide fresh air" is selected to derive detailed

bleed ECS and bleedless ECS subsystem specification. As a result, the sub-system specification realizes subcomponents such as compressors, heat exchangers, and turbines in the air-conditioning system.

The Capella subsystem-transition add-on is used for this purpose, and the selected subsystem is transformed into the system of interest in the SA. From there, LA and PA are defined with realized subcomponents. The *vertical transition (LA to SA)* method defines in detail the low-level architecture for the main components by realizing the subcomponents. The advantage of vertical transition is that the main sibling components (components in the LA that directly interact with the selected subsystem) become system actors. In this way, each subsystem can be modelled with subcomponents and only realize the input-output interfaces. Thus, interdependencies and interfaces are inherited in the new model. Furthermore, the vertical transition method renders the model clean and efficient. The vertical transition is more suitable for specifying a subsystem because only the sibling subsystem, component, or actor is realized, which allows for more focus on the subsystem and its interfaces.

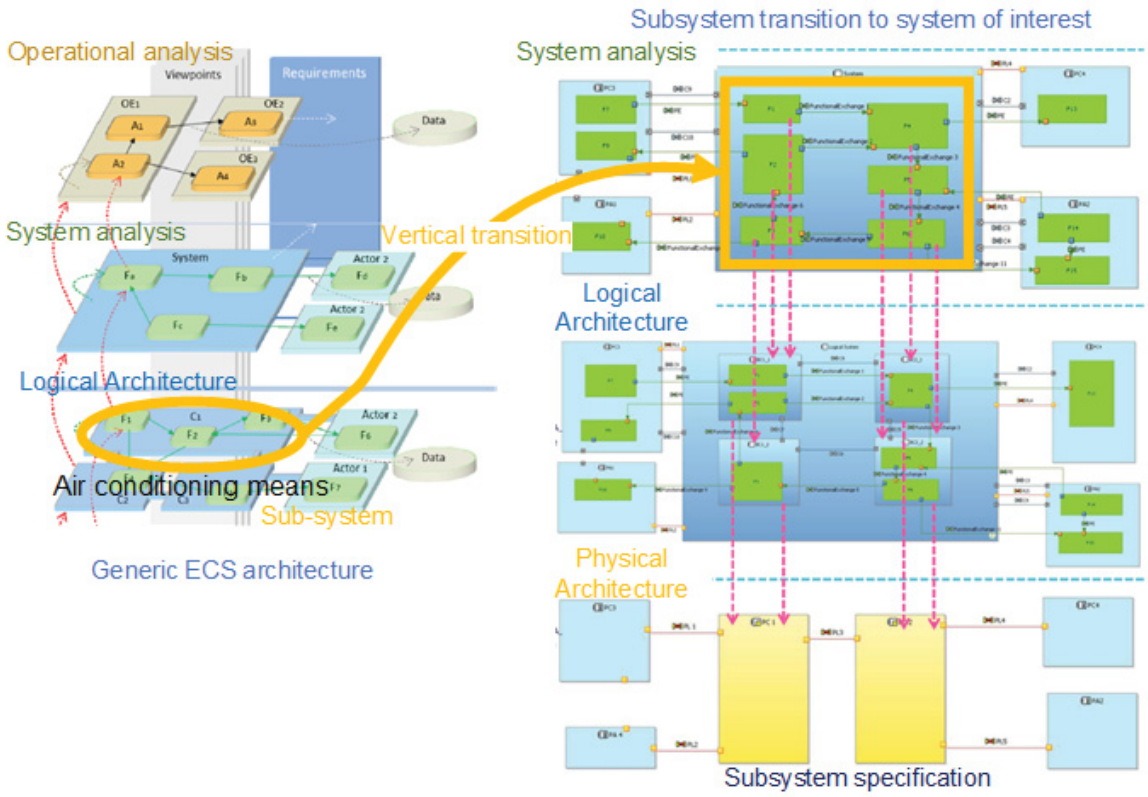

Figure 3.8: The vertical transition method for variability management

Figure 3.9 presents a comparison between the vertical transition and horizontal adaptation method. In vertical transition, an overview of the entire ECS system is not possible. This method also requires the definition of an additional interface in the generic architecture before creating the vertical transition in order to be consistent with the new implementation. On the other hand, horizontal adaptation provides insight to the solution without losing system overview. However, for effective system engineering, each subsystem requires a separate model. Therefore, horizontal adaptation is not suitable for sub-system specifications. Horizontal adaptation is only used when the specification of subcomponents or behaviour is not necessary. In the ECS case study, this method is applied only for the controller and aircraft-energy specifications.

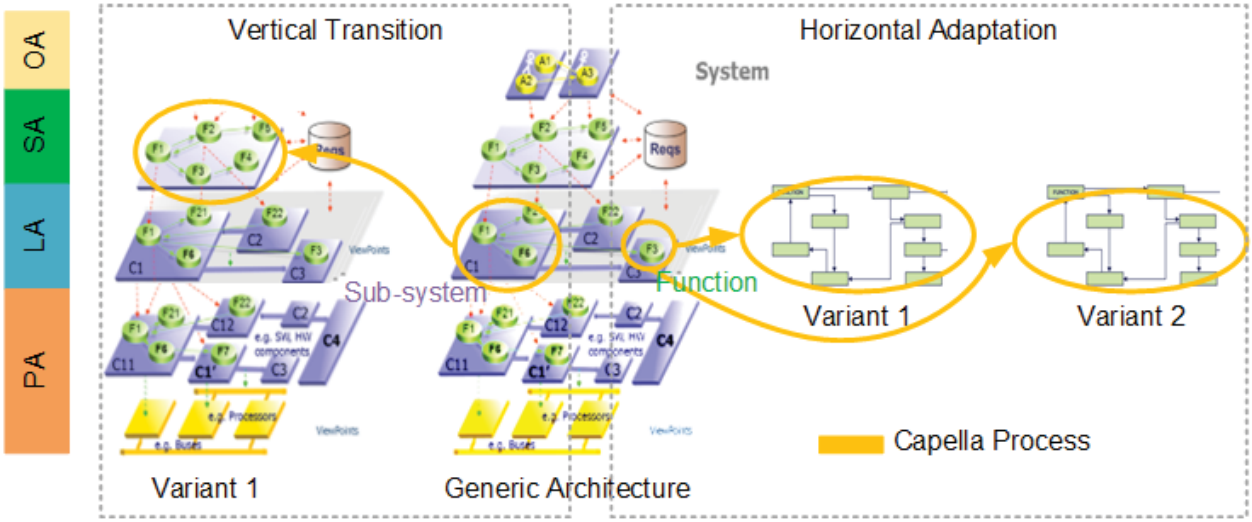

Figure 3.9: Comparison between vertical transition and horizontal adaptation method

3.3 Summary

This Chapter presented the proposed methodology for developing complex aircraft systems using MBSE Framework. The methodology is based on two main aerospace guidelines: ARP4754A and DO-178C. Hence, the proposed methodology addresses the ARP4754 development process with multi-level engineering and DO-178C development process with DO-331

model-based supplement. To support the variability aspect two methods are defined: horizontal adaptation and vertical transition. The horizontal adaptation can provide an insight to the solution without losing the overview of the system. The vertical transition provides a detailed low-level view of the sub-system. Although the methodology is defined for ARCADIA/Capella, any MBSE framework can adopt the methodology for aerospace system development.

In sum, the effectiveness presented methodology of dealing with generic and variant models needs to be explored. However, subsystem engineers are required to not miss any inconsistencies on the system level. In the following Chapter, the vertical transition method is implemented for sub-system specification, and horizontal adaption is implemented to specify certain aspects of sub-systems.

Chapter 4

Specification Model Implementation

This chapter describes the MBSE implementation for the ECS case study following the methodology described in Chapter 3. As depicted in Figure 4.1, all system architecting levels are covered in this case study, but the emphasis is on system-level and item-level modelling.

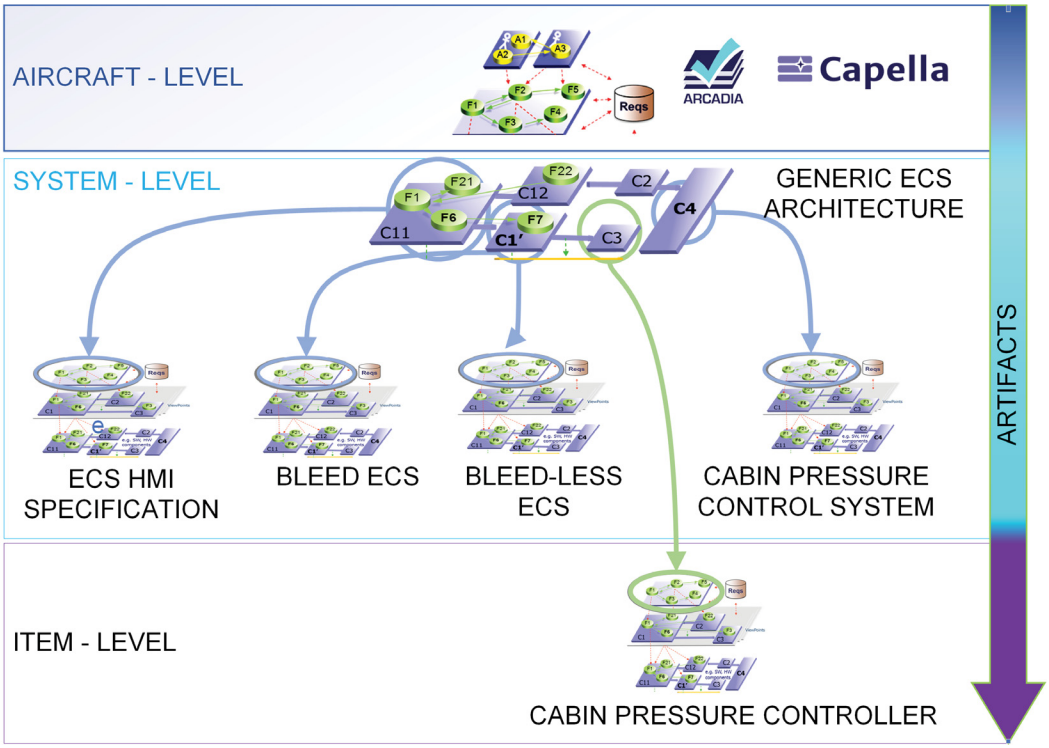

Figure 4.1: ECS case study implementation overview in Capella

The process begins at aircraft-level at which all systems are in a model and represent the context to the ECS. Then, a vertical transition is performed to obtain modelling artifacts specific to the ECS system at system-level. Next, the ECS generic architecture is developed.

From the generic architecture, the ECS HMI specification, bleed and bleedless ECS variants and the cabin pressurization system specification are developed. Further, a vertical transition is performed on the controller component to obtain system SRATS at the item level. Finally, the cabin pressure controller specification is developed from SRATS.

4.1 Aircraft-level specification

For the ECS case study, the relevant aircraft-level functions and requirements developed by the Bombardier aircraft-level engineers were reviewed. The functions were formulated using company internal manuals and were specified in spreadsheets. Moreover, the requirements were handled through word documents and reference management add ons. Table 4.1 provides an insight into the aircraft-level functions adapted from [77].

Table 4.1: List of aircraft level function for ECS use case

	Aircraft-Level functions
1	**Provide operational awareness**
1.a	Provide HMI for flight crew
1.b	Provide HMI for cabin crew
1.c	Provide HMI for Maintenance crew
2	**Provide centralized computing and data sharing capabilities**
3	**Generate and distribute power**
3.1	provide pneumatic power generation and distribution
3.2	provide electric power generation and distribution
4	**Provide a controlled environment**
4.a	Control internal ambient temperature
4.b	Control internal ambient pressure
4.c	Provide and control air quality
4.d	Provide equipment cooling

There exist a full aircraft-level architecture model in Capella. However, this model was not matured enough to perform a vertical transition, as described in Figure 3.6. Therefore, it was decided to obtain aircraft-level functions and requirements from aircraft-level team and start modelling at system-level. The data from aircraft-level will be specified at OA and SA as a need analysis for ECS. The reason is that the scenario will be the same if a vertical transition of the ECS aircraft-level component was performed. The transition will result in ending up of logical aircraft-level ECS functions in system-level SA. The additional task will be to define the operational analysis.

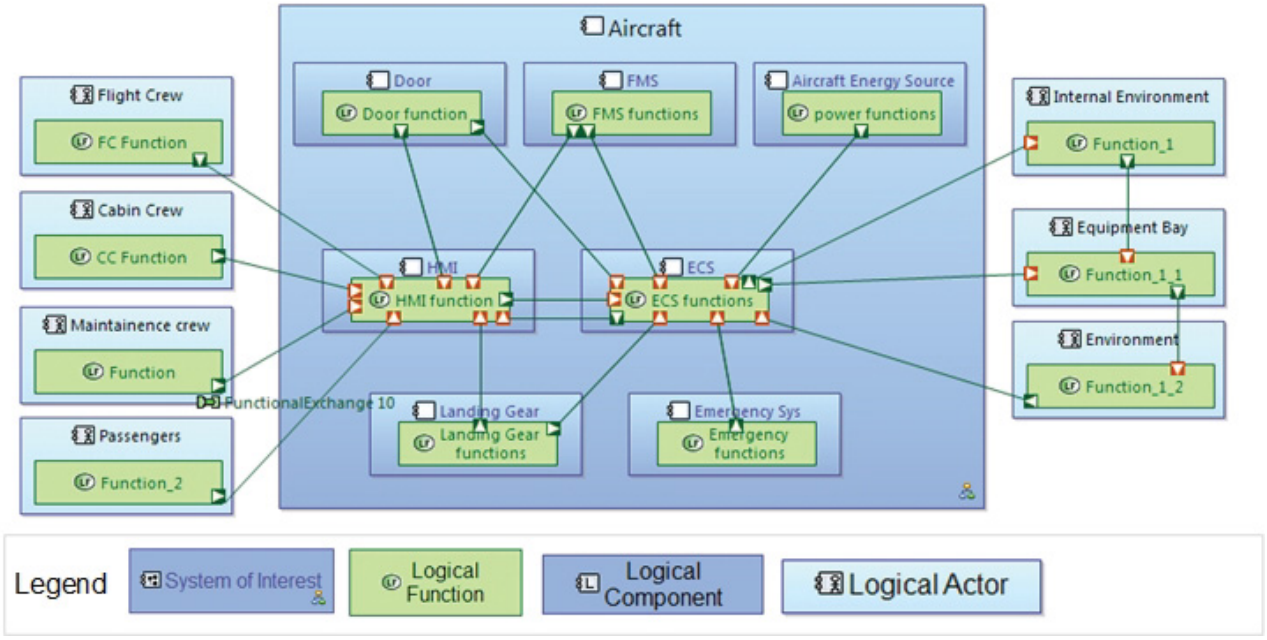

Figure 4.2: Example of the aircraft-level logical architecture

Figure 4.2 representative to an aircraft-level logical architecture with aircraft as a system of interest and ECS as a component. When the vertical transition is performed on the ECS component, the resulting SA will be the same as presented in subsection 4.2.2

4.2 System-level specification

In this section, model specification using ARCADIA/Capella at system-level is presented.

4.2.1 Operational analysis level

The objective of the operational analysis (OA) is to identify the operational needs and objectives of users of the system, such as the pilot, maintenance engineers, and passengers. The aim is to capture all user needs to guarantee the adequacy of the system. An OA must define the activities of the users and the operational scenario in which they perform the activity.

According to the multi-level methodology presented in Figure 3.6, the OA for the system-level (ECS) is derived from the aircraft level to ensure traceability, by using the vertical transition. If a complete aircraft-level is not available in the MBSE environment, the OA is based on aircraft-level specification documents and reviews with stakeholders.

The first step is to define the operational actors and operational entities of the system with reference to aircraft-level requirements. For the ECS case study, flight crew, cabin crew, maintenance crew and passengers are the identified actors. Whereas aircraft, aircraft energy source, internal environment, equipment bay, environment, other systems, and airport facilities are the identified operational entities. The next step is to define the operational capabilities of the identified actors/entities. The operational capabilities define the capability of an actor or entities to deliver a high-level service to fulfil an operational need. For example, for ECS case study the flight crew should have the capability to regulate internal ambient temperature and pressure. The operational capabilities of the users and entities can be captured in an Operational Capability Blank diagram (OCB), as shown in Figure 4.3. These capabilities help users provide high-level services to achieve operational need of providing a comfortable internal ambient atmosphere.

For the users or entities to achieve their capability, the users and entities are required to perform some operational activities. Operational activities are actions carried out by the user to deliver a high-level service. Therefore, the next step is to define the activities of users/entities. The list of activities is captured in the Operational Activities Breakdown Diagram (OABD), as shown in Figure 4.4.

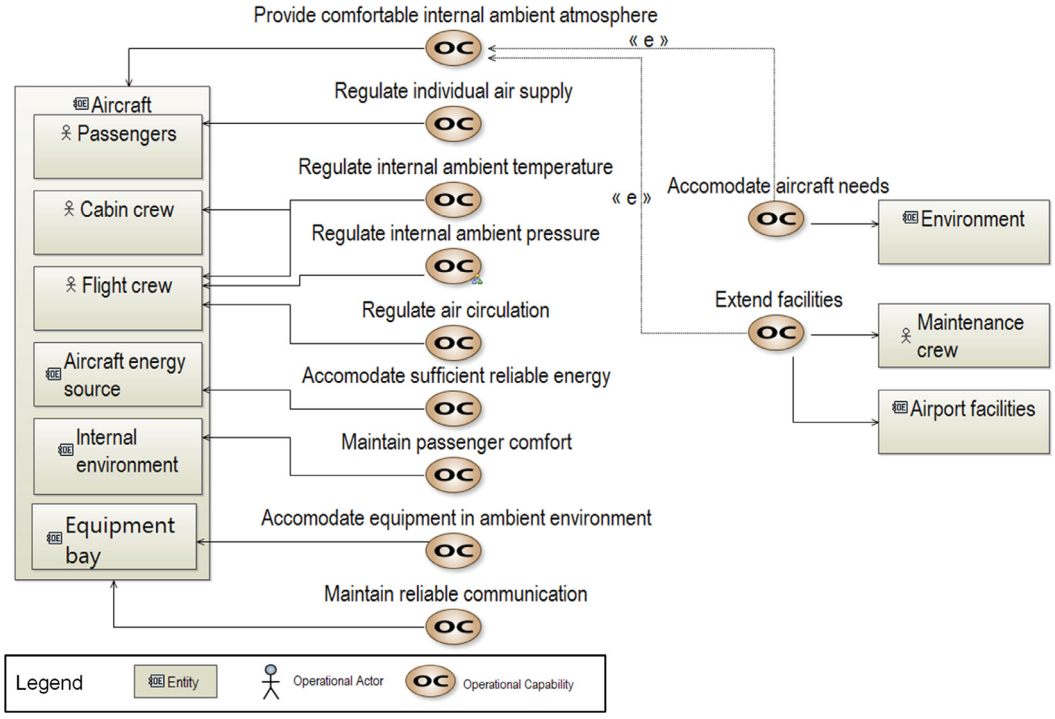

Figure 4.3: Operational capabilities of users (operational capability – OCB)

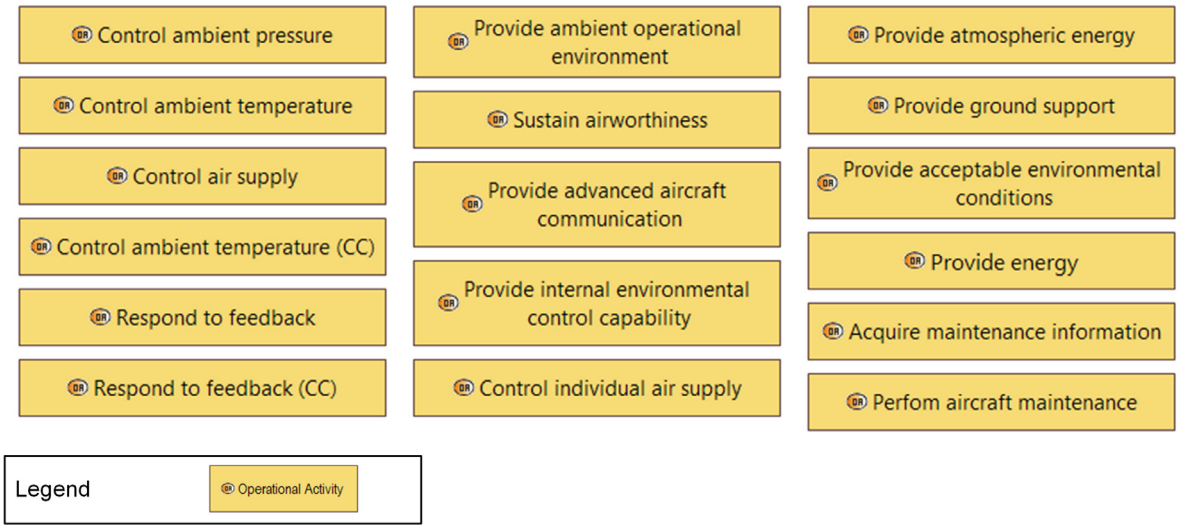

Figure 4.4: Operational activities of users (OABD – Operational Activities Breakdown Diagram)

Further, the interaction between activities is specified. This specification helps to understand how the activities perform together to deliver the operational need. One way is to specify the operational process for a specific capability. The operation process defines all the operational

activities and interactions that contribute to a specific process or capability. The interactions are defined using the Operational Activity Interaction Blank diagram (OAIB) and shown in Figure 4.5.

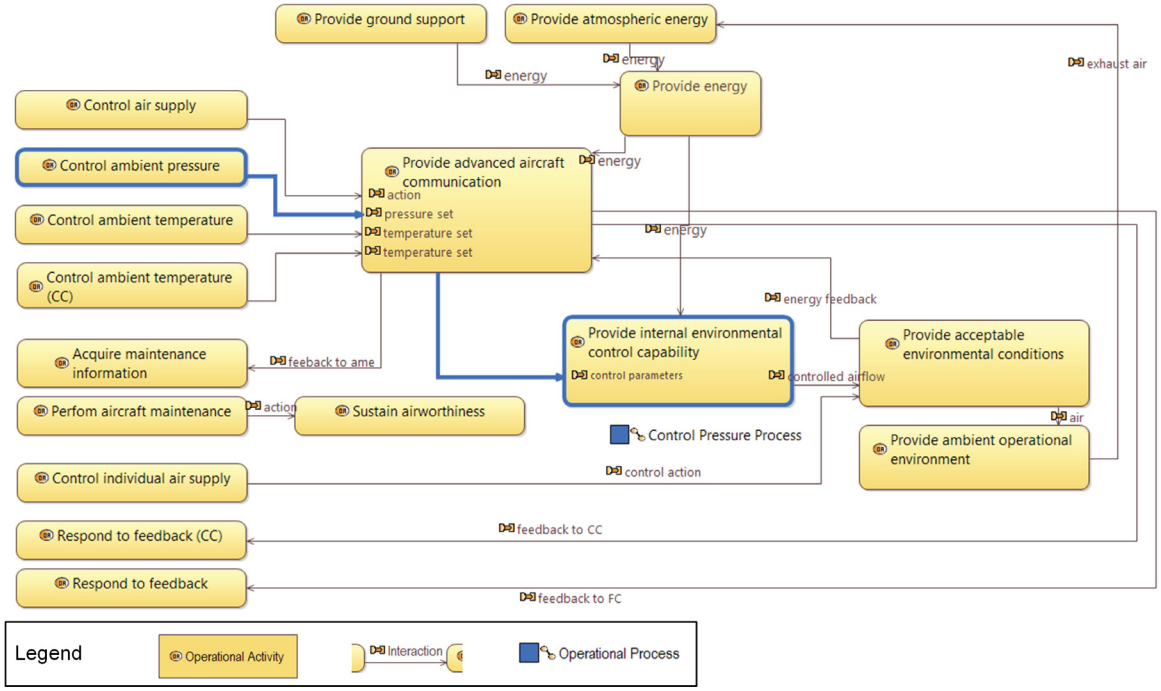

Figure 4.5: Interactions between ECS operational activities - (OAIB) diagram

The OAIB highlights the identified operational process. The next step is to allocate the activities to concerned users to complete the operational architecture. An example of the operational architecture block diagram for the generic ECS is presented in Figure 4.6. The activity "Provide internal environmental control capability" represents the ECS system. The OA focuses on the DIMA use case for the CPCS. In Figure 4.6, the blue line shows the pressurization control process chain that needs to be carried out to meet the required operational objective of cabin pressure control. The flight crew initiate the activity "ambient control pressure (1)". The activity "provide advanced aircraft communication (2)" accounts for the HMI and controller aspects. The activity "Provide internal environmental capability (3)" completes the operational objective. The final step is to validate the diagram. Validation makes sure that the architecture followed all the system architecting rules. The validation rules are explained in section 4.4. When performed for the first time, Capella validation will

give warning that the model elements are not realized in the higher working level (SA). Once the SA is completed the systems engineer is supposed to come back to OA and validate the diagram to make sure every element is traced.

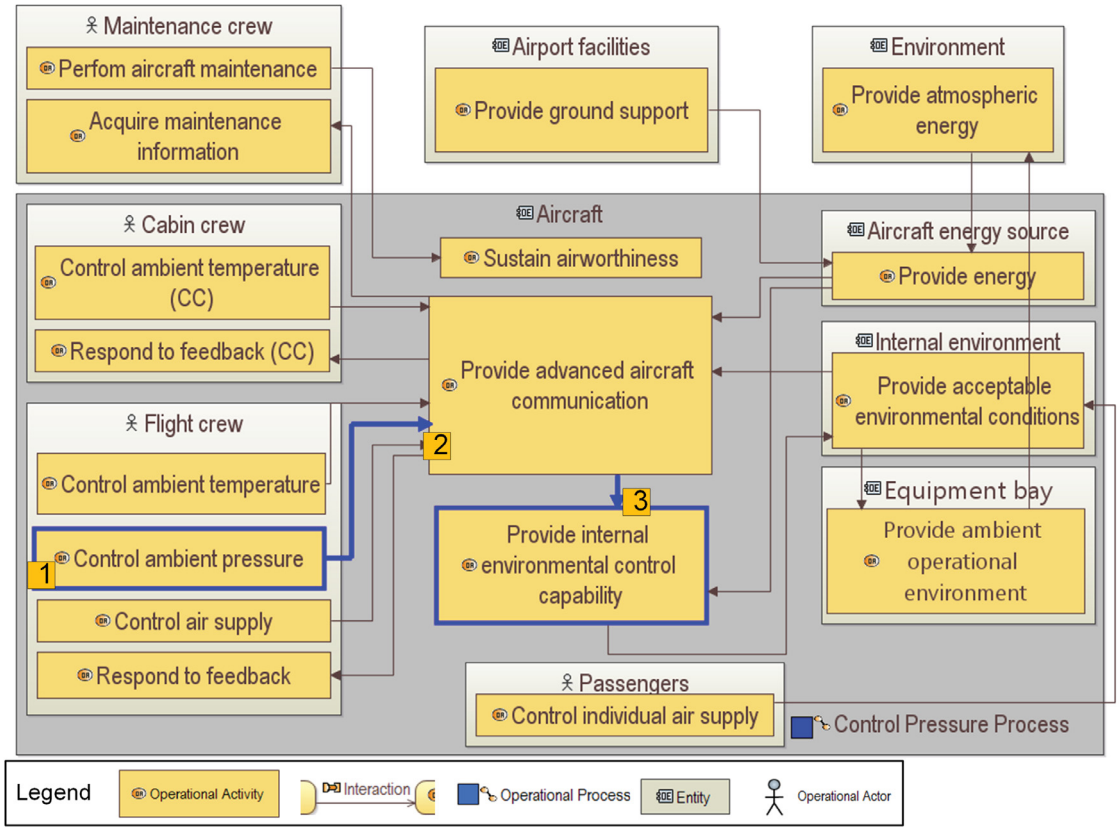

Figure 4.6: Operational architecture for the ECS (operational architecture –OAB)

4.2.2 System analysis level

The SA reflects the aircraft-level logical ECS functions. At SA the first process is to acquire the operational needs defined in OA through the automated transition. Therefore, the operational activities, operational entities, actors and requirements are acquired. During the transition the activities are converted to functions. This is to maintain the traceability of operational needs and support aircraft-level functional analysis. Also, the operational actors and entities are transformed into system actors. The reason is that in a system engineering

perspective anything other than the system of interest are actors. A representative transition is shown in Table 4.2.

Table 4.2: Transition of OA elements to SA

Operational Analysis	**System Analysis**
Operational actors	**System actors**
Flight crew	Flight crew
Cabin crew	Cabin crew
Passengers	Passengers
Maintenance crew	Maintenance crew
Operational entities	
Aircraft	Aircraft
Aircraft energy source	Aircraft energy source
Equipment bay	Equipment bay
Internal environment	Internal environment
Environment	Environment
Airport facilities	Airport facilities
Activities	**Actor functions**
Control ambient temperature (CC)	Control ambient temperature (CC)
Respond to feedback (CC)	Respond to feedback (CC)
Control ambient temperature	Control ambient temperature
Control air supply	Control air supply
Provide advanced aircraft communication	Provide advanced aircraft communication
Provide acceptable environmental conditions	Provide acceptable environmental conditions
Provide ambient operational environment	Provide ambient operational environment
Control ambient pressure	Control ambient pressure
Control air supply	Control air supply
Respond to feedback	Respond to feedback
Operational interactions	**Function exchanges**

In SA, the system of interest appears for the first time and also begin the functional analysis stage. The second step is to define the mission for the ECS system along with the actors and capability of the system to accomplish the mission. For instance, the actors accomplish the ECS mission, which is to provide a comfortable internal ambient atmosphere. The actors use their identified capabilities to fulfil the mission, as shown in Figure 4.7.

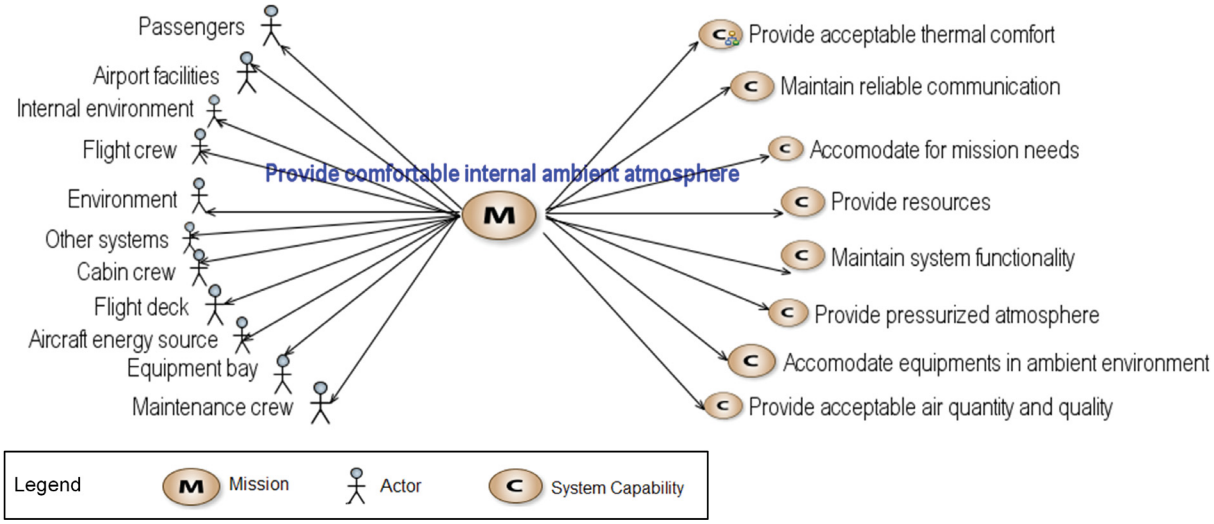

Figure 4.7: ECS mission with system capabilities (mission capabilities -MCB)

System capabilities provide the objectives for use case. In Capella, the use case or capability is expressed mainly through functional chains and scenarios. More about functional chains and scenarios have been presented in Section 4.4. The aircraft-level functions obtained are controlling pressurization, providing and controlling ventilation, providing equipment cooling, and controlling internal ambient temperature. These functions will be leaf functions to the function "Provide internal environmental control capability". Apart from aircraft-level ECS functions, functions obtained through the automatic transition from the OA as shown in Table 4.2 are also present. The information flow between the aircraft-level functions and actors is defined as data-flow diagrams and is categorized into control, energy, and feedback flows. Then functions are allocated to their respective actors and ECS system, and the ECS system architecture is created as shown in Figure 4.8.

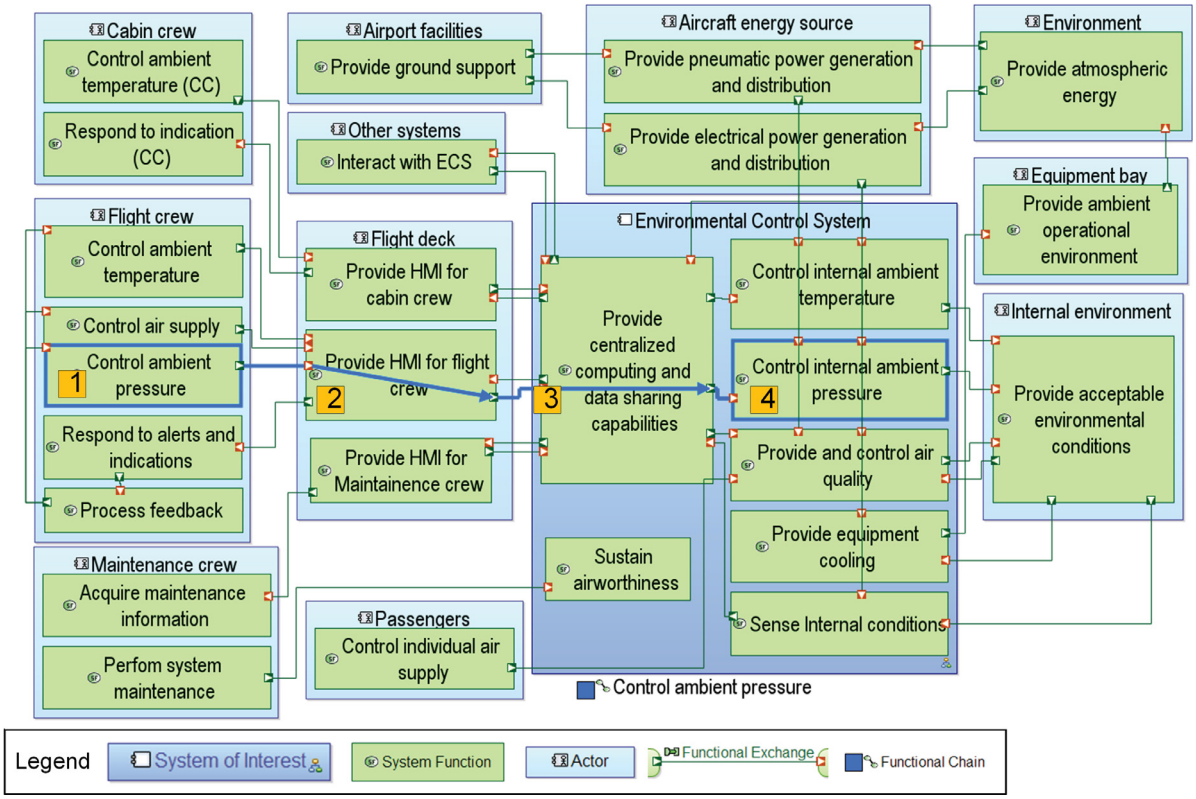

Figure 4.8: System analysis for the ECS (system architecture – SAB)

In system architecture diagram 4.8, the ECS is the system of interest in dark blue. The functions within reflect what the ECS has to do for the users. The users are *flight crew, cabin crew, maintenance crew* and *passengers* that are interacting with ECS. They interact with the system through the flight deck. Further, the aircraft energy source provides the required pneumatic and/or electrical power. The need for IMA is expressed through the function "Provide centralized computing and data sharing capabilities." The blue line shown in Figure 4.8 is the functional control chain to express cabin pressure control functionality. As numbered, the first function in the functional chain is the command by the flight crew to "Control ambient pressure (1)". The HMI provides an interface with the ECS system (2). Then, the function "Provide centralized computing and data sharing capabilities (3)" processes the command and forwards it to the function "Control internal ambient pressure (4)". To summarize, the functional chain shows the pressurization control flow and what the system has to do for flight crew is to control internal ambient pressure.

The final step of SA is to validate the system architecture. The process makes sure that every model element has traceability to the operational need specified in OA. After validation, the process can proceed to LA.

4.2.3 Logical architecture level

The aim of the LA is to define a generic architecture for ECS that can be used to define variants. In general, LA aims to define logical components and conduct a functional analysis inside the system. While functions in SA defines the need in terms of what ECS has to do; the functions in LA defines the solution in terms of how the has to perform. First, the artifacts from SA are transitioned through Capella automated transition. As a result, all the elements in SA are available in LA as logical elements. This helps to define how the system works to fulfil the expectations. Next, the technical TRD requirements are imported into the LA through requirements management add ons as presented in 4.4. As mentioned in 2.1, the TRD documents provide derived system-level requirements needed for defining logical elements. Further, the TRD level requirements help identify all the actors interacting with the system. Some of these actors are generalized in SA as "other systems."

The next step is updating the logical functions, and all functions necessary to describe how the system works must be present. In this stage, more system specific functions are developed. Then, a dataflow diagram is developed to define the interaction between functions. Next, functional chains are defined to highlight control, feedback, and resource flows. The data flow diagrams provide consequence loops help determine the safety aspects and hazard levels. Finally, all the logical components are developed to represent the sub-systems. With the defined functions allocated to components, the generic ECS architecture is created. The generic ECS architecture includes a generic controller component, air-conditioning components, pressurization components, distribution components, and emergency components, as shown in Figure 4.9.

To account for a real-time operating system that hosts the system application software, a

generic controller model is adapted and introduced as a controller component to the LA as highlighted with the red box in Figure 30. The controller has three subcomponents:

1. *Operational interface component:* The operational interface acts as an interface between the controller and other systems. The data needed for control and feedback are often delivered through the operational interface.

2. *Control and management component:* This component executes the management of the ECS and controls the system components.

3. *Physical interface component:* This component provides an interface between the controller and the physical system being actuated or controlled (PA level).

The subcomponents are processes or tasks defined in software that can be allocated to different host platforms. It is a realization of the ARINC 653 implementation of allocation, assigning pieces of software to different applications in a distributed system. For instance, the physical interface can be allocated to a host platform close to sensors, or the operational interface close to the cockpit. The red container in Figure 4.9 also contains an ECS HMI component, which again is a controller component that manages the HMI functionality.

The blue line in Figure 4.9 shows the functional chain for control. The function chain presents the functions and interactions that contribute to control cabin pressurization. As numbered, the chain starts with flight crew giving the command to "control ambient pressure (1)". The flight deck provides an interface between crew and system (2). The ECS HMI component provides an operational definition to the crew command (3). Further, the command is forwarded to the ECS controller. The controller processes the command (5) and performs the control algorithm (6). Finally, the controller sends the actuation signal to the CPCS (7) and pressurization means control the cabin pressure.

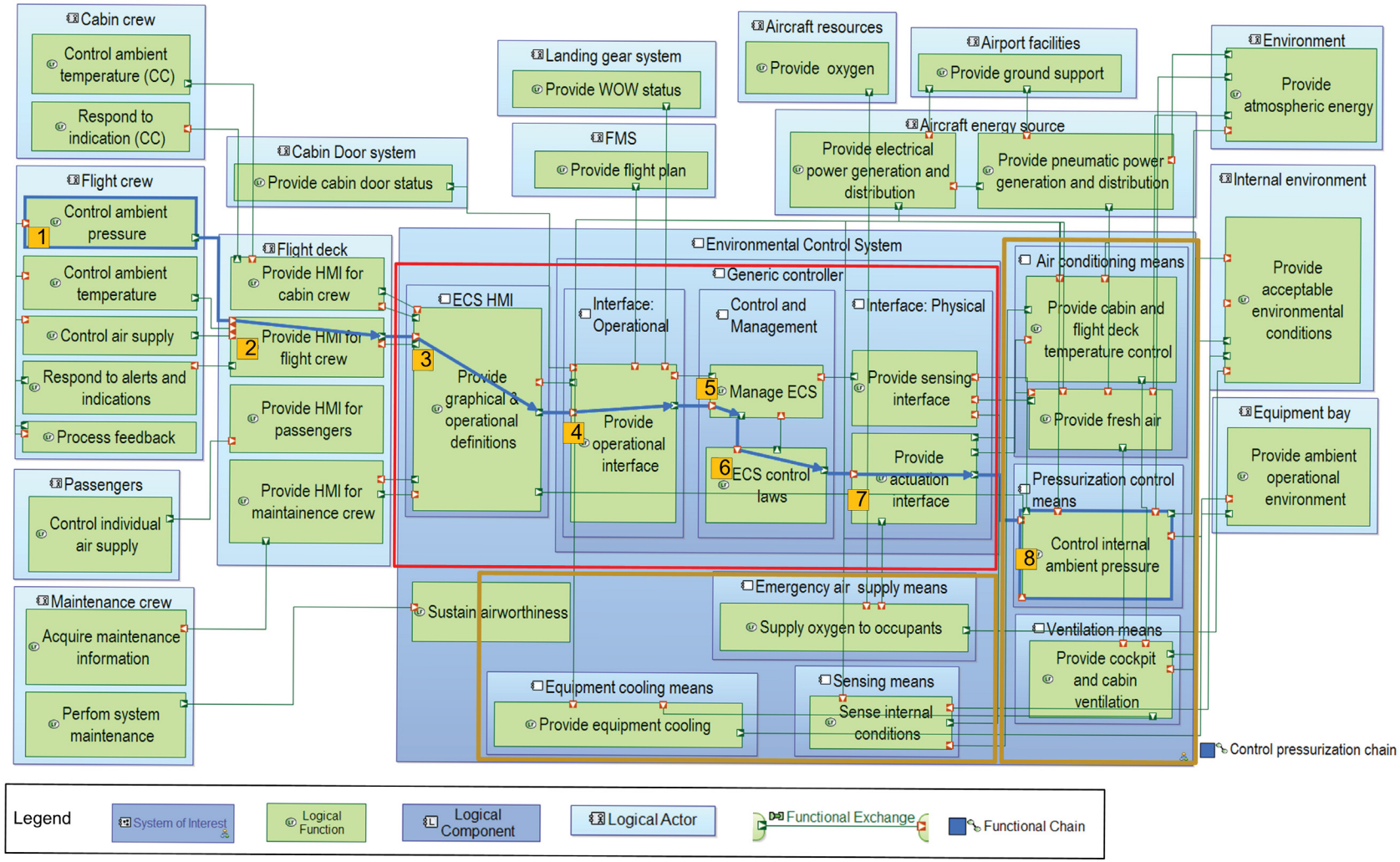

Figure 4.9: Logical generic architecture for the ECS (logical architecture LAB)

Finally, the generic ECS system is validated to make sure that system architecting rules are followed. One of them is traceability to the SA. The validation helps to identify the elements that do not contribute to the needs specified in SA.

For detailed system-level design, a vertical transition is performed in each sub-system components defined in brown box. Through the vertical transition, system-level specifications for bleedless ECS architecture, bleed-driven ECS architecture, cabin pressure control subsystem architecture, and bleedless ECS HMI specifications are defined for the system level. During the transitions, only sibling actors and components are transitioned. Sibling components or actors are elements that have a direct interface with the system of interest. For instance, Figure 4.10 shows the SA obtained through the vertical transition for a bleed-driven ECS.

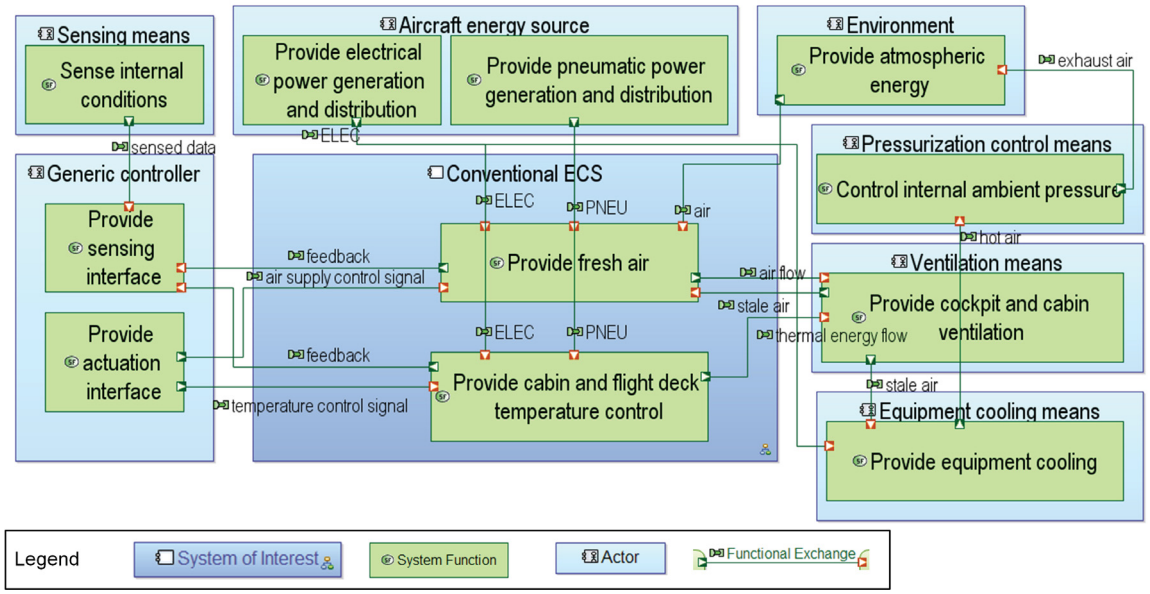

Figure 4.10: System analysis obtained after transition – functional system architecture diagram for the conventional bleed-driven ECS (SAB))

An automatic transition to the logical level is performed, and logical functions are developed. Then, the principal components are identified. Finally, the bleed-driven ECS and bleedless ECS LA is defined by allocating functions to the identified components. A sample bleed-driven ECS logical architecture is presented in Appendix B and that of bleedless ECS in

Appendix F.

Aircraft energy sources are defined through the horizontal adaptation method, which helps maintain the abstraction of the energy sources and the encapsulation of energy-source processes in the functions “provide electrical power generation and distribution” and “provide pneumatic power generation and distribution.”

By subsystem-transitioning the ECS HMI, the SA is obtained. The HMI is defined for the bleedless ECS and consists of physical panel definitions and widget-layer definitions. There are two physical panels for the bleedless ECS: (1) the air-conditioning panel definitions and (2) the pressurization panel definitions. Regarding the widget-layer definitions, there is (1) the bleedless ECS synoptic layer with an air-parameter widget and a synoptics widget, (2) the EICAS ECS layer with an air-parameter widget, and (3) the cabin-crew temperature layer with a temperature-setting widget.

4.2.4 Physical architecture level

The PA provides insights into how the system is built and developed. The physical working level provides the most detailed representation of the system and its associated components. Logical elements are transformed into behavioural components and allocated to physical components. Sometimes, logical components and elements may appear to belong together, but in the PA, they need to be allocated to different physical components depending on various factors, such as latency, budget, or configuration efficiency. For the ECS case study, the PAs for the bleedless and bleed-driven configurations and the cabin pressure system have been developed.

An example of a high-level PA developed for the CPCS is shown in Appendix C. Electrical outflow valves have been incorporated into the model. The system has two sets of valves to account for any electrical failure.

4.3 Item-level specification

One of the primary objectives of this thesis is to investigate the MBSE approach for the software aspects of the controller. Primarily, two specification models have derived in item level namely platform architecture and application software architecture. The platform architecture contains components related to target computers such as processors, memory, and networks. Further, the application software architecture contains software components such as threads and subprograms, the dynamic behaviour of the components and the communication between components.

The vertical transition method is carried out to reach the item level. The requirements are divided into two categories: (1) SRATS and (2) SRATH. Through the transition, the system-level requirements are inherited and then allocated to software functions (SRATS) and hardware components (SRATH). At SA, the HLR is developed from the SRATS as a part of the software-requirement process. The HLRs specify what is to be implemented in the application. For instance, the activity "provide pressurization control" is an HLR of the controller, as is the "verification of sensor data." The SA provides the specification model. The next phase is that software design was design model is defined. As mentioned in subsection 3.2.2, the design model consists of software architecture and LLRs. The software architecture is developed from the HLRs. Software functions are developed and allocated to generic-controller parent functions. A sample data-flow diagram (application software architecture) is defined and embedded in the parent function through horizontal adoption and can be found in Figure 4.12. Figure 4.11 shows the LA for the cabin pressure controller. A detailed LLR model is also specified for CPCS controller at Bombardier.

Once the system architecting is complete, the validation of the diagram is performed. The validation helps in model checking process as specified in subsection 3.2.2. The following section presents other validation capabilities available in Capella.

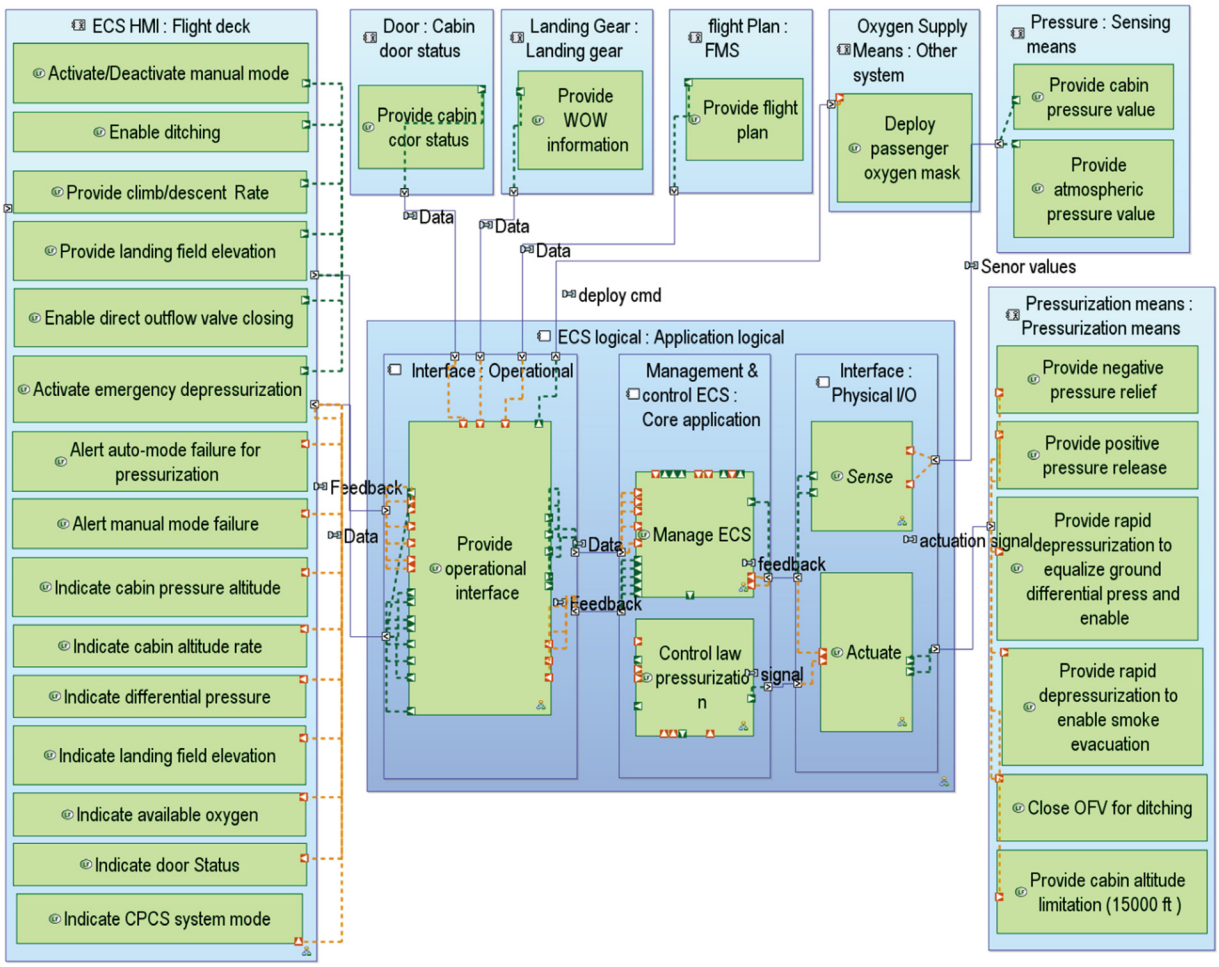

Figure 4.11: Item-level logical architecture for the cabin pressure controller (LAB)

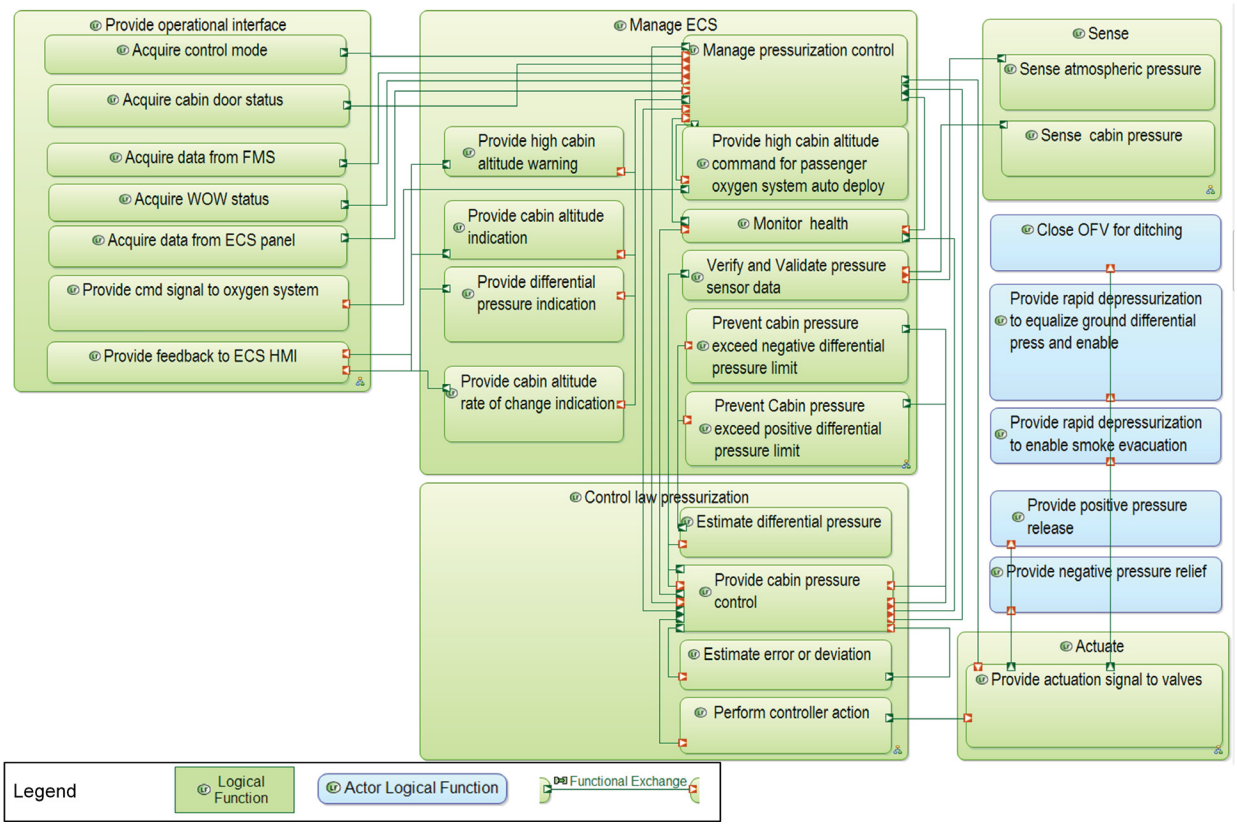

Figure 4.12: Logical data-flow diagram for cabin-pressure control functions (LDFB)

4.4 Architecture validation

The validation of an architecture defines the maturity of the specification. Each working level is validated for traceability through various model validation features:

1. *Validation rules*: The validation rules are rules defined in Capella to support effective design, integrity, quality and transition at each working level. Any element that does not follow the rule will be tagged with a warning sign and warning message. Table 4.3 gives an overview on Capella validation rules. The rules can be enabled or disabled according to the scope of the model. However for the thesis work all rules were enabled.

Table 4.3: Capella validation rules [78]

Validation rules		**Examples**
Design	Consistency	Check if element is up to date compared to original
Design	Completeness	Makes sure a capability is always involved in scenario or an interaction/ functional exchange is not connected to parent function.
Design	Coverage	Check if a capability is involved by at least one actor. Or a port is present without any exchange or link
Design	Well-formedness	Checks if use case, capabilities, interfaces, components, data, state machines, dataflows and scenarios are well-formed.
Integrity	Integrity	Ensure realizations or traceability in upper and lower working levels
Quality	Quality	Check if each element has a description or summary and is reviewed or not.
Transition	Consistency	Ensures there is consistency in components, dataflows, interfaces, scenarios and state machines at all working levels
Transition	Justification	Ensures realization or traceability of LA, PA, and SA elements during transition.

2. *Requirements allocation*: Once the bleedless ECS and the conventional ECS architecture are defined, requirements are allocated to the developed functions. This process validates whether the developed functions are sufficient to fulfill the requirements. The Capella requirement management wizard aids in the process of allocating requirements to functions as shown in Figure 4.13. The process of importing requirements is explained in section 4.6.

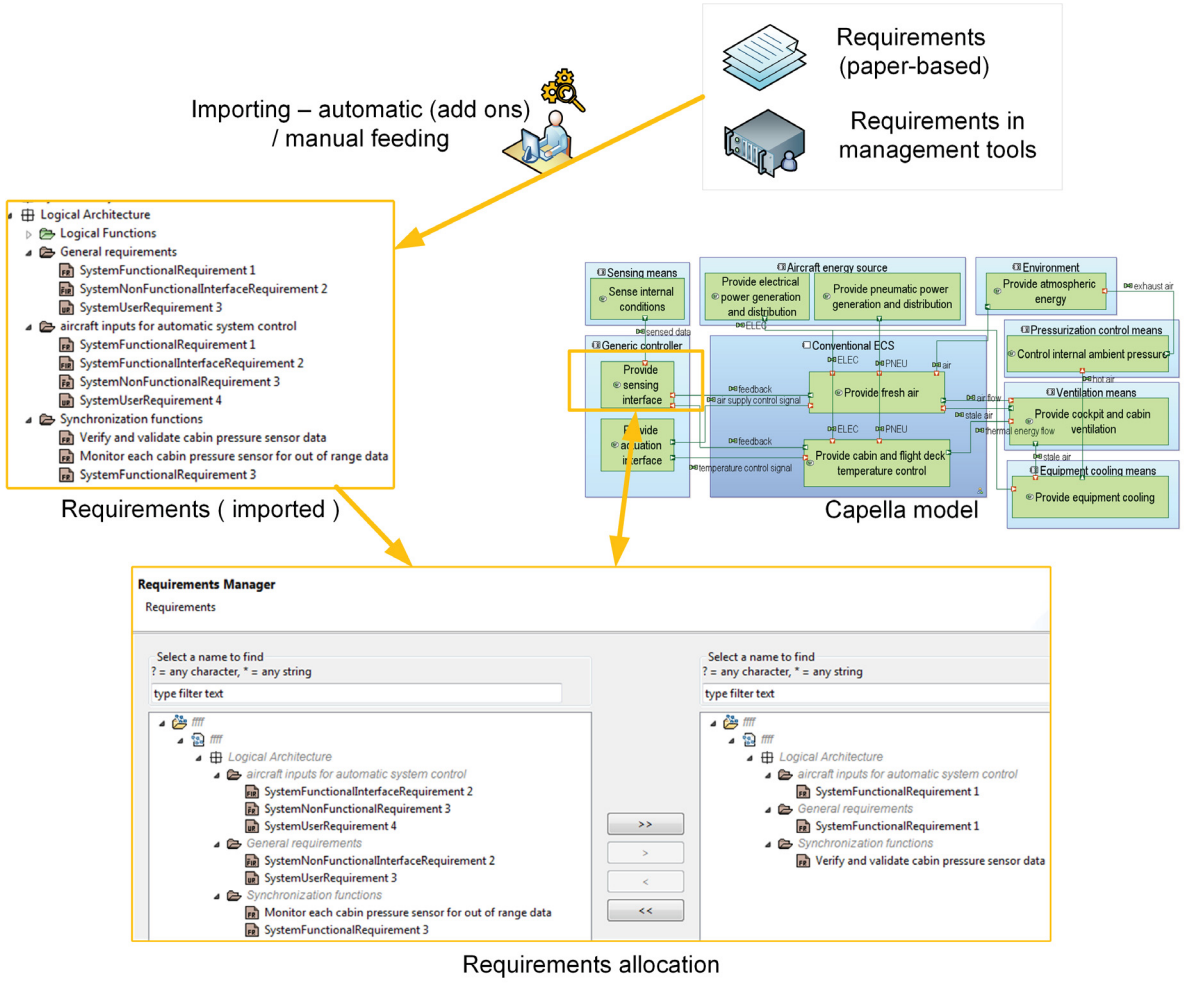

Figure 4.13: Requirement management and allocation

3. *Functional chains*: The functional chain helps to designate or highlight a specific path. It consists of a combination of functions and functional exchanges related to a specific task. In the case of the ECS, functional chains are divided into command, feedback, and resource paths. For the DIMA case study, the pressurization control system and feedback are defined as functional chains. Functional chains also enrich the specifications with constraints such as latency, particularly crucial for the specifications of the DIMA platform. Moreover, the basic performance viewpoint is used to define the required function-execution time and then map it to the allocated budget.

4. *Scenarios:* Scenarios are sequential diagrams describing how interactions between functions or logical components or actors are executed to accomplish a capability. The

difference between a scenario and a functional chain is that a scenario is a closed-loop sequence made up of different functional chains. For instance, in the ECS pressurization scenario is a combination of a control chain, a feedback chain, and a resource-flow chain. Scenarios are created for early validation to confirm that the identified functions can support the system to fulfil the system capabilities. Any missing function or interaction to complete a capability can be identified through scenarios. Modes and states, explained in section 4.5 also help give deep meaning to scenarios. The transition between operational modes is also depicted as a scenario. An example scenario of automatic to manual pressurization is presented in Appendix E. This scenario is defined in LA and shows a detailed solution for transition.

Also, a representative scenario of temperature control defined at SA is shown in the Figure 4.14.The scenario is divided into two sections. The first section is feedback loop, where the the temperature is updated to the user in fixed intervals. The second section is control sequence, where the flight crew or cabin crew can control the temperature alternately.

(a) The feedback loop: The ECS sense temperature from internal temperature and forwards the information to user. The HMI supports the ECS to convey information to user.

(b) The control sequence (alternative) : (1) Flight crew set the required temperature. The HMI convey user requirement to ECS. The ECS controls the internal ambient temperature to provide acceptable environmental conditions to user. (2) Cabin crew set the required temperature. The HMI convey user requirement to ECS. The ECS controls the internal ambient temperature to provide acceptable environmental conditions to user.

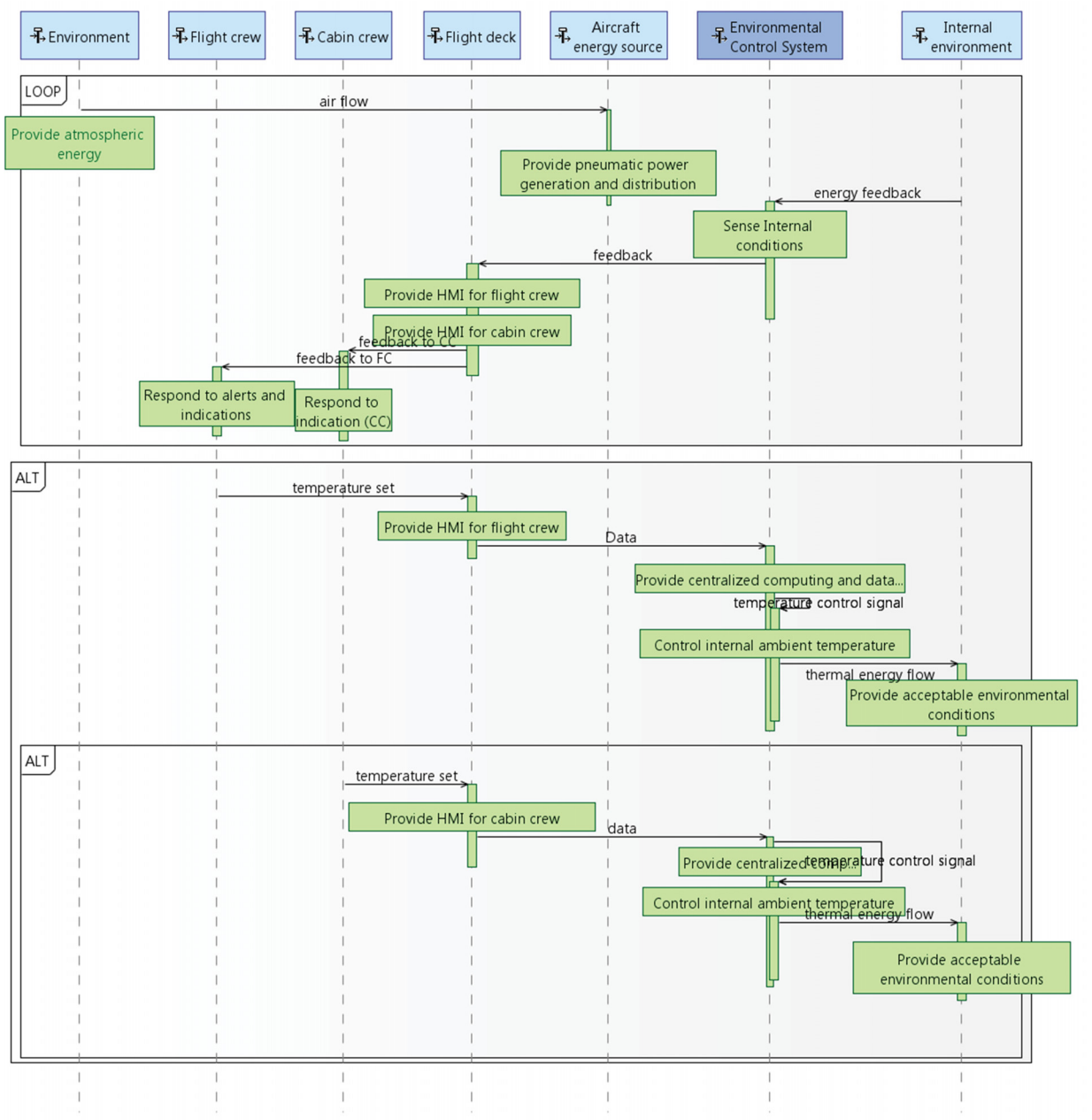

Figure 4.14: Entity scenario [ES] for temperature control at SA

Further, the time duration can be defined between two exchanges in scenario to represent the performance and timing constraints.

4.5 Transverse modelling

Transverse modelling features can be utilized throughout all four working levels of ARCADIA. These transverse modelling techniques are important to increase the effectiveness of the model-based specifications. In the following section, the two main features of transverse modelling available in Capella and applied in the ECS use case are presented.

1. **Modes and states diagram**

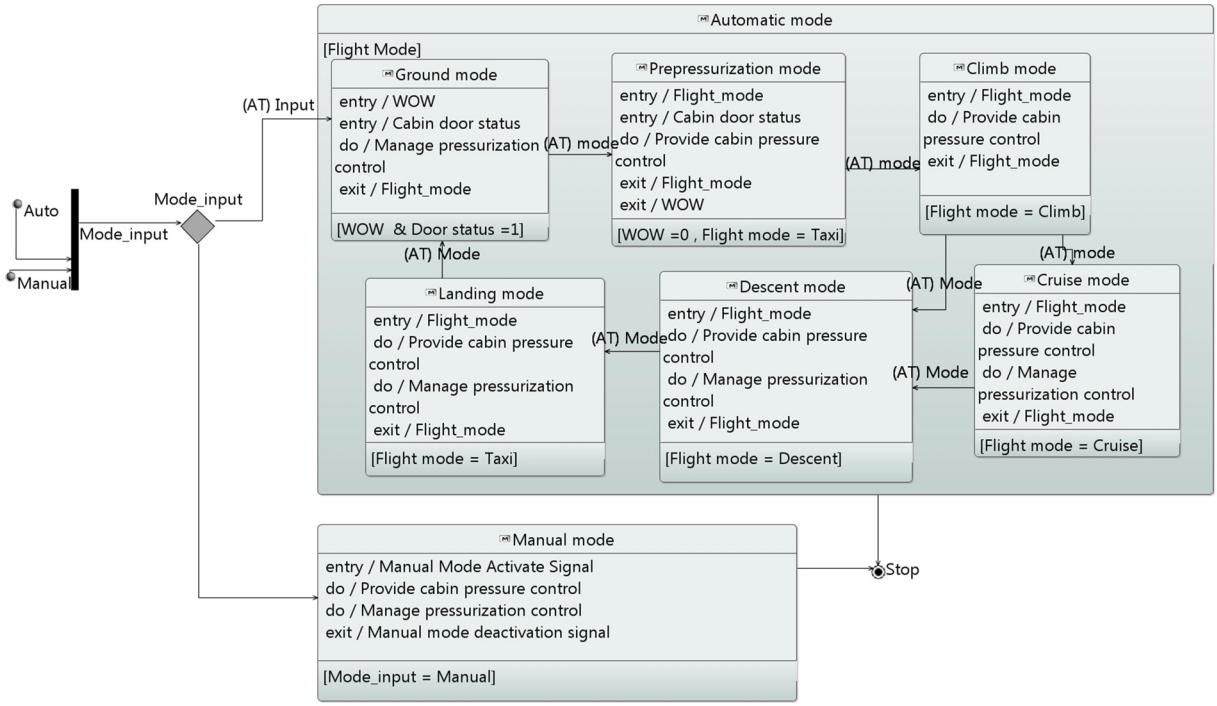

Figure 4.15: Mode diagram for pressurization control (MSM)

As part of the architecture maturing, the modes and states of components and actors are specified. The ARCADIA definition of modes and state is different from usual INCOSE definition. System modes are definition of the expected behavior of the system (or of its actors, or of its components) in situations foreseen at design time [79]. However, A state represents an operating condition or status on structural elements of the system: operational, failed, degraded, absent, etc [79]. The definition of the modes and states

starts at the operational level (here with the definition of flight phases, for example) and is continuously enriched throughout the specification process. For instance, the flight mode contains parking, taxis, take-off, climbing, cruising, descent, and landing modes. Similarly, the air-conditioning means have automatic, standby, and manual modes. Pressurization also has modes that depend on the flight phase, as shown in Figure 4.15. States and modes are also important to analyze failure scenarios and plan specific test scenarios.

2. **Data-class diagrams**

   Data-class diagrams include all the data structures needed to complete the specifications of each subsystem. Often, data-class diagrams compress of exchange items or data parameters utilized in a system. The data includes the following items:

   - *Time factors:* duration, data update rate, etc.
   - *Data factors:* properties of data; accuracy, range, and scale of data
   - *Interface factors:* cockpit control and display factors
   - *System factors:* Status, modes, and availability
   - *Type:* control (commands), feedback, or resources (air, electrical)
   - *Exchanged data:* cabin temperature, cockpit temperature, differential pressure
   - *Units:* Degree Celsius, voltage

An instance of a cabin pressure controller is presented in Appendix D. The benefit of the data structure is the significant reduction of the model complexity on the interface level. The number of exchange items between two functions in a DIMA system is huge. Furthermore, increased exchange links between functions render the model difficult to understand.

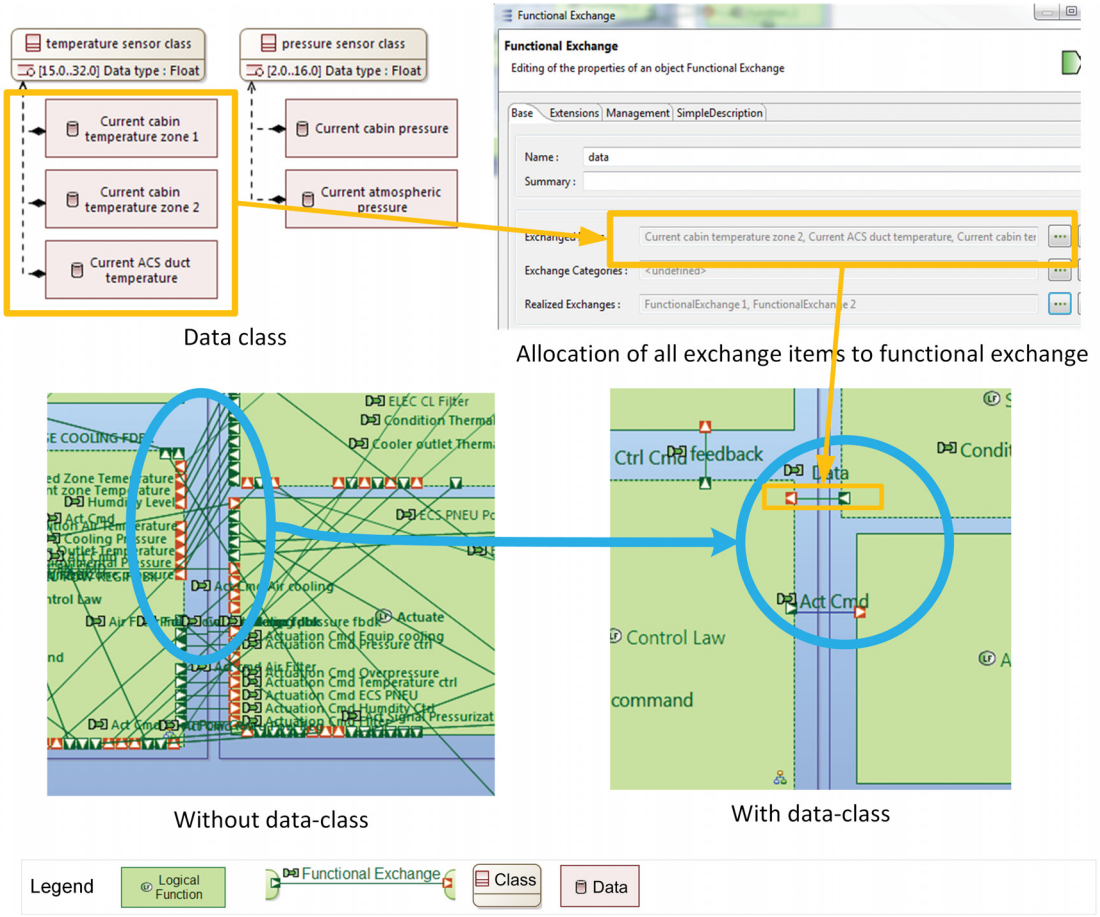

Figure 4.16: Illustration of the benefits of using data classes for exchange items

A sample class diagram for cabin pressure controller is presented in Appendix D.

In this context, a data structure minimizes the link to one direction by allocating exchange items between two functions to a single link and thus reducing the overall complexity, as shown in Figure 4.16.

4.6 Capella viewpoints

Capella viewpoints are extension or add on infrastructures to perform additional engineering activities on the models [74]. Capella viewpoints are different from viewpoints specified in section 2.1. However, Capella viewpoints are extensions that facilitates the same services as

the viewpoints in system engineering. The engineering activities involve creating a shared environment for models, automatic generation of documents, management of requirements and properties, multi-level transition, and assessment of safety, performance, cost, mass and scheduling.

For the thesis, it was important to have extension to other engineering activities. The idea is to slowly incorporate Capella model as a replacement or complement to current specifications. Therefore, automatic document generation extension was explored. The methodology also needed an extension to support importing requirements. Further, the DIMA case study required to have an extension to MARTE to perform the timing analysis at item-level. As mentioned in section 3.2.4, the vertical methodology is carried out with the sub-system transition add on. Following are the various viewpoints[1] supported by Capella as shown in Figure 4.17.

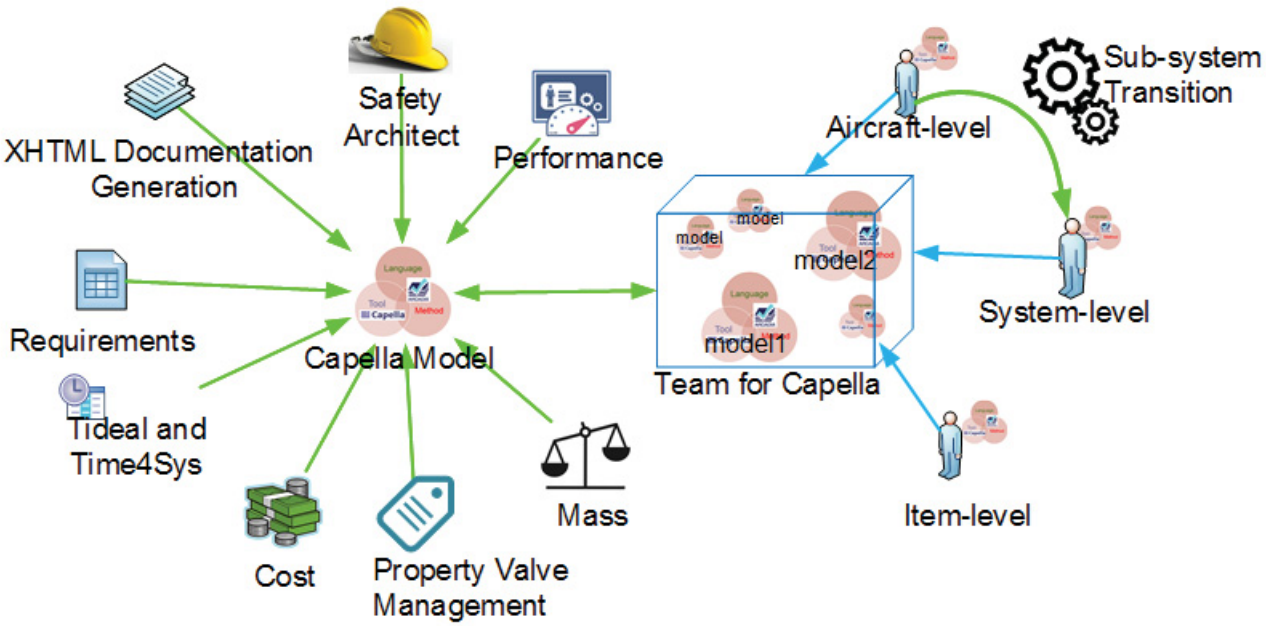

Figure 4.17: Engineering activities supported by Capella extensions

1. **Requirements addon**: the add on helps to import requirements files such as RegIf defined using a requirements management add on like IBM DOORS. Figure 4.13 shows the requirements management process used for ECS case study. During the case study,

[1] Not all the Capella viewpoints presented were explored.

the aircraft-level requirements were tried to import using requirements add on. However, the add on was just launched and not mature enough to implement. Later the add on was tested by aircraft-level engineer and found to be effective.

2. **XHTML Documentation Generation**: The add on helps to generate HTML documents for Capella models that can be shared with all stakeholders. Figure 4.18 shows an example of the generated HTML document. For the case study, the HTML document was shared between the model review team along with models. The document contains all the model elements, and the information allocated to the element. Moreover, the document showed the traceability to upper and lower working levels. The document was effective in providing a complete system specification. However, the problem faced was that the model should be present along with the document to view the document.

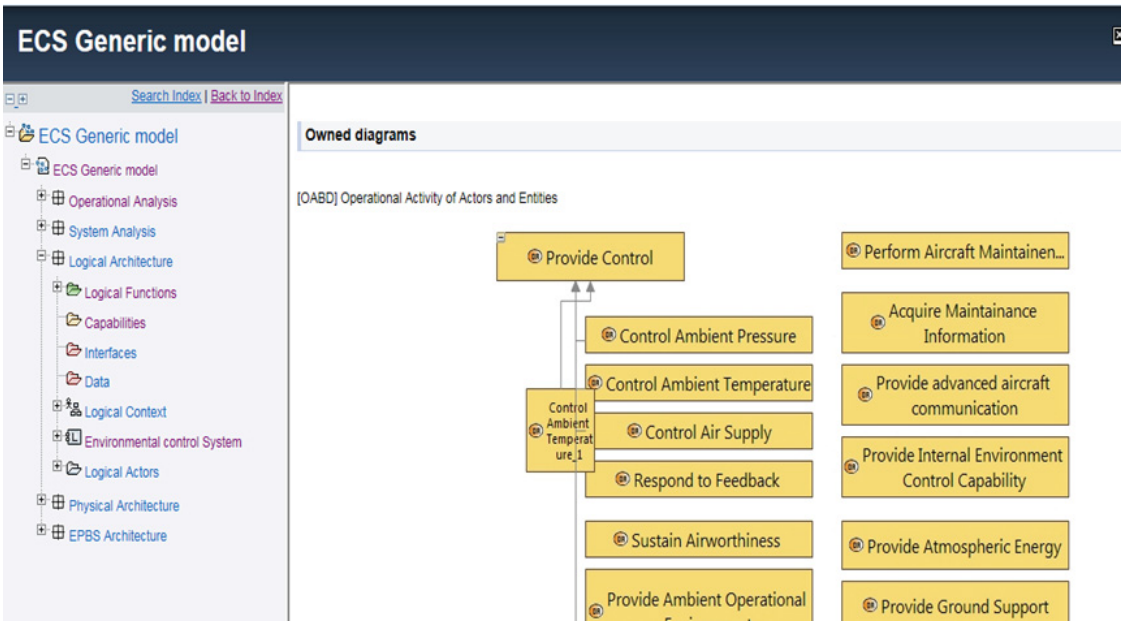

Figure 4.18: ECS model HTML document generated using viewpoint

3. **Basic viewpoints**: The add on enables to define the non-functional aspect of cost, mass and performance of components (physical/logical) and compares the maximum acceptable by the component to the current value of the components. The cost and mass analysis can be performed at PA. For example, the PA can define the maximum

value and the supplier can add required value to do the analysis. The logic for the cost and mass analysis is as follows:

(a) Value > Max Value; the physical component and associated annotation are displayed in *red*.

(b) Value = Max Value; the physical component and associated annotation are displayed in *orange*.

(c) Value < Max Value; the physical component and associated annotation are displayed in *Green/pale*.

Figure 4.19 shows the mass analysis of the hydraulic system using the logic as mentioned above.

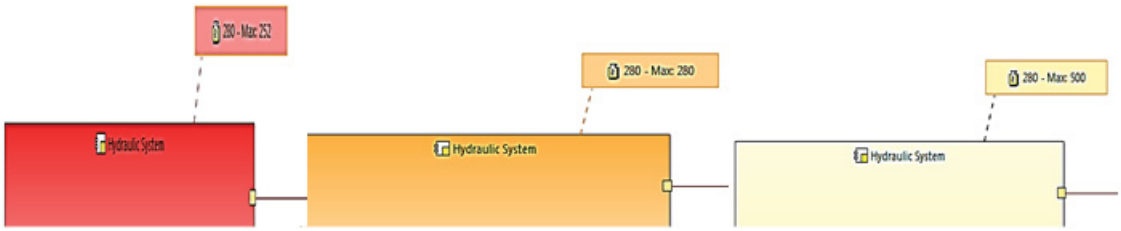

Figure 4.19: Non- functional mass analysis using Capella viewpoint

Further, performance analysis for time constraints and compares the Function Time Consumption of all functions in Functional chain (Total Execution Time) with the maximum allowed execution time limit (Execution Limit). Figure 4.20 shows the performance analysis done on information transfer functional chain between functions "*Display information and receive commands*" and "*Receive system indication*" using the following logic.

(a) Total Execution Time < Execution Limit: The Execution Limit is displayed in *green/yellow*.

(b) Total Execution Time = Execution Limit: The Execution Limit is displayed in *orange*.

(c) Total Execution Time > Execution Limit: The Execution Limit is displayed in *red*.

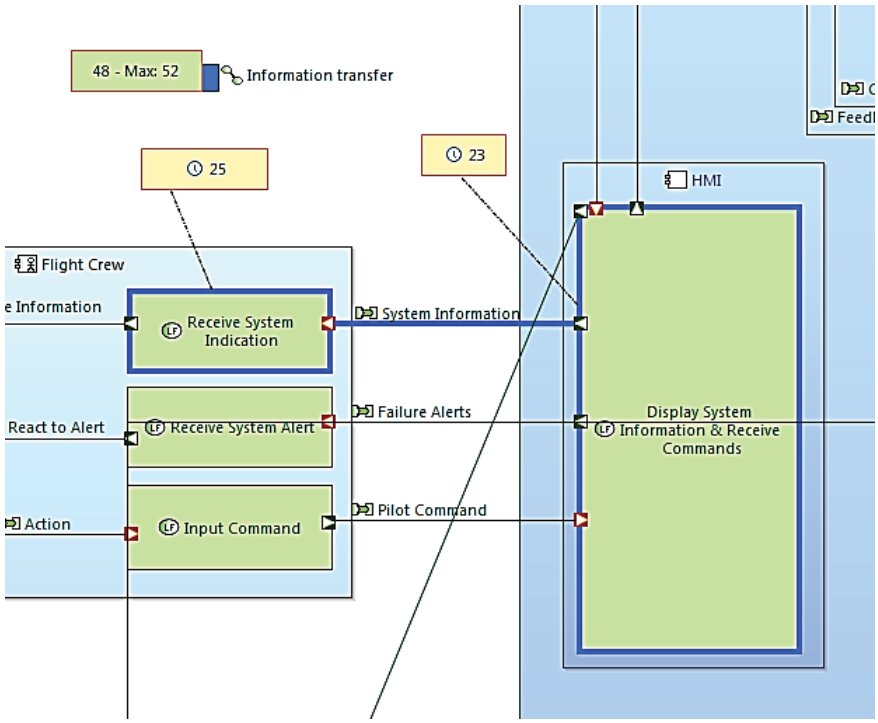

Figure 4.20: Performance analysis using Capella viewpoint

In the above mentioned basic viewpoints, performance viewpoint is more useful in the case study, as it helps to define the maximum allocated time for a process. This viewpoint was the first attempt to introduce timing analysis in Capella and paved the way for more mature and functional add ons.

4. **Tideal and Time4Sys**: The add on contains two viewpoints namely Tideal which is used to define timing properties and Time4Sys which creates a bridge with Time4sys a software that enables the architect to perform schedulability analysis or simulation. Time4Sys proposes four capabilities: modelling and viewing the Time4sys model, a dedicated meta-model based on the MARTE standard, model transformations able to transform and adapt Time4sys model for verifications add on and connectors to import and /or export models from design add ons and verifications add ons [80]. For the ECS use case, only Tideal was used to define timing properties to software architecture. Figure 4.21 shows a sample timing property definition using Tideal viewpoint. The

purple elements are schedulable resources, and all the functions appear in green colour. A watchdog timer alarm is defined highlighted in red with timer function as in orange. Furthermore, the add on helps to provide the model checking process explained in subsection 3.2.2.

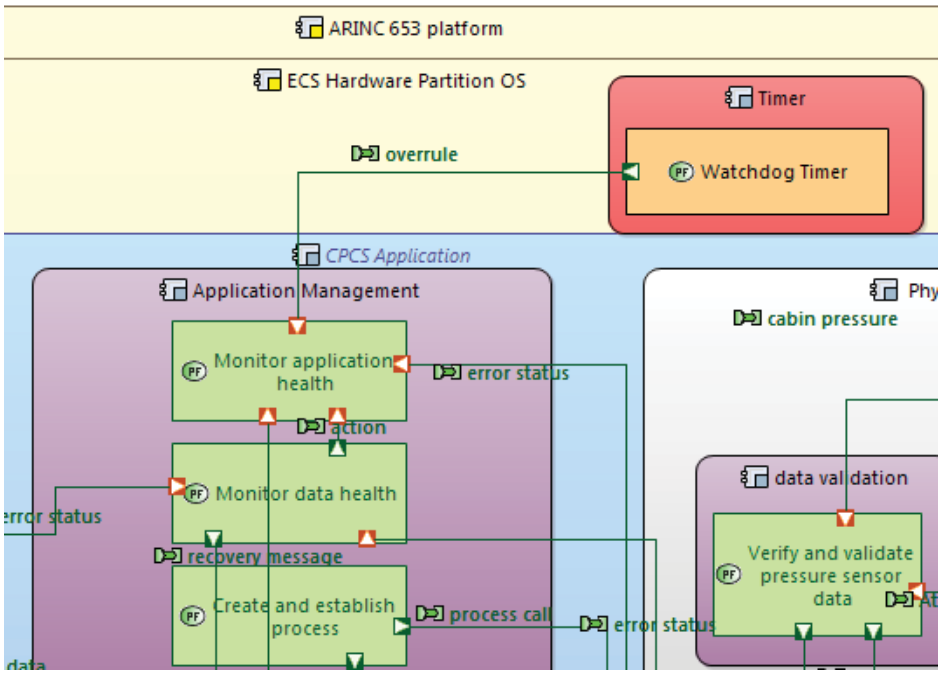

Figure 4.21: Timing properties defined using Tideal

5. **Team for Capella**: Team for Capella enables system engineers of multi-levels to collaborate and share Capella models. Team for Capella provides a fine-grained concurrency management policy: only the minimum set of elements are locked when modified and automatically released on saving [81]. The add on provides a cloud repository to share models with strict permissions.

   For a complex system like DIMA the interfaces are important. When a derived requirement is formed in a system, the actor system interacting with the system should be informed. Moreover, if a derived requirement forms in item-level, the system-level needs to know about the change as mentioned in subsection 3.2.1 to assess the impact. With a common shared environment, if the system engineer has limited access to actor system's model, the engineer can inform about the change under a request

and approve method. After the assessment, if the change is not approved the system engineer can come up with another solution. Further, when the model is available to multi-disciplinary team, they can use viewpoint to perform analysis. Therefore, with the Capella viewpoints, stakeholders in multi-level can perform analysis on a model and ensure that the concerns of the stakeholders are addressed.

6. **Capella Studio:** Capella Studio provides a fully integrated development environment (IDE) which aims at facilitating the development of extensions for Capella MBSE. Capella Studio offers the capability to create Capella add-ons in a standard way with Java and the Eclipse Modeling Framework (EMF) [59]. To explore the feasibility to create a new add -on, a quality assessment viewpoint was created using Capella Studio. Figure 4.22 shows the quality viewpoint created using the IDE. The viewpoint has the attributes to define the maturity level and the confidence level of the component.

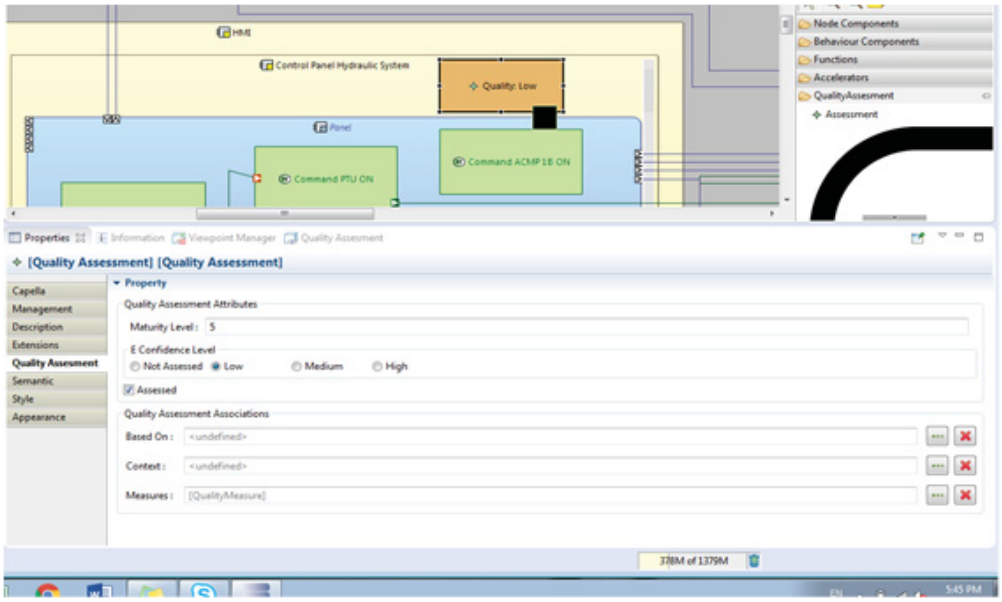

Figure 4.22: Quality viewpoint developed using Capella studio

Further, there are viewpoints like Safety Architect [82] which enables safety assessment and Pure Variants which enables variability and property valve management to complement data class diagrams.

To summarize the viewpoint enables to perform additional engineering activities on the same

reference and moreover, new viewpoints can be developed that fits the requirements of the concerned systems engineering process.

4.7 Summary

In this Chapter, the implementation of MBSE, with the methodology defined in Chapter 3 was discussed. For the case study, a generic architecture for ECS was developed. The generic architecture included all the needed functionality for an ECS system. The architecture can be updated to incorporate the development of technologies. To illustrate this, a bleedless and bleed-driven ECS specification was derived. Further, a CPCS specification and state of the art ECS HMI specification was also derived. All the derived models used the vertical transition to maintain consistency and traceability at system-level. At item-level, a CPCS controller architecture is specified.

The validation tools in Capella help to ensure traceability and consistency is maintained in all the working levels. When started the modelling activity, it was hard to differentiate the OA and SA. Especially the capability definitions. Moreover, there was difficulty in comprehend between activity and capability. The main reason is that the capability was defined after the activity was defined. Moving on to LA, horizontal adaption was used in the beginning to define technological variants such as bleed and bleedless ECS. However, the dataflow diagrams must be kept unsynchronized. This was learned during the case study when a synchronization was performed to update a function. After synchronization the whole model was turned into a unreadable diagram, due to the high number or interfaces. The reason was that once synchronized all the existing interfaces are updated automatically to meet the Capella validation rule. Therefore, a new method was needed, and thus vertical transition was used to define sub-systems.

One of the advantages of defining specifications in Capella is that ARCADIA is viewpoint driven. For the case-study, apart from the need and solution viewpoint in ARCADIA additional viewpoints were explored. XHTML documentation generation tool was beneficial as an

architecture description document. However, at the moment there are add ons that can generate model description in specific templates which are independent of models. The case study initiated timing analysis for DIMA architecture using Tideal. The coupled Time4Sys add on support MARTE modelling and simulations can be performed to check the time constraints. Overall, MBSE is powerful and effective for designing complex aircraft systems.

Chapter 5

Integration and Demonstration

This section presents the integration demonstration that is under development at Bombardier to reflect the full development cycle from the aircraft level to the software application in an ARINC 653 platform. The thesis covers the following scope of the demonstration.

1. Develop a plant model in Simulink to emulate the physical system and cabin atmosphere behaviour.

2. Develop a controller model for CPCS application

3. To support the development of ECS HMI to emulate CDS

The demonstration platform helps explore the optimal specifications of the partitions and the required computing resources. The demonstration is implemented through a Hardware-in-the-Loop (HIL) simulation and therefore provide a virtual test environment for the DIMA platform. HIL simulation helps in testing complex real-time embedded systems. That is, HIL simulation provides a platform that emulates the complexity of the physical system under control. For instance, HIL simulation will provide a platform to emulate the outflow valves in CPCS along with the complex behaviour of the cabin atmosphere. The controller to be tested will be connected to the platform or plant simulation.

Figure 5.1 shows the overview of integration and demonstration currently under development for Cabin Pressure Controller (CPC). The HIL simulation is divided into three components. First is ECS HMI section that consist of the flight deck and avionics application that manages the HMI aspects. Second is the plant model that act as a real-time simulator for physical systems and air flow characteristics. The third and most important component is the CPCS

application (controller). All three components are connected through TTETHERNET communication protocol.

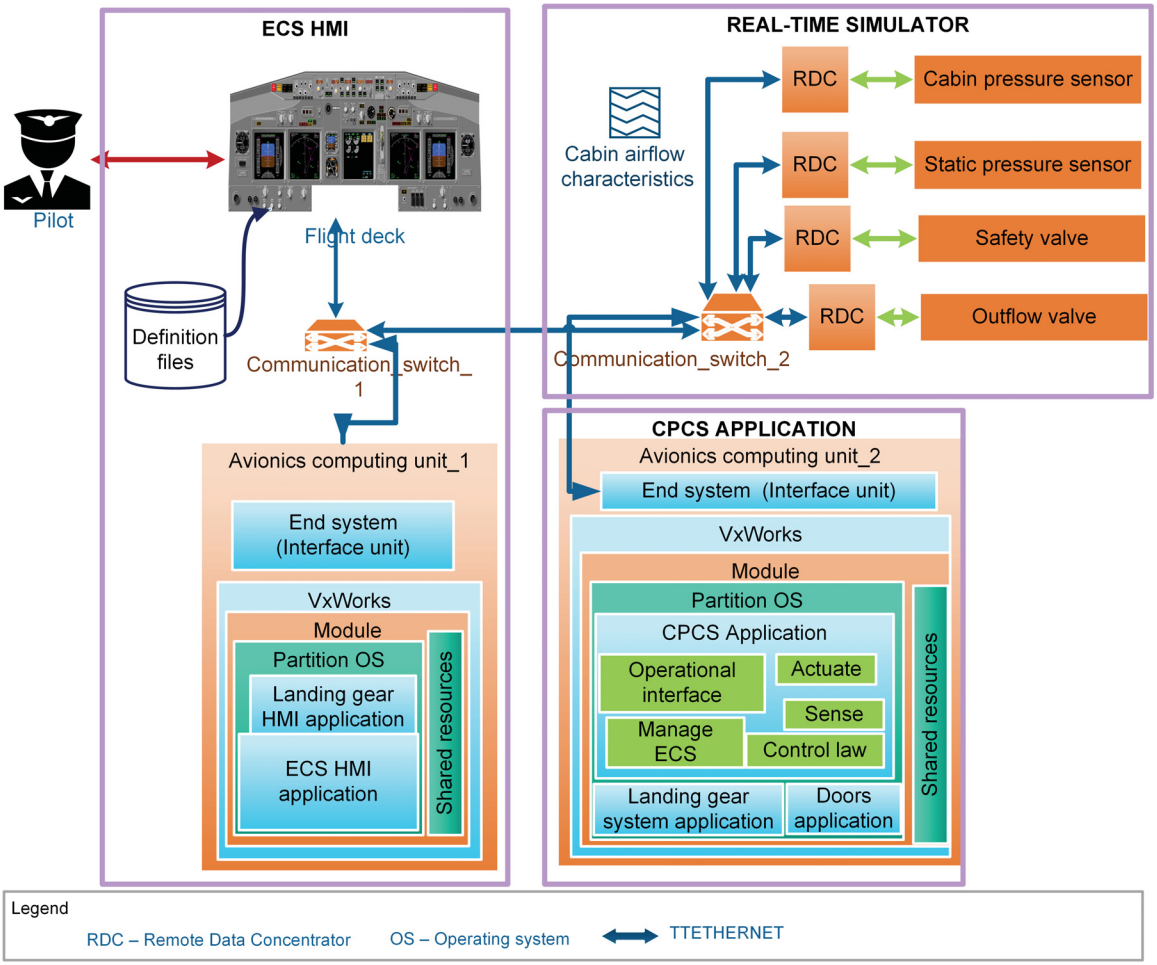

Figure 5.1: Integration and demonstration overview

The Integration and demonstration aim following three objectives.

1. To validate and verify the CPCS controller and HMI specification defined using Capella through HIL simulation

2. To develop and test CPCS application

3. To develop and test HMI defined using the ARINC 661 protocol.

The following section provides insight to each component in HIL simulation.

5.1 Plant model development

The simulator will emulate the physical system and behaviour of CPCS. The first step is to develop a Simulink model of the CPCS. The CPCS controls the pressurization by regulating the exhaust air through outflow valves. The pressurization can be automatic or manually controlled. To control pressure automatically, the controller acquires pressure data from the pressure sensor. The controller calculates the deviation from the scheduled pressure. For manual control, the deviation is calculated between flight crew command and pressure data. However, the differential pressure between cabin pressure and atmospheric pressure is always given prior. The differential pressure should be maintained at all cost. Next, the controller sends the actuation signal to the outflow valve electric actuator The actuation signal is a reference voltage. The actuator opens the outflow valve to lease the air and pressure is brought to the required value.

The system can be modeled as a mathematical equation of balance of air mass [83]. The change in cabin air mass is the sum of the mass of air flowing into the cabin from air conditioning system (ACS), the mass of exhaust air and mass of air leaked through the structure as depicted in equation (1)

$$dM_{cabin} = M_{air-in} + M_{air-out} + M_{air-leaked} \tag{1}$$

where:

M_{cabin} – *Mass of air in the cabin*

$M_{air-out}$ – *Mass of air exhaust from ACS (-ve)*

M_{air-in} – *Mass of air forward from ACS(+ve)*

$M_{air-leaked}$ – *Mass of air leaked through structural leakage (-ve)*

The cabin air is assumed to obey the ideal gas equation.

$$P_{cabin} = \frac{R_{air} * T_{cabin}}{V_{cabin}} \int (M_{air-in} - M_{air-out} - M_{air-leaked})dt \tag{2}$$

where:

R_{air}– *Specific constant for air*

V_{cabin}– *Volume of the pressurized zone*

P_{cabin} – *Cabin pressure*

T_{cabin} – *Cabin temperature*

The actuator dynamics follows a 2^{nd} order dynamics represented by the following transfer function with specific values, according to ref [84].

$$Transfer\ function = \frac{K_v}{(1 + T_e S)(1 + T_m S)} \tag{3}$$

where: T_e , T_m – *Mechanical and Electrical time constant*

K_v – *actuator gain*

The characteristics of the outflow valve are assumed to be linear. Thus, the amount of air exhausted is linearly proportional to the valve opening. The equations 2 ,3 are used to develop the Simulink model. Physical systems such as outflow valves, actuators, and pressure sensors are also modelled in Simulink, as depicted in Figure 5.3. The cabin air flow function is defined using the ANSI/ASHRAE Standard [73]. The structural leakage function determine the leakage based on cabin pressure and atmospheric pressure and is specific to Bombardier.

The performance of the plant model satisfied with the requirements specified in the ECS Capella model as shown in Figure 5.2.

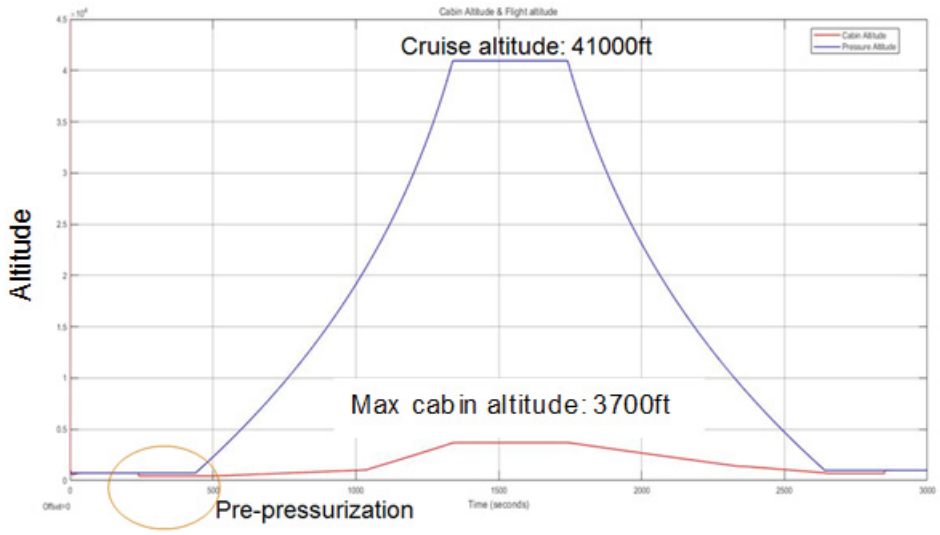

Figure 5.2: CPCS Simulink model output

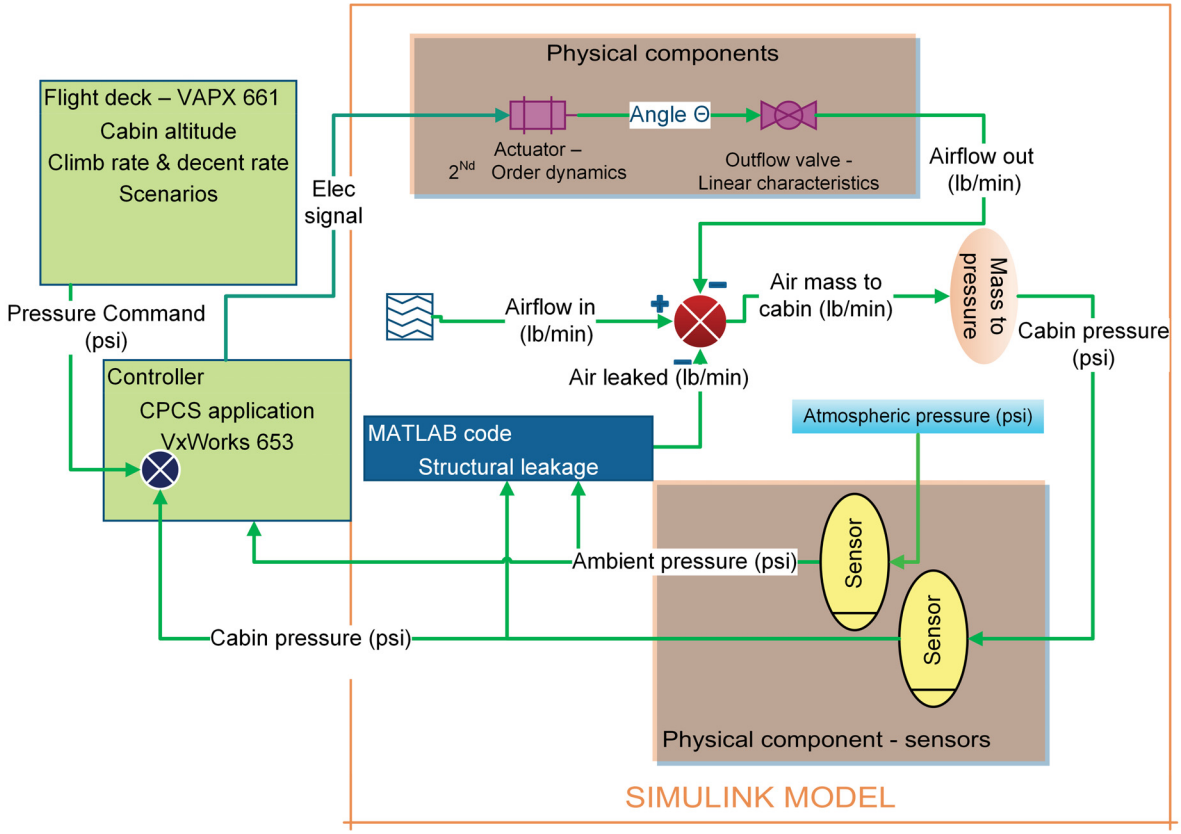

Figure 5.3: Simulink model for the CPCS

The plant model will be adapted and deployed in RT-LAB for real-time simulations.

5.2 The CPCS application

The application is the software component and will process the CPCS data and perform the control algorithm. The ECS application is developed through a Simulink model at Bombardier. To do so, the software model needs to be optimized. The first step is to perform a timing analysis to find out the Worst-Case Execution Time (WCET), execution profiles for developed functions. The data will be used for optimization of WCET and to validate that the Simulink model and the derived software program will adhere to timing requirements. Further, the code for the model is generated using the Embedded Coder in MATLAB [85]. The generated code will be adapted and deployed to an ARINC 653 platform to create the CPCS application. The model is an intellectual property of Bombardier.

5.3 The ECS HMI

The ECS human-machine interface emulates flight deck with real synoptics for ECS. The flight deck or CDS is developed using the VAPX XT 661 tool [86]. VAPS XT is a complete, object-oriented C++ avionics software-development tool for all types of avionics cockpit displays, including ARINC 661. The CDS acts as a rendering engine that deploys the updated data from UA and graphical presentations defined in definition files as shown in Figure 5.4. The developed HMI will demonstrate the capabilities of ARINC 661 and the physical hardware such as buttons, knobs etc. HMI is defined for EECS and consists of physical panel definitions and layer-widget definitions. There are two physical panels for EECS. (1) Air conditioning panel definition and (2) Pressurization panel definition.

In layer-widget definition, there is (1) EECS synoptics layer with air parameter widget and Synoptics widget (2) EICAS ECS layer with air parameter widget and (3) Cabin crew temperature layer with temperature setting widget.

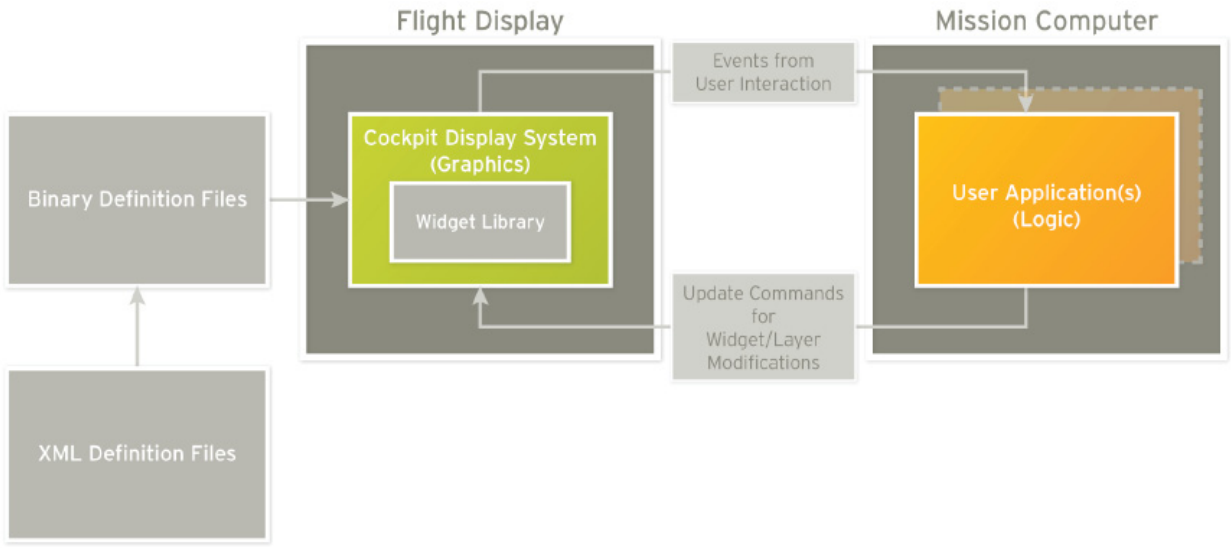

Figure 5.4: Overview of ARINC 661 implementation presented by Presagis [87]

To implement the HIL simulation demo, efficient communication between the Real-Time simulator (Plant model), CPCS application and ECS HMI is crucial. Therefore, TTETHERNET will be deployed as a communication network.

Chapter 6

Conclusion and Future Works

An MBSE methodology for the specifications of the implementation of the aircraft-control system on a DIMA avionics platform is developed in this thesis.

At present, the system engineering process for a complex system of systems like aircraft is not well developed and is one of the reasons why aircraft development programs are time-consuming and resulting in high product development cost and period. Further, the aircraft complexity has been increased rapidly with the arrival of IMA architecture. Although IMA provides benefits regarding SWaP, IMA also brings complex interfaces and interdependences. Traditional paper-based system engineering is unable to account the considerable complexity introduced by DIMA architectures. The majority (70%) of the faults are introduced in the early design and analysis phase and are due to errors in requirements and system interaction or interface definitions. However, MBSE promising to address these challenges. MBSE can provide consistency, end-to-end traceability and integration throughout the development phases for complex systems like DIMA.

The scope of the thesis focuses on the improvement of early analysis and design phase for implementing networked aircraft control system in an IMA architecture. To do so, an ECS case study has been implemented. The state of the art explored the ECS and various technologies used by subsystems. Further, IMA technology is studied along with the aerospace standards and guidelines that must be followed for the use-case. State of the art also explored existing MBSE frameworks and concluded that ARCADIA/Capella supports the use case by providing the functional analysis as specified by ARP4754A. ARCADIA/Capella has four working levels in every engineering level. The first two levels of operational analysis and

system analysis specify the need and remaining two levels logical architecture and physical architecture specifies the solution. However, state of the art realized that the existing MBSE use-cases focused on the feasibility studies and a gap in methodology exists.

The proposed methodology defines a systematic MBSE approach by strictly following the development process defined in the aerospace guidelines ARP4754A and DO-178C. Further, the methodology supports system engineering in multi-level and defines a proper transition in a top-down approach. Although there exist commercial variability management tools, the methodology addresses demonstrate how systems with few variants can be efficiently managed using the generic and derived specific models while still assuring consistency between subsystems right down to the item level. Horizontal adaptation and vertical transition are the two methods defined in this thesis. The horizontal adaptation deals with variability through abstraction and encapsulation thus providing an insight into the solution without losing the overall picture. The vertical transition deals with variability and level transitioning and provides a detailed low-level specification.

The specification models are defined for generic ECS architecture with the generic controller, ECS HMI architecture, bleed and bleedless ECS architecture, CPCS architecture for the fully electric and electro-pneumatic system and CPCS application. Also, several viewpoints were also explored to project the capability of Capella tool to address other engineering activities such as Tideal, XHTML documentation generation and performance viewpoint.

To conclusion, the methodology covers a complete spectrum, from aircraft-level specification to controller implementation on the avionics platform. The various steps of the methodology are illustrated for the aircraft ECS and the CPCS in particular. The thesis shows how the MBSE approach using the Capella tool can be used to implement all the process steps required by the SAE ARP4754A and DO-178C. A demonstration platform is presented that enables virtual testing of the developed controller using a Simulink plant model to represent the system. Overall, the presented work significantly contributes to the further development of a DIMA platform by improving the specification capabilities. With the established

methodological framework and the demonstration platform, it is possible to perform predictions of how many additional resources might become available at runtime, considering a more granular definition of the control system's operating conditions. The MBSE approach enables to provide a centralized architecture definition with a unified information model. Furthermore, the model-based system specifications are much more complete, allowing for virtual testing of the integrated system, which leads to more mature specifications in the real product and hence reduces rework in later design phases and thus also development costs. The presented methodology applies to other aircraft-control systems and contributes to the model-based development of future DIMA platforms.

6.1 Future works

The immediate future work is to validate the ECS controller specifications through the completion of the demonstration. The controller code generated by Simulink needs to be adapted for an application deployed in the VxWorks environment. Once an application is deployed, the communication between ECS HMI, CPCS controller and real-time plant simulator needs to be established through TTETHERNET protocol. In the modelling aspect one interesting work to be done is to continue exploring the Time4Sys viewpoint to perform time analysis on the model. Another major work is to implement the Team for Capella, so that a shared environment can be created to collaborate multi-level system engineers.

The proposed methodology specified in this thesis has also investigated requirements management. However, a guideline should be developed on how to define requirements at each engineering level. Further, the guideline should also address the standardization of MBSE nomenclature. For example the standardization in formulation of function names, exchange names and other nomenclatures. This will further help in the wide spread adoption of MBSE in an organization. Another important aspect is to schedule a maintenance period to update to the newest versions of the tool. The tool upgrades provide more capabilities to model specifications. For instance, while in thesis most of the models were made in Capella 1.1.1

to keep the model accessible to specialist while Capella 1.2 was already in the market. The reason is because a large number of users were adapting to MBSE and the few specialists used MBSE tool only to support model review process. As a result, the upgrading of the tool was hard to implement throughout the organization. The organization can make use of the Capella Studio to develop the much-needed bridge between Capella and Simulink. This will facilitate a complete Model-Driven Engineering (MDE) framework. Moreover, a bridge between the tool and Product Lifecycle Management (PLM) will effectively help to track the versions and provide a library and repository for models to be shared.

The case study was performed with the open-source tool Capella but could also be implemented in other MBSE tool frameworks using similar concepts. However, the efficiency of the overall process highly depends on the tool infrastructure. For example, the full benefit of an architecture-centric approach only becomes available if other engineering activities such as safety assessments, performance assessments and optimization, and requirements engineering are also performed using the same reference. The efficiency of this approach strongly depends on the tool infrastructure. Future work can be performed to fully benefit from the developed framework and to investigate in-depth aspects such as the development of customized viewpoints, the investigation of bridges between tools for virtual integration (e.g. to Simulink), safety analyses, and DIMA resources analyses.

In summary, a significant contribution was made by providing a practical but reusable extensive example of MSBE methodology for aircraft control system specifications for DIMA platforms.

List of Publications

- George Mathew, P., Liscouet-Hanke, S., and Le Masson, Y., "Model-Based Systems Engineering Methodology for Implementing Networked Aircraft Control System on Integrated Modular Avionics - Environmental Control System Case Study," SAE Technical Paper 2018-01-1943, 2018, https://doi.org/10.4271/2018-01-1943.

- Presented at Aerospace Systems and Technology Conference (ASTC) in UK, London, November 2018.

Bibliography

[1] "Bombardier Aerospace Granted Authority to Offer CSeries Aircraft to Customers - Bombardier." *Bombardier Inc*, [Online], Available : `https://bit.ly/2ExwzRw`, [Accessed:07-Jan-2019].

[2] S. Trimble, "Transport Canada announces type certification for CSeries." *FlightGlobal*, [Online], Available : `https://bit.ly/2EyZeFN`, [Accessed:07-Jan-2019].

[3] P. Jackson and K. Munson, "IHS Jane's all the world's aircraft. Development & production," 2019.

[4] S. Holt, P. Collopy, and D. DeTurris, "So It's Complex, Why Do I Care?," in *Transdisciplinary Perspectives on Complex Systems*, pp. 25–48, Cham: Springer International Publishing, 2017.

[5] H. Salzwedel, "Mission level design of avionics," in *The 23rd Digital Avionics Systems Conference (IEEE Cat. No.04CH37576)*, pp. 9.D.2–91–10, IEEE.

[6] H.-H. Altfeld, *Commercial aircraft projects : managing the development of highly complex products*. Ashgate Pub., 2010.

[7] B. Annighoefer, V. Posternak, and F. Thielecke, "Empirical investigations on avionics scaling laws," in *2016 IEEE/AIAA 35th Digital Avionics Systems Conference (DASC)*, pp. 1–10, IEEE, September 2016.

[8] A. J. Kornecki, "Airborne Software: Communication and Certification," *SCPE*, vol. 9, no. 1, pp. 77–82, 2008.

[9] Federal Aviation Administration, "Handbook for Ethernet-Based Aviation Databuses: Certification and Design Considerations," tech. rep., 2005.

[10] R. E. McShea, *Test and Evaluation of Aircraft Avionics and Weapon Systems.* 2014.

[11] P. Eremenko, "Historical schedule trends with complexity by DARPA," *Adaptive Vehicle Make (AVM). Proposers' Day BriefingTactical Technology Office, DARPA*, 2017.

[12] SAE International, "ARP4754 A Guidelines for Development of Civil Aircraft and Systems," 2010.

[13] J. Gausemeier and S. Moehringer, "VDI 2206- A New Guideline for the Design of Mechatronic Systems," *IFAC Proceedings Volumes*, vol. 35, pp. 785–790, December 2002.

[14] B. Lewis and P. H. Feiler, "Incremental Verification and Validation of System Architecture for Software Reliant Systems Using AADL(Architecture Analysis & Design Language)." Software Engineering Institute, Layered Assurance Workshop, Carnegie Mellon University, Pittsburgh, 2010.

[15] P. H. Feiler, "Model-based validation of safety-critical embedded systems," in *2010 IEEE Aerospace Conference*, pp. 1–10, IEEE, March 2010.

[16] B. W. Boehm, *Software Engineering Economics (Prentice-Hall Advances in Computing Science & Technology Series).* 1981.

[17] Research Triangle Institute (RTI), "The Economic Impacts of Inadequate Infrastructure for Software Testing Final Report 02-3," tech. rep., National Institute of Standards & Technology (NIST), U.S Department of Commerce, Technology Administration, Project Number 7007.011, 2002.

[18] D. Galin, *Software Quality Assurance: From Theory to Implementation.* Pearson/Addison-Wesley, 2003.

[19] Federal Aviation Administration, "Handbook for Real Time Operating Systems Integration and Component Integration Consideration in Integrated Modular Avionics Systems," 2008.

[20] P. E. Gartz, "Systems Engineering." *Boeing*, tutorial at 14th DASC, Boston/MA,

November 1995.

[21] C. R. Spitzer, "Digital Avionics - an International Perspective," *IEEE Aerospace and Electronic Systems Magazine*, vol. 7, no. 1, pp. 44–45, 1992.

[22] J.-B. Itier, "A380 Integrated Modular Avionics," in *Network of Excellence on Embedded Systems Design*, (Rome), pp. 6–19, 2007.

[23] M. Morgan, "Integrated modular avionics for next generation commercial airplanes," *IEEE Aerospace and Electronic Systems Magazine*, vol. 6, pp. 9–12, August 1991.

[24] A. Mairaj, "Preferred choice for resource efficiency: Integrated Modular Avionics versus federated avionics," in *2015 IEEE Aerospace Conference*, pp. 1–6, IEEE, March 2015.

[25] Boeing Commercial Airplane Group, "777 Application Specific Integrated Circuits (ASIC) Certification Guideline,"

[26] F. M. G. Dorenberg, "Modular avionics." lecture at UCLA , 1997.

[27] Kafyeke,Fassi, "Quand l'environnement inspire l'innovation - rapport final de fin de projet," tech. rep., 2015. *SAGE-2 Consortium*.

[28] P. E. Gartz, "Avionics Development and Integration System Methods," *IEEE Aerospace and Electronic Systems Magazine*, vol. 2, pp. 2–8, June 1987.

[29] D. D. Walden, G. J. Roedler, K. Forsberg, R. D. Hamelin, T. M. Shortell, and International Council on Systems Engineering., *Systems engineering handbook : a guide for system life cycle processes and activities*. 4th ed., 2015.

[30] National Aeronautics and Space Administration (NASA), *NASA Systems Engineering Handbook*. revision 1 ed., 2007.

[31] J. S. Macias, J. M. G. Rey, C. A. Gonzalez, C. E. Roso, J. S. Velandia, J. P. Barreto, N. Ochoa, C. F. Rodriguez, and A. Garcia-Rozo, "Design and implementation of a Lunabot using NASA Systems Engineering," in *2012 IEEE 4th Colombian Workshop on*

Circuits and Systems (CWCAS), pp. 1–6, IEEE, November 2012.

[32] B. W. Boehm, "A spiral model of software development and enhancement," *Computer*, vol. 21, pp. 61–72, May 1988.

[33] W. W. Royce, "Management of the Development of Large Software Systems: Concepts and Techniques," in *Wescon Conference 1970, International Conference on Software Engineering*, 1987.

[34] A. J. Shenhar, V. Holzmann, B. Melamed, and Y. Zhao, "The Challenge of Innovation in Highly Complex Projects: What Can We Learn from Boeing's Dreamliner Experience?," *Project Management Journal*, vol. 47, no. 2, pp. 62–78, 2016.

[35] M. Schulte, "Model-Based Integration of Reusable Component-Based Avionics Systems - A Case Study," in *Eighth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC'05)*, pp. 62–71, IEEE.

[36] J. Yin, B. Lawler, and H. Jin, "Application of model based system engineering to IMA development activities," in *2017 IEEE/AIAA 36th Digital Avionics Systems Conference (DASC)*, pp. 1–7, IEEE, September 2017.

[37] D. Dori, *Object-Process Methodology*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002.

[38] INCOSE, "Systems Engineering Vision 2020. version 2.03 edn.," *Technical Operations, International Council on Systems Engineering*, vol. INCOSE-TP-, 2007.

[39] "System Engineering Methodology for Implementation of Networked Aircraft Control Systems on Distributed Integrated Modular Architecture." *Concorida University, Bombardier Aerospace, Mitacs*, [Online], Available : `https://bit.ly/2EHJRL8`, [Accessed:02-Jan-2018].

[40] International Organization for Standardization, "Standard Atmosphere ISO 2533:1975," 1975.

[41] Bombardier Aerospace, "Bombardier Global Express - Altitude and Temperature Operating Limit - Flight Crew Operating Manual," tech. rep. November 2006.

[42] M. Sinnett, "787 No-Bleed Systems -Saving Fuel and Enhancing Operational Efficiencies," *AERO Magazine QTR_4.07*, [Accessed:20-Jan-2019].

[43] R. Burkhart, "Modeling Standards Activity Team - MBSE Initiative," in *MBSE International Workshop - INCOSE*, (Phoenix, AZ), 2011.

[44] "MBSE:standards ." *MBSE Wiki*, [Online], Available : `https://bit.ly/2tvksy3`, [Accessed:28-Jun-2018].

[45] "SysML Open Source Project - What is SysML? Who created SysML?." *Eclipse Foundation*, [Online], Available : `https://sysml.org/`, [Accessed:02-Feb-2019].

[46] "What is UML ." *Unified Modeling Language*, [Online], Available : `http://www.uml.org/what-is-uml.htm`, [Accessed:05-Jan-2019].

[47] C. J. Paredis, Y. Bernard, R. M. Burkhart, H.-P. de Koning, S. Friedenthal, P. Fritzson, N. F. Rouquette, and W. Schamai, "An Overview of the SysML-Modelica Transformation Specification," 2010.

[48] "OMG UML Profile for DoDAF/MODAF." *Object Management Group*, [Online], Available : `https://www.omg.org/updm/`, [Accessed:05-Jan-2019].

[49] "OMG MARTE Web site." *Object Management Group*, [Online], Available : `https://www.omg.org/omgmarte/`, [Accessed:05-Jan-2019].

[50] S. Gérard, J. Medina, and D. Petriu, "MARTE: A New Standard for Modeling and Analysis of Real-Timhttps://www.overleaf.com/project/5c782592b91383699645afa0e and Embedded Systems," in *Euromicro Conference on Real-Time Systems (ECRTS 07)*, (Pisa, Italy), 2007.

[51] E. J. Barkmeyer, A. B. Feeney, P. Denno, D. W. Flater, D. E. Libes, M. P. Steves, and E. K. Wallace, "Concepts for automating systems integration," tech. rep., National

Institute of Standards and Technology, Gaithersburg, MD, 2003.

[52] CCSDS, "Reference Architecture for Space Data Systems.," in *Recommendation for Space Data System Practices (Magenta Book)*, vol. Issue 1, (Washington, D.C), 2008.

[53] ISO/IEC JTC 1/SC 7 Software and systems engineering, "ISO/IEC 19793: Information technology – Open Distributed Processing – Use of UML for ODP system specifications," 2015.

[54] ISO/IEC JTC 1/SC 7 Software and systems engineering, "ISO/IEC/IEEE 42010:2011 - Systems and software engineering – Architecture description," 2011.

[55] D. Huart and O. Olechowski, "Towards a Model-Based Systems Lifecycle: CPCS from Design to Operations," *International Workshop on Aircraft System Technologies*, 2017.

[56] J.-L. Voirin, *Model-Based System and Architecture Engineering with the Arcadia Method*. Elsevier, first ed., 2017.

[57] J.-L. Voirin, S. Bonne, D. Exertier, and V. Normand, "Simplifying (and enriching) SysML to perform functional analysis and model instances," in *Annual INCOSE International Symposium*, 2016.

[58] J.-L. Voirin, "Method and tools to secure and support collaborative architecting of constrained systems," in *27th Congress of the International Council of the Aeronautical Science (ICAS 2010)*, (Nice, France), 2010.

[59] B. Langlois and J. Barata, "Extensibility of Capella with Capella Studio." *Eclipse Foundation*, [Online], Available : `https://bit.ly/2NjZPOb`, [Accessed:25-Jan-2019].

[60] L. Batista and O. Hammami, "Capella Based System Engineering Modelling and Multi-Objective Optimization of Avionics Systems," in *ISSE -Int. Symp. Syst. Eng.*, 2016.

[61] "pure::variants." *Pure Systems*, [Online], Available : `https://www.pure-systems.com/products/pure-variants-9.html`, [Accessed:03-Aug-2018].

[62] H. Jahanara, S. Liscouët-Hanke, and J.-L. Bauduin, "A Model-Based Systems Engineering Approach for the Development of Test Means for Flight Control Systems," in *Aerospace Systems and Technology Conference (ASTC)*, 2018.

[63] A. Jeyaraj and S. Liscouët-Hanke, "A Model-Based Systems Engineering approach for efficient flight control system architecture variants modelling in conceptual design," in *Recent Advances in Aerospace Actuation Systems and Components*, (Toulouse, France), 2018.

[64] S. Liscouët-Hanke, B. R. M. Mohan, P. J. Nelson, C. Lavoie, and S. Dufresne, "A Model-Based Systems Engineering Approach for the Conceptual Design of Advanced Aircraft High-Lift System Architectures," in *CASI AERO conference, Toronto*, 2017.

[65] M. Sadri, "Modeling of Landing Gear System," *Bombardier Aerospace, Concordia University.* internship report, 2017.

[66] RTCA, "DO-297 Integrated Modular Avionics (IMA) Development Guidance and Certification Considerations," 2005.

[67] RTCA, "DO-178C, Software Considerations in Airborne Systems and Equipment Certification," 2012.

[68] RTCA, "D0-331,Model-Based Development and Verification Supplement to DO-178C and DO-278A," 2011.

[69] Aeronautical Radio Inc., "ARINC 651 Design Guidance for Integrated Modular Avionics," 1997.

[70] Aeronautical Radio Inc., "ARINC 653 Specification Avionics Application Standard Software Interface," 2015.

[71] Aeronautical Radio Inc., "ARINC 661 Cockpit Display System Interfaces to User systems," 2016.

[72] SAE International, "SAE AS6802: Time-Triggered Ethernet," 2016.

[73] ASHRAE Standards Committee, "ASHRAE Standard 161-2013: Air Quality within Commercial Aircraft," 2013.

[74] "Capella Add-ons." *Pure Systems*, [Online], Available : `https://polarsys.org/capella/addons.html`, [Accessed:20-Nov-2018].

[75] "M2Doc - M2Doc." [Online], Available : `http://www.m2doc.org/`, [Accessed:25-Jan-2019].

[76] L. Wagner, "Formal Methods for Certification: Why and How?," 2016.

[77] SAE International, "APPENDIX E - Contiguous Aircraft/System Development Process Example," 2010.

[78] "Capella Guide." Help - Capella, Capella 1.3.0.

[79] S. Bonnet, J.-L. Voirin, D. Exertier, and V. Normand, "Modeling system modes, states, configurations with Arcadia and Capella: method and tool perspectives; Modeling system modes, states, configurations with Arcadia and Capella: method and tool perspectives," in *27th Annual INCOSE International Symposium*, (Adelaide, Australia), 2017.

[80] "Time4Sys." *PolarSys*, [Online], Available : `https://www.polarsys.org/time4sys`, [Accessed:25-Jan-2019].

[81] "Team for Capella." *Obeo.*[Online].Available : `https://bit.ly/2SQK5IM`. [Accessed:22-Jan-2018].

[82] "Safety Architect." *ALL4TEC*, [Online], Available : `https://www.all4tec.com/safety-architect`, [Accessed:25-Jan-2019].

[83] Fábio Yukio Kurokawa, Cláudia Regina de Andrade, and Edson Luiz Zaparoli, "Determination of the outflow valve opening area of the aircraft cabin pressurization system," in *18th International Congress of Mechanical Engineering*, (Ouro Preto, MG), 2005.

[84] *SAE Aerospace, Applied Thermodynamics Manual ; 6: Characteristics of equipment*

components, equipment cooling system design, and temperature control system design. Aerospace information report, Society of Automotive Engineers, 3. ed ed., 1994.

[85] "Embedded Coder - MATLAB & Simulink." *MathWorks*, [Online], Available : `https://www.mathworks.com/products/embedded-coder.html`, [Accessed:05-Feb-2019].

[86] "VAPS XT - COTS Modeling & Simulation Software." *Presagis*, [Online], Available : `https://www.presagis.com/en/product/vaps-xt`, [Accessed:28-Jun-2018].

[87] Y. Lefebvre, "Mastering the arinc 661 standard," in *SAE Technical Paper*, SAE International, October 2011.

[88] J. F. Asfia, K. R. Williams, W. A. Atkey, C. J. Fiterman, S. M. Loukusa, and C. Y. Ng, "Electric air conditioning system for an aircraft Patent: US6526775B1," September 2001. *Boeing Co*, [Online], Available : `https://patents.google.com/patent/US6526775B1/en`.

Appendix A

Capella diagrams

This sections provides an overview on Capella diagrams in all ARCADIA working levels. It is recommended that the creation of diagrams follow the order in which diagrams are introduced in this appendix.

A.1 Operational Analysis (OA)

The first level is Operational Analysis, where the stakeholder needs are expressed. The OA provides a need understanding on *' What the users of the system need to accomplish '*. Following seven diagrams specific to OA.

A.1.1 Operational Entity Diagram [OEBD]

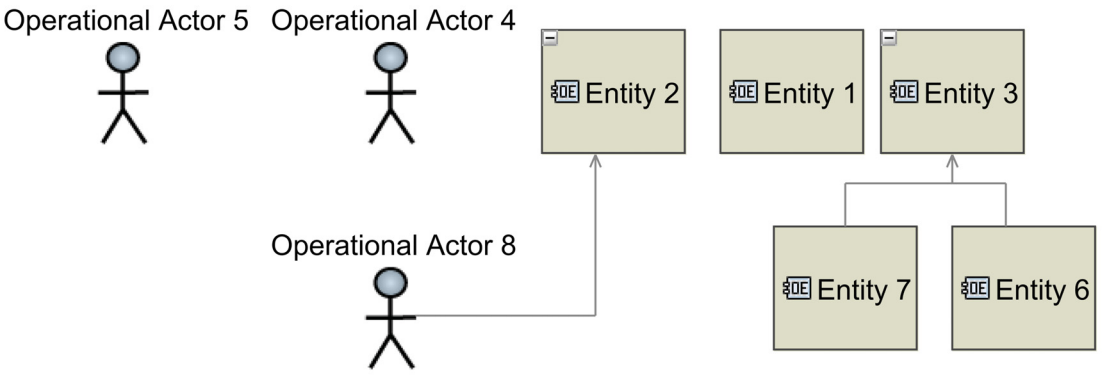

Figure A.1: [OEBD]- Operational Entity Breakdown Diagram

The entity diagram shows all the actors and entities in an OA and any relationship between them. For instance,*Operational Actor 8* is contained in *Entity 2*. Further, *Entity 7* and

Entity 6 is contained in *Entity 3*.

A.1.2 Operational Capability diagram [OCB]

The Capability diagram in OA shows the capabilities of actors and entities as shown in Figure A.2. For this example, the *Operational Capability 3* is contained in *Operational Capability 2*. That is, to provide a specific use-case *Operational Capability 2* would need the *Operational Capability 3*.

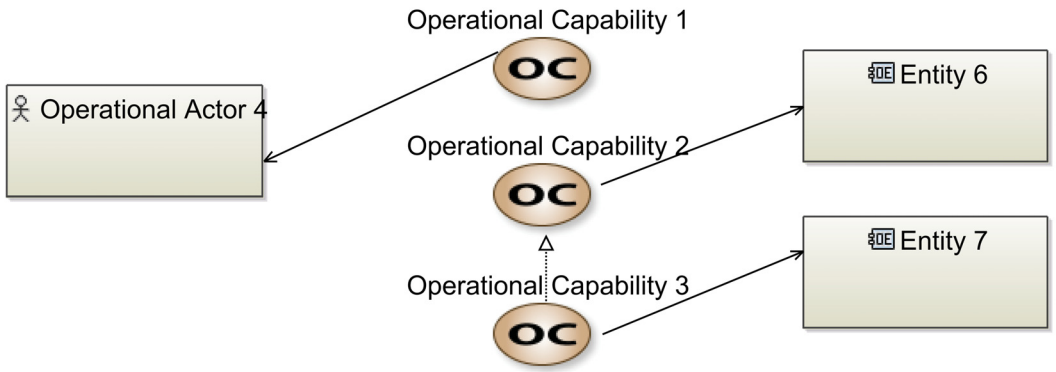

Figure A.2: [OCB]- Operational Capabilities Blank diagram

A.1.3 Operational Activity Breakdown diagram [OABD]

The *Operational Activity Breakdown diagram* shows all the activities identified in OA. Furthermore, the diagram also shows the nested activities.

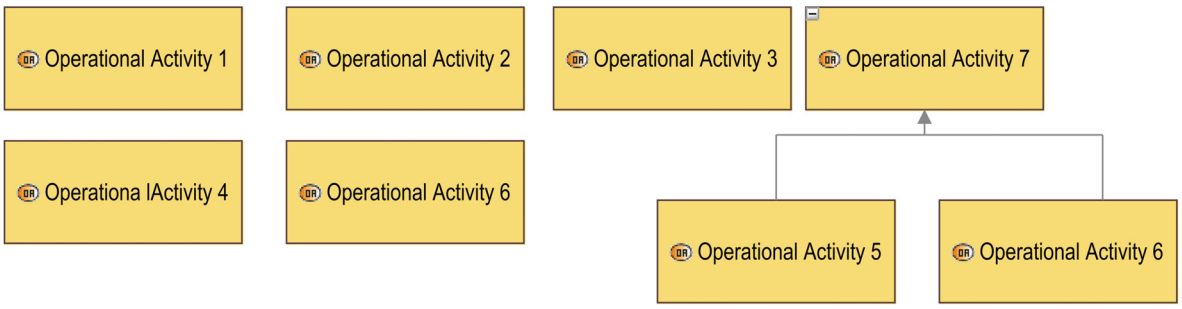

Figure A.3: [OABD]- Operational Activity Breakdown diagram

A.1.4 Operational Activity Interaction diagram [OAIB]

Operational Activity Interaction diagram presents the interaction or exchange between the activities as shown in Figure A.4. Each connection between an activity is a one directional flow of information. For instance, in Figure A.4 *Operational Activity 1 & 6* has a one way forward interaction to *Operational Activity 4* through *Interaction 1 & 2*. The green port is output and red port is input to the function.

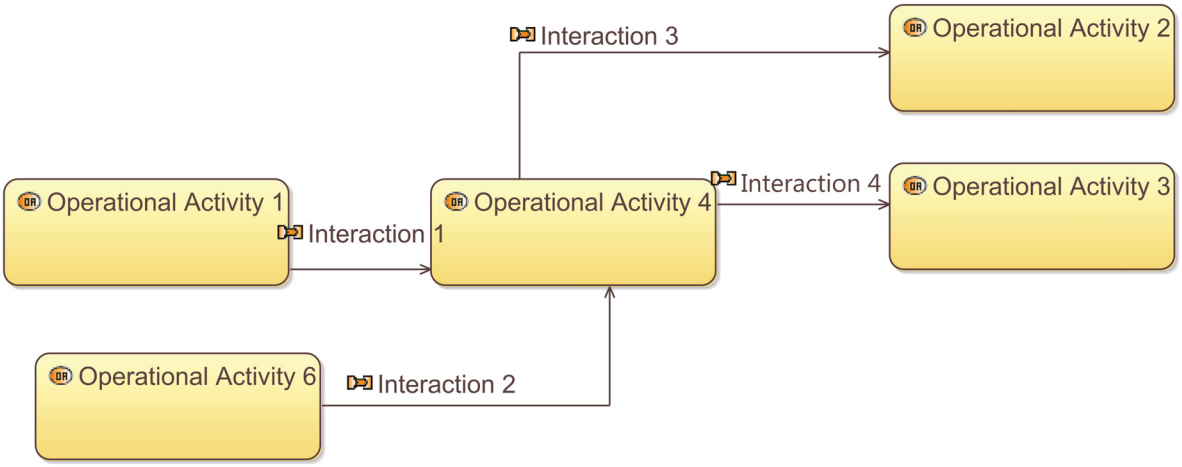

Figure A.4: [OAIB]- Operational Activity Interaction Blank diagram

A.1.5 Operational Activity Scenario [OAS]

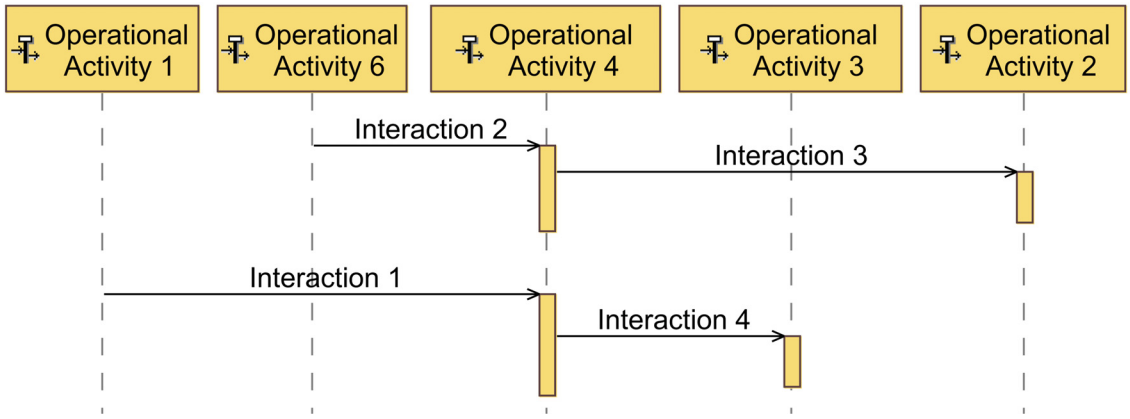

Figure A.5: [OAS]- Operational Activity Scenario diagram

The *Operational Activity Scenario* diagram presents the sequence by which the activities interact to provide a specific capability. For example, in Figure A.5 the sequence of activities is as follows:

Operational Activity 6 ⇒ *Operational Activity 4* ⇒ *Operational Activity 2* ⇒ *Operational Activity 1* ⇒ *Operational Activity 4* ⇒ *Operational Activity 3*

A.1.6 Operational Architecture diagram [OAB]

The *Operational Architecture diagram* presents the overall architecture at OA. As shown in Figure A.6, the architecture includes identified *actors*, *entities*, *activities*, *interactions* and *communication means*.

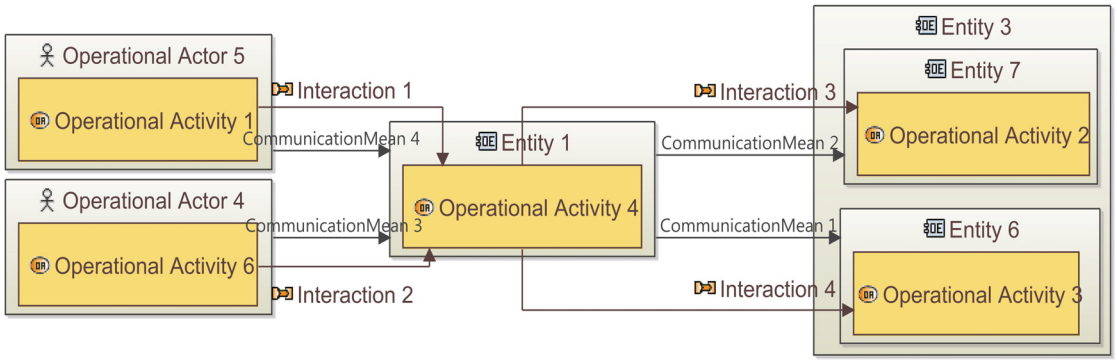

Figure A.6: [OAB]- Operational Architecture Blank diagram

A.1.7 Operational Exchange Scenario diagram [OES]

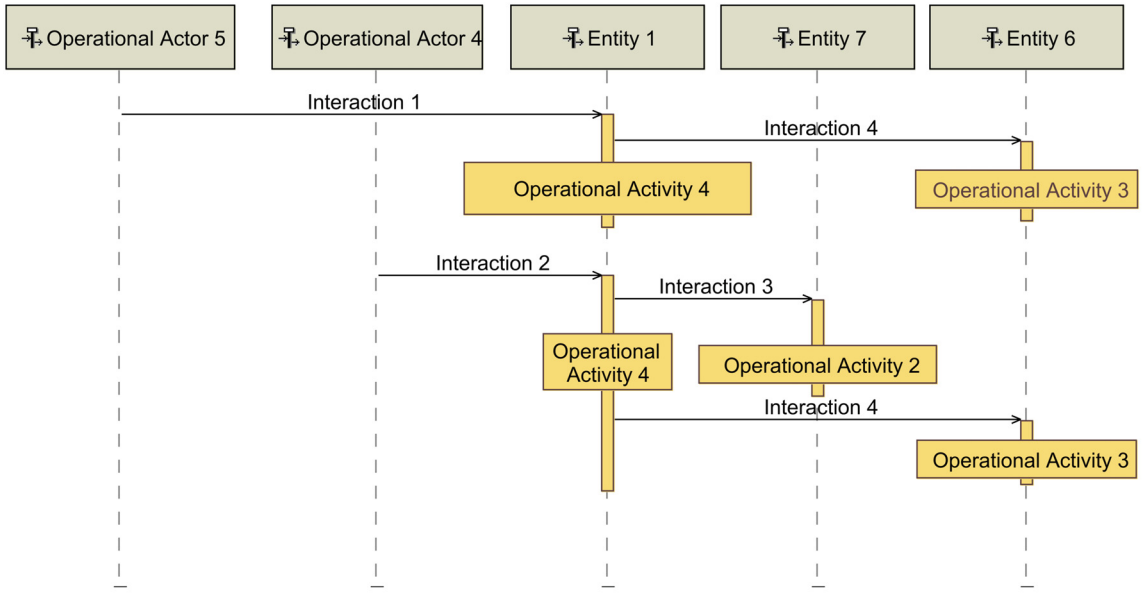

Figure A.7: [OES]- Operational Entity Scenario diagram

The *Operational Exchange Scenario* diagram presents the sequence by which the actors and entities interact to provide a specific capability. For example, in Figure A.7 the sequence of interactions is as follows:

Operational Actor 5 $\Rightarrow$ *Entity 1* $\Rightarrow$ *Entity 6* $\Rightarrow$ *Operational Actor 4* $\Rightarrow$ *Entity 1* $\Rightarrow$ *Entity 7* $\Rightarrow$ *Entity 6.*

A.2 System Analysis (SA)

The System Analysis provides the need understanding on *' what the system has to do for users'*. SA is a sum of operational need understanding and system need understanding. Following seven diagrams are specific to SA.

A.2.1 System Contextual Actor diagram [CSA]

The *System Contextual Actor* diagram shows all the actors in SA level along with system of interest. It should be noted that all the actors and entities identified in OA will be transformed into actors in SA.

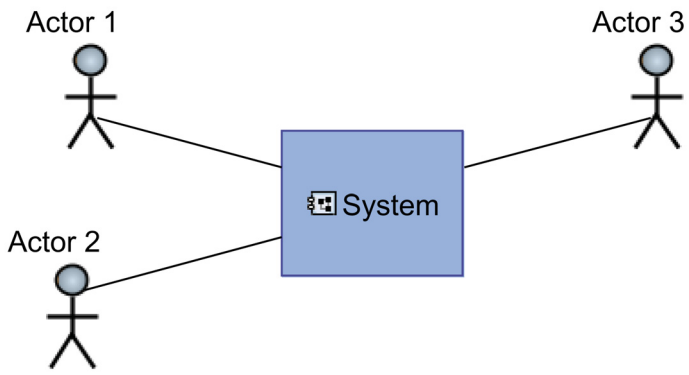

Figure A.8: [CSA]- System Contextual Actor diagram

A.2.2 Mission diagram [MB]

The *Mission diagram* presents the mission of the system and the capabilities and actors that needs to complete the mission as shown in Figure A.9.It should be noted that the operational capabilities will be acquired during transition from OA. The operational capability can be adapted for SA and additional system capabilities can be defined.

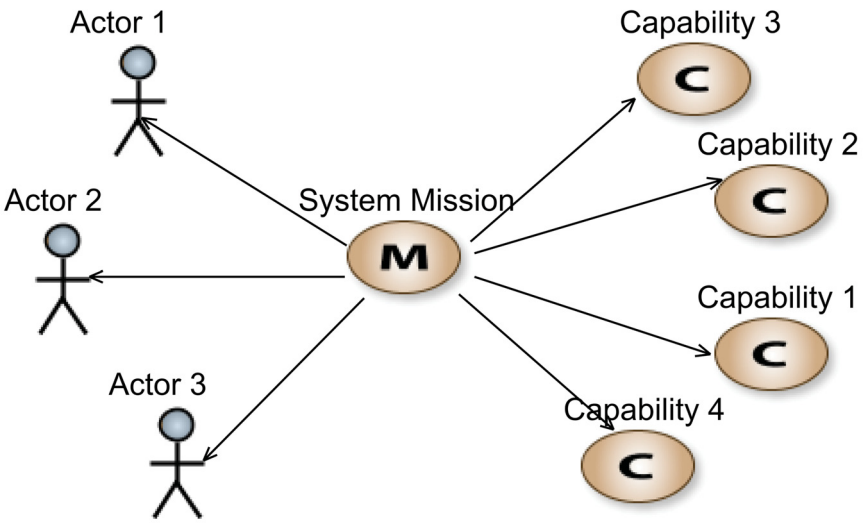

Figure A.9: [MB]- Mission Blank diagram

A.2.3 System Functional Breakdown diagram [SFBD]

The *System Functional Breakdown diagram* presents all the functions needed to represent the system need. Further, as shown in Figure A.10 the also contains the operational activities defined at OA. During transition to SA the activities are transformed into functions to support the system need.

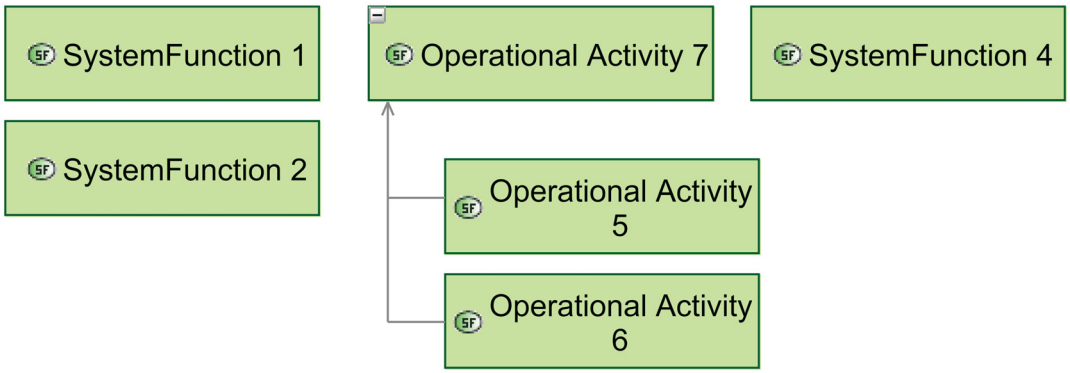

Figure A.10: [SFBD]- System Functional Breakdown diagram

A.2.4 System Functional Dataflow diagram [SDFB]

The *System Functional Dataflow* diagram shows the realized exchanges between functions. It is to be noted that only leaf function can exchange information. Each connection or functional exchange is uni-directional. Moreover, once allocated, all the actor functions will be presented in blue colour in a dataflow diagram.

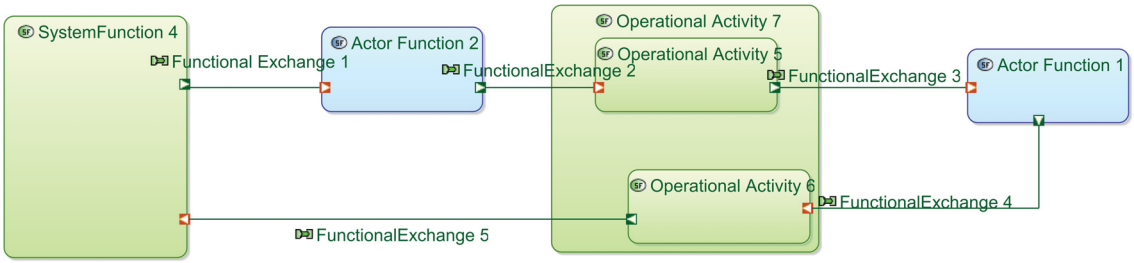

Figure A.11: [SDFB]- System Functional Dataflow Blank diagram

A.2.5 Functional Scenario [FS]

The *Functional Scenario* diagram presents the sequence by which the functions interact to provide a specific capability. For example, in Figure A.12 the sequence of functions is as follows:

Actor Function 1 ⇒ *Operational activity 6* ⇒ *System Function 4* ⇒ *Actor Function 2* ⇒ *Operational Activity 5* ⇒ *Actor Function 1.*

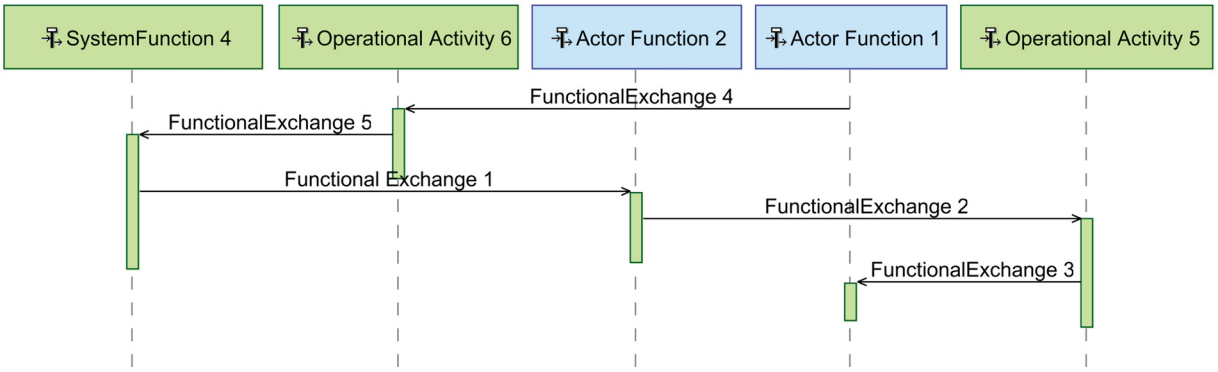

Figure A.12: [FS]- System Functional Scenario Blank diagram

A.2.6 System Architecture diagram [SAB]

The *System Architecture* diagram presents the overall architecture at SA. As shown in Figure A.13, the architectural diagram consist of actors, system of interest, actors & system functions and exchanges between them.

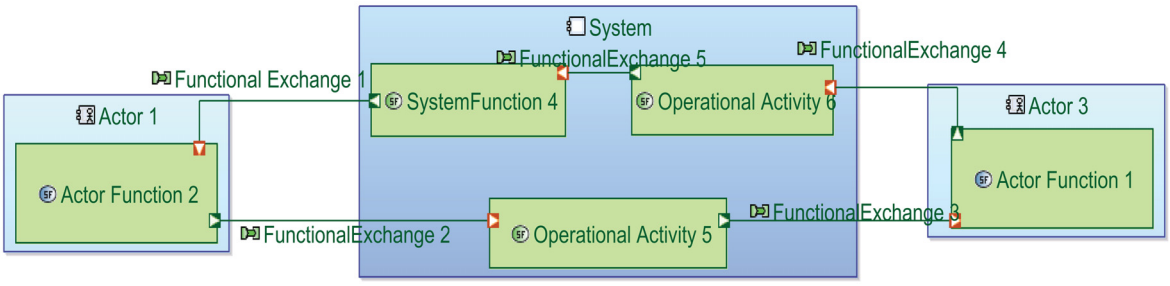

Figure A.13: [SAB]- System Architecture Blank diagram

A.2.7 System Exchange Scenario diagram ES

The *System Exchange Scenario* diagram presents the sequence by which the actors and system of interest interact to provide a specific capability. For example, in Figure A.14 the sequence of interactions is as follows:

System Actor 3 $\Rightarrow$ *System* $\Rightarrow$ *System* $\Rightarrow$ *System Actor 1* $\Rightarrow$ *System* $\Rightarrow$ *System Actor 3* .

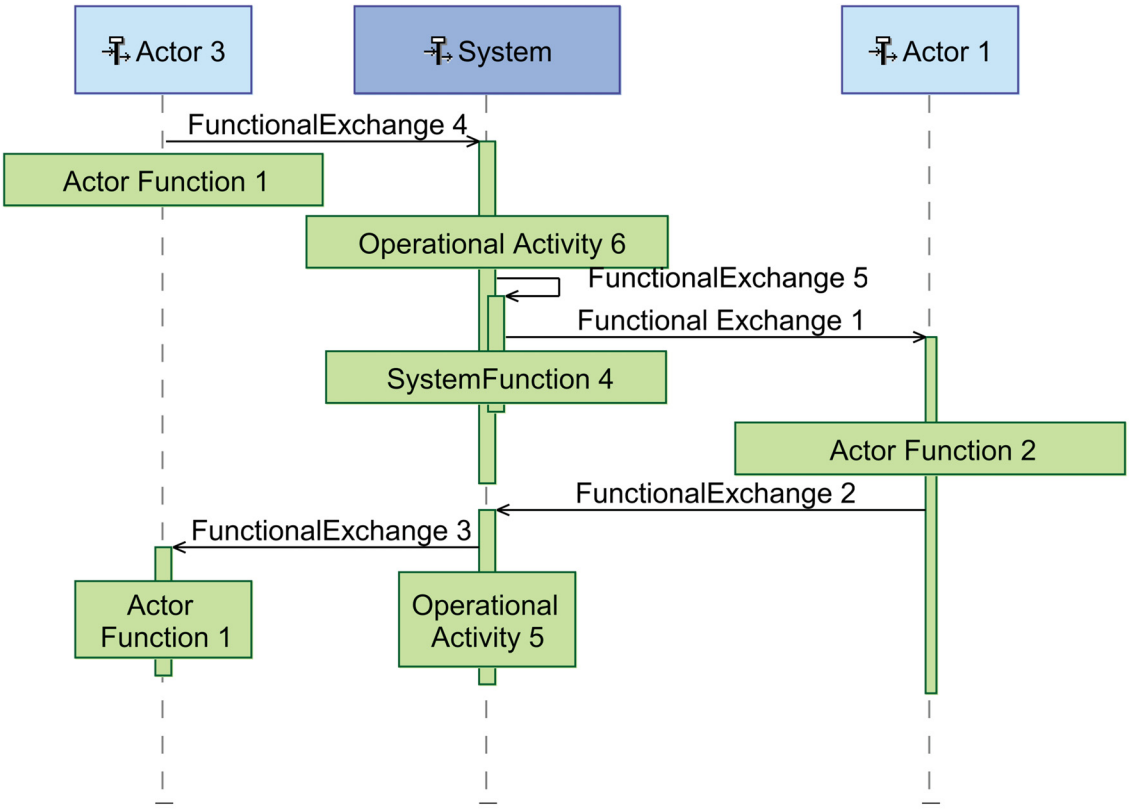

Figure A.14: [ES]- System Entity Scenario diagram

A.3 Logical Architecture (LA)

The Logical Architecture provides the solution on *' how the system works to achieve the required performance.'*. LA is the logical solutions for needs specified in OA, SA. Following diagrams are specific to LA.

A.3.1 Logical Functional Breakdown diagram [LFBD]

The *Logical Functional Breakdown diagram* presents all the logical functions needed to represent the system solution.

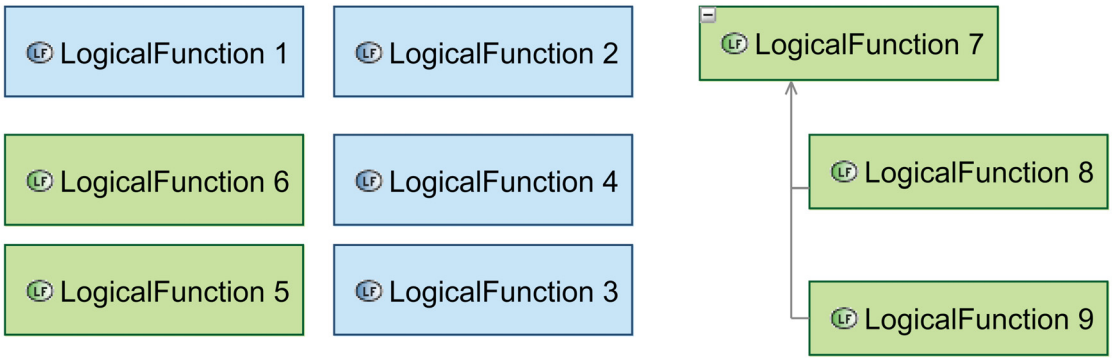

Figure A.15: [LFBD]- Logical Functional Breakdown diagram

A.3.2 Logical Functional Dataflow diagram [LDFB]

The *Logical Functional Dataflow* diagram shows the realized exchanges between functions. It is to be noted that only leaf function can exchange information. Each connection or functional exchange is uni-directional. Moreover, once allocated all the actor functions will be presented in blue colour in a dataflow diagram.

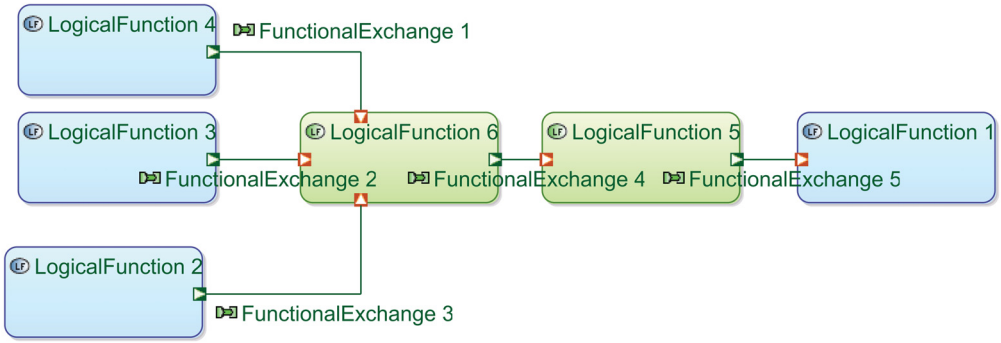

Figure A.16: [LDFB]- Logical Functional Dataflow Blank diagram

A.3.3 Functional Scenario [FS]

The *Functional Scenario* diagram presents the sequence by which the logical functions interact to provide a specific capability. For example, in Figure A.17 the sequence of functions is as follows:

(1) *Logical Function 4* ⇒ *Logical Function 6*, (2) *Logical Function 2* ⇒ *Logical Function 6*, (3) *Logical Function 3* ⇒ *Logical Function 6*, (4) *Logical Function 6* ⇒ *Logical Function 5*, (5) *Logical Function 5* ⇒ *Logical Function 1*.

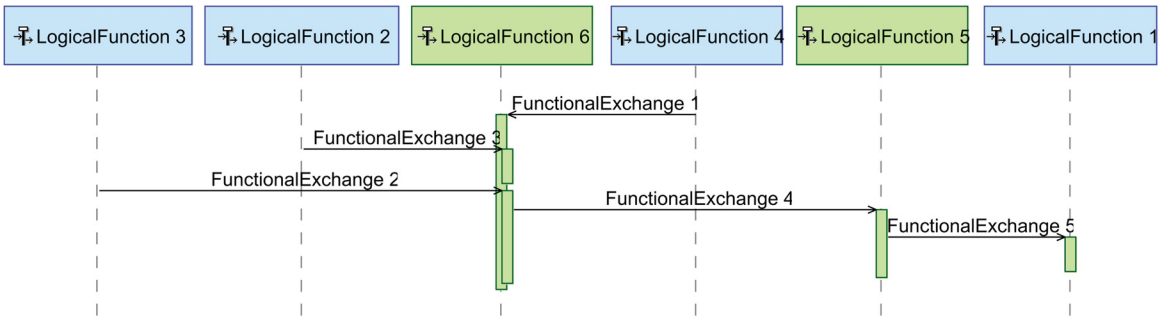

Figure A.17: [FS]- Logical Functional Scenario diagram

A.3.4 Logical Component Breakdown diagram [LCBD]

The *Logical Component Breakdown Actor* diagram shows all the components in LA level.

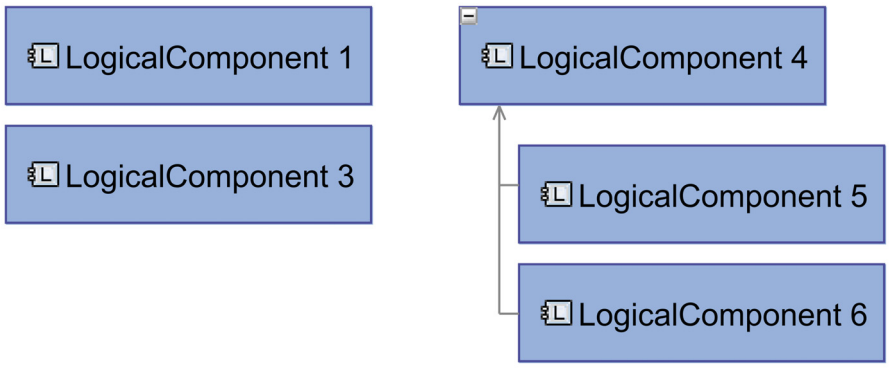

Figure A.18: [LCBD]- Logical Component Breakdown diagram

A.3.5 Logical Architecture diagram [LAB]

The *Logical Architecture* diagram presents the overall architecture at LA. As shown in Figure A.13, the architectural diagram consist of actors, logical components, system of interest, logical functions and exchanges between them.

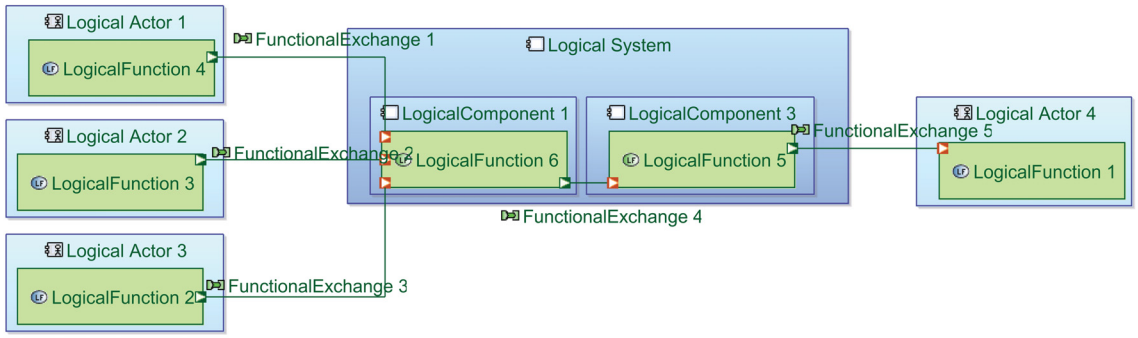

Figure A.19: [LAB]- Logical Architecture diagram

A.3.6 Logical Exchange Scenario diagram [ES]

The *Logical Exchange Scenario* diagram presents the sequence by which the actors and logical components interact to provide a specific capability. For example, in Figure A.20 the sequence of interactions is as follows:

(1) *Logical Actor 1* ⇒ *Logical Component 1*, (2) *Logical Actor 3* ⇒ *Logical Component 1*, (3) *Logical Actor 2* ⇒ *Logical Component 1*, (4) *Logical Component 1* ⇒ *Logical Actor 5*, (5) *Logical Actor 5* ⇒ *Logical Actor 4*.

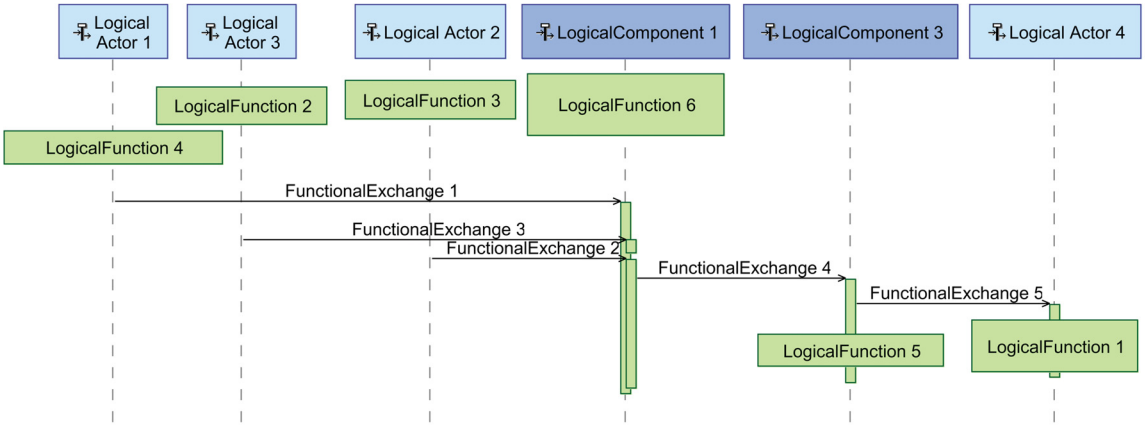

Figure A.20: [ES]- Logical Entity Scenario diagram

A.4 Physical Architecture (PA)

The Physical Architecture provides the solution on *' how the system will be developed and built..'* PA is the developed and built solution from the selected logical solution. Following diagrams are specific to PA.

A.4.1 Physical Functional Breakdown diagram [PFBD]

The *Physical Functional Breakdown diagram* presents all the physical functions needed to represent the system build and development.

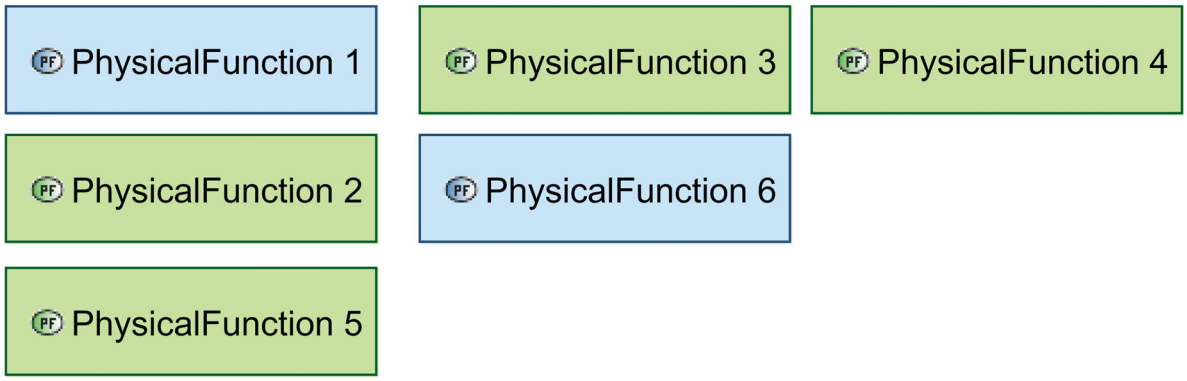

Figure A.21: [PFBD]- Physical Functional Breakdown diagram

A.4.2 Physical Functional Dataflow diagram [PDFB]

The *Physical Functional Dataflow* diagram shows the realized exchanges between Physical functions. Moreover, once allocated, all the actor functions will be presented in blue colour in a dataflow diagram.

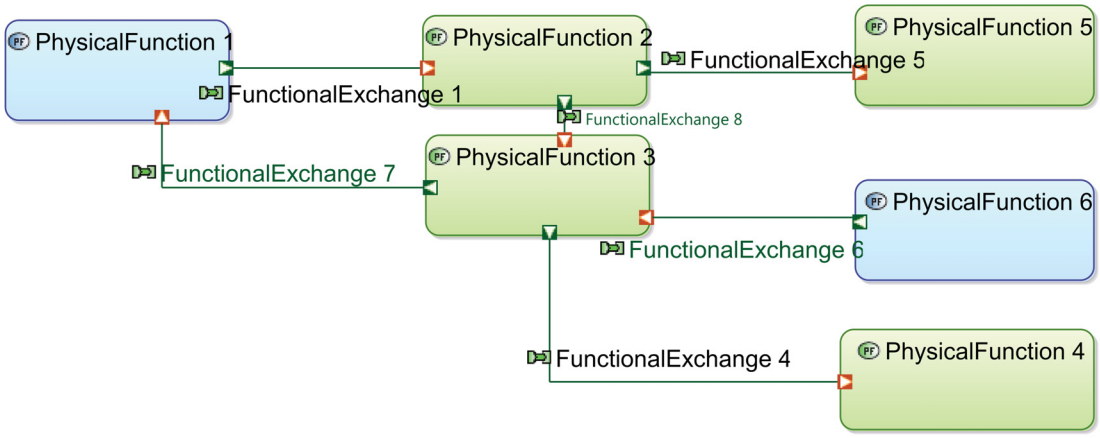

Figure A.22: [PDFB]- Physical Functional Dataflow Blank diagram

A.4.3 Functional Scenario [FS]

The *Functional Scenario* diagram presents the sequence by which the Physical functions interact to provide a specific capability. For example, in Figure A.23 the sequence of functions is as follows:

(1) *Physical Function 1* ⇒ *Physical Function 2*, (2) *Physical Function 2* ⇒ *Physical Function 3*, (3) *Physical Function 3* ⇒ *Physical Function 4*, (4) *Physical Function 6* ⇒ *Physical Function 3*, (5) *Physical Function 3* ⇒ *Physical Function 1*.

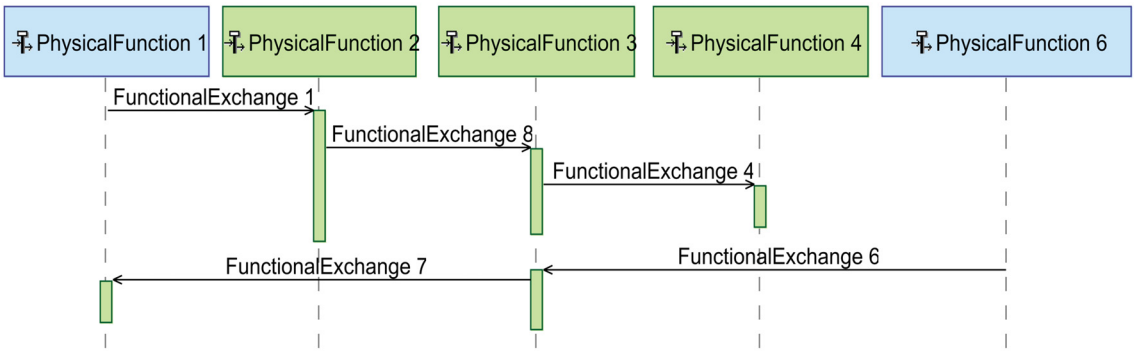

Figure A.23: [FS]- Physical Functional Scenario diagram

A.4.4 Physical Component Breakdown diagram [PCBD]

The *Physical Component Breakdown Actor* diagram shows all the components in PA level. The components include node physical component and behavior physical component.

Figure A.24: [PCBD]- Physical Component Breakdown diagram

A.4.5 Physical Architecture diagram [PAB]

The *Physical Architecture* diagram presents the overall architecture at PA. As shown in Figure A.25, the architectural diagram consist of actors, physical components, physical functions and exchanges between them.

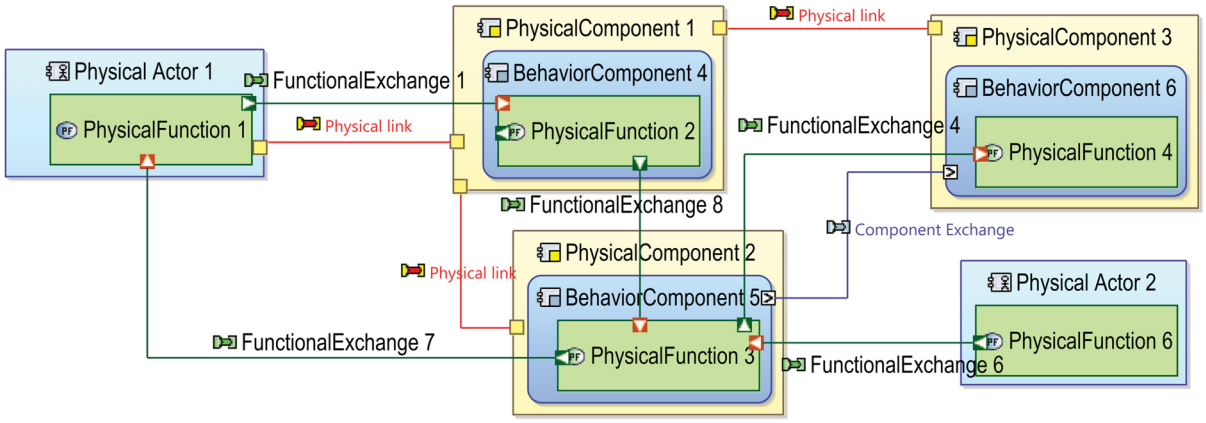

Figure A.25: [PAB]- Physical Architecture diagram

Appendix B

Logical architecture for the bleed-driven ECS

Figure B.1 shows a Bleed ECS architecture based on Challenger 605. A conventional bleed ECS uses bleed air from the engine as an existing source of pneumatic energy. First, the bleed air is filtered for contaminants. Next, the pneumatic power is preconditioned using ram air. Then, the preconditioned air is transferred to the compressor and compressed. Next, the hot, compressed air is cooled in the heat exchanger down to the atmospheric temperature (in ideal conditions). The cooled air is then expanded in the expander. The temperature of the air emitted from the expander is below the atmospheric temperature. Finally, the low-temperature air coming out of the expander gets mixed with the hot air extracted from the preconditioning process, and thus conditioned air is generated. In Figure B.1, three functional chains are shown:

1. Cold air generation chain represented by orange chain.

   This chain highlights the manipulation of airflow to generate cold air. The functional flow is as follows:

   Provide pneumatic power generation and distribution ⇒ Modulate Pneumatic power flow ⇒ Provide protection against FOD ⇒ Regulate Pneumatic power flow to ECS ⇒ Provide primary cooling ⇒ Limit pre-cooled air flow ⇒ Provide primary cooling to hot air ⇒ compress air ⇒ Provide secondary cooling to compressed air ⇒ Provide moisture exaction⇒ Provide air expansion ⇒ Provide humidity control ⇒ prevent backward air flow ⇒ Provide cockpit and cabin ventilation

2. Temperature control flow chain represented by red chain.

   This chain highlights the flow of hot air to control the temperature. As shown in Figure

B.1 a portion of the hot air is used to prevent ice formation in ECS system. The main hot air flow for cabin temperature control is as follows:

Provide primary cooling ⇒ Limit pre-cooled air flow ⇒ Provide cabin and flight deck temperature control ⇒ Limit high temperature in conditioned air ⇒ prevent backward air flow ⇒ Provide cockpit and cabin ventilation

3. Redundant cold air air generation chain represented by yellow chain

   The redundant chain is a measure to facilitate cross flow between two air conditioning systems in case of failure in one. The pre-cooled air is taken by a redundancy mean (usually a cross bleed valve) and feed it to the second air conditioning system.

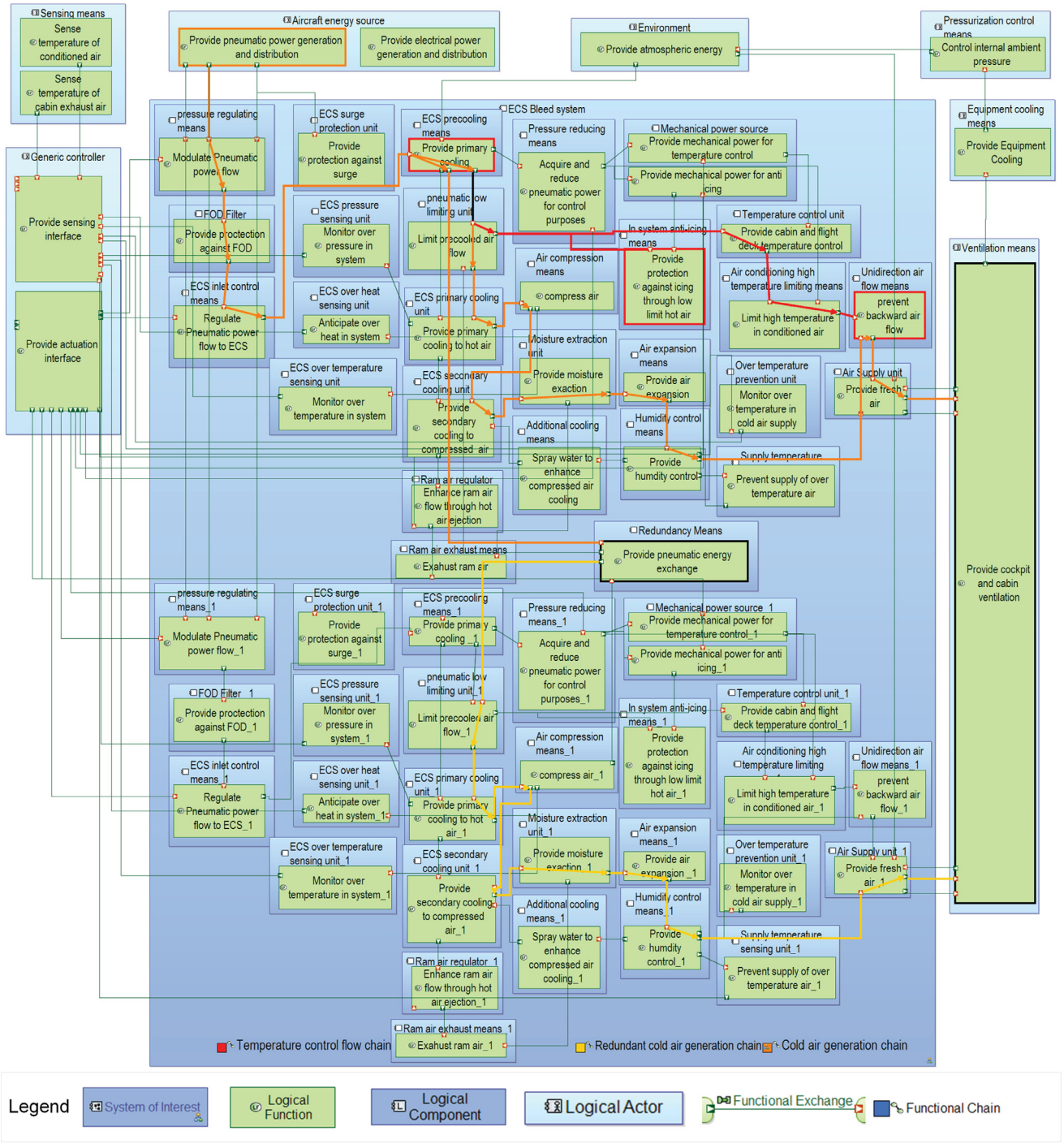

Figure B.1: Logical architecture for the bleed ECS

Appendix C

Physical architecture for the CPCS

The Figure C.1 shows a high-level PA of fully electric CPCS and also reflect the demonstration overview presented in 5.1. The actors of the CPCS are flight crew, flight deck, FMS, door, landing gear system and oxygen system. The communication switch provides integrated communication between the components. There is two avionics computing unit representing LRM. The unit one is dedicated for HMI and is placed near the flight deck. Also, unit one host the ECS HMI application and also contains the ECS HMI definition files. Unit two host the CPCS application. The remote data concentrator (RDC) forward the actuation command to the outflow valves and also receive pressure data from sensors. The CPCS has two sets of outflow valves with each having a primary and secondary valve.

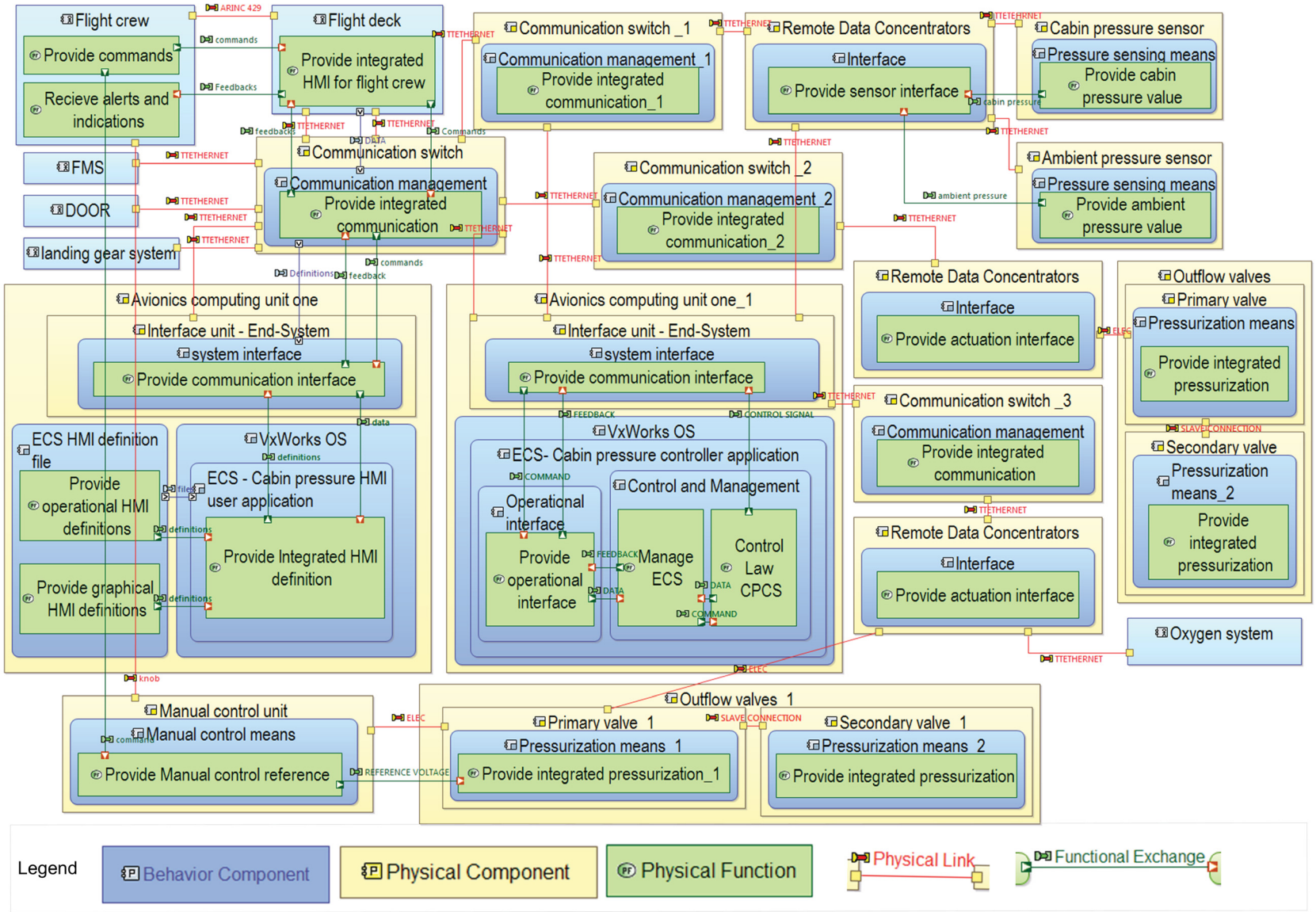

Figure C.1: Physical architecture for the CPCS

Appendix D

Data class for cabin pressure controller

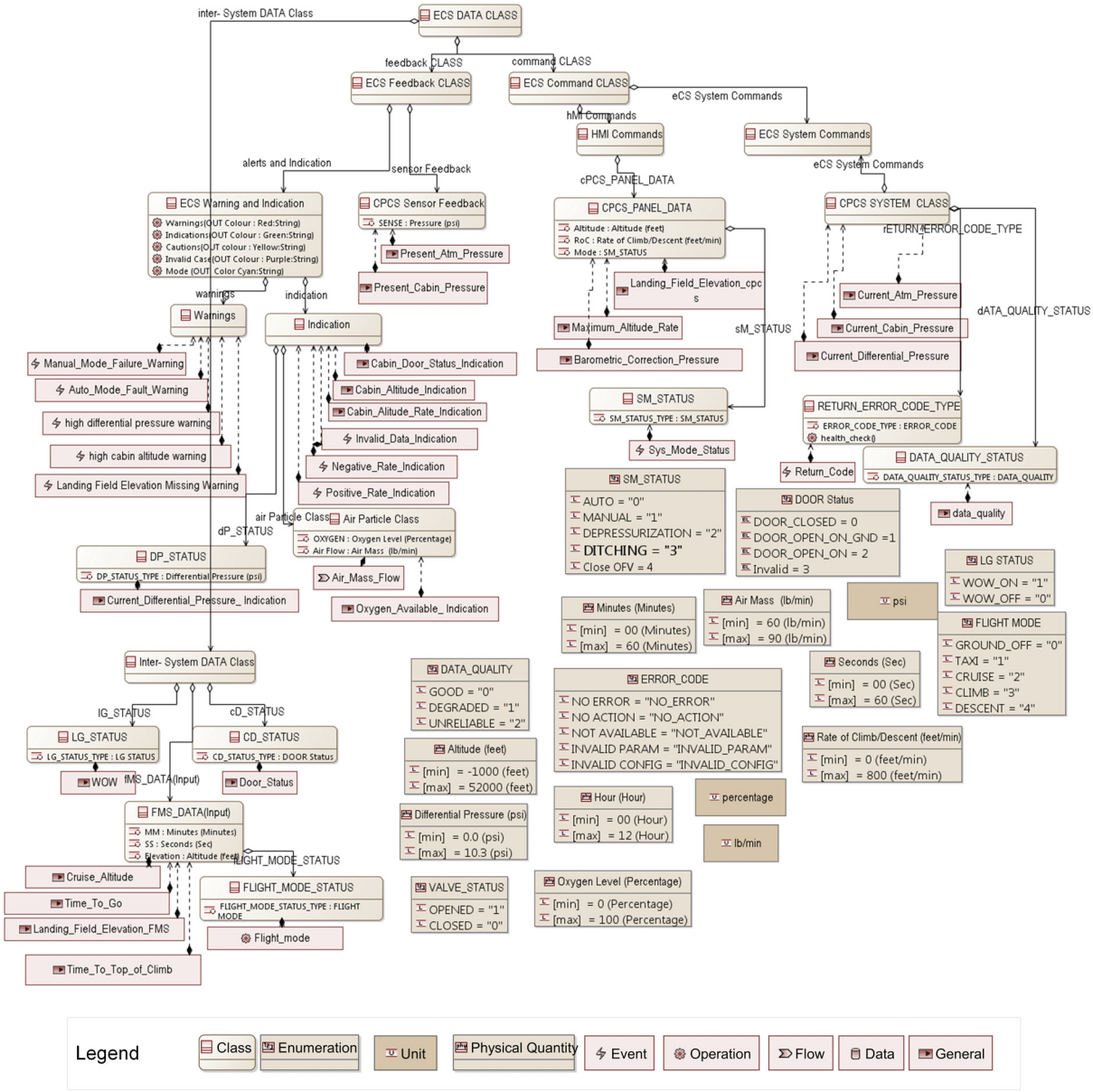

Figure D.1: Data class for cabin pressure controller

Appendix E

Example for logical scenario

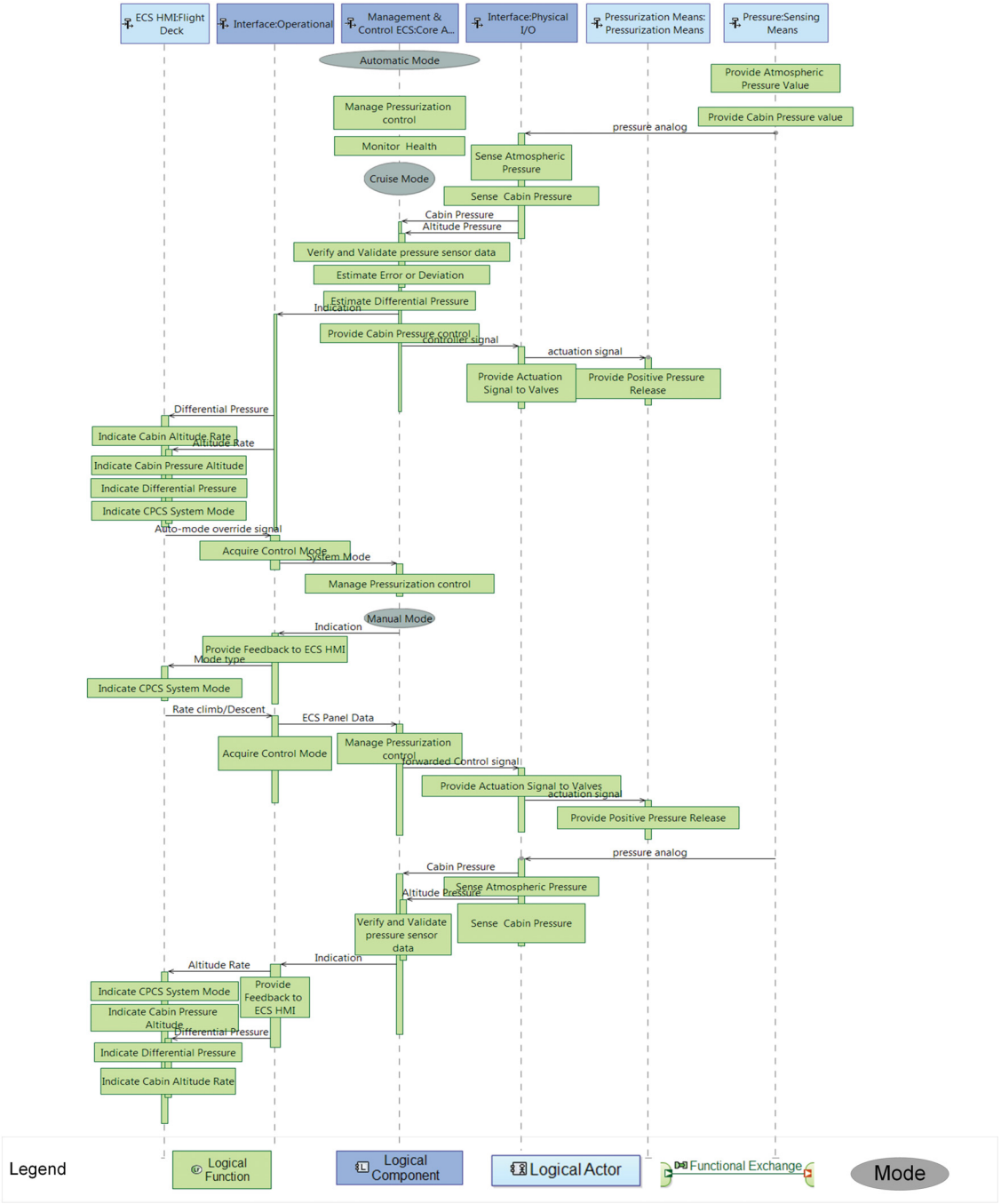

Figure E.1: Logical scenario for automatic-to-manual pressurization control for the electric system

Appendix F

Logical architecture for the bleedless ECS

The logical architecture for the Bleedless ECS in Figure F.2 is adapted from the Boeing patent for electric air conditioning system for an aircraft [88] presented in Figure F.1.

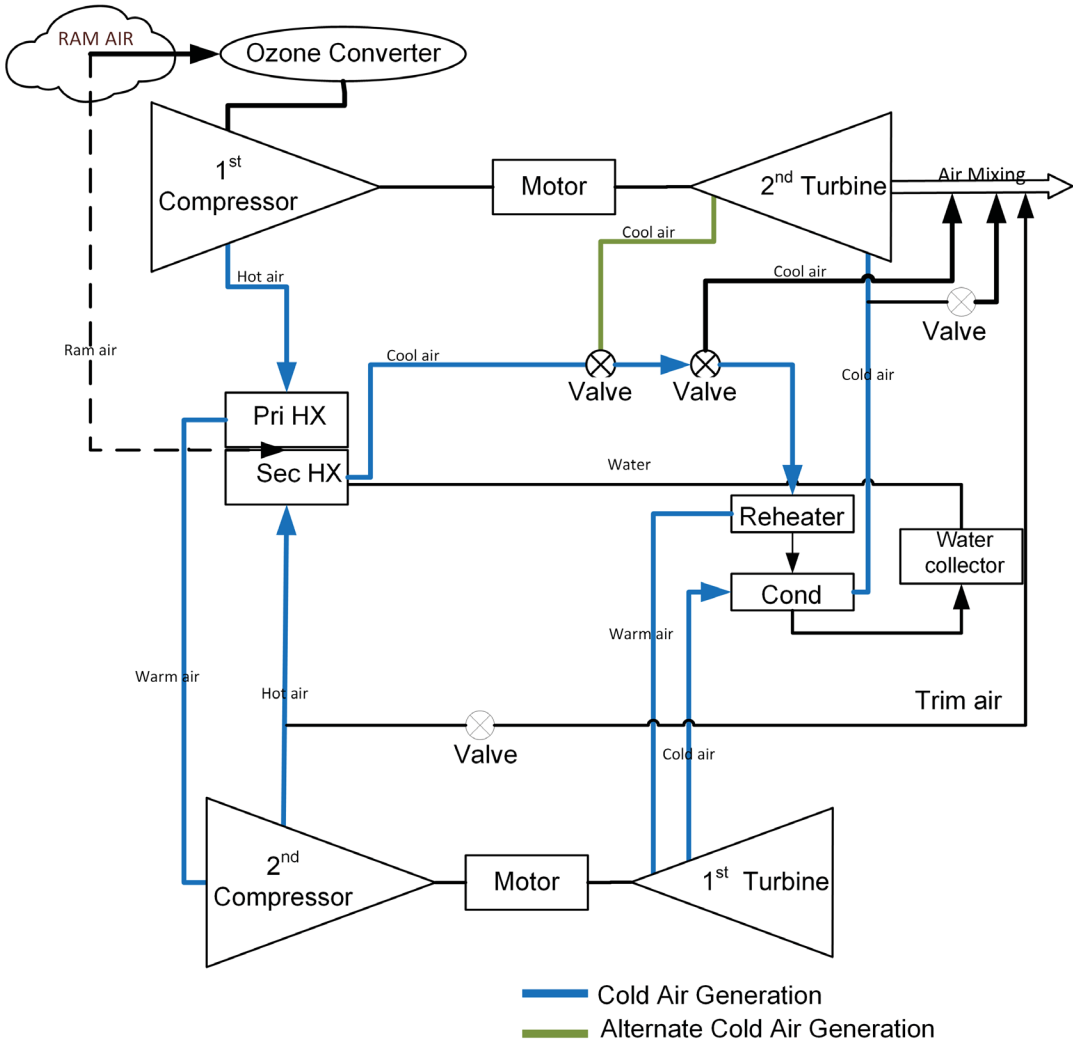

Figure F.1: Electric air conditioning system adapted from [88]

There are four functional chains defined in Figure F.2.

1. Cold Air Generation chain is represented by the blue line The chain highlight the manipulation of ram air to generate cold air. The functional flow is as follows:

   Generate pneu power ⇒ Provide cooling to compressed air ⇒ Compress warm air ⇒ Provide cooling to recompressed air ⇒ Reheat cooled air ⇒ Cool warm air through expansion (primary) ⇒ Condense water from cold air ⇒ Cool air through expansion (secondary) ⇒ Provide fresh air through mixing.

2. Alternate Cold Air Generation chain is represented by olive green in Figure F.2 & F.1. In this chain, the air from secondary heat exchanger (Sec HX) is feed to secondary turbine by bypassing the primary turbine.

3. Control Signal Flow chain is represented by pink colour. The chain highlights the valves that are directly controlled by the ECS controller

4. Electrical Flow chain is represented by yellow colour. the chain shows possible electrical supply to the required elements

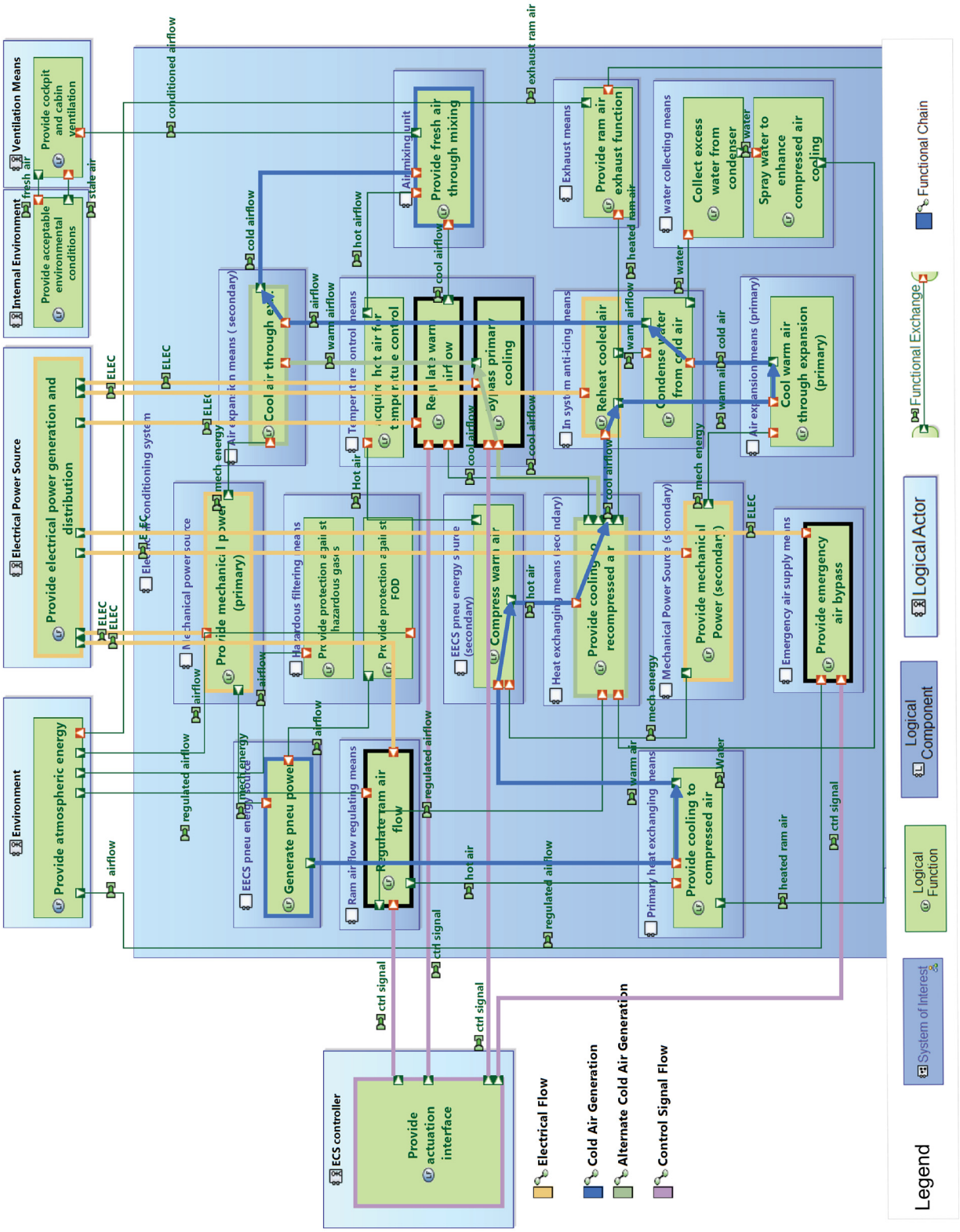

Figure F.2: Logical architecture for the bleedless ECS