Improving Model-Based System Architecture Specification to Enable Fault Tree Analysis

Enes Kolip

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

May 2024

Concordia University

School of Graduate Studies

This is to certify that the thesis prepared

By: Enes Kolip
Entitled: Improving Model-Based System Architecture Specification to Enable Fault Tree Analysis

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:

_____	Dr. Rajamohan Ganesan	Examiner & Chair
_____	Dr. Catharine Marsden	External Examiner
_____	Dr. Susan Liscouët-Hanke	Supervisor

Approved by _____
Dr. Muthukumaran Packirisamy, Chair
Department of Mechanical, Industrial and Aerospace Engineering

_____ 2024

_____
Dr. Mourad Debbabi, Dean
Gina Cody School of Engineering and Computer Science

Abstract

Improving Model-Based System Architecture Specification to Enable Fault Tree Analysis

Enes Kolip

The aviation industry explores innovative aircraft technologies and concepts aiming to reduce its emissions and meet environmental targets. The advanced technologies result in high complexity in aircraft systems, necessitating novel system architecting and safety assessment methods. Model-Based Systems Engineering (MBSE) presents a promising approach, offering more efficient systems development than document-centric methods. At the same time, safety assessment is an integral part of the system development process and can also benefit from a model-based approach. Model-Based Safety Assessment (MBSA) emerges to enable the analysis of the system architecture from a safety perspective and automate segments of the process, and by doing so, it improves efficiency by reducing development time and errors. The objectives of this thesis are to integrate MBSE and MBSA to construct a system model with an architecture specification that can help build safety models for fault tree analysis (FTA). This thesis focuses on the transition from system to safety models and explores various methods to enhance architecture specification in support of MBSA. The approach presented in this thesis utilizes the Capella workbench and extends the logical architecture levels of the Architecture Analysis and Design Integrated Approach (ARCADIA) to represent the system architecture at the appropriate level of granularity to support MBSA. The presented methodology involves enriching a system specification model by integrating safety properties into Capella elements with the help of the property values management tools (PVMT). The flap system is selected as a test case, and a system model of the flap control and actuation system is developed and used to construct safety models in the AltaRica 3.0 language. Specific failure scenarios are introduced by adding observers to the safety models, enabling FTA with AltaRica 3.0. The minimal cutsets and failure rates of the FTA are examined to validate the results of the safety analysis, ensuring the transition between the system and the safety model is correct. Overall, the presented thesis helps to improve coherence and collaboration between system and safety engineers designing complex systems, such as advanced flight control system architectures.

Acknowledgments

I am profoundly grateful to my family for their unconditional love and encouragement. To my mother, Sema, whose unwavering faith in me has been my greatest strength, and to the memory of my father, Hasan, whose guidance continues to inspire me. I dedicate this thesis to both of you as a token of my deepest gratitude and love.

I extend my heartfelt gratitude to my supervisor, Dr. Susan Liscouët-Hanke, whose invaluable guidance and support have been instrumental in shaping this thesis. I am deeply thankful for her patience and encouragement throughout this academic endeavor. I am also grateful to her for providing me with the opportunity to conduct this research project.

I am indebted to my admirable friends in the Aircraft Systems Lab, Hasti Jahanara, Mohammed Mir, and Noble Muyenzikazi, for their support. Their contributions have made this journey enjoyable. Special thanks to Andrew Jeyaraj for his constructive suggestions and intellectual input throughout the period of work.

I also sincerely thank Alvaro Tamayo and Vincent Saluzzi from Bombardier for their invaluable support and expertise. Their generous provision of time and practical insights have greatly enhanced this research.

Finally, I would like to express my heartfelt gratitude to everyone who has supported me throughout this journey. Your encouragement has been invaluable to me. Thank you for being a part of this milestone in my academic and personal growth.

Table of Contents

List of Figures

List of Tables

Nomenclature

AADL	Architecture Analysis and Design Language
AFHA	Aircraft Functional Hazard Analysis
AILE	Aile Intelligente et Légère pour l'Environnement
ARCADIA	Architecture Analysis and Design Integrated Approach
ARP	Aerospace Recommended Practice
ASA	Aircraft Safety Assessment
CCA	Common Cause Analysis
CU	Control Unit
DODAF	Department of Defense Architecture Framework (the USA)
EICAS	Engine Indication and Crew Alerting System
FCS	Flight Control System
FEM	Failure Effects Modeling
FHA	Functional Hazard Analysis
FLM	Failure Logic Modeling
FMEA	Failure Mode and Effects Analysis
FMECA	Failure Modes, Effects, and Criticality Analysis
FMES	Failure Modes and Effects Summary
FPM	Failure Propagation Modeling
FTA	Fault Tree Analysis
GTS	Guarded Transition Systems
HIP-HOPS	Hierarchically Performed Hazard Origin and Propagation Studies
INCOSE	International Council on Systems Engineering
ISAAC	Improvement of Safety Activities on Aeronautical Complex System
JPL	Jet Propulsion Laboratory
L0	Logical Architecture at Level 0
L1	Logical Architecture at Level 1
L2	Logical Architecture at Level 2
LA	Logical Architecture
LAB	Logical Architecture Breakdown
LABRI	Computer Science Laboratory of the University of Bordeaux
MARTE	Modeling and Analysis of Real-time and Embedded
MBD	Model-Based Design
MBSA	Model-Based Safety Assessment
MBSE	Model-Based Systems Engineering
MODAF	Ministry of Defence Architecture Framework (the UK)
NSERC	Natural Sciences and Engineering Research Council of Canada
OA	Operational Analysis
PA	Physical Architecture
PAB	Physical Architecture Breakdown
PASA	Preliminary Aircraft Safety Assessment
PDU	Power Distribution Unit
PSSA	Preliminary System Safety Assessment
PVMT	Property Value Management Tool
SA	System Analysis
SAB	System Analysis Breakdown

SAE	Society of Automotive Engineers
SEE	Single Event Effects
SFHA	System Functional Hazard Analysis
SMF-FTA	Safety Modelling Framework for Fault Tree Generation
SOI	System of Interest
SSA	System Safety Assessment
SysML	Systems Modeling Language
UML	Unified Modeling Language
UPDM	Unified Profile for DoDAF/MODAF
XML	Extensible Markup Language

1. Introduction

The global population has been seeing an increase in the accessibility of aviation with a growing demand for air traffic [1]. Thus, it results in concerns about the environmental impact of the industry [2]. CO_2 emissions from aviation have grown significantly faster in recent years than rail, road, and shipping industries, accounting for 2% in 2022 [3]. For this reason, the industry is attempting to reduce carbon emissions by half by 2050 [4]. Consequently, novel aircraft technologies that can help reduce carbon emissions and increase the efficiency of aircraft are being developed. These technologies, however, add significant complexity to aircraft systems, resulting in a need for advanced methods for design, management, system, and safety assessments [5], [6], [7].

1.1 Background and Motivation

An aircraft development, from the conception phase to the market, takes about 10 years on average [8]. The efforts to develop more efficient aircraft increase the complexity of aircraft systems and the development process, causing delays in the aircraft development of industry pioneers, Boeing [9] and Airbus [10]. Thus, the aerospace industry needs novel methodologies and approaches to reach environmental goals and bring more efficient solutions to aircraft systems.

SAE Aerospace Recommended Practice ARP4754 [11] is a guideline for developing civil aircraft and systems. It presents an aircraft development process following a systems engineering approach to cope with complexity. The process begins by designing and validating artifacts at three levels: aircraft, system, and item levels. Later in the development, the engineers focus on implementation, integration, and testing.

Also, ARP4761, a guideline for conducting the safety assessment process on civil aircraft, states that safety engineers perform safety assessments simultaneously to support the aircraft development process. Safety engineers work on safety analyses exclusively with the information available on the system model. These analyses, however, take place late in the system design process. System designs constantly evolve until they are finalized. Therefore, safety analyses cannot participate in important design choices [12]. There is often a gap between the system engineering and safety assessment because of the evolving nature of the system model. If safety analyses are done based on an outdated system model, it can result in cost increases and time waste.

A model-based approach has the potential to reduce the gap between systems engineering and safety assessment disciplines and reduce faults caused by this gap. Model-based methods can increase the capability of system engineering activities by improving traceability in system development, providing numerous system diagrams and perspectives, and facilitating organization [13].

Figure 1.1 shows the interaction between safety and development processes. Safety assessment supports aircraft development by introducing safety requirements at aircraft, system, and item levels. The safety assessment process consists of Preliminary Aircraft Safety Assessment (PASA), Common Cause Analysis (CCA), Preliminary System Safety Assessment (PSSA), System Safety Assessment (SSA), Aircraft Safety Assessment (ASA), and Functional Hazard Assessment (FHA).

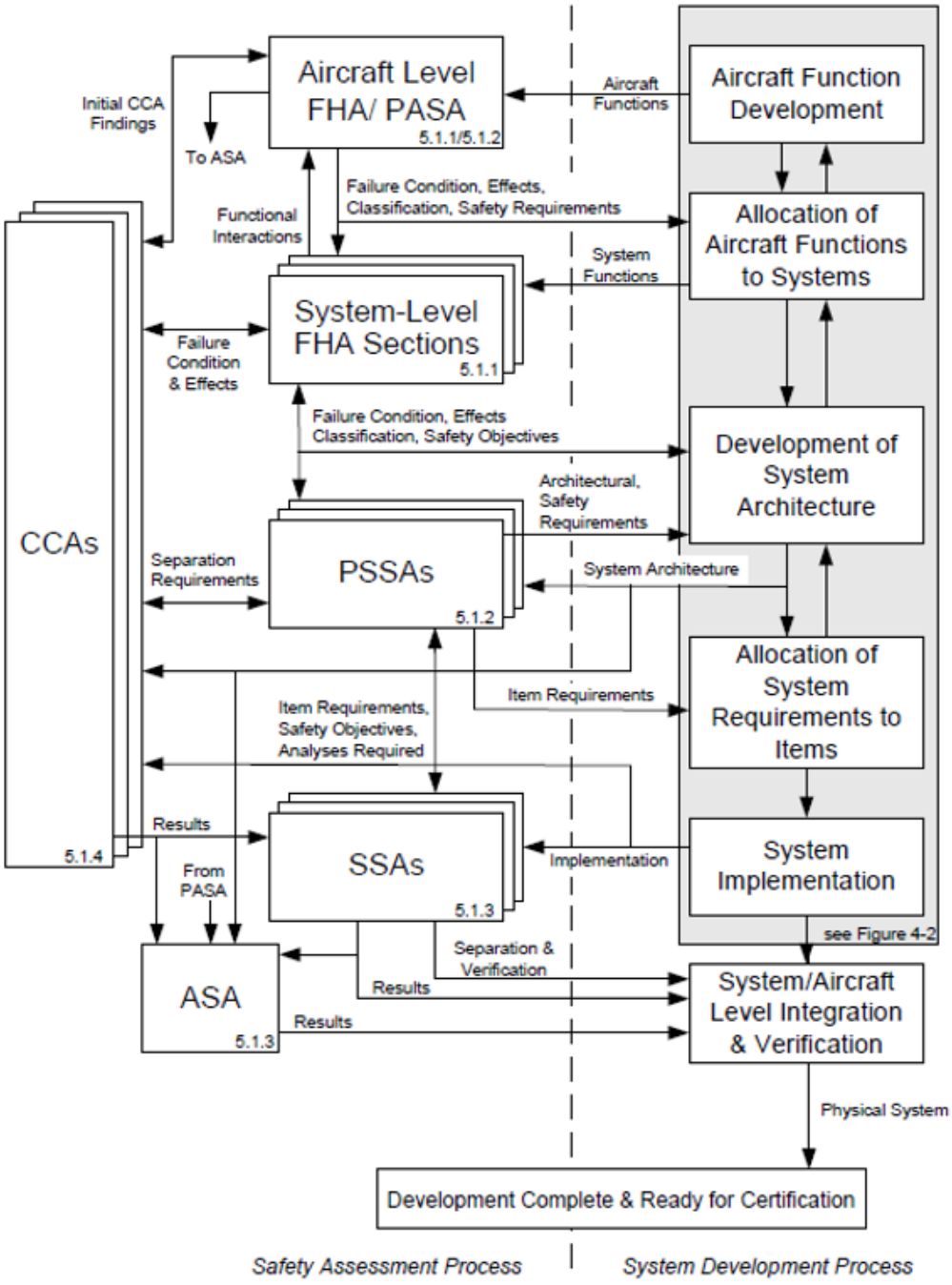

Figure 1.1 - Safety Assessment Process Model in [11]

At the aircraft level, FHA and PASA provide safety objectives based on the aircraft-level functions. After aircraft functions are allocated to different systems, they become system functions. System-level FHAs take system functions as inputs and provide system-level safety objectives for designing individual aircraft systems. Later in the process, component-level safety requirements are derived from these system-level safety objectives to support the design or selection of components that are part of the systems. All safety requirements support meeting the safety objectives set after system-level FHAs and PSSA, as shown in Figure 1.1. Validation activities are

occurring, starting at the system level, for the previous level to confirm that specifications are developed as intended. FHA identifies potential hazards associated with functions and defines the risks related to each hazard identified. After each FHA, an FTA takes place to validate the FHA results. FTA helps to understand the failure of systems and find ways to reduce risks identified by FHA.

ARP4754 and ARP4761 define system development and safety assessment as interdependent. In addition, Figure 1.1 depicts the close relationship between the processes of system development and safety assessment. ARP4761 describes a detailed safety assessment process occurring concurrently with the development of the aircraft and its systems. The safety assessment can be classified into two approaches: Qualitative and quantitative. The qualitative approach detects the dependencies between component failures and hazards on the system level. On the other hand, the quantitative method provides probabilities and rates of failure events. Traditional safety analysis that works with these two approaches consists of techniques such as FTA, FHA, Failure Modes, and Effect Analysis (FMEA). In addition, independent safety tools, languages, and methods such as the Unified Language Model (UML) and Matlab-Simulink are often used to conduct these traditional safety analyses [14]. Also, safety analyses are often performed manually, they are performed without the aid of automated tools or software, meaning that these analyses can depend on the skill of the safety analysts [15].

ARP4761 defines MBSA as an analysis method used to model system architecture to show system behavior if any failures occur. MBSA allows automation by introducing models to traditional safety assessment. For example, AltaRica, an MBSA language, can enable automated FTA [16]. Thus, MBSA, with the automatic safety analysis, improves efficiency by decreasing the time needed for development and errors [17].

1.2 Objectives and Scope of the Thesis

This thesis is conducted as part of the project ‘Aile Intelligente et Légère pour l'Environnement’ (AILE) funded by Bombardier and ‘Natural Sciences and Engineering Research Council of Canada’ (NSERC) (under the grant number CRDPJ542298-19) to research new methodologies that will contribute to bringing novel flight control system architectures to fruition in the next generation of aircraft in Canada by advancing the state-of-the-art of virtual design and virtual testing in three areas: (1) model-based systems engineering (MBSE), (2) model-based safety assessment (MBSA) and (3) model-based design (MBD). The following sections present the background and motivation for the conducted research and exhibit this thesis's objectives and scope.

The gap between MBSE and MBSA due to separate environments performing safety assessments can be reduced by incorporating safety analysis and artifacts into an MBSE approach and workflow. Traditionally, the system architecture specification model does not include the artifacts and findings of safety analysis. This thesis aims to enhance model-based system architecting by incorporating safety artifacts into the system model to perform automated model-based safety analyses. The generated system model with safety information can be used for other safety analyses and assessment activities with the evolving architecture specification and safety information embedded into it. For example, if a system model includes FHA results such as failure conditions, affected functions, and failure rate objectives, then the following FTAs can be

performed based on this system model. If the system model evolution is not only limited to system architecting but also safety assessment, embedding FTA results into the system model can enable performing FMEAs. Accommodation of an MBSE model with safety information can make the model a reference for performing safety analyses. In addition, employing MBSA frameworks to run these analyses would maximize the power of safety tools, enabling the automatic generation of safety artifacts.

This thesis presents an approach for MBSE to accommodate model-based FTA. It focuses on enriching the system architecture with safety artifacts and performing automated FTAs with standalone safety models. This thesis applies the methodology presented on a flap system test case as the industrial partner Bombardier addresses advancements in flight control system architectures. The flap system test case involves components with high technology levels that reflect the aerospace industry's complexity. However, the methodology applies to other systems as well.

The proposed methodology involves different phases of development: (1) the architecture specification and then a system model representation of the architecture. (2) with a safety model that can generate automated FTAs. The research objectives are the following:

- Develop methods to use the MBSE specification model to perform safety assessments. The thesis specifically focuses on the transition between the Capella, an MBSE tool, system model and the AltaRica safety model, aiming to maximize the power of safety tools by utilizing standalone safety models.
- Exploring architecture representations and modeling artifacts in the ARCADIA/Capella MBSE framework to support the transition between MBSE and MBSA disciplines.
- Exploring modeling methodologies that can capture the aircraft development process in ARP4754 while accommodating safety analyses in ARP4761.

1.3 Organization of the Thesis

This thesis is structured as follows: Chapter 2 presents the state of the art in MBSE, MBSA, and their tools. It also introduces the ARCADIA approach and different MBSA techniques in the literature. Chapter 3 shows a modeling approach extending the ARCADIA/Capella logical level from one to three. It also involves the enrichment of the Capella system model and the integration of FTA. Chapter 4 provides the application of the methodology representing a flap system test case. The last chapter, Chapter 5, concludes the thesis by summarizing the main points and outlining potential future works.

2. State of the Art

This section introduces Model-Based Systems Engineering (MBSE) and Model-Based Safety Assessment (MBSA). It covers the previous studies on bridging MBSE to MBSA and presents a gap analysis.

2.1 Model-Based Systems Engineering

The traditional approach, document-based systems engineering, establishes the development of a system by managing the documentation of requirements, design, analysis, verification, and validation activities [18]. On the other hand, MBSE shifts the focus to developing models of the system of interest (SoI) [19]. Since the introduction of modeling methods to systems engineering, utilizing MBSE for complex systems engineering has gathered attention from industries building complex systems because it provides models that can store information on the system's functions, requirements, architecture, and behavior [20], [21]. Also, The International Council on Systems Engineering (INCOSE) decided that MBSE is a central element in the primary vision for 2025 [22].

Model-based approaches to systems engineering ensure better communication by providing all stakeholders with a systematic examination of the system model [23]. Design teams communicate using the same modeling language to develop a system model. Thus, productivity increases with the quality while the risk associated with the development is reduced [24], [25], [26]. Since the modeling languages follow the traditional document-centric process, they support validation and verification efforts throughout the lifecycle of a project [21]. Also, MBSE helps the development process standardized by storing the model and its elements in a common model repository [27].

Since MBSE provides a systematic modeling approach, systems with advanced technologies and high complexity are the main targets of MBSE. Adopting MBSE is becoming more common due to its help in the ability to understand problems related to the design process, hence increasing the efficiency of the development process [28]. The MBSE approach is applied by many companies developing complex systems, such as NASA on the Europa Project [29], The Jet Propulsion Laboratory (JPL) [30], and Boeing [31]. In addition, surveys in [32] and [33] indicate that the usage of MBSE has increased both in government operations and industry. Consequently, numerous disciplines and domains are adopting MBSE to develop highly complex systems.

Moving to applications of MBSE on the aircraft systems architecture, which is aligned with the scope of the study, a study by Liscouët-Hanke and Jeyaraj uses MBSE to represent system architectures in conceptual design [34]. An application of MBSE to a test case study, flight control systems, similar to the case presented in this thesis, is shown by Jeyaraj [35]. Likewise, Liscouët-Hanke et al. developed an MBSE approach for test rig architectures of flight control systems [36]. Mathew et al. [37] and Tabesh [38] presented different MBSE approaches for developing and supporting the system architecture on integrated modular avionics and early aircraft design stages, respectively.

The study presented in this thesis continues the work done by Tabesh [38] by extending the scope to FTA from FHA. Tabesh proposes an alternative method that adopts a different approach from this thesis to capture failure relationships within the system model. Tabesh utilizes the FHA outcomes by identifying the failure conditions of a function and specifying the impacted functions in the properties created by a Capella add-on.

Overall, the use of MBSE is increasing to address the challenges faced in system architecting for complex systems due to its capabilities of storing information and providing different viewpoints for stakeholders [20], [21]. Compared to the traditional document-centric systems engineering approach, MBSE offers traceability throughout the development process, easy-to-manage requirements, and architectures [39]. The system models created with MBSE frameworks provide a single source of truth for all engineers and, as a result, achieve an increased coherency between different disciplines [40].

2.1.1 MBSE Methods and Languages

Different MBSE approaches aim to solve the problems that document-based systems engineering brings. They can be methods, tools, or processes to support systems engineering discipline [12]. There are frameworks designed to address specific problems for particular systems, such as Modeling and Analysis of Real-time and Embedded (MARTE) for software and hardware systems or Unified Profile for the USA Department of Defense Architecture Framework and the UK Ministry of Defence Architecture Framework DoDAF/MODAF (UPDM) [41], [42]. There are also methodologies for mapping specifications between domains that help make models compatible in different environments [43].

Numerous modeling frameworks and languages exist to build and share models. MBSE utilizes two main model types: descriptive and analytical models [44]. While descriptive models describe a system's logical relationships, interfaces, and functions, analytical models explain mathematical relationships in a system. On the other hand, a system model that provides a cohesive system representation can be a hybrid model of descriptive and analytical models [44]. Another method to implement MBSE is the modeling tools and languages to create the mentioned models.

Modeling languages and methods are considered to be enablers of MBSE [25]. The most common MBSE enablers are UML, System Modelling Language (SysML), and ARCADIA/Capella [45], [18]. UML visualizes complex software structures by designing and documenting their elements with diagrams. It is an object-oriented language focusing primarily on software development, but its models can address the complexity of the systems for systems engineering. Therefore, SysML was created by extending UML 2 for system engineering applications [25], [46]. Although these languages aim to develop complex systems, they can be structured differently. ARCADIA/Capella supports functional analysis by implementing requirements for functions and functional flows [47], while SysML and UML are structured with activity diagrams with no functional hierarchy [48].

The methods, tools, and processes must be collaboratively used to enable MBSE [49]. Therefore, a great endeavor is required to implement MBSE, especially in the aerospace industry, where novel technologies bring high complexity to systems. Consequently, ARP4754, a system engineering recommended practice by the aerospace industry, addresses the complexity of aircraft systems [11]. Engineers should select which MBSE environment to use for a project, considering the application of the principles outlined in ARP4754 and the complexity of the system they are working on. This thesis chooses Thales's ARCADIA approach and the associated Capella tool to implement the MBSE framework [47]. Therefore, the next section is centered around ARCADIA/Capella.

2.1.2 ARCADIA/Capella

Functional analysis and safety assessment are essential, as ARP4754 suggests, for developing complex systems. ARCADIA/Capella helps integrate requirements for functions and functional exchanges to enable functional analysis, unlike other methods [48], [50]. In addition, creating a functional structure by defining functions and exchanges is the first activity in ARCADIA/Capella, followed by allocating them to structural components [51]. On the other hand, in SysML, structural blocks are utilized to show functions, resulting in no significant difference between the functions and structural components, unlike ARCADIA/Capella.

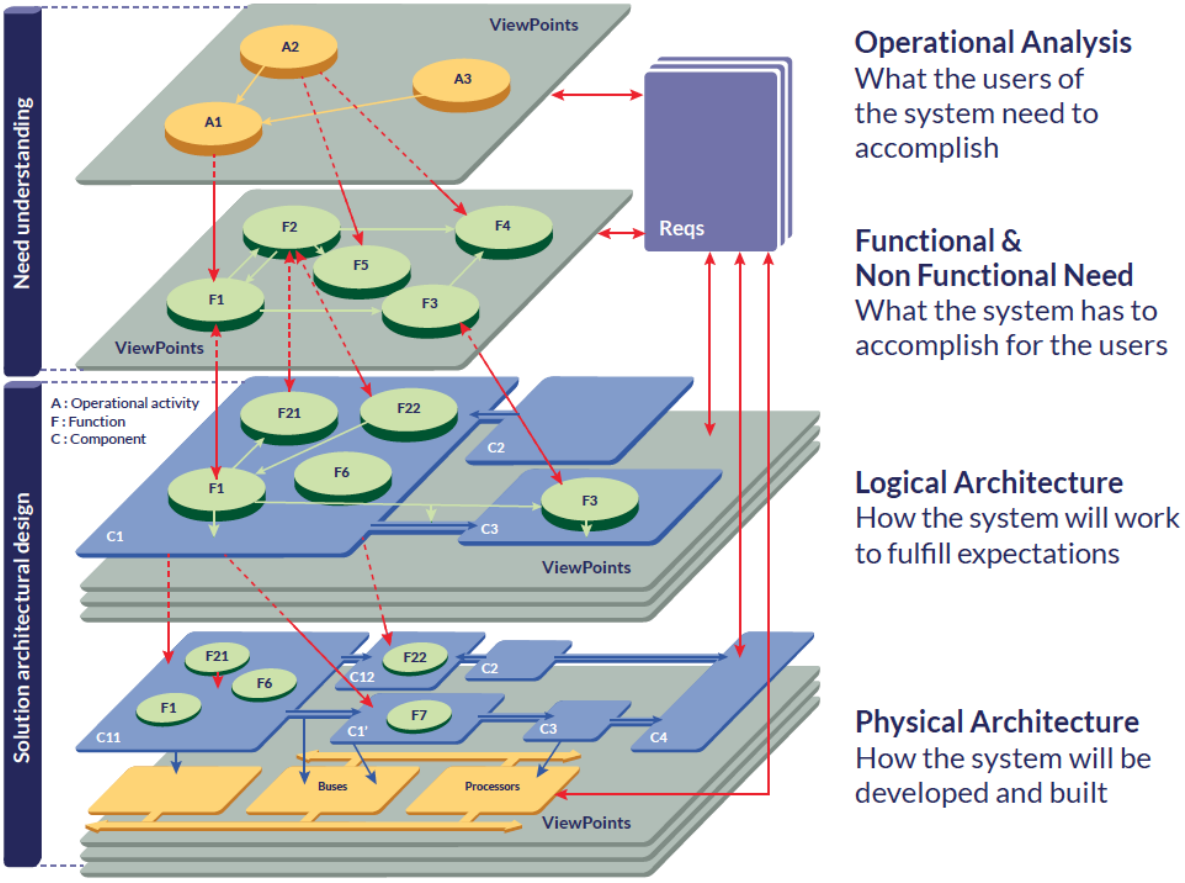

Figure 2.1 - ARCADIA Viewpoint-Driven Approach from [52]

Figure 2.1 shows that the ARCADIA/Capella has four different architectural levels. System specification is created by following the process along with allocating the requirements to system elements. Chapter 3 – Methodology section of the thesis presents a more elaborate explanation of these levels. A brief description of each level is as follows [53]:

- Operational analysis (OA) is the level where outer entities (named as actors in Capella) that have interactions with the SoI are defined.
- System analysis (SA) defines what the system has to accomplish for the outer entities, such as users and actors.

- Logical architecture (LA) outlines how the system works to achieve the expected performance.
- Physical architecture (PA) defines how the system develops and should be built.

By accommodating the four-level ARCADIA approach, the customer needs are kept consistent with the system specifications since the requirements are shared at each level with specific systems and components.

2.2 Model-based Safety Assessment

Like MBSE, MBSA has seen interest gradually increase from industry and academia [54], [55], [56]. The notable languages developed by academia for MBSA are AltaRica [57], [58], [59], Hierarchically Performed Hazard Origin and Propagation Studies (HiP-HOPS) [60], and the Architecture Analysis and Design Language (AADL) [61], [62]. T. Prosvirnova et al. present AltaRica 3.0, the newest version of AltaRica, to express traditional risk modeling methods such as fault trees and Markov chains [16]. Improvement of Safety Activities on Aeronautical Complex System (ISAAC) utilized MBSA tools such as AltaRica to support safety assessments [63]. In addition, the most updated revision of ARP4761, 2023, presents an appendix for MBSA, where an overview of the concepts and processes associated with performing a safety analysis using MBSA is presented.

MBSA is an approach that depicts a system's architecture and functional design. Its primary goal is to define how the system behaves in case of failures [64]. Unlike traditional safety analysis methods, the MBSA method is based on a common model (an extension of a system model or two models — a system model and a safety model transformed from it — [65]) in which system development and safety assessment efforts transpire simultaneously. This is due to the disadvantages of traditional safety analysis approaches in the following:

First, safety engineers use a system model to obtain safety information. Due to the evolving nature of the system model, the information exported from the model can become obsolete in time [66]. Second, if the system design is complete before the safety assessments, there is no further timeframe to change the design with the results of the analyses [67]. These problems can lead to late design changes with sharp cost increases [68]. Also, the most common safety analysis methods, such as FTA and FMEA, depend on the experience of the safety engineers and are thus prone to error and time-consuming [69]. Accommodating a common model, therefore utilizing MBSA, enabling system and safety engineers to work together on a system model with safety information, tackles these drawbacks of the traditional safety methods.

There are two classifications of MBSA methods. One depends on the system model and the other on the component interactions [65]. Standalone and extended models differ in system model [56],[65]. Previously mentioned common models that capture safety and systems engineering activities are used only to extract system structure or integrate safety artifacts into the system model. Therefore, standalone models require a transformation of the system model to be used for the MBSA tools. Thus, utilizing already developed and improved safety languages, methods, and tools after the transition takes place is one of the key advantages of the standalone models.

Several researchers have studied standalone models. For instance, Yakymets et al. [70] created the Safety Modelling Framework for Fault Tree Generation (SMF-FTA). The framework transforms the SysML system model into AltaRica language and then integrates safety information back into SysML. It provides a verification algorithm to detect any errors in the transformation. After the

transformation, AltaRica's tools can be used for safety analysis. Another study [71] developed an algorithm for extensible markup language (XML) to analyze the system model and transform the model into the AltaRica language. Their method creates preliminary FMEA reports automatically with the help of AltaRica.

Extended models, on the other hand, aim to expand the MBSE environment to accommodate safety analysis without the need to create a separate model. MBSE tools can enable safety assessment within themselves via tool extensions for safety by adding safety artifacts into the system model. An approach named SafeSysE is presented in [72]. The method introduces new attributes to store several safety artifacts. Then, the system model is exported as XML metadata interchange to a specific Python tool built for reading this data. Lastly, the tool runs FTAs and FMEAs. Also, Helle extends the MBSE environment, SysML [73]. The study illustrates failure cases as SysML Use Cases. It uses IBM Rhapsody, allowing the use of Rhapsody API to construct a program to extract the failure case data. Then, the program can calculate failure rates for different failure cases.

The other classification of MBSA methods depends on component interaction construction. Failure logic modeling (FLM), failure effects modeling (FEM), and fault injection modeling (FIM) are the main examples that fall under this classification. FLM models are written in a traditional FTA approach where the exchanges between components are defined with failure modes only. In FLM, if a component has a failure occurring, it generates a specific failure mode for another component that has interactions with it. On the other hand, the FEM approach involves the construction of a simplified model of the system where the component exchanges are captured in terms of flow characteristics (energy and/or information) with boolean values. While standalone models are constructed with FLM and FEM (e.g., AltaRica [16] and HipHops[74]), FIM supports the extended models. FIM helps safety analysts determine failures that result in safety requirements not being met [54]. FIM achieves this by configuring the elements of the extended models for specific failures so that the components present erroneous exchanges at the concerned areas or interfaces in the extended models. FIM has been utilized in several safety assessment studies by Bozzano et al. [75]. This modeling method provides a coherence advantage with the system model compared to other techniques. However, it relies on the skills of the safety engineer to build extended models. Also, FIM has limitations on the capability of handling time-dependent faults that are seen in many aircraft component failures.

While extended models allow system architecting and safety analysis in one unified environment, MBSE tool add-ons for safety analysis must be built. On the contrary, standalone models require effort to be transformed from system models, but they use existing safety analysis tools.

2.3 Safety Modelling Tools, Languages and Methods

This section presents several MBSA modeling tools, languages, and methods available developed by academia and industry.

Safety Architect is an MBSA tool developed by ALL4TEC to assess system architectures in various industries [76]. It can use functional or physical system architectures built in SysML or Capella to run local Failure Modes, Effects, and Criticality Analysis (FMECA). For the failure cases identified, it automatically runs FTA [77]. First, the user must import a system model into the environment to perform a local analysis. The local analysis allows users to link the failure modes of different blocks of the model with their inputs and outputs. Along with the local analysis, users are expected to add safety barriers that prevent feared events from occurring. After these

steps, the failure modes for selected feared events must be specified. Finally, the user requests the tool to run an overall analysis. Safety Architect then runs an automatic analysis that spreads the failure modes to the system and traces their combinations, resulting in the feared event. This results in an enriched system model consisting of assumptions made in the local analysis, a summary of the results attained in the overall analysis, and a fault-tree visualization [78].

AltaRica 3.0 is a constraint automata language designed at the Computer Science Laboratory of the University of Bordeaux (LaBRI) and used for safety analysis [79], [80]. It is an event-driven complex systems modeling based on Guarded Transition Systems (GTS). AltaRica models consist of component hierarchies that capture GTS. The main principle of AltaRica is a set of rules that translate (flattening in AltaRica) component hierarchies, called boxes, to a GTS [81]. The tool consists of component notions called nodes. Each node is defined with several flow and state variables and events. While events trigger transitions between different states on the automata modeled by the state variables, flow variables are interfaces of the nodes, defining inputs and outputs [65], [82]. More information on Altarica and its main principles, with an overview of GTS, is presented in section 3.2.

The Architecture Analysis and Design Language (AADL) is a language extended from MetaH developed by Binns et al. [83], [84]. SAE standardized AADL as an architectural description language for analyzing embedded software [85]. AADL is tailored to support architectures and software patterns for distributed processor platforms with hybrid automata [86]. It consists of specific structures for designing embedded software. Its syntax includes components such as software subprograms, hardware processors, memory, bus, etc [85].

HiP-HOPS is a method that enables integrated assessment of complex systems by analyzing the failure behavior of components through utilizing interface-focused-FMEA (IF-FMEA), a modified version of classical FMEA [87]. Applying this method results in a table providing a list of failure modes for components. The failure modes can be observed from the outputs of the components. The approach then captures the causes of output failures as combinations of internal failures, component malfunctions, or input deviations. The final stage of the analysis is determining the structure of the fault propagation process in the system. This is done by examining the functional failures identified in functional failure analysis and their combinations arising from component failure modes identified in the IF-FMEAs. Eventually, analyzing the expressions in the IF-FMEAs generates an FTA [87], [88].

2.4 Summary and Gap Analysis

According to the literature, MBSE and MBSA are promising solutions to tackle the problems that complex systems bring and to increase the efficiency of development and safety assessment processes. However, there are several issues exist due to the separation of environments in which systems engineers and safety engineers operate. Systems engineers typically work in an MBSE environment, such as Capella, while safety engineers work in an MBSA environment, such as AltaRica. This separation results in several challenges: there can be delays between analyses, leading to the system model being outdated by the time the safety analysis is performed. Additionally, errors can occur during the transmission of data between these environments, potentially compromising the integrity of the analyses. Also, the lack of synchronization between the system models and safety models can create inconsistencies that undermine the effectiveness of both MBSE and MBSA.

Several researchers have addressed these challenges by using either so-called *extended* or *standalone* models for MBSA. Extended models have the benefit of accommodating system design and safety analysis in the same environment. However, standalone model approaches use the capabilities of the existing tools to the maximum since system architecting and safety assessment are performed in their own environments.

A third way is integrating safety artifacts into the MBSE environment that can enable MBSA and reduce the above-mentioned challenges. Few researchers have investigated this option; this thesis will explore this avenue.

Regarding the scope of MBSA, most of the research focuses on the FHA, fewer work on the various subsequent stages. As previous work in the Aircraft Systems Lab addressed the FHA [38], this thesis will build on this prior work and focus on the FTA.

In summary, the analysis presented in this section identifies several gaps in the current methodologies for system architecture specification and safety analysis integration:

- Lack of Integration of Safety Properties: The current methodologies lack the capability to perform safety analyses based on the information stored in an MBSE environment. This gap highlights the need for developing methods to incorporate safety artifacts directly into the MBSE framework, enabling the conduction of safety assessments.
- Modeling Methodologies for ARP4754 and ARP4761: Existing modeling methodologies do not fully capture the aircraft development processes outlined in ARP4754 while also accommodating the safety analyses specified in ARP4761. This represents a gap in creating comprehensive models that address both system architecture and safety requirements effectively.
- The transition between Capella and AltaRica: There is insufficient research on the transition between the Capella MBSE tool and the AltaRica safety modeling tool. This gap indicates a need to explore methodologies for the transition between these tools.

To address these gaps, the primary research question guiding this thesis is: How can MBSE be enhanced to support FTA and enable MBSA in complex system architectures?

The objectives of this research are:

- To develop a system model using ARCADIA/Capella to enable hosting various safety properties.
- To integrate safety properties into system models using Capella's PVMT add-on.
- To create standalone safety models in AltaRica 3.0 with the information stored in the Capella system model for performing FTAs.
- Validate the methodology’s effectiveness through a flap system case study.

3. Methodology

This chapter introduces the methodology that aims to enable MBSA by introducing several enhancements to the MBSE environment. The methodology is based on an iterative process. This chapter presents a methodology overview by illustrating a mapping between safety assessment, aircraft development process, and system architecting in Capella, followed by an introduction to safety analysis with AltaRica. The next topics discussed in this chapter are system model development with Capella and safety artifact integration. Finally, the chapter ends with a summary of the methodology.

3.1 Methodology Overview

This section outlines the iterative approach utilized in this research to incorporate safety analysis into the system development lifecycle. Each iteration plays a role in improving and enhancing the system model. The iterative steps aim to reflect the connection between the system and the safety model. While developing the aircraft, these two models continuously provide input to each other to finalize system design and reach the safety objectives. The system model provides system architecture and safety properties to the safety model; the safety model outputs drive the system design decisions in a safety context.

The process depicted in Figure 3.1 starts with creating a test case architecture. Next, a Capella system model is developed based on the test case architecture, marking the first stage of development. However, the initial Capella model is representative of a typical system architecture specification model, not particularly tailored for safety assessment.

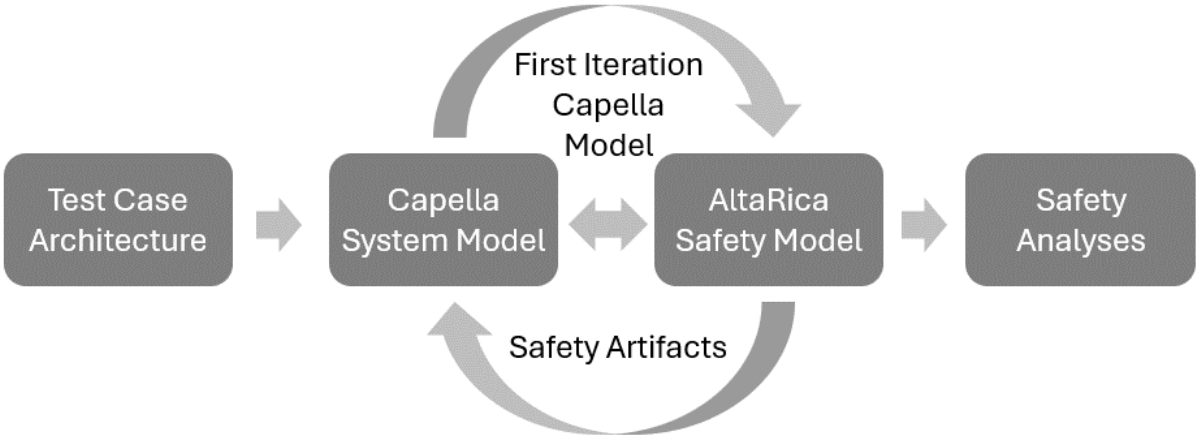

Figure 3.1 - Process to Obtain the Methodology

In the first iteration, an AltaRica safety model is developed alongside the Capella system model but with limited information, leading to qualitative and quantitative safety analyses that present failure rates with orders of magnitude only. The system and safety models are built, taking the test case architecture as a baseline. Hence, the initial process involves identifying safety elements for integration into the Capella system model, laying the groundwork for future safety integration efforts.

The next iteration has a more detailed system model following the development process depicted in Figure 3.2. Understanding the mapping between safety assessment, development, and ARCADIA/Capella process is vital for the next steps of the methods, especially additional logical architecture levels created in Capella.

In the first stage of the development process, which is aircraft function and requirement development, the functions are developed with the requirements as well as the experience and knowledge of the engineers on the system developed. The created functions define the capabilities of the aircraft. Here, the first iteration of AFHA and aircraft FTAs takes place.

In the next step, numbered 4.3 in Figure 3.2, the *Development of Aircraft Architecture and Allocation of Aircraft Functions to Systems*, system engineers break down these functions to obtain parent and subfunctions and define the functional exchanges. This step involves defining actors that are entities interacting with the SoI and allocating the functions to every system and actor. Breaking systems into subsystems corresponding to the logical architecture level in Capella is one of the main activities done here.

The *Development of System Requirements* stage is the step in which system engineers shape the architecture of the systems with the matured functions and their allocations. The SFHA is being done at this level with the system functions provided by the architecture. Also, the interactions and interfaces between different systems are defined, making early validation and verification possible.

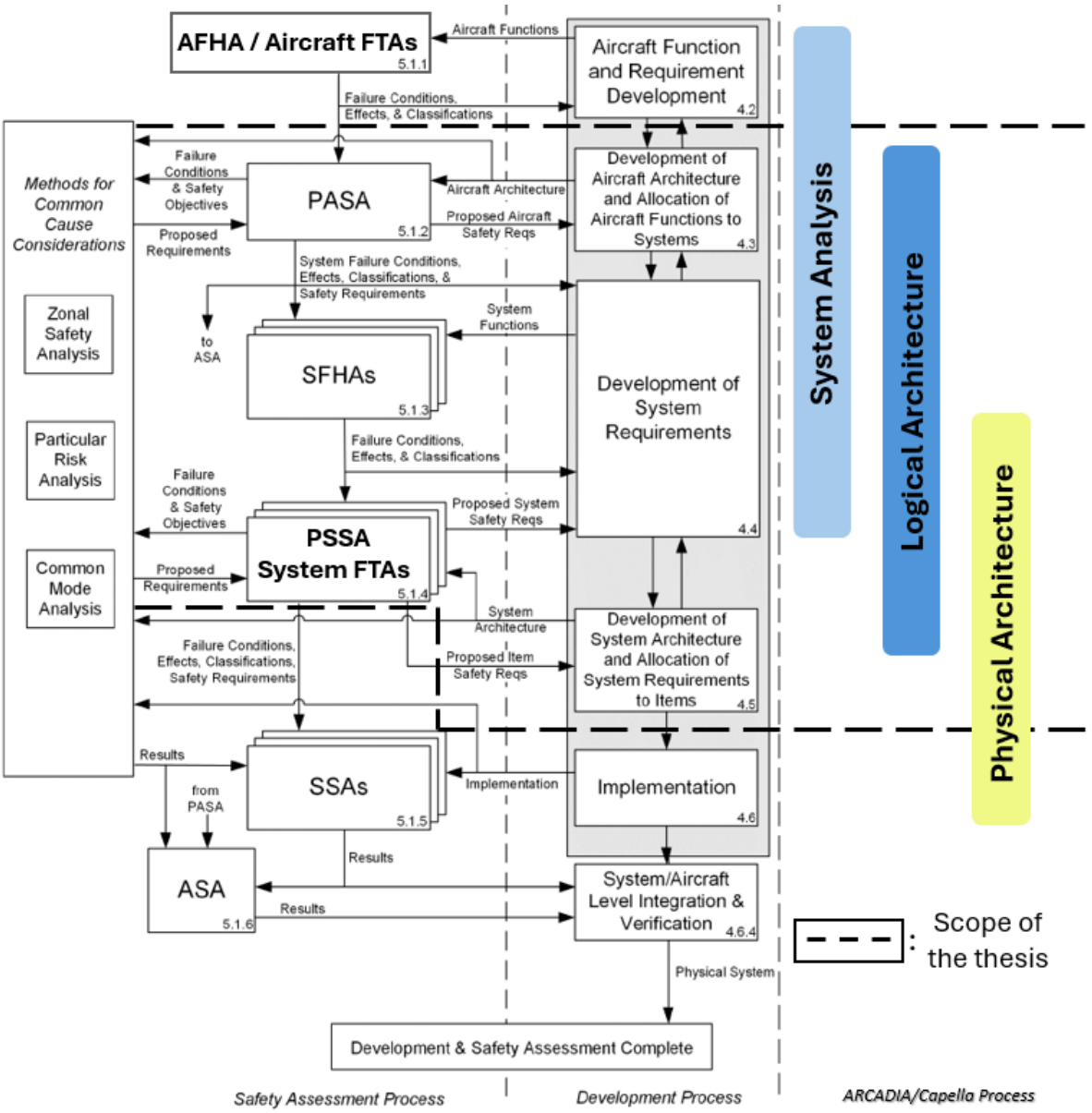

Figure 3.2 - ARP Safety Assessment Process Model Mapping to ARCADIA adapted from [11]

The *Development of System Architecture and Allocation of System Requirements to Items* is the stage where system engineers finalize the architecture by capturing and allocating requirements to the systems, components, and functions. The allocated requirements cover various topics, such as safety and performance. Safety engineers run system-level FTAs at this level with required inputs obtained from the matured system architecture. Depending on the results of the FTAs, the process can be iterative until the safety and reliability requirements are met.

The last stages of the development process do not fall under the scope of this thesis because the main activities done here are implementation and integration. It is the stage where a transition occurs from a higher (system) to a lower (item and component) engineering level. Safety engineers perform FTAs with real test data in the SSA for verification.

To accommodate system FTAs in the corresponding stage of the development process presented in Figure 3.2, the logical architecture in the Capella/ARCADIA approach is organized into three levels - L0, L1, and L2 - representing various stages of aircraft development, adapted from [38]. The three different logical levels are discussed in section 3.3.

3.2 Introduction to Safety Analysis with AltaRica

AltaRica is a modeling language designed for safety, reliability, and performance analysis. It is a tool for Model-Based Safety Analysis. It allows engineers to represent complex, modular, and dynamic systems, capturing the interface of components, events, and states. AltaRica is a safety tool that enables failure propagation modeling (FPM) [89].

The FPM depicts the system architecture with the dysfunctional behavior of the systems and components. The safety model created with FPM should illustrate both the design of the system and its failure characteristics from a safety perspective, as well as consider factors such as design maturity and assumptions about failure independence [64].

The foundation of AltaRica's modeling approach is the concept of Guarded Transition Systems (GTS) [64]. GTS provides a formal framework for representing system behaviors, transitions, and states. It captures a system's operation by outlining various transitions, each distinguished by particular conditions, actions, and events [90]. Thus, GTS enables the modeling of systems' dysfunctional behavior.

The representation of a repairable component that can take two different states in a guarded transition system can be indicated in Figure 3.3. Assume that it can be either *WORKING* or *FAILED*. In the initial state, the component works, meaning it prints *output* as true. When an event *failure* occurs, the state of the component changes to *FAILED*, and the output of the component becomes *false*. Similarly, the event *repair* makes the state variable transition from *FAILED* to *WORKING*. The events of *failure* and *repair* are executable with specific rates: λ and μ. Components in this thesis, however, have only one event, which is 'failure'.

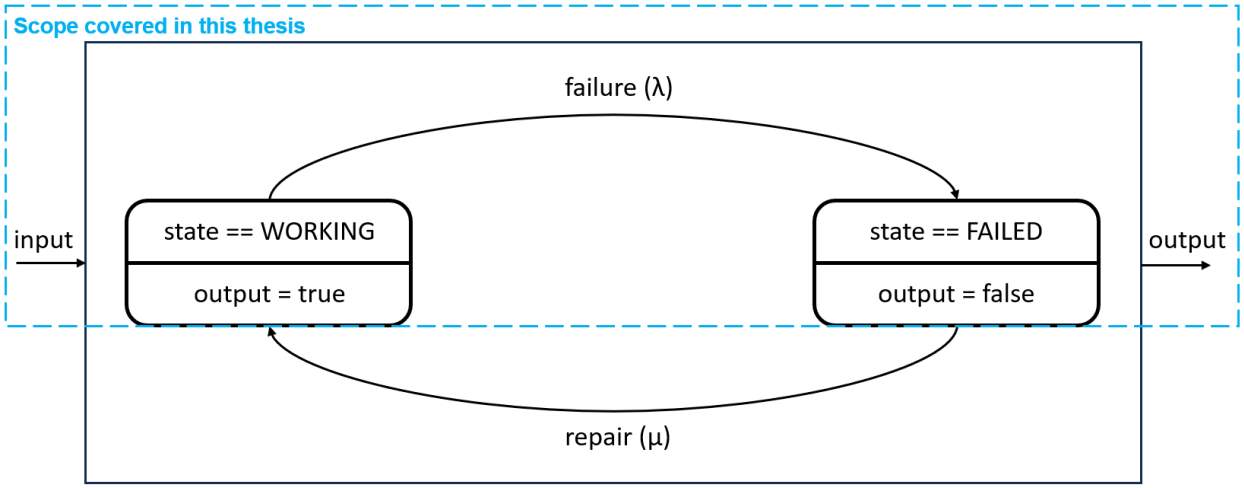

Figure 3.3 - Guarded Transition Systems Representation of a Repairable Component

A component named *lever* is depicted in Figure 3.4. The variable s represents the state of the component *lever*. It takes its value, which is *WORKING* or *FAILED*, from the domain ComponentState. The attribute init gives its initial value and is equal to *WORKING*. Its value is modified by the transition labeled by the event *failure*. When the event *failure* occurs, the state's value changes to *FAILED*, and then the flow variable values are updated.

```
class Lever
        ComponentState s (init = WORKING);
        parameter Real mu = 0.000002753;
        event failure ( delay = exponential ( mu ));
        transition
                failure : s == WORKING -> s := FAILED;
        Boolean input , output1, output2 ( reset = false );
        assertion
                output1 := s == WORKING and input;
                output2 := s == WORKING and input;
end
```

Figure 3.4 – An Example of a Component defined in AltaRica 3.0

The flow variables represent the inputs and outputs of a component and are named “input” and “output” in the examples provided in Figures 3.3 and 3.4. The attribute *reset* sets the initial value for the flow variables. Attribute *boolean* makes the flow variables have true or false values. The attribute of the flow variables depends on the state variables and can only be edited in the assertion section. The assertions set the way for how flow variables work in different conditions. Here, the assertion dictates that the output is true if the state s equals *WORKING* and an input comes to the component. The output value becomes false if the state “s” variable equals *FAILED* or no input comes to the component.

3.2.1 Fault Tree Analysis with MBSA

A fault tree is a graphical representation used in safety engineering to analyze the potential causes of system failures. Fault trees provide a structured approach to assess various events that could lead to a specific undesirable outcome, known as the top event. Events are connected to each other by gates. While the events connected by an “AND” gate must occur simultaneously, the “OR” gate states that one of the events would lead to the failure of the event above. The decision to primarily use “AND” and “OR” gates in this thesis is based on the need for simplicity in modeling the failure logic. While other types of gates (e.g., XOR, NOT, NAND) exist, they might introduce additional complexity that is not necessary for the scope of this study.

In the fault tree illustrated in Figure 3.5, the top event, *Loss of a Function*, represents the undesirable outcome being analyzed. Connected to this top event is a basic event, *Sensor Malfunction*, indicating one potential cause of the system failure with a failure rate of 1.00e-05. The failure rates for the components in this thesis are modeled using an exponential failure distribution. This approach assumes a constant failure rate over time, which simplifies the analysis. Although other distributions, such as the Weibull distribution, can provide more detail by accounting for varying failure rates over time, they require additional parameters (such as mean, standard deviation, shape and scale parameters, etc.) that are difficult to access in the literature for the components presented in the test case.

The triangles attached below, in Figure 3.5, denote that the fault tree structure continues beyond the scope of this diagram. All branches must lead to basic events with failure rates to calculate the probabilities of failures for intermediate events and, ultimately, the top event.

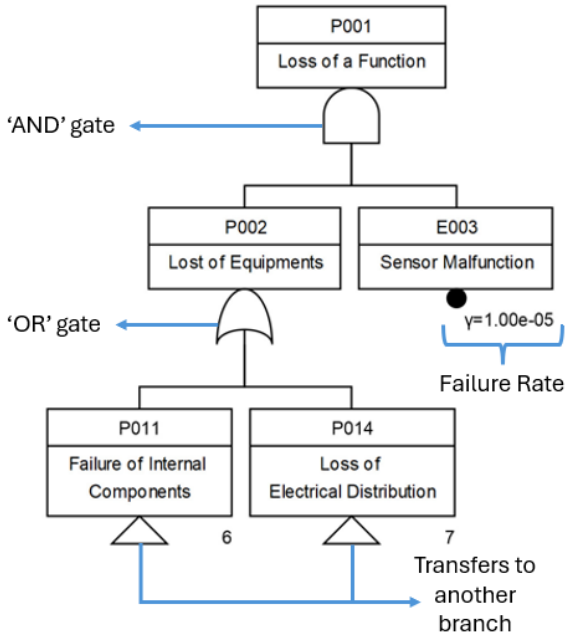

Figure 3.5 - A Simple Example of a Fault Tree (image extracted from Arbre Analyst)

As described in ARP4761, functions are associated with specific failure conditions after an FHA process, and their effects are delineated. These identified functions are the cornerstone of FTA, providing a starting point for analyzing failures since they are the top event functions for FTA. Once the top event for the FTA is identified, the next task is to establish the intermediate events that will form the pillars of the fault trees. These intermediate events are the indicators of potential failure modes or conditions that contribute to the occurrence of the top event. Depending on the level of detail available in the system model, the intermediate events may vary in complexity and granularity.

The first step for performing FTAs is to select a failure scenario and the top event for this specific failure case. System architecture layouts all functions, including the top events with failure conditions as a result of FHA. However, there needs to be a transition of the top events between system and safety models. This transformation is achieved through observers, which track the system's behavior and record specific events or conditions in AltaRica safety models. Observers operate like flow variables but have some differences: they cannot be employed in transitions and assertions to define the system's behavior, instead serving as quantities for observation. They are regularly updated after each transition executed, offering insight into the system's state and enabling dynamic monitoring of events or conditions. The top events for fault tree analysis are specified via the observers, allowing for identifying critical failure scenarios within the system. Additionally, several observers can be defined for the same AltaRica 3.0 model, enabling the generation of multiple fault trees from a single model. This flexibility ensures that various failure scenarios and their associated top events can be analyzed within the safety model.

3.2.2 Inputs and Outputs of MBSA

After presenting the principles of AltaRica, the next step is discussing what elements are needed to run safety analyses and what is expected for the outcome. By delineating these elements, there will be a clear picture of what kind of enhancements need to be made to the system model.

ARP4761A states that MBSA utilizes inputs and outputs comparable to conventional safety analysis methods. The specific inputs and outputs needed depend on the analysis type and the detail level. Minimal cutsets are one type of output of FTAs and are a group of sets consisting of the smallest combinations of basic events that result in the occurrence of the top event. They represent all the ways in which the top event occurs based on the basic events. For example, conducting a traditional FTA would necessitate FHA failure conditions for analysis and anticipate minimal cutsets and failure probabilities in return. Therefore, it can be concluded t hat conducting an FTA in an MBSE environment entails similar components.

Also, the type of analysis performed with FPM sets the boundaries of the FPM and the inputs and outputs of MBSA [64]. Figure 3.6 shows the inputs and outputs of the safety models created in this thesis. Capella, as an MBSE tool, provides the system requirements and architecture. However, the Capella system model is enhanced to provide other input elements as well. An FPM block consists of events, states, and transfer functions. The FPM drives failure condition observers with its flow and state variables and outputs minimal cutsets and failure probabilities for FTA with the assumptions presented in this thesis, as seen in Figure 3.6.

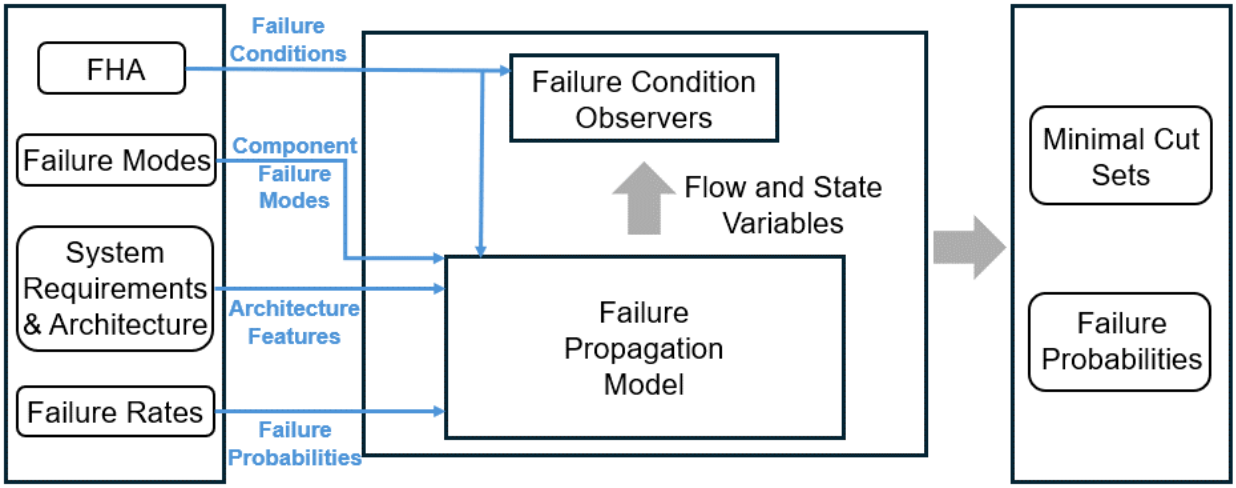

Figure 3.6 - Safety Model Inputs and Outputs Tailored for the Test Case FTA adapted from ARP4761

On the other hand, safety model elements have defined objects to work. The components in an MBSA environment need the information of inputs, outputs, events, states, and transfer functions. [64]. GTS's quintuple notation (V, E, T, A, I) illustrates a system's dynamics and what needs to be defined to create components in an AltaRica safety model.

- V (Variables): Variables are categorized into state variables (S) and flow variables (F). State variables encapsulate the system's current state, while flow variables represent dynamic interactions or events.

- E (Events): Events are symbols denoting occurrences that trigger transitions within the system. These events define the dynamics and state changes of the system.
- T (Transitions): Transitions represent the system's evolution. Each transition is a triplet comprising an event, a guard, and an action.
- A (Assertions): Assertions are instructions built on variables of V. They are actions that occur after a transition. They express the consequences of the transition.
- I (Initial Assignment): The initial assignment defines the system's starting state, providing a foundation for subsequent transitions.

Capturing the elements of the GTS notation for each component in a system enables building a safety model. The class presented in Figure 3.4 has all quintuple notations defined. Thus, the relevancy and importance of each element can be understood by referring to the explanation of the figure. Transfer functions defined in the ARP4761 share the same functionality with the assertion section of AltaRica, determining the output based on a component's inputs and states. Table 3.1 depicts the namings of MBSA elements stated in ARP4761 and GTS of AltaRica. This thesis denominates MBSA artifacts using both sources interchangeably.

Table 3.1 - Matching Elements of GTS and an ARP4761 MBSA Model

ARP4761	GTS
Inputs, outputs	Flow variables
States	State variables
Events	Events
Transfer functions	Assertions

3.3 System Model Development with Capella

This section provides information on the selection of the MBSE environment, explains different levels of ARCADIA in subsections, and emphasizes the extended logical levels of Capella.

This thesis utilizes ARCADIA/Capella for the following reasons:

1. While ARCADIA is the method to define system architecture, Capella is the language to apply the ARCADIA approach. Many languages and tools for modeling come without a method of modeling, unlike Capella.
2. ARCADIA/Capella is that it shifts the focus from the modeling languages to the method and its procedure. Therefore, systems engineers are not expected to be experts in the modeling language.
3. ARCADIA/Capella is an open-source solution used in aerospace domains by Thales and is being evaluated by Bombardier, the industrial partner of the thesis.

Capella/ARCADIA has four different levels of system development: Operational Analysis, System Analysis, Logical Architecture, and Physical Architecture. Since the Operational Analysis level defines high-level interactions among entities and actors and captures the operational needs of stakeholders [91], it does not fall under the scope of this thesis. Instead, this thesis focuses on the Logical Architecture level more because this stage covers the system and system architecture development, as Figure 3.2 shows.

The modeling in the thesis follows a top-down approach in compliance with the ARP4754. The following subsections explain each Capella stage in order.

3.3.1 System Analysis

SA is the starting phase involving the decomposition and analysis of the system architecture to ensure that it meets the specified requirements [53]. The SA level is fundamental for understanding the system's structure, behavior, and interactions. Breaking down the system into manageable components for in-depth analysis helps engineers to understand and create logic. The main activities at this level are identifying and defining system functions, allocating functions to systems and components, identifying systems interacting with the system of interest, and defining interactions between systems and actors [47].

Figure 3.7 captures the main activities of the SA process by illustrating the structural organization and functional relationships within the system architecture. The system of interest, which represents the focus of the development, is located at the center of the design. System actors represent external entities interacting with the system of interest and among themselves. Each function in the design captures a distinct aspect of system functionality, contributing to the overall capabilities and objectives of the system. Moreover, functional exchanges between functions signify the flow of information and interactions within the system architecture. These exchanges represent the relationships between different systems, subsystems, and actors.

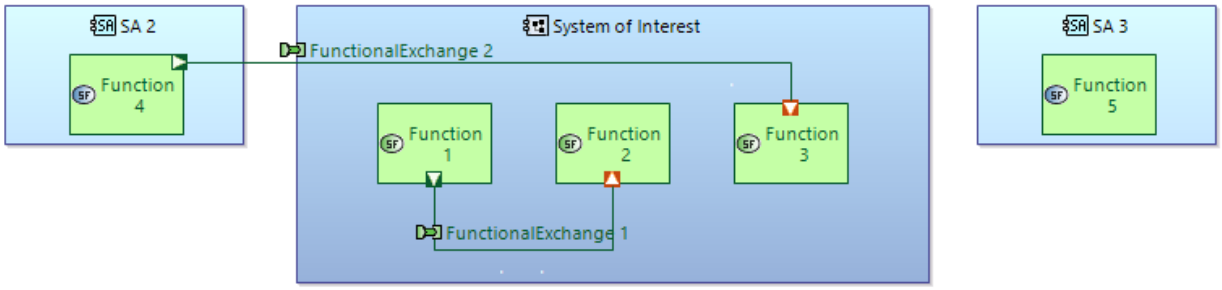

Figure 3.7 - Sample of a System Architecture Diagram in Capella

In the safety assessment process, SA-level functions correspond to aircraft-level functions analyzed in AFHA. The outputs of AFHA (failure conditions, effects, and classifications) support generating functions and requirements for the next steps of development.

3.3.2 Logical Architecture

Following a top-down approach, the next analysis step is the logical architecture level, where how the system works to fulfill expectations is defined, and a more detailed exploration of the system's architectural elements takes place [47], [52]. Continuing the development from the system analysis drives system engineers to refine the system elements. The refinement not only includes the components, functions, and their exchanges but also defines logical subsystems and components. Therefore, this stage must establish the system's internal organization, interfaces, and interactions and develop its logical structure. Like the system analysis stage, the logical architecture level helps comprehend and form the system's design. However, here, the focus switches to outlining the logical parts and subsystems of the system architecture.

The level of detail of the system elements is more refined in the logical architecture diagram compared to the system analysis level. The refinement that takes place in Figure 3.8 achieves the objectives of the logical architecture level. *Function 5* in *SA 3* has been decomposed into *SubFunction 5.1* and *SubFunction 5.2*. Similarly, *Function 1* now becomes a parent function, meaning a combination of its subfunctions achieves the same functionality. The decomposition of *Function 1* is done by three subfunctions, *SubFunction 1.1*, *SubFunction 1.2*, and *SubFunction 1.3*, and they are allocated to a newly defined *Logical Subsystem 1*. This decomposition reflects a more granular architecture.

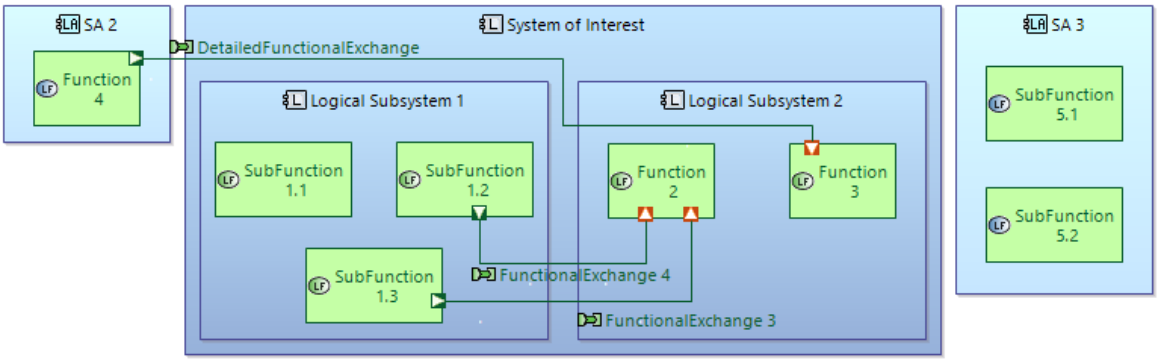

Figure 3.8 - Sample of a Logical Architecture Diagram in Capella

Furthermore, *Function 2* and *Function 3*, located in the system of interest in the system analysis, have now been allocated to a newly defined *Logical Subsystem 2*. The allocation signifies the establishment of logical subsystems for capturing related functions and components. Thus enhancing modularity within the system architecture.

A similar refinement takes place in functional exchanges. The exchange from *Function 4* to *Function 3*, previously labeled *FunctionalExchange 2*, has been elaborated into *DetailedFunctionalExchange*, reflecting a more detailed exchange specification. Additionally, the functional exchange from *Function 1* to *Function 2* is broken down into two different exchanges to reflect the decomposition of *Function 1* into its subfunctions. *FunctionalExchange 4* and *FunctionalExchange 3* together in the logical architecture satisfy the interaction between *Function 1* and *Function 2* in the system analysis.

3.3.3 Physical Architecture

In the context of aircraft development, the physical architecture level represents the stage where the system's logical design transforms into physical components. The level defines how the system should be built and developed. The emphasis on the physical architecture level is limited in the thesis. This is due to aircraft manufacturers outsourcing physical components' design and development efforts to specialized supplier companies. Supplier companies are tasked with translating the functional and performance requirements outlined by the aircraft manufacturer into viable design solutions based on the physical architecture. In this way, aircraft manufacturers leverage the expertise of external suppliers in component design.

Therefore, the physical architecture level in this thesis corresponds to the implementation efforts in the development process and the System Safety Assessment (SSA) in the safety assessment

process. The physical architecture phase should capture SEE, FMEA, and FMES that are out of the scope of the thesis.

Figure 3.9 illustrates the PA-level architecture diagram that is the continuation of the LA in Figure 3.8. The *System of Interest* in Figure 3.8 goes under a physical implementation phase in PA. As a result, physical components form the system by carrying the functions from LA to PA. *SubFunction 1.3* in Figure 3.8 becomes a parent function with two subfunctions in the PA level, which are *Physical Function 1* and *Physical Function 2*. *FunctionalExchange 3* is captured in the PA level by allocating it between *Physical Function 2* and *Function 2*. *Component 1* and *Component 2* form *Logical Subsystem 1* and carry the same functional capacities as the logical subsystem. The physical implementation of *Logical Subsystem 2* is done with *Component 3* and *Component 4*. Actors go through a transition from LA to PA and become physical actors in this stage.

In addition, logical components that are transferred from LA to PA become behaviour physical components (named *Behavior PC* in Figure 3.9). While these *Behavior PCs* are allocated to the physical nodes, yellow boxes, that represent the physical implementation of the logical components, physical functions are appointed to the *Behavior PCs*.

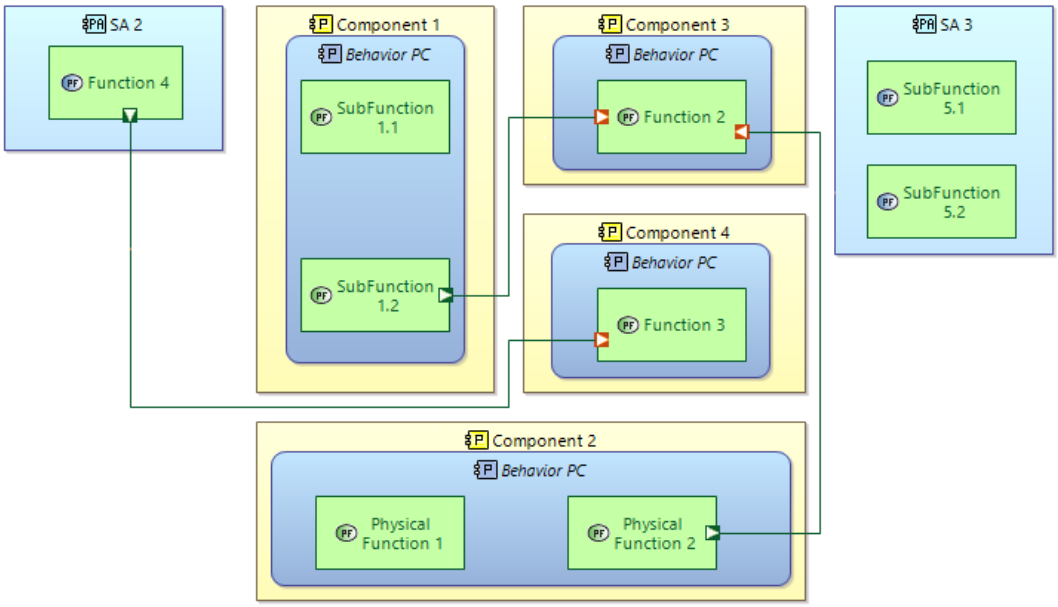

Figure 3.9 - Sample of a Physical Architecture Diagram in Capella

3.3.4 LA and Safety Process Alignment

This thesis refines the system architecture by extending the LA level to three phases in Figure 3.10 —L0, L1, and L2. Table 3.2 summarizes the key activities done at each level. The utilization of three logical levels helps to facilitate a gradual refinement and specification of the system architecture. The three-level approach is a representation of the development in system architecture. It also enables capturing essential information necessary for conducting thorough safety analyses. The method starts with L0 and ends with L2. Phase L1 captures iterations between the other two levels. As the methodology advances to L2, the level of detail achieved approaches

that of the physical architecture, reducing the need for extensive focus on the physical architecture level in Capella.

This thesis builds on the method presented by Tabesh [38]. Tabesh proposes an MBSE approach for early aircraft design aimed at unconventional architectures such as hybrid aircraft. Therefore, it describes a multi-leveling approach to accommodate different technological choices an architecture can capture. As the method of Tabesh progresses to L0, it shows a system for SoI and other systems interacting with the SoI. Once the level L1 is reached, aircraft-level model propagation starts. There are multiple derivative models are created at L1 and they are called as System-Logical (Sys-L1).

While the multi-leveling methodology that Tabesh proposes aims to manage the model variants for better accessibility and traceability, in this thesis, the goals are to reflect refinement in the aircraft development process and allow performing various safety assessments at each logical level to influence the system design. Also, Tabesh integrated FHA artifacts into a Capella system model by utilizing PVMT, whereas the methodology of this thesis uses PVMT to integrate safety properties for FTA.

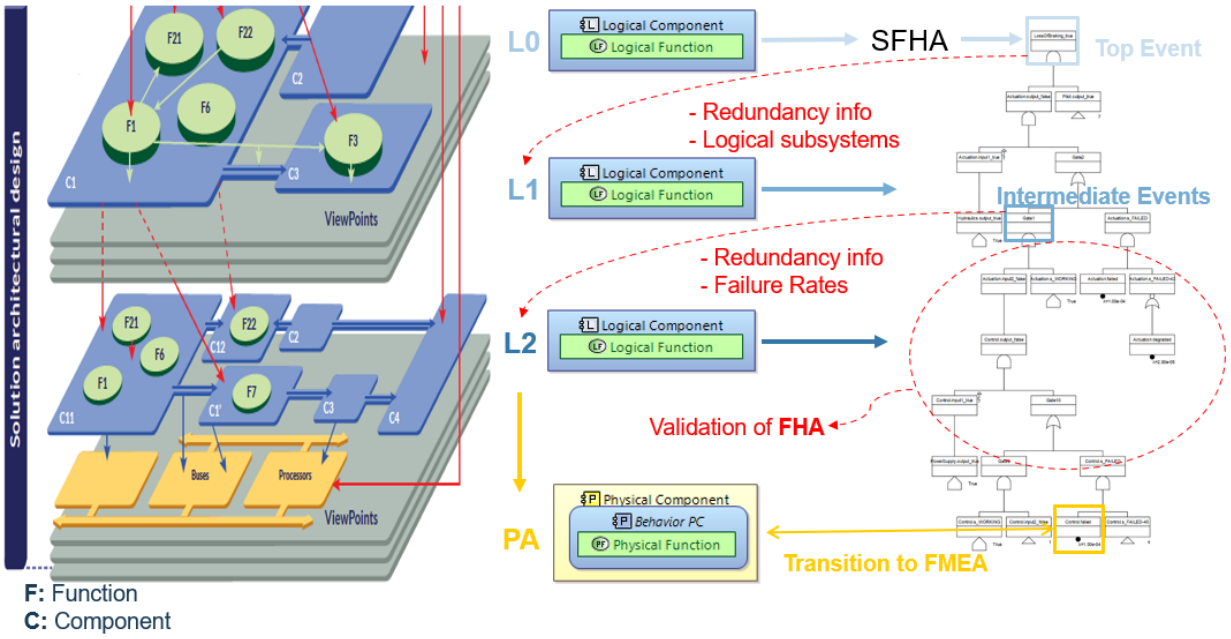

Figure 3.10 - Capella Logical Architecture and Safety Analyses Relationship

At the LA level, systems and functions are decomposed into sub-systems and subfunctions. SFHA is being done at this stage, as Figure 3.10 shows. The SFHA focuses on identifying hazards and assessing risks precisely with the details available. Each identified hazard and associated function is then mapped onto a fault tree, forming the basis for subsequent FTAs. These fault trees begin with top-level events derived from the SFHA results. They help to facilitate running different safety analyses for various failure scenarios.

From a safety point of view, each logical level—L0, L1, and L2—serves a distinct purpose in enhancing the system architecture while facilitating different types of safety analyses. Beginning with L0, this level represents the initial stage where fundamental functions are identified and classified as top-level events through the FHAs. L0 is particularly suited for qualitative analysis

because it identifies critical functions without going into detailed quantitative data. Also, L0 and L1 levels offer quantitative analysis involving component/subsystem failure rates with their orders of magnitude instead of precise failure rates.

Moving to L1, a higher level of refinement is achieved, characterized by increased redundancy and availability of failure rate information to some extent. At this stage, the interactions between systems are delineated with greater detail. However, the system does not yet have the full redundancy information, and there might be missing information on specific component-level elements. Thus, L1 remains suitable for qualitative analysis and, at the same time, might be offering quantitative analysis for certain failure scenarios.

Finally, at L2, the system architecture reaches its most detailed and refined state. It encompasses all levels of redundancy. All subsystems, functions, functional exchanges, and interfaces are fully defined. Due to constant logical subsystem breakdowns along the process, component-level information is reached, allowing the storage of failure rate data for each component. L2 is particularly well-suited for quantitative analysis because it has detailed information on component-level system engineering elements.

Table 3.2 - Key Activities to be done at each Logical Architecture Level

Logical Architecture Level	Key Activities for System Development Adapted from [38]	Key Activities for Safety Assessment
L0 (starting point)	Define logical systems, making a transition from system functions to logical functions, allocate the functions to logical systems.	Identification of the FTA's top events through FHA, qualitative FTA to form a fault tree structure
L1 (several iterations)	Define logical subsystems and components, breakdown of logical functions, allocating the detailed logical functions to logical subsystems and components.	Quantitative FTA with both precise failure rates and orders of magnitude, qualitative FTA for fault tree structure and minimal cutsets
L2 (endpoint)	Define logical components, allocate logical components to logical subsystems, breakdown of the logical functions, allocate the detailed logical functions to logical components.	Quantitative FTA with precise failure rates, qualitative FTA for minimal cutsets

3.4 Safety Artifact Integration

This thesis facilitates integrating safety artifacts into the Capella system model by utilizing the capabilities of the PVMT addon. PVMT is an extension designed to define specific properties of the different system model elements in Capella [92]. Refer to Appendix B to examine a guide on how to use PVMT. This thesis uses the PVMT as a bridge to introduce the essential elements of the GTS quintuple and the defined inputs of an MBSA model in the ARP4761A document into the Capella modeling environment. In the methodology presented, this add-on allows engineers to create additional features for the system model components, customize safety properties, and attribute values to these properties for individual system elements.

Figure 3.11 below shows different safety artifacts embedded into the Capella system model. The SFHA results provide information about the top events of FTA, as previously mentioned. Therefore, each logical level, starting with L0, accommodates FTA. The L2 phase involves quantitative FTAs that must validate the SFHA. AltaRica helps perform automated FTAs with components. Therefore, the logical components of the system model must encompass the safety properties. However, the same properties can be added to other system model elements, such as functions, exchanges, etc. Hence, the system model can accommodate different safety analyses. There are five safety features integrated into Capella by PVMT: Failure rates, state variables, transfer functions, flow variables, and functional failure chains. While functional failure chains support the safety assessment activities, the other safety properties integrated are necessary to perform FTAs according to the latest version of ARP4761A. Also, the safety artifacts integrated, except the functional failure chains, fulfill the GTS quintuple that forms the framework of AltaRica. The rationales for choosing these safety artifacts to embed into the system model can also be examined by referring to sections 3.2 and 3.2.2.

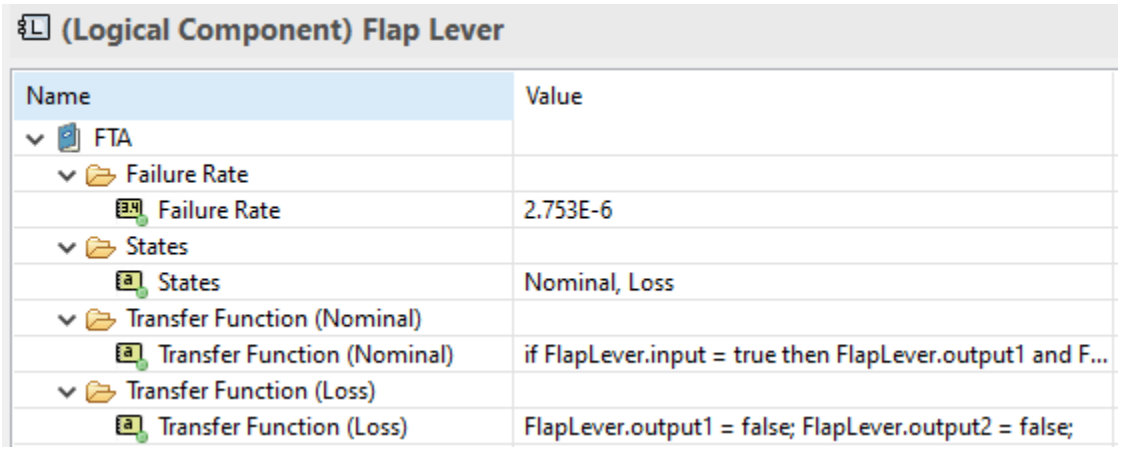

Figure 3.11 - Additional Properties added to a Logical Component in Capella by PVMT

1. Failure Rates:

Embedding failure rates is a key objective because it integrates quantitative reliability information directly into the system architecture. This integration enables engineers to assess the likelihood of component failure by assessing different failure scenarios with FTA to validate FHA. The failure rates used in this thesis are expressed in failures per hour (fph). This unit indicates the likelihood of a component failing during a one-hour period. For example, a failure rate of 1.00E-05 fph means that there is a 0.00001 probability of failure per hour for that component.

2. State Variables:

The state variables should indicate all the states a component can take triggered by different events. Nominal represents the working state (indicated as WORKING in AltaRica), while loss (indicated as FAILED in AltaRica) shows the failed state in the work presented. The event failure triggers the change between these states.

3. Transfer Functions:

Integrating assertions, referred to as transfer functions, into the Capella model enhances the system's analytical capabilities. By integrating them using AltaRica syntax, engineers can effectively define the behavior of components within the system. When engineers specify a component's possible states, the system model automatically prompts for corresponding transfer functions for each state. For instance, the component in Figure 3.11 has two states, nominal and loss, so engineers must define transfer functions for both states in AltaRica syntax. Using the AltaRica syntax provides traceability between the safety model and the system architecture since there would be uniformity in the language used in both models. The transfer functions of the *Flap Lever* component shown in Figure 3.11 dictate that the component loses its outputs when its state variable is *loss*. Nominal transfer functions state that the outputs are provided by this component while there are inputs coming from other components and the state variable is *nominal*.

4. Flow Variables:

Figure 3.12 shows a component exchange in red edited by PVMT to make it a command flow variable. Component exchanges are selected to map flow variables in the system model because their ports indicate the flow of information between components.

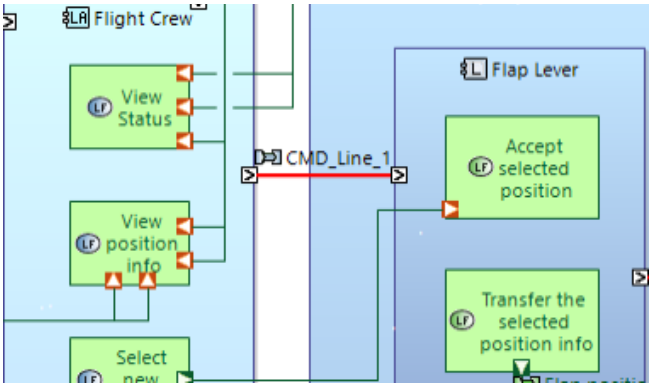

Figure 3.12 - Representation of a Flow Variable, colored in red, in Capella with an edited Component Exchange

Flow variables define the connections between different components in a safety model. Thus, two elements are required for definition. To emphasize this, component exchanges are edited by the PVMT to show the notion of flow variables. Engineers can modify component exchanges with PVMT to change the definition and make them applicable to address the failure connections between components. These edited exchanges are colored red, green, and blue to indicate command, power, and data connections, respectively. Giving them different color codes distinguishes them from the original component exchanges that show interactions between two components

5. Functional Failure Chains:

The last method to enhance the system model is incorporating functional chains in Capella to articulate the failure relationships across different system components. It aims to understand the potential impact of component failures on system functionalities by investigating the system model's different types of functional exchanges. Regardless of the specific failure mode of a selected component to be inspected, the focus remains on understanding which functionality is

affected. It is done by investigating the top event function for a particular failure case and its associated functional exchanges.

Figure 3.13 depicts a simple example of a functional chain in Capella. *Function 4* has two outgoing functional exchanges. The functional chain follows the functions connected to *Function 4* and their exchanges. The chain ends with reaching functions with no outgoing or incoming exchanges. In Figure 3.13, *Function 3* and *SubFunction 1.1* connects the chain to *SubFunction 5.1* and *SubFunction 5.2*. The chain ends with *SubFunction 5.1* and *SubFunction 5.2* because there is no other functional exchange to follow. By creating functional chains in Capella, the exchanges among specific functions can be highlighted. Functional chains are used to identify failure relationships among functions in this thesis.

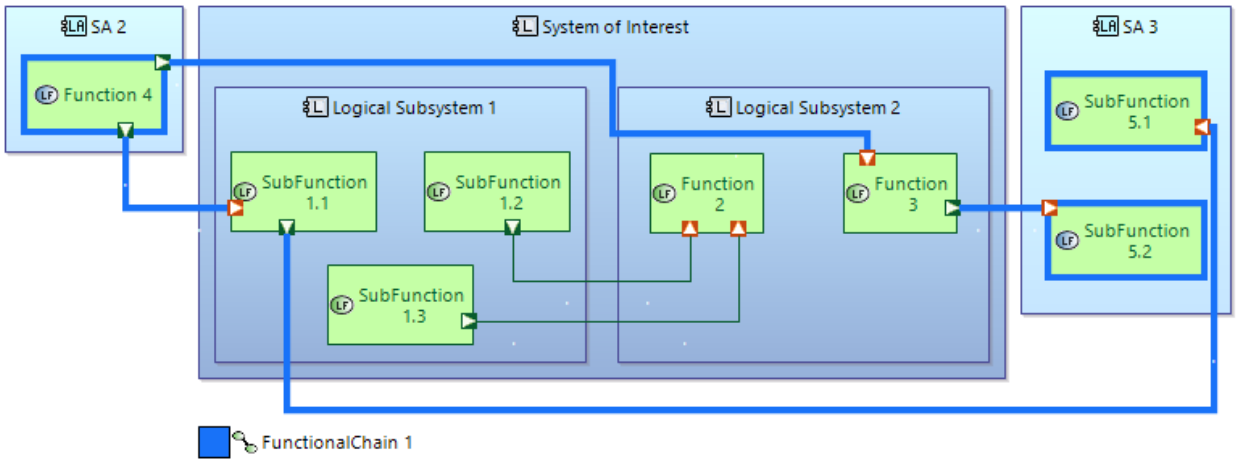

Figure 3.13 - Example of a Functional Chain

For instance, if a control unit fails due to a command-related issue- assuming it is an intermediate event in an FTA -the focus shifts to examining the top event function within the system model, specifically focusing on the functional exchanges that present command connections. After identifying the functions that have command-related functional exchanges with the top event function, the same identification must be done for each function. It continues until there are no other functional exchanges to inspect. Finally, there is a map of functions with their exchanges to represent failure relationships between them for certain failure scenarios. The map identifies the functions that might be affected by the control unit failure, including those allocated to the control unit component, to understand which functions are lost depending on the failure mode. This particular method is built since there is not enough FHA information present for the test case presented in the thesis.

3.5 Methodology Summary

This chapter has introduced a methodology to enhance model-based system architecting by incorporating safety artifacts into the system development lifecycle to run model-based safety analyses. The methodology consists of iterative phases, each contributing to the refinement of the system model.

Initially, the methodology focuses on establishing a test case architecture and developing a simple Capella system model. Subsequent iterations involve enhancing the system model's specificity and detail, integrating safety elements, refining the architecture through logical levels (L0, L1, and

L2), and performing safety analysis in AltaRica 3.0, as Figure 3.14 depicts. Overall, the process includes creating AltaRica safety models after each logical level and performing FTAs before the creation of the next level in the Capella system model. The main steps of the methodology implementation are in the following:

- Step 1: The system model development progresses through three levels: System Analysis, Logical Architecture, and Physical Architecture. The thesis focuses on the logical architecture levels (L0, L1, and L2), which involve defining logical systems, subsystems, and components, breakdown and allocation of functions, and refining the architecture to facilitate safety analyses. After the development of each level, safety elements are added to system components by utilizing PVMT, which is the second step.
- Step 2: The thesis introduces various enhancements to the Capella system model, including embedding safety properties, defining functional chains to express failure relationships between functions, and modifying component exchanges to show flow variables. The integration of safety analysis into the Capella system model is facilitated using the PVMT addon. This Capella extension enables engineers to embed essential safety properties into system model elements, including failure rates, state variables, assertions, and flow variables. This step constitutes the main contribution of the thesis.
- Step 3: AltaRica, a modeling language for MBSA, is utilized to build three safety models to run FTAs. The language's guarded transition systems framework enables the formal representation of system dynamics and state changes, facilitating safety analysis. The creation of the safety models in this environment is possible with the safety information stored in Capella using PVMT. The transitions from Capella to AltaRica and from AltaRica to automated FTA are manual. The safety properties of Capella are put into classes and blocks of the AltaRica safety model; then, to perform automated FTAs, parameters such as flight hour and the observers to inspect are selected manually.
- Step 4: FTAs are conducted for each logical level to assess potential safety issues associated with the system architecture. For each failure scenario, an observer is defined, enabling quantitative and qualitative fault tree analysis in AltaRica 3.0. The fault trees are visualized using Arbre Analyst to ease the qualitative analysis. The quantitative assessment at L2 validates the FHA results.

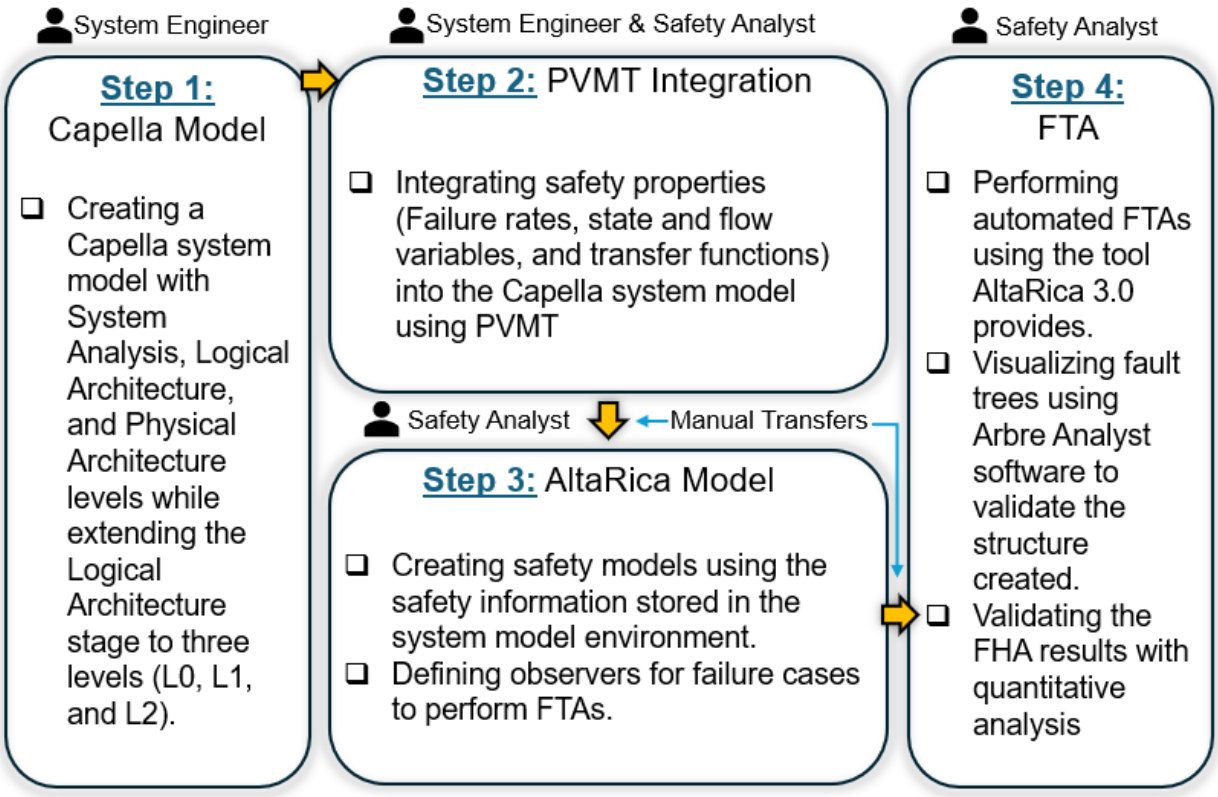

Figure 3.14 - Methodology Implementation Process

It is important to note that initial assignments, events, and transitions are not integrated into the system model due to the scope of the thesis is limited to FTA. The decision to exclude events and transitions is based on their dynamic nature, which varies depending on the specific failure scenario being analyzed. Different failure cases may trigger different events and transitions within the system, making embedding them into the system model impractical. Additionally, since only nominal and loss states are considered in the test case analysis, setting every component's initial assignment as nominal in the safety model is sufficient.

The next chapter presents the application of the proposed methodology on a Capella system model built on a test case. The process illustrated in Figure 3.1 is followed by a test case integration, including different safety models created and conducted FTAs with selected failure scenarios in the MBSA environment AltaRica.

4. Modeling and Integration

Building on the foundation established in the previous chapter, the following section takes a practical approach by presenting a test case: modeling and integrating a conventional flap system. This chapter explains in detail the process of representing a system architecture within the MBSE framework, Capella, and creating model-based AltaRica safety models in line with the system architecture. The chapter explains the flap system architecture and every enhancement made to the system model in detail.

4.1 Test Case: Flap System

A flap system is selected as the SoI to build a system model and run safety analyses. Figure 4.1 illustrates a simple architectural view of the selected flap system. A comprehensive understanding of the flap system in aviation is essential before a detailed description. Flaps, which are aerodynamic components installed on an aircraft's wings, modify the lift and drag properties of the aircraft, especially during takeoff and landing. The test case adapts the flap system of Global 5000, an aircraft of the industry partner Bombardier. The selected flap system is an architecture to model the system architecture and analyze safety aspects in this thesis. The technological choices of the flap system and the components used for these choices to form an architecture are taken from Global 5000 [93],[94],[95].

For this test case, three failure scenarios were selected to perform FTAs: Annunciated loss of flap extension/retraction, unannunciated loss of flap extension/retraction, and flap panel disconnection. To develop a complete set of failure scenarios, a comprehensive FHA needs to be conducted. Since the FHA is out of the scope of this thesis these failure scenarios were derived by expert consultations from Bombardier. Each selected failure scenario is described as follows:

- Annunciated loss of flap extension/retraction: Involves a failure in the flap system that is detected and indicated to the flight crew. The annunciation allows the pilots to take corrective actions, mitigating potential risks.
- Unannunciated loss of flap extension/retraction: The flap system fails without any indication to the pilots. This can lead to a more dangerous situation as the pilots are unaware of the malfunction and unable to respond promptly.
- Flap panel disconnection: Involves the physical separation of the flap panel from the wing. Such a failure can cause significant aerodynamic issues and pose a severe safety risk due to the potential for further structural damage.

Safety models must include failure rates for each component to perform FTAs. Table 4.1 lists each component with failure rates adapted from [96] and [97]. In the case that a precise rate for a component cannot be found, the failure rate magnitude of similar components is selected.

Table 4.1 - Failure Rates for Each Component in the Flap System (per Flight Hour)

Component	Failure Rate
Flap Lever	2.75E-6
Control Units	5E-5
DC Power Source	1E-5
DC Motor	7.31E-6
Air Data Computer	5E-7

EICAS	1E-5
Torque Tube	1E-7
Speed Summing Gear	1.47E-5
Branch Gear	1.47E-5
Bevel Gear	1E-7
Flap Panel	1E-9
Flap Ballscrew Actuator	1E-5
Flap Position Sensor	5E-7
Position Transducer	5E-7
Wing Tip Brake	5E-5

The redundancy defined for the SoI aims to reduce the risk of single-point failures by incorporating multiple components that can take over in case one fails. The system configuration includes a Power Drive Unit (PDU) featuring two DC motors powered by electrical sources placed inside it [94]. These motors are powered by separate electrical sources within the unit. This means that if one motor or its power source fails, the other motor can still operate, ensuring continuous functionality of the power drive unit. The system mitigates the risk of a single-point failure because the failure of one motor does not lead to the total loss of function.

The primary components of the test case include Flight Control Units (FCUs), a PDU, a flap lever, driveline components, wing tip brakes, flap panels, actuators, flap position sensors and position transducers. The primary components of the flap system are described in the following:

- Flight Control Units (FCUs): The FCUs serve as the central controllers responsible for translating commands from the flap lever into physical flap movement. The system architecture includes two FCUs. While the presence of two FCUs reduces the risk of total system failure, it does not guarantee uninterrupted operation in all scenarios. If one FCU fails, the other can take over the control, thereby maintaining system functionality in the event of a single FCU failure. However, other factors, such as simultaneous failures or failures in interconnected components, can still impact the operation of the system.
- Power Drive Unit (PDU): The PDU is a distribution hub for the flap system. The DC motors, housed within the PDU, are responsible for actuating the flaps by converting electrical power into mechanical motion. The PDU integrates a speed summing gear mechanism to aggregate the rotational speeds of the individual DC motors, ensuring continued system operation even in the event of a single motor or FCU malfunction.
- Flap Lever: A flap lever is an interface for flight crew members. The flap lever enables pilots to adjust the position of the flaps. Commands from the flap lever are relayed to the FCUs, initiating flap movement based on pilot inputs. Flap position indicators placed next to the lever, as seen in Figure 4.1, provide real-time feedback to flight crew members regarding flap angle and deployment status. The flap angle indicator displays the angular orientation of the flaps relative to their neutral position, while the position indicators announce whether the flaps are retracted or deployed, shown as *IN* and *OUT*.

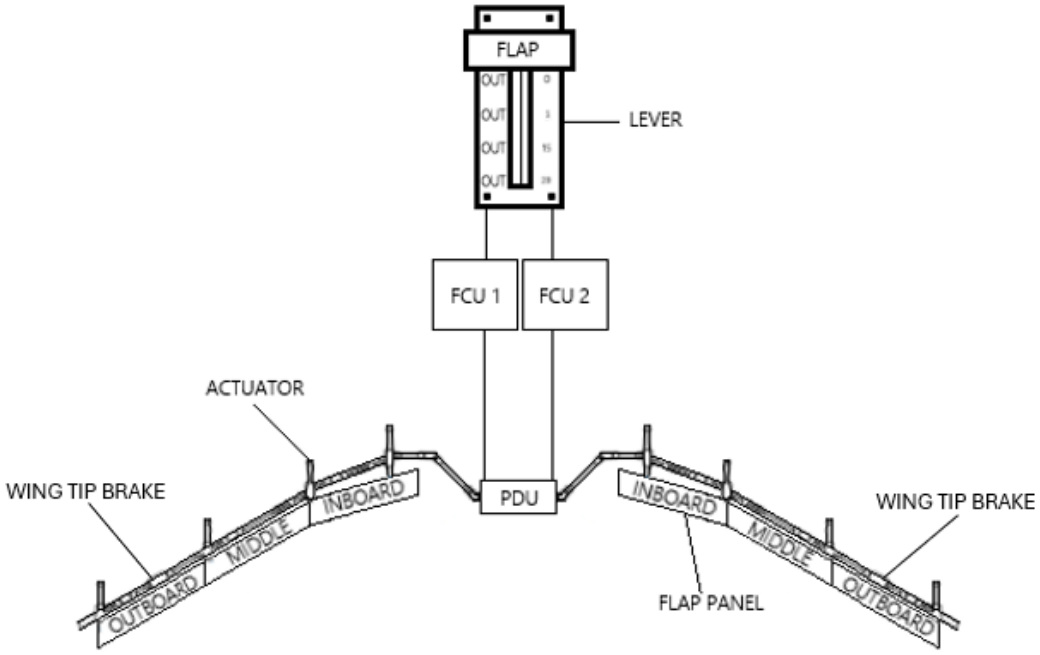

Figure 4.1 - Flap System Architecture adapted from publicly available Global 5000 documentation [94]

- Driveline: The driveline transmits the power the PDU generates to the actuators to move the flap panels. It consists of multiple torque tubes and bevel gears located before the first inboard and after the second actuators. The bevel gears adjust the angle of the driveline while transmitting power. The driveline includes two types of brakes to ensure safety: wing-tip brakes and an asymmetry brake integrated into the PDU.
    - Wing Tip Brakes: Positioned between the middle and outboard panels, wing tip brakes serve as a safety measure to halt the driveline's operation. They can interrupt the transmission of power from the PDU to the actuators in case of an emergency, such as the asymmetric deployment of flaps, or during maintenance work on the system.
    - Asymmetry Brake Integrated into the PDU: This brake is designed to arrest the complete driveline system in case of a shaft failure or asymmetry, working in conjunction with the wing tip brakes to provide safety.

  Two position transducers (sensors) are located at the end of each driveline to monitor the alignment of the driveline components. The transducers provide feedback to the flap control units on skewing encountered during operation.

- Flap Actuators: The flap system's architecture includes eight ball-screw actuators, each connected to one for outboard or two for inboard flap panels. These actuators are responsible for deploying or retracting the flap panels in response to commands from the FCUs. The distribution of actuators per flap panel reflects a standard configuration employed in many aircraft, including Global 5000, with the inboard flap panels typically actuated by two actuators due to their higher load requirements than the outboard panels. Also, flap position sensors are located next to the actuators for each actuator. They provide position feedback on the flaps.

- Flap Panels: Flap panels are positioned along the trailing edge of the aircraft's wings and can be deployed or retracted to alter the wing's shape and lift characteristics. There are six flap panels, three panels for each side, positioned on the wings in the architecture.

Figure 4.2 depicts that the command chain begins with the flap lever in the system. It is the interface for flight crew members to change the position of the flaps. It sends commands received from the pilots to the FCUs that activate the PDU afterward. The PDU executes the commands of applying the desired changes in the flap position by running the driveline. Driveline components are the physical means to transform these commands to change the flap panel angles.

In addition, as shown in Figure 4.2, the accommodation of two FCUs and DC motors combined with a speed summing gear located in the PDU provides continued system operation in the event of a DC motor failure or FCU malfunction. The primary function of speed summing gear is to sum the rotational speeds of the individual DC motors driven by each FCU. In the event of an FCU or DC motor failure, the speed summing gear compensates by adjusting the output speed, thus deploying the flaps at half speed, ensuring continued operation. Additionally, the FCUs also manage the flap sensors and brakes. The flap sensors provide position data to the FCUs, which use this information for monitoring and control. The FCUs also control the brakes to ensure the flaps can be stopped or held in position after deployment.

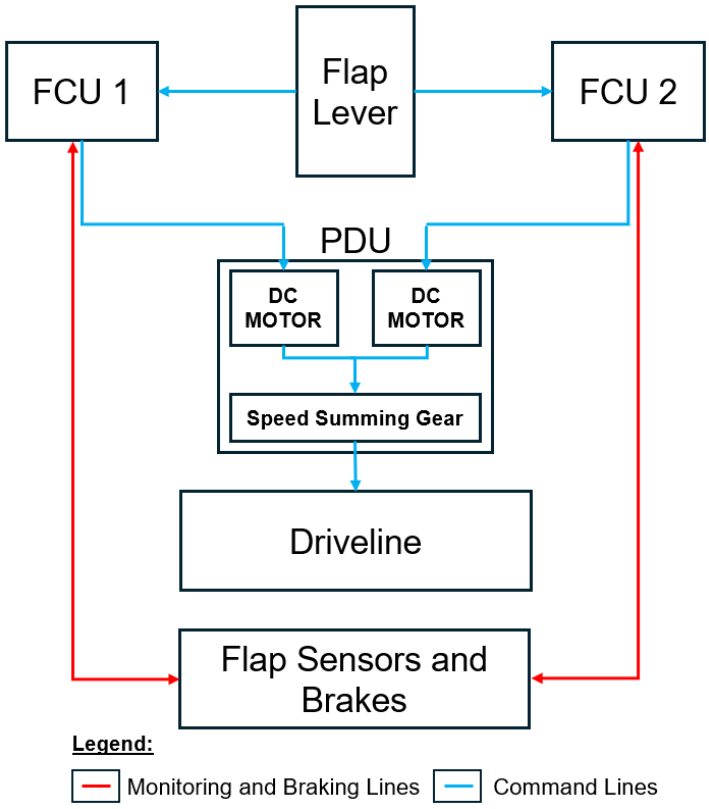

Figure 4.2 - Flap System Test Case Schematic

Overall, the chosen components and their arrangement within the architecture are designed to represent a traditional configuration and demonstrate the practical application of the methodology. Each component fulfills a specific function to facilitate the operation of the flap system.

4.2 Capella System Model

This section presents different levels of system development in Capella / ARCADIA: System analysis, logical architecture, and physical architecture. The operational analysis level is omitted, as discussed in the previous chapter.

This thesis utilizes the Capella 5.1.0 with the addon PVMT 50.5.1 versions. The Diagram Styler feature helps to apply the safety viewpoints created by PVMT on the Capella diagrams. Functional chains and color-coded component exchanges are only visible if activated on the selected diagram with the diagram styler. Table 4.2 lists different diagrams and viewpoints that each Capella level accommodates. While SA and PA levels only display the architectural diagrams, LA levels (L0, L1, L2) consist of three different architectural diagrams, a functional breakdown diagram to help engineers understand the refinement, and safety viewpoints that show the enhancements made on the system level.

Table 4.2 - Diagrams and Viewpoints used in each Capella Level

Level	Diagrams & Viewpoints
System Analysis	System Analysis Breakdown (SAB)
Logical Architecture	Logical Architecture Breakdown (PAB) (L0, L1, L2) Logical Functional Breakdown Diagram (L0) Safety Viewpoints with Diagram Styler (L0, L1, L2)
Physical Architecture	Physical Architecture Breakdown (PAB)

The first step is developing an SA level from the test case architecture. Next, L0 is built with the help of automated transitions for functions and actors in Capella. L1 architecture is shaped by introducing more redundancy information. L2 is reached with great detail of specifications and redundancy, representing an architecture that can accommodate quantitative analyses with the enhancements. Finally, a transition is made from L2 to PA, keeping the same structure and layout as L2. The system model development progression is described in detail in the following sections labelled after each development level of Capella.

While developing the system model, requirements provided by the industry partners help shape the architecture. The functions from a requirements document for a slat flap control system in [98] are carefully examined to design the architecture so the FTA failure scenarios can be accommodated with the system model. The requirements were derived from publicly available maintenance manuals of several Bombardier aircraft, including the Global 5000 [98]. Table 4.3 illustrates which requirements document functions are satisfied by the system model. The functional requirements are not satisfied at one specific level. Different logical architecture levels introduce certain components and logical functions to capture the requirements. While introducing new elements to the model to satisfy the functions, the functions are tailored to the scope of the thesis. For instance, not all safety functions are introduced for the flap system, but the ones needed for selected failure scenarios. In addition, only the parts related to the flap system are considered in this thesis. Requirement number 5 can be an example of this consideration because the system model has functional exchanges that provide flap system status, data, and interface to EICAS while excluding any other aircraft systems.

Table 4.3 - Functional Requirements Document adapted from [98]

1) The Flight Control System (FCS) shall respond to pilot commands for a change in High Lift configuration
2) The FCS shall move the High lift Surfaces to the selected configuration at a controlled rate of motion.
3) The FCS shall hold the high lift surfaces at all selected positions.
4) The FCS shall provide safety functions.
5) The FCS shall provide the required system status, system data, and interface to EICAS and other aircraft systems.
6) The FCS shall provide means to aid installation and trouble-shooting problems for the maintenance crew.

4.2.1 System Analysis

The goal at the system analysis level is to define the system's contributions to satisfy users' and stakeholders' needs [47]. The purpose of the SA stage is achieved by defining high-level functions with their exchanges and allocating them to different system elements that are system actors and the SoI.

Figure 4.3 shows system-level functions attributed to both the SoI and system actors. These functions define the system's operational scope, and their exchanges define the interactions between the SoI and its surrounding actors.

The systems interacting with the flap system are presented as actors and will stay as actors throughout development. However, more actors can be introduced with the advancement of each level in Capella. The actors defined in the SA are the Flight Crew, Engine Indicating and Crew Alerting System (EICAS), Air Data Computer, and Maintenance Crew.

The Maintenance Crew is incorporated into all phases of Capella to recognize its role in system maintenance and safety. While it is not directly part of the test case for FTA, it emphasizes the importance of considering the system's operational environment. The Maintenance Crew represents a stakeholder whose actions may impact system reliability and maintenance procedures. Their inclusion in the SA phase ensures that all system stakeholders are accounted for, even if they do not have direct involvement in specific test cases for FTA. Incorporation of the Maintenance Crew for LA and PA levels supports the efforts of presenting a top-down development approach since this actor is getting more detailed going through the system model levels.

For the SoI, functions on this level are high-level generic functions. They will be broken down into more detailed sub-functions at the next levels of development, along with the system components. Thus, every component and actor has at least one function allocated to them at SA to represent the overall capabilities of the flap system. Flight Crew sends commands to move the flap surfaces, starting with selecting a flap position. The flap system is responsible for accepting the commands from the flight crew and operating the actuation according to these commands. It has sensors to monitor and control units to control itself. Air Data Computer feeds the flap system with flight information to inform whether the selected position falls under a suitable range for the flight phase. Otherwise, flaps can face structural damage due to too much load. EICAS provides the status and position information of the flaps for pilots to check.

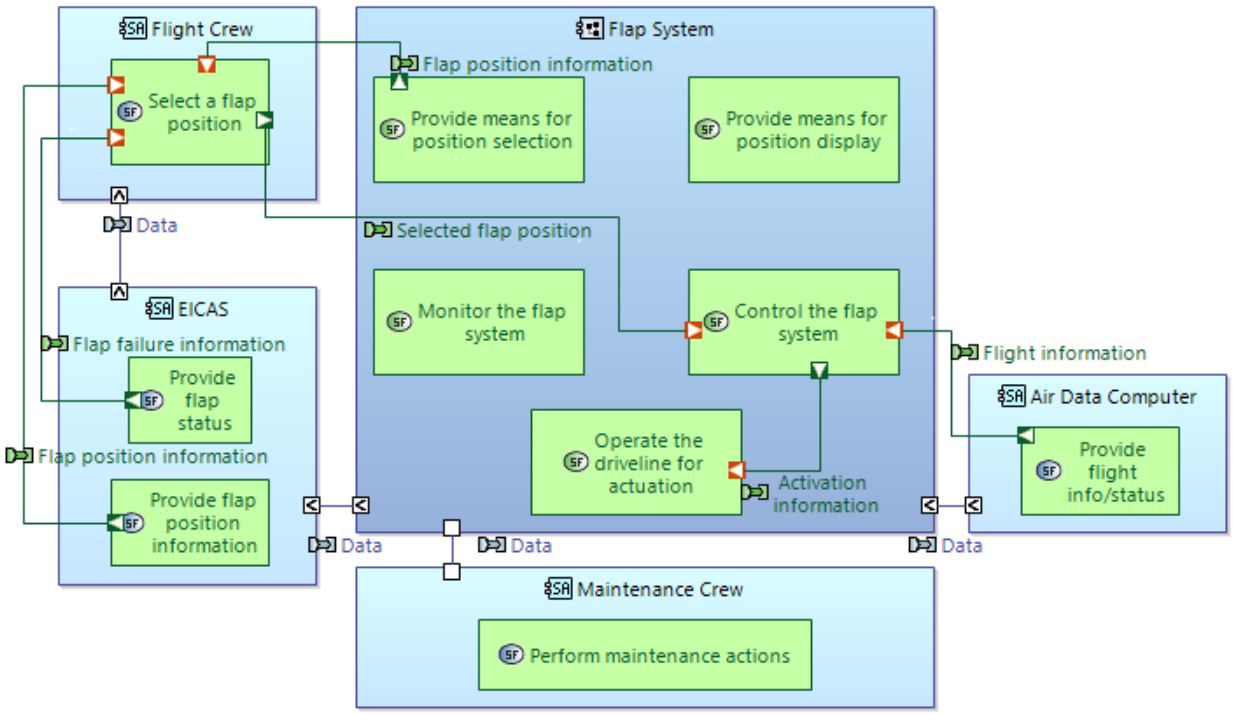

Figure 4.3 - System Architecture Diagram [SAB]

Functional exchanges represent the interactions between different functions. These interactions can be the flow of information, signals, or data [53]. The flight crew selects a flap position by getting all the information needed from different components. Before choosing a new flap position, they must check the EICAS for warnings and the flap lever. The flight crew's *select a flap position* function receives *flap position information*, and *flap failure information* exchanges while it sends selected *flap position information* by the pilots to the flap system. Another example of functional exchanges is the *control the flap system* function in the flap system, in Figure 4.3. *Flight information* and *selected flap position* are needed to start controlling the system. An activation signal/information is expected from the *control the flap system* function to operate the driveline and actuation.

Component exchanges in SA are utilized to represent the flow of data, signals, energy, etc., between the components. Hence, the component exchanges in Figure 4.3 show data interactions between components and their direction of flows (indicated by arrows). If an interaction between two components is both ways, the component exchange ports have no arrows inside them, e.g., *Flap System* and *Maintenance Crew* component exchange. Facilitating component exchanges at the SA level is done by providing a visual representation of the interaction patterns within a system. The next stages of the development present component exchanges edited by PVMT to address failure exchanges.

Since the architectural diagrams at SA must represent the high-level capabilities of the system, the flap lever has not been introduced yet. However, the flap system should capture the functionality of the flap lever since it is the component creating the interface between the flight crew and the SoI. Provide means for position selection function in flap system describes the interaction. Thus,

it is defined and allocated at SA. In later stages of the development, it can be allocated to the lever once it is introduced.

4.2.2 Logical Architecture – L0

The level of detail and key activities in this level of system architecture development are explained in Chapter 3. While the objective of SA is defining what the flap system has to accomplish for the stakeholders and entities (actors in Capella), LA aims to unfold how the system has to perform to fulfill the expectations defined at SA by defining subsystems/components and their functions [53], [91]. L0 has its components, actors, and functions transitioned from SA with the help of Capella's automated transition, meaning that the system elements at the SA level become logical elements at the LA level. In addition to SA, high-level logical subsystems are introduced at L0.

Figure 4.4 depicts the architectural view of L0. The high-level logical subsystems defined at this level are the flap lever, monitoring system, control system, actuation system, LHS (left-hand side) flap, and RHS (right-hand side) flap. The distinction between LHS and RHS flaps aligns with the progressive nature of the aircraft development process, where early stages focus on defining subsystems before specifying detailed components, such as the number of panels.

The subsystems mentioned in the previous paragraph are created inside the flap system logical box based on their critical roles in the functionality of the flap system. The flap lever is essential as it serves as the primary interface for pilot input. The monitoring system provides feedback on the flap system's status. The control system is responsible for processing commands within the system. The actuation system translates control commands into the mechanical movement of the flaps. By defining these subsystems at L0, a structured foundation is established that can host more subsystems and components for further development in subsequent logical levels.

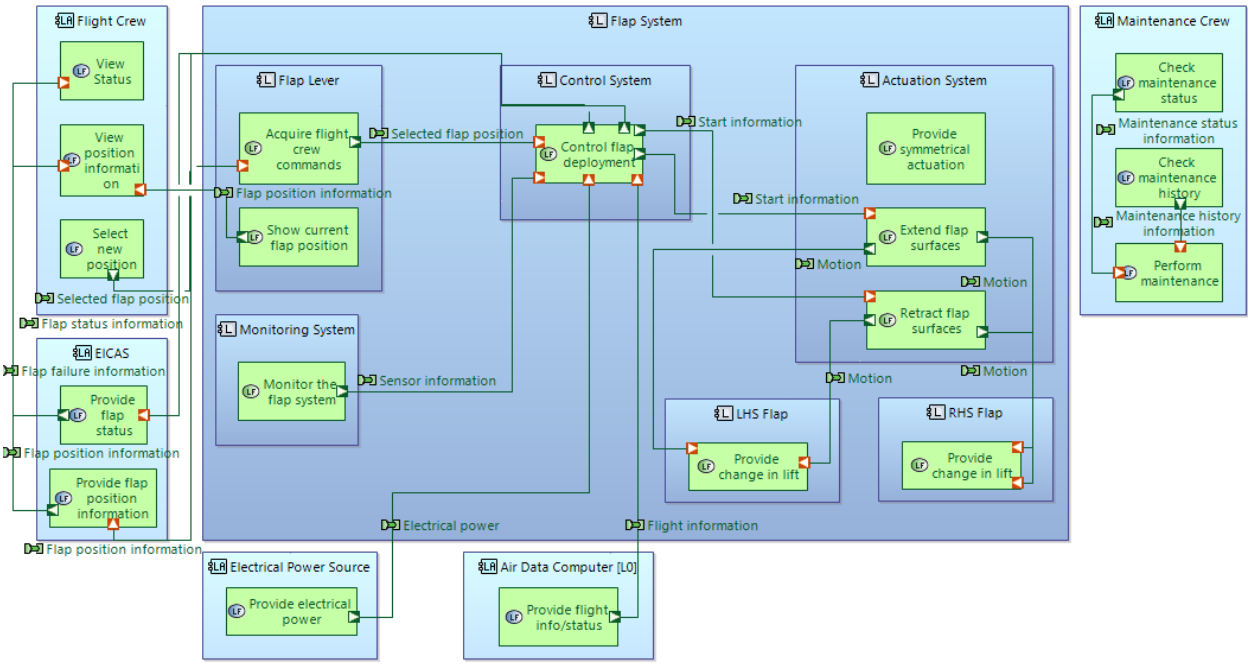

Figure 4.4 - Logical Architecture Diagram [LAB] of L0

Also, a new actor, an electrical power source, is introduced. The newly defined logical subsystems and some of their functions will face a breakdown in the following stages. The intention is here to reflect the gradual improvement of the flap system discussed in Chapter 3. In creating the logical subsystem, the engineers already have an understanding of what possible components would be appointed to which logical subsystems in the next steps of development. Therefore, the allocated functions need to capture the functionalities of those components, which will be associated with the logical sub-system in later stages. For example, the *Monitoring System* in Figure 4.4 will involve sensors late in development; the *Monitor flap system* function should capture the functionalities of sensors.

Regarding functions, engineers must allocate the system functions defined at SA to the logical subsystems at L0. The first step for functions is using the automated transition of system functions in Capella to convert them into logical functions. Next, some of these functions can be converted to parent functions, meaning they have subfunctions. The last step is the allocation of these subfunctions to the related systems. Table 4.4 lists the transition of functions between SA and L0 and their allocations to logical systems. Some functions are kept the same since no detail about their corresponding system exists yet at this level such as the monitoring system.

Table 4.4 - Functional Transitions from SA to L0

System Analysis	**Logical Architecture – L0**
Flight Crew Actor	
Select a flap position	Select new position View status* View position information*
EICAS Actor	
Provide flap status Provide flap position information	Provide flap status Provide flap position information
Air Data Computer Actor	
Provide flight info/status	Provide flight info/status
Maintenance Crew Actor	
Perform maintenance actions	Perform maintenance Check maintenance status* Check maintenance history*
Flap System	**Subsystems of the Flap System**
Provide means for position selection Provide means for position display	➢ *Allocated to **Flap Lever*** Acquire flight crew commands* Show current flap position*
Monitor the flap system	➢ *Allocated to **Monitoring System*** Monitor the flap system
Control the flap system	➢ *Allocated to **Control System*** Control flap deployment*
Operate the driveline and actuation	➢ *Allocated to **Actuation System*** Provide symmetrical actuation* Extend flap surfaces* Retract flap surfaces*

**indicates the function is created at L0*

Figure 4.5 depicts the *Control flap deployment* function allocated to the *Control System* logical system at L0 as a parent function and the functions at L1, which are allocated to *Control Unit 1* and *Control Unit 2* as subfunctions of the *Control flap deployment* function. Since *Control Unit 1* and *Control Unit 2* have the same functionalities, they share subfunctions with the same names, except the *Start DC Motor* 1 and *Start DC Motor* 2 functions. This is because *Control Unit 1* activates *DC Motor 1,* and *Control Unit 2* activates *DC Motor 2*. Functional Breakdown diagrams can help engineers understand the hierarchy of functions. Since the LA phase with three levels has the most number of functions, a functional breakdown diagram is used in the LA. Although component and function breakdown diagrams support users to see the hierarchy of system elements, architecture diagrams are the most informative. Architecture diagrams can also be used to see hierarchy information. For example, the *Control System* has *Control Unit 1* and *Control Unit 2* as subsystems in Figure 4.6, showing a component hierarchy. Likewise, the function allocated to the *Control System,* which is *control flap deployment,* has subfunctions distributed to these subsystems*: Control Unit 1* and *Control Unit 2*. Therefore, this thesis puts emphasis on the architectural diagrams, which give the most information about the system and facilitate safety viewpoints.

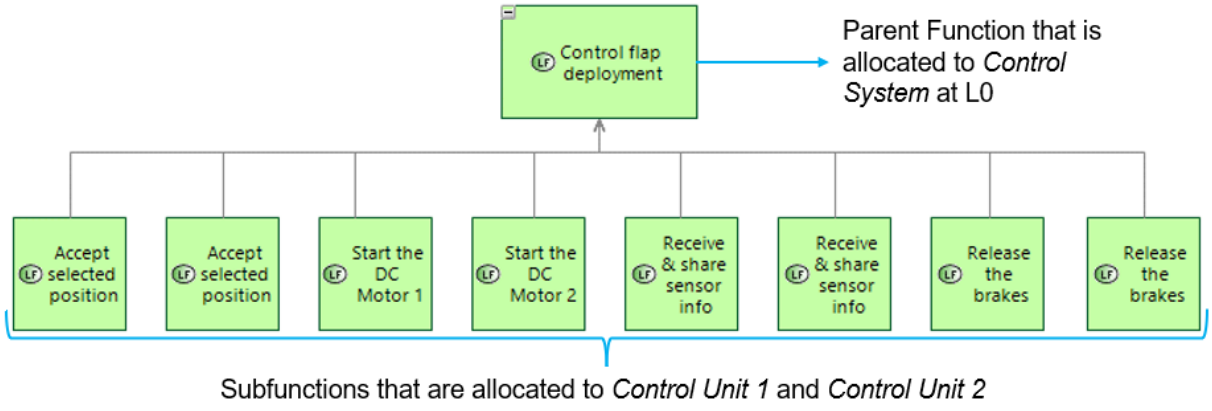

Figure 4.5 - Control Flap Deployment Function in the Logical Functional Breakdown Diagram

Moving on to functional exchanges, they do not change from SA to L0 if they have the same level of detail on their functions that can be seen from EICAS. However, if the function is broken down into several subfunctions, the subfunctions inherit the functional exchanges from their parent function. *Select a flap position* function allocated to the *Flight Crew* actor at SA in Figure 4.3 has both incoming and outgoing exchanges; at L0, in Figure 4.4, the exchanges are allocated to their new subfunctions: *View status*, *View position information*, and *Select new position.* New functional exchanges are defined with the introduction of new logical functions. These exchanges are expected to have more details in the following stages of the development. This is due to the refinement of the functions in different phases of the aircraft development. For instance, EICAS has more detailed functions at L1. Therefore, the functional exchanges between the Flight Crew and EICAS functions also become more detailed.

4.2.3 Logical Architecture – L1

At this stage of the development, high-level redundancy information, new power sources, and system technologies are introduced. Including the electrical power sources is vital since they power the most critical components in the flap system, the PDU and flap control units. They are defined

as actors and kept generic because the thesis focuses solely on the flap system. Also, introducing more details to other entities would make the safety model more intricate. The redundancy information included aims to design a safe system by preventing a failure of a subsystem, resulting in a failure of the whole system and, ultimately, the aircraft.

Figure 4.6 displays the L1 logical architecture with all Capella elements included, while Figure 4.7 represents the L1 without actors. In both figures, most of the functional exchanges are hidden to provide a clear view of the architectures. The refinement process of the functions, functional exchanges, and components is identical to the previous levels and continues at this level before it takes its final shape in the next stage. There are two control units appointed under the control system. The *Actuation System* at L1 houses a *PDU* consisting of two *DC motors* and a *speed-summing gear*. *Control Unit 1* drives the *DC Motor 1*, and *Control Unit 2* drives the *DC Motor 2*. In case of a control unit failure, the flap system does not fail. The system works, but the deployment takes place at half-speed due to only one operating DC Motor. This condition is possible with a speed-summing gear that sums the motion received from both motors. The *transmit motion to both wings* function is associated with the *PDU*, but no logical components have been created for it yet. It is because the component that should inherit the *transmit motion to both wings* function must connect to the components in the driveline, but there is not enough detail to capture that yet. Therefore, the function is defined and allocated to the *PDU*. Later in the development, a branch gear should be placed inside the *PDU*, and the function must be allocated to that gear.

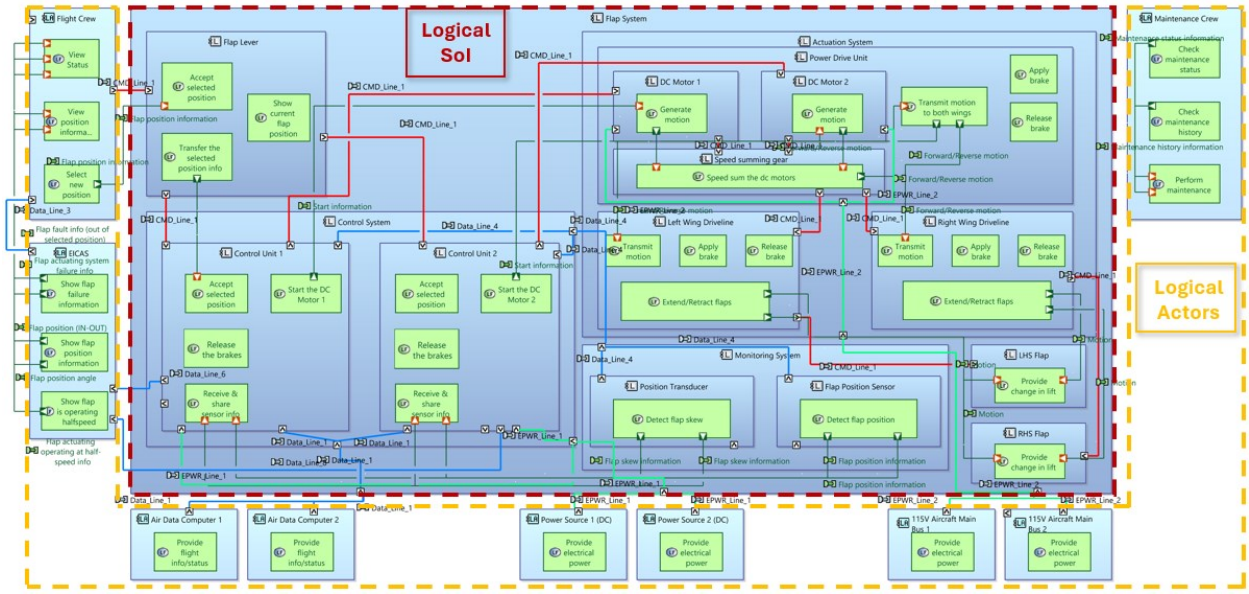

Figure 4.6 - Logical Architecture Diagram [LAB] of L1

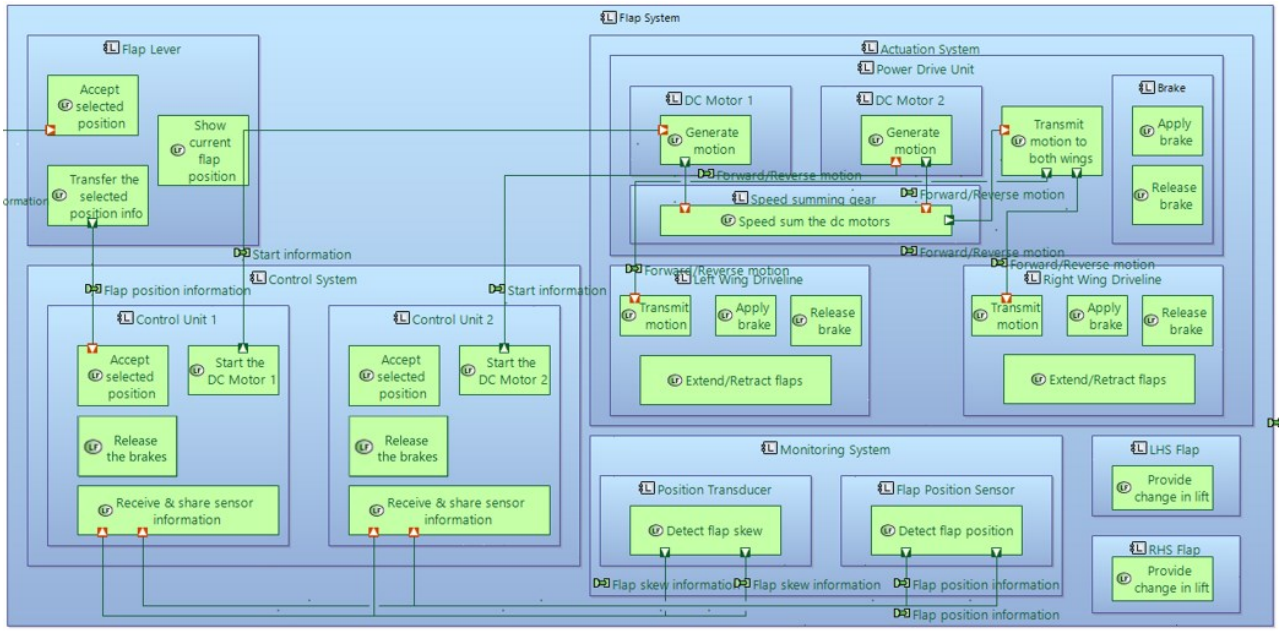

Figure 4.7 - Logical Architecture Diagram [LAB] of L1 – Simple View

Left and right-wing drivelines are placed between the PDU and the flaps. These drivelines represent every component between the PDU and flaps. The main functionalities of the driveline and actuation are allocated to these components. The differentiation between the right and left wings is to capture the symmetric/asymmetric deployment cases and to reflect a layout of the system with clarity. The *Monitoring System* now has *Position Transducers* and *Flap Position Sensors*. The *Position Transducers* convert physical displacement into electrical signals, providing a measurement of flap positions to detect skewing. Skewing means the unequal movement of the flaps, where one side moves differently than the other, leading to asymmetry in the flap positions. The *Position Transducers* detect such skewing by comparing the positions of the flaps on both sides. The *Flap Position Sensors*, on the other hand, directly measure the position of the flaps and provide feedback for the control system. However, the architectural layout still needs to be completed where these sensors are placed. Therefore, the number of sensors still needs to be determined.

Like functional transition Table 4.4, high-level functions appointed to the logical subsystems at L0 cannot be seen in the L1-level diagrams. Instead, the diagrams display subfunctions of the functions at L0. For example, the *control flap deployment* function associated with the control system at L0 is satisfied with the combination of eight subfunctions allocated to control units under the control system at L1. In addition, these subfunctions can host sub-logical functions themselves in the next stage, which is L2. Realizations (connecting system elements at different levels by selecting the corresponding parent element for the sub-element in the Capella environment) and utilizing the parent functions are key factors for enabling traceability.

Like functions, functional exchanges are refined through the L0, L1, and L2 levels. Figure 4.8, as an example of the refinements, shows that the functional exchanges of *EICAS* are now more numerous, and they specify what kind of flap information is available to display.

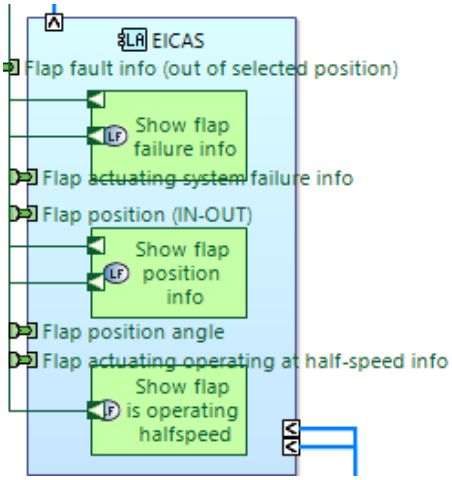

Figure 4.8 - Logical Architecture L1 EICAS and Its Functions & Functional Exchanges

4.2.4 Logical Architecture – L2

Refer to Appendix A to see a readable version of the L2 architectural diagram. The layout of the logical components is presented in L2, as seen in Figure 4.9. This includes components' location, connections, and any physical constraints that impact their placement. For example, the PDU is placed between the two wings. Also, every logical component in the system is fully specified with detailed descriptions of its functionality, internal structure, and any dependencies it may have on other components at this stage. L2 is a reference for the flap system's detailed design and implementation phases. The information at L2 guides engineers in translating the logical architecture into a physical realization.

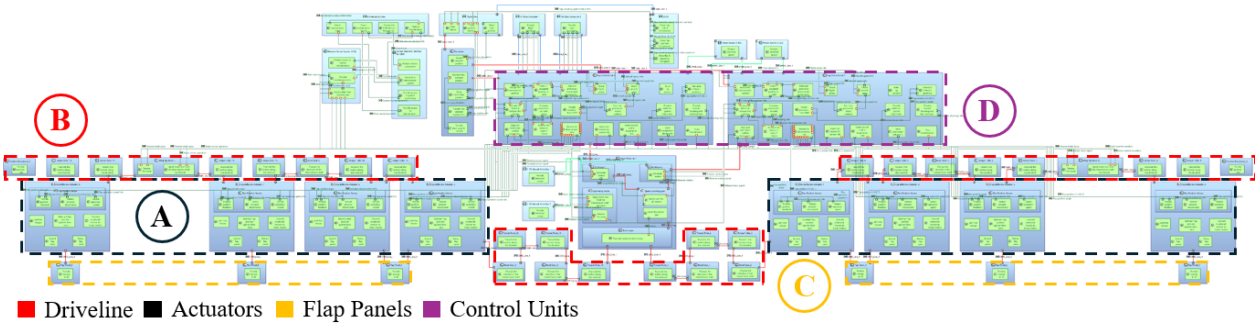

Figure 4.9 - Logical Architecture Diagram [LAB] of L2

L2 provides detailed information on the types of components required for the flap system. This includes specifying the technology and functionality of each component, such as the type of motors, sensors, and control units that must be used. PA is the level where the solutions are outsourced to supplier companies at the item level. For example, actuator types are defined as ball-screw actuators at L2 with certain functionalities. At PA, supplier companies must define the solutions to capture the functions allocated to the actuators.

The driveline at L2 has logical components; contrary to L1, it is not represented with a logical system. Left and right-wing drivelines from L1 become torque tubes, bevel gears, actuators, and wing-tip brakes. The functions *transmit motion*, *apply brake*, *release brake*, and *extend/retract flaps* from L1 are allocated to these logical components in L2, as seen in Figure 4.10.

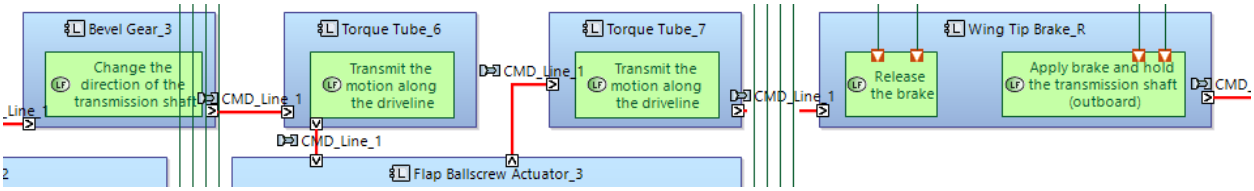

Figure 4.10 - Right-Hand Side of the Outboard Driveline at L2

As an example, flap ball-screw actuators are introduced at this stage. Their functionalities are represented by the driveline logical subsystems at L1. Now, *extend/retract flaps* functions are allocated to the actuators, and new functions are defined inside the *Flap Ballscrew Actuator_1* and outside the *Flap Position Sensor* for safety purposes to reflect how a ball-screw actuator works. Later, at the physical architecture level, more components that are parts of the actuators are defined, and the functions at L2 are appointed to those components. It indicates that refinement still takes place going from L2 to physical architecture. However, the flap position sensors are placed into the actuators at this stage, as Figure 4.11 shows, since they are components that can potentially appear in some of the fault tree failure cases examined in the thesis. Also, the aircraft manufacturer can require component or item-level elements from the supplier companies.

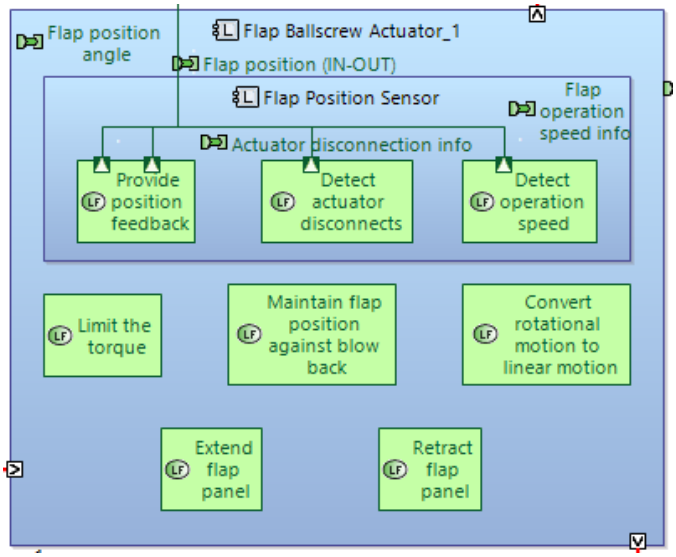

Figure 4.11 - Flap Ballscrew Actuator at L2

Apart from allocating existing functions to newly defined logical components, new functionalities can also be introduced at different levels of Capella. Since L2 has many detailed components, functions, and exchanges, it has introduced new functions. Flap lever and control units have more functions with more detailed functional exchanges at this level. Defining new functions at different logical levels occurs in the thesis for two reasons:

The first reason is that new lower-level functions often represent specialized tasks or detailed functionalities inherent to specific components. These functions provide a more detailed breakdown of the overall system behavior and address particular requirements at a finer level of detail. The second reason for introducing new functions at lower levels is to allow adaptability for changes in design. If alterations or enhancements are needed, introducing new functions at a lower level can be a practical way to address these changes without affecting the overall system

architecture. For example, if a new type of sensor needs to be integrated into the flap system for better performance, this can be achieved by adding a new function at the component level that handles the sensor data. This approach minimizes the impact on the higher-level system design and is important to this thesis as it demonstrates the flexibility of the methodology presented.

Following the same logic, new logical components can be introduced, and the functions allocated to logical systems or subsystems can now be allocated to these newly created components. Figure 4.12 shows that the logical components of the *Asymmetry brake* and *Branch gear* are defined and appointed to the *PDU* at L2. These logical components inherit *Transmit motion to both wings, Apply brake*, and *Release the brake* functions from the *PDU* at L1.

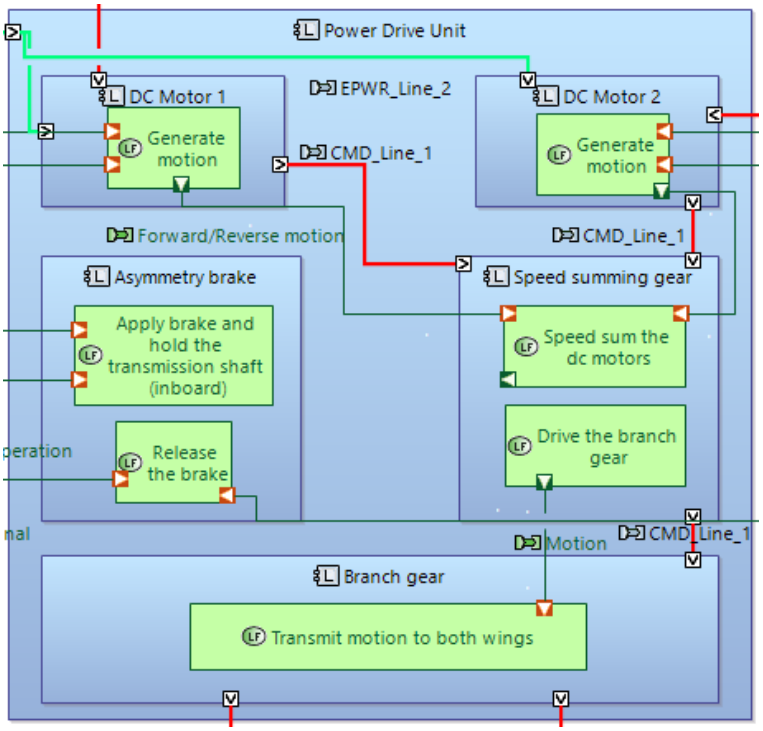

Figure 4.12 - Power Drive Unit at L2

4.2.5 Physical Architecture

Engineers define how the system is built and developed at the PA level. At this stage, software and hardware allocation, interface specification, and deployment configurations are described [53]. The primary activities that drive the development process at the PA level are identifying, defining, and decomposing physical components and subsystems.

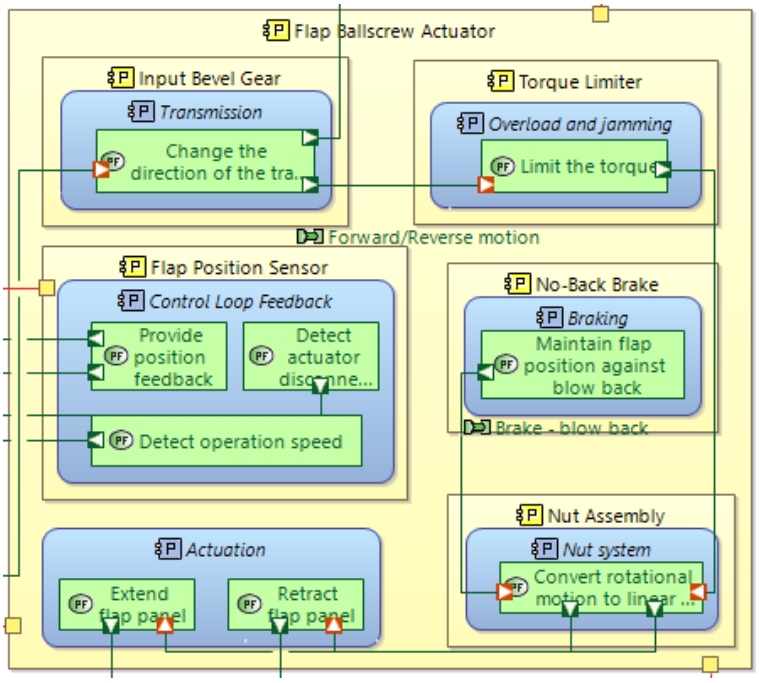

Figure 4.13 - Flap Ballscrew Actuator at PA

The only refinement designated at PA is, as Figure 4.13 illustrates, on the flap ball-screw actuators to show decomposition progress. This distinction is made between the two levels to illustrate the ongoing development process in aircraft systems, which often involves work conducted by suppliers. This thesis highlights how certain components, such as actuators, continue to evolve even at the PA stage by showing this additional detail. However, typically, aircraft manufacturers appoint these duties to their suppliers and use the PA to capture design decisions at the detailed level. After the decomposition is done to the equipment level, the suppliers run FMEA. However, the FMEA is out of the scope of this thesis. Therefore, the physical layer carries the same level of detail as the L2 except for the actuators. Similar to L2, PA has the layout of the system depicted, centered with the PDU, as Figure 4.14 shows.

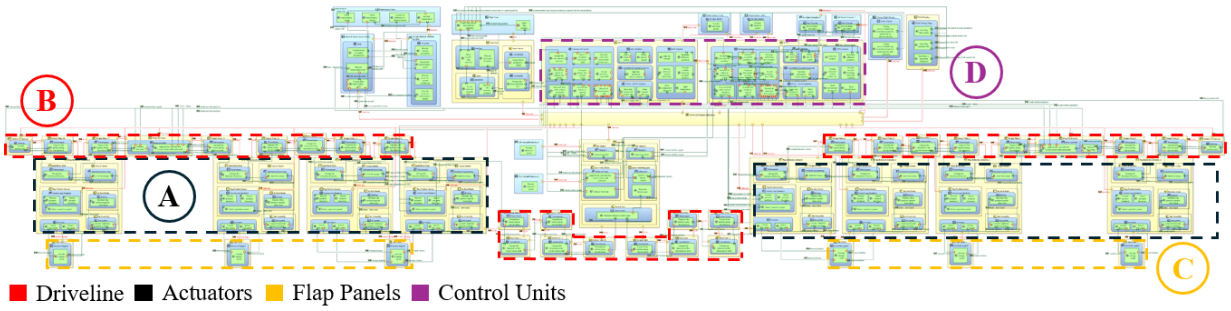

Figure 4.14 - Physical Architecture Diagram [PAB]

4.3 Enhancements in System Architecture Specification

Section 3.4 explains the safety artifacts that need to be integrated into the system model in detail. This section presents the enhanced system model for the flap system test case with the proposed methodologies. The safety artifacts embedded in the Capella system model are state variables, flow

variables, transfer functions, and failure rates. In addition, functional chains are created to capture the failure relationships across different system components. While the failure rates are integrated at L1 and L2 and functional chains at L0, the rest are embedded throughout all logical levels. Failure rates for different components are taken from [96] and [97]. The maintenance crew actor is left out from the safety perspectives at all levels because the failure scenarios inspected do not involve it.

Figure 4.15 shows the component exchanges edited with PVMT to represent flow variables at L0. The blue lines correspond to data lines, while the red ones are for command, and the green ones are for electrical power exchanges. The naming of exchanges is numbered to separate different command, power, and data lines. The test case involves one command line numbered with 1 at all levels. The component exchange ports are crucial since they are the means to represent the flow, and the coloring and naming show the type of flow. The flap system command line starts with the flight crew sending commands to the control system via the lever. Then, the actuation system transfers these commands to move the flaps. The control system lies in the center of the SoI, getting critical data from the air data computer, such as airspeed and altitude and power from the electrical power source and sending data to EICAS for display.

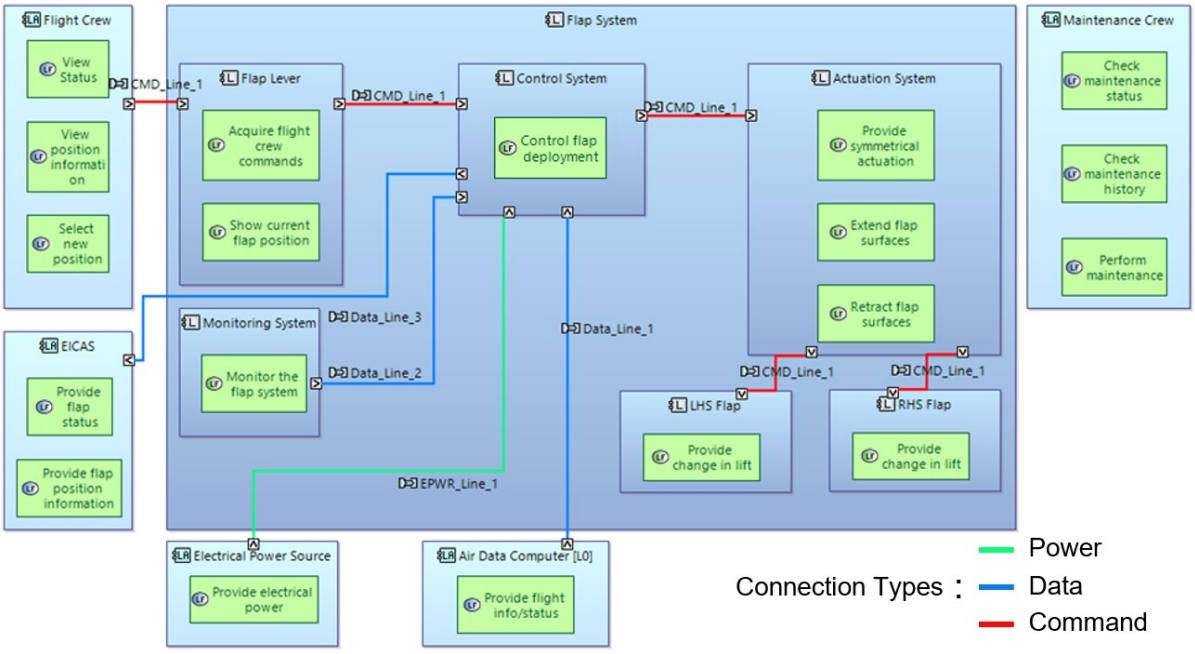

Figure 4.15 - Logical Architecture Diagram [LAB] of L0 – Safety Viewpoint

The *annunciated loss of flap extension* failure case is mapped with a functional chain in Figure 4.16. The top event function for this particular case is the *extend flap surfaces*, which is examined by focusing on the functional exchanges that present command, power, and data connections. The functions that have exchanges with the top event function are also examined similarly until there is no exchange to capture. This method provides to identify the potential cause-and-effect connections between the functions. Later, with the FTA, failure event combinations that are causing a failure condition are identified. Therefore, it stands as a bridge between FHA and FTA.

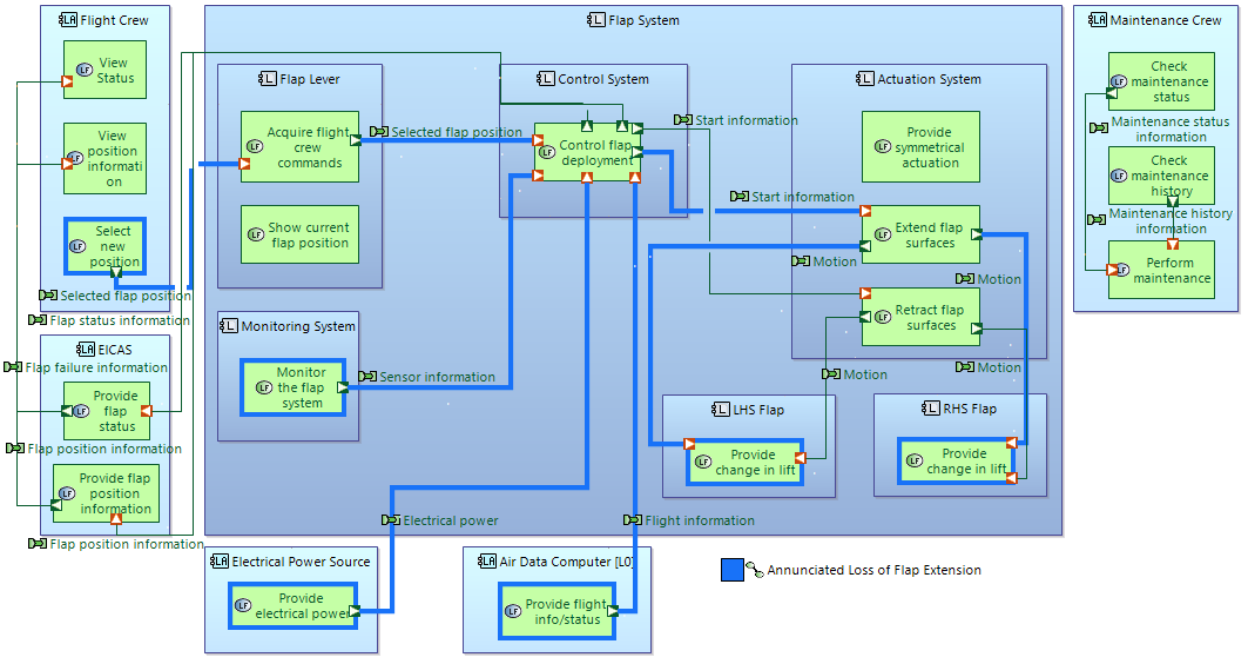

Figure 4.16 - Logical Architecture Diagram [LAB] of L0 with a Failure Chain

Figure 4.6 in Section 4.2.3 shows the L1 with most of the system and safety artifacts, while the presented safety viewpoints of the diagrams in this chapter exclude functional exchanges and ports for clarity. Figure 4.17 illustrates the edited component exchanges allocated to sublogical systems created at L1. At L0 control system has the most incoming and outgoing flow variables. Defining control units under the control system at L1, the flow variables are connected to control units. Comparing the command connection between the control and actuation systems at two levels, L0 and L1, the flow variables defined are between the DC motors and control units. It means that the pathways of the component exchanges at L0 stay the same at L1 and L2, but since new components are introduced, they need to be allocated to the new components under the high-level logical systems. Also, to capture the connections at L0, more component exchanges are created at L1 and L2.

The architecture is at its most detailed version at L2. Here, the PVMT component exchanges representing commands start from the flap lever and end with connecting flap panels with actuators. Failure rates are given to each component for quantitative analysis.

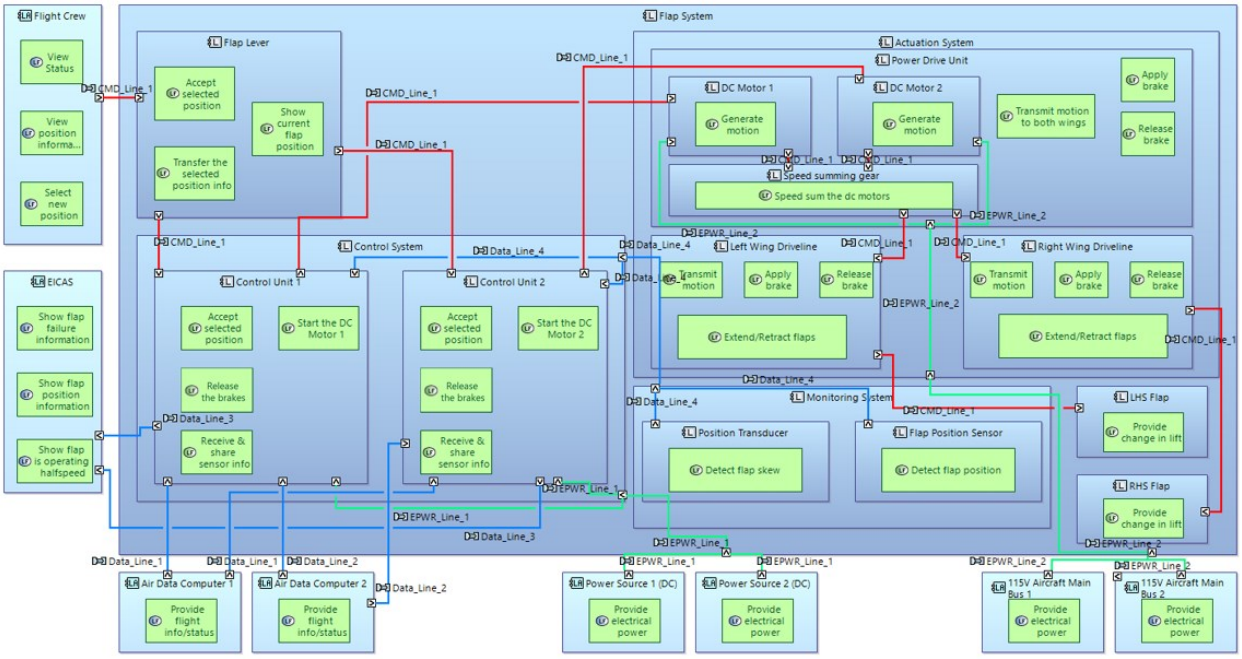

Figure 4.17 - Logical Architecture Diagram [LAB] of L1 - Safety Viewpoint

The same efforts are made to refine and allocate the edited component exchanges for the transfer functions throughout the logical levels. Transfer functions carry information on the component's behavior according to the state and flow variables. Since more detail is introduced at each logical level, the assertions must also be adjusted for the detail.

At L0, the change in the inputs and outputs of the actuation system is defined. The transfer functions for the logical components under the actuation system at L1 should match the information presented at the previous level. Therefore, if the incoming command exchanges are not lost and the state is nominal for the components, they should transfer the command to the drivelines at L1 since the behavior defined for the actuation system represents the same logic at L0.

Figure 4.18 shows the progression of safety properties added by PVMT for the *Control System* through the logical levels. Although no failure rate is available at the L0 level for the *Control System,* the magnitude of the failure rate can be embedded if needed. L1 and L2 levels involve failure rate information. However, the transfer functions of the *Control System* at L0 need to be transitioned to the *Control Units* according to the other components that interact with the *Control Units* at L1. At the L2 stage, every component with its redundancy information is present. Therefore, the transfer functions must be edited accordingly (e.g., L1 includes one position transducer and flap position sensor while L2 has the complete numbers).

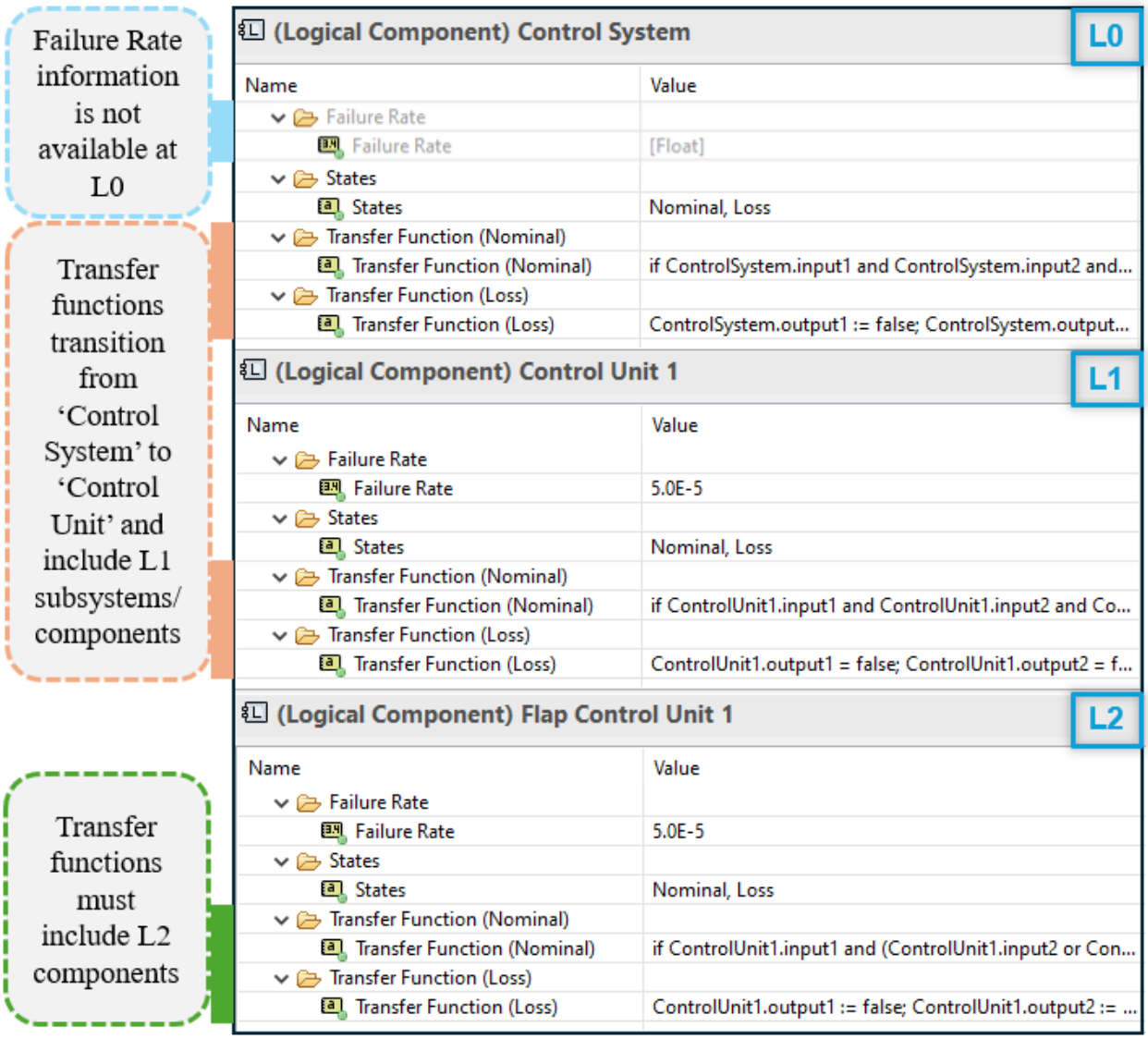

Figure 4.18 - Progression of Safety Artifacts for a Component through L0, L1, and L2

4.4 AltaRica Safety Model

AltaRica is a modeling language to build safety models for analyzing potential safety hazards within complex systems. The proposed method in this thesis involves systematical construction of safety models tailored to each logical level - L0, L1, and L2, meaning that the safety models are created independently using the Capella logical level diagrams as a reference.

To demonstrate the effects of the enhancements made to the Capella system model, this thesis formulates a test case to demonstrate the safety model's capabilities in the FTA context. Three scenarios are presented: the annunciated loss of flaps extension/retraction, unannunciated loss of flaps extension/retraction, and flap panel disconnection. These failure scenarios represent critical safety aspects within the system architecture.

Several assumptions are made during the development of the safety model to simplify the modeling process and focus on the practical analysis and representation of critical failure scenarios within the system. Below are the key assumptions:

- Assumption 1: Only failures causing loss of function are modeled; failures causing erroneous behavior of sensors, control units, and displays are not considered.
  Rationale: The available literature lacks specific information regarding failure modes and corresponding failure rates for individual components within the flap system. Therefore, modeling only the failures that lead to a loss of function prioritizes critical failures over those causing erroneous behavior of components.
- Assumption 2: Position transducers and flap position sensors are not used to close the loop on flap control
  Rationale: Building safety models that represent FPM with control loops may cause problems [89]. The typical construction of a fault tree is to follow input and output dependencies. If a control loop is attached to the system, with the typical fault tree logic, a circular logic appears. Solving circular equations is possible in numerous ways but is often left to the strategic choices of safety analysts [89]. Therefore, in this thesis, the sensors are not used to close the control loop.
- Assumption 3: Flaps are stopped, and flap fail annunciation is posted in case of loss of output from flap lever position sensors, detected by the flap control unit
  Rationale: This assumption aligns with the approach taken in assumption 2 to avoid circular logic and potential complications in safety modeling. This prevents the system from relying on sensor input to maintain control, which could otherwise lead to circular dependencies and complex fault tree logic. The control unit can safely halt operations without the need for continuous feedback from position sensors, thus simplifying the fault tree analysis and avoiding potential modeling issues associated with circular logic.
- Assumption 4: If both flap control units fail, the flap surfaces are stopped due to command loss to motors and brake release command. Flap fail annunciation is posted due to loss of communication detected by EICAS.
  Rationale: If both flap control units were to fail simultaneously, it would result in a loss of command signals to the motors responsible for controlling the flap surfaces. Additionally, the assumption is that a failure of both control units would lead to a loss of communication, as detected by the EICAS. In such a scenario, it is assumed that the flap surfaces would be stopped to prevent unintended movement.
- Assumption 5: Regarding the flap panel disconnection failure case, the flap panel's failure rate is set to a value that reflects an aggregate of potential failure modes and rates, incorporating elements such as hinges and connection arms, which are not explicitly modeled but are components affecting the flap panel's failure behavior.
  Rationale: The scope of the system model does not capture all components necessary to calculate the failure scenario. Therefore, an average failure rate with an order of magnitude is given to each flap panel. Consequently, each minimal cut set corresponds to the failure of a specific flap panel
- Assumption 6: Only ‘AND’ and ‘OR’ gates are used in the fault tree analysis.
  Rationale: This decision is based on the need for simplicity in modeling the failure logic. While there are other types of gates (e.g., XOR, NOT, NAND), they might introduce additional complexity that is not necessary for the scope of this study. ‘AND’ and ‘OR’ gates are sufficient to capture the primary failure modes and their interactions.

4.4.1 Elements of Safety Model

This section of the thesis explains AltaRica safety model elements and how they are mapped to FTA. Refer to Appendix D to see classes and blocks of different levels of AltaRica safety models. The use cases for AltaRica elements, observers, classes, blocks, and assertions and their refinement throughout the development process are shown.

Once the identification of the top event for the FTA is complete, which in this case is the three different failure scenarios presented, the next task is to establish the intermediate events forming the pillars of the fault trees. The intermediate events are the indicators of potential failure conditions that contribute to the occurrence of the top event. Depending on the level of detail available in the system model (L0, L1, L2), the intermediate events may vary in complexity and granularity. For instance, at the L0 level, where the system model provides a broader overview, the intermediate events may represent general failure categories such as loss of flap components with specifying left or right-hand side distinctions. However, as safety engineers examine the levels where the modeling ascends to more detail, such as L2, the intermediate events must be the loss of each flap instead of left and right-hand side distinctions.

The logical framework described with the selection of intermediate events lays the foundation for FTAs. The next essential step is translating this logical framework into a representation within the AltaRica safety model. The translation occurs with the help of observers. The observers monitor the system's behavior and capture the occurrence of specific events or conditions defined within the logical framework. Observers function similarly to flow variables, albeit with some distinctions. While observers cannot be used in transitions and assertions to describe the system's behavior, they serve as quantities to be observed. They are updated after each system action or event, providing feedback on the system's state and facilitating the dynamic monitoring of critical conditions [79].

The top events for FTA are specified via the observers, allowing for identifying critical failure scenarios within the system. Additionally, several observers can be defined for the same AltaRica safety model, enabling the generation of multiple fault trees from a single model [82]. This flexibility ensures that various failure scenarios and their associated top events can be analyzed within the same safety model.

Furthermore, in alignment with the flexibility offered by observers, the thesis has developed three distinct safety models, each corresponding to a different system level: L0, L1, and L2. Within each safety model, multiple observers have been defined to capture failure scenarios and their associated intermediate events. Observers listed in Table 4.5 highlight the refinement across different levels for the *annunciated loss of flap extension* failure scenario. In the L0 and L1 safety models, the elements LHSFlap and RHSFlap represent the left-hand and right-hand side flaps, respectively. Conversely, in the L2 safety model, observers from FlapPanel1 to FlapPanel6 are showcased, indicating a higher level of detail in both the system and safety models.

Table 4.5 - 'Annunciated Loss of Flap Extension' Failure Case Observers at L0, L1, L2

observer Boolean AnnunciatedLossOfFlapExtension =		
L0	**L1**	**L2**
EICAS.output and (not LHSFlap.output or not RHSFlap.output);	EICAS.output and (not LHSFlap.output or not RHSFlap.output);	EICAS.output and (not FlapPanel1.output or not FlapPanel2.output or not FlapPanel3.output or not FlapPanel4.output or not FlapPanel5.output or not FlapPanel6.output);

A single block named flap system in Figures 4.19 and 4.20 represents the SoI. Within this block, various classes are instantiated to capture the components displayed in the Capella system model. Different classes represent distinct components in the system, and each class is dedicated to defining its respective component's unique attributes. For instance, their shared characteristics and functionalities drive the decision to instantiate power sources under the same classes. Power sources, whether powering the PDU or the control units, exhibit consistent attributes across different instances, including identical inputs and outputs, uniform operational states, and the same failure rates. The model consolidates these shared attributes within a single class. Compared to having numerous blocks, this approach facilitates more straightforward modification and updates, as changes to the class properties propagate uniformly across all components that fall under the same class.

```
block FlapSystem
        StartingPoint FlightCrew;
        Lever FlapLever;
        EPWER ElectricalPowerSource;
        Control ControlSystem;
        Actuation ActuationSystem;
        Flaps LHSFlap, RHSFlap;
        Sources AirDataComputer, MonitoringSystem;
        Endings EICAS;
```

Figure 4.19 - Components Initiated from Classes under the Flap System block at L0

Figures 4.19 and 4.20 outline the instantiation of various components within the safety model. Each component corresponds to specific elements observed within the system model developed in Capella. The elements are named the same between the safety and system models. The instantiation of components occurs by placing component names after class names (e.g., *Control* for the class name and *ControlUnit1* and *ControlUnit2* for the component names in Figure 4.20). Also, it is expected to facilitate more components as the logical levels progress because subsystems transition to components, and this refinement can be seen by examining Figures 4.19 and 4.20.

```
block FlapSystem
        StartingPoint FlightCrew;
        Lever FlapLever;
        PowerSource DCPowerSource1, DCPowerSource2, mainBUS1, mainBUS2;
        Control ControlUnit1, ControlUnit2;
        Motors DCMotor1, DCMotor2;
        Data AirDataComputer1, AirDataComputer2, PositionTransducer,
FlapPositionSensor;
        Endings EICAS;
        Drivelines LeftWingDriveline, RightWingDriveline;
        Gear SpeedSumGear;
        Flaps LHSFlap, RHSFlap;
```

Figure 4.20 - Components Initiated from Classes under the Flap System block at L1

In the assertion section in Figure 4.21, the information on connections between various components is transferred from Capella to AltaRica. The figure outlines assertions that dictate how components are interconnected. These connections ensure that the system functions as intended, with data or signals flowing appropriately between components to fulfill the system's operational objectives. While classes carry information on the types of inputs and outputs, assertions can give insights into where certain connections begin and end. For instance, the pathway of the command type flow at L0 is below, and this pathway can be indicated in Figure 4.21 by following the inputs and outputs of components.

Pathway of 'command' at L0: Flight Crew > Flap Lever > Control System > Actuation System > LHSFlap and RHSFlap

```
assertion
        FlapLever.input := FlightCrew.output;
        ControlSystem.input1 := FlapLever.output;
        ControlSystem.input2 := ElectricalPowerSource.output1;
        ControlSystem.input3 := AirDataComputer.output;
        ControlSystem.input4 := MonitoringSystem.output;
        EICAS.input := ControlSystem.output2;
        ActuationSystem.input1 := ControlSystem.output1;
        ActuationSystem.input2 := ElectricalPowerSource.output2;
        LHSFlap.input := ActuationSystem.output1;
        RHSFlap.input := ActuationSystem.output2;
```

Figure 4.21 - Block Assertions at L0

4.4.2 FTA Results

This section summarises the output of safety models and FTA results in Figures 4.22, 4.23, and 4.24. The elements displayed in the fault tree figures are explained in section 3.2.1. All the figures in this section show fault trees for the *annunciated loss of flap extension* failure case. Fault tree diagrams for other specified failure scenarios are available in Appendix E.

At level L0 of the safety model, conducting both quantitative and qualitative FTAs is possible. Firstly, at level L0, the system model provides a broad overview of the system architecture,

capturing high-level functionalities and their interfaces. Due to this level's lack of detail for the system components, obtaining precise quantitative data such as failure rates and probabilities may be challenging. That is why the rates are on the orders of magnitude level at L0. Qualitative FTA, however, allows for a preliminary exploration of potential failure scenarios without the need for detailed quantitative information. Furthermore, with qualitative assessments, engineers can assess the overall system safety and prioritize critical areas for further analysis and refinement. Therefore, the qualitative approach guides the development and refinement of the system design for the following stages.

Figure 4.22 depicts the fault tree of the *Annunciated Loss of Flap Extension* failure scenario and the activities done at the L0 stage. Here, the qualitative analysis focuses on identifying possible events that can lead to a system failure, defining gates, and eventually constructing a fault tree with the basic event and gate information. Figure 4.22 shows that the *Annunciated Loss of Flap Extension* top event can only occur if one of the flap panels fails to execute their functions while EICAS is working (e.g., unannunciated loss of flap extension/retraction occurs if one of the flap panels fails to execute their functions while EICAS is not working). Defining the "AND" and other gates depicted in Figure 4.22 is a part of qualitative analysis and constructing fault trees. The fault tree structure created at this stage is used in the following stages for the same failure cases.

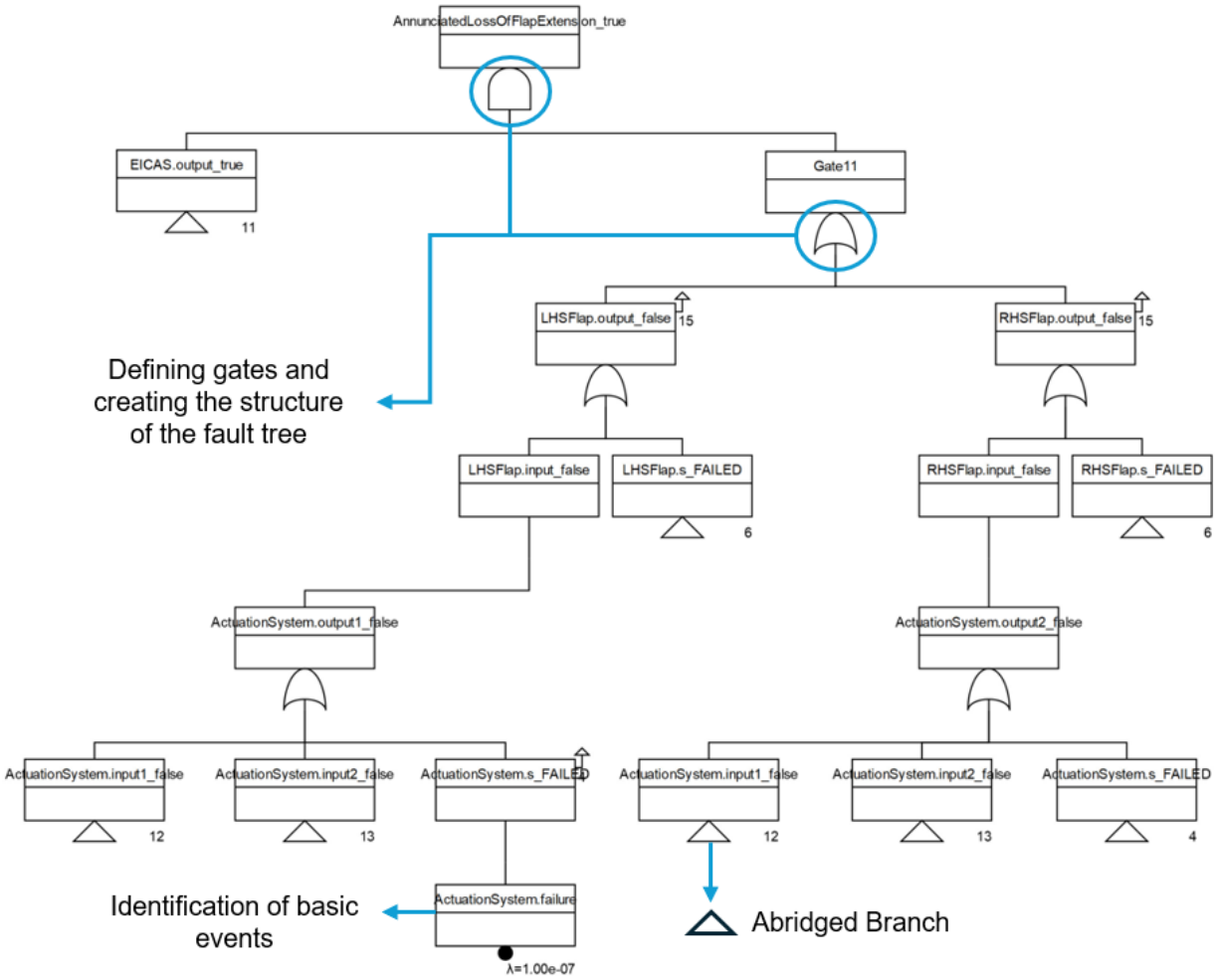

Figure 4.22 - Fault Tree of Annunciated Loss of Flap Extension at L0

Level L1 of the system model offers a more detailed representation of the system architecture than level L0. However, the structure at L1 is not yet in its final shape, with specific details still undergoing refinement. For instance, while the overall layout and types of sensors may be defined, specifics such as the exact number and placement of sensors remain uncertain at this stage.

Redundancy contributes by influencing the calculation of failure probabilities in quantitative analysis. Complete redundancy can significantly reduce the likelihood of the top event failure. The lack of complete redundancy information poses a challenge for conducting a full quantitative analysis at level L1. Therefore, instead of precise failure rates, orders of magnitude can be used at this level; the safety model can help to specify the required number of components and more accurate failure rates.

Leveraging quantitative analysis at level L1, even with incomplete redundancy information, can guide decisions regarding redundancy enhancements. While the quantitative analysis may not yield precise failure rates for top events, it can highlight areas of potential vulnerability and indicate where additional redundancy measures may be warranted. Also, the level of detail in system architectures is not uniformly distributed. The system model development might be close to complete for certain points of the architecture, while the other sections are still in early development. Hence, engineers can run quantitative analyses on the sections where enough information is stored at L1. Therefore, a hybrid approach is adopted, wherein qualitative analysis is complemented by quantitative elements tailored to the available level of detail.

Figure 4.23 shows that the calculated probability of the top event is higher than the expected probability. This means that the architecture is not safe, and redundancy must be increased in the critical areas of the architecture. The *contribution* indicates how much each minimal cutset contributes to the overall probability of the top event occurring. For the *LHSFlap*, *RHSFlap*, *LeftWingDriveline*, and *RightWingDriveline* events, the contribution is high. This is expected at the levels of L0 and L1 since the redundancy information for the architecture is not finalized. Each flap panel and driveline element must be introduced to lower the contributions of these events and lower the calculated probability of the top event.

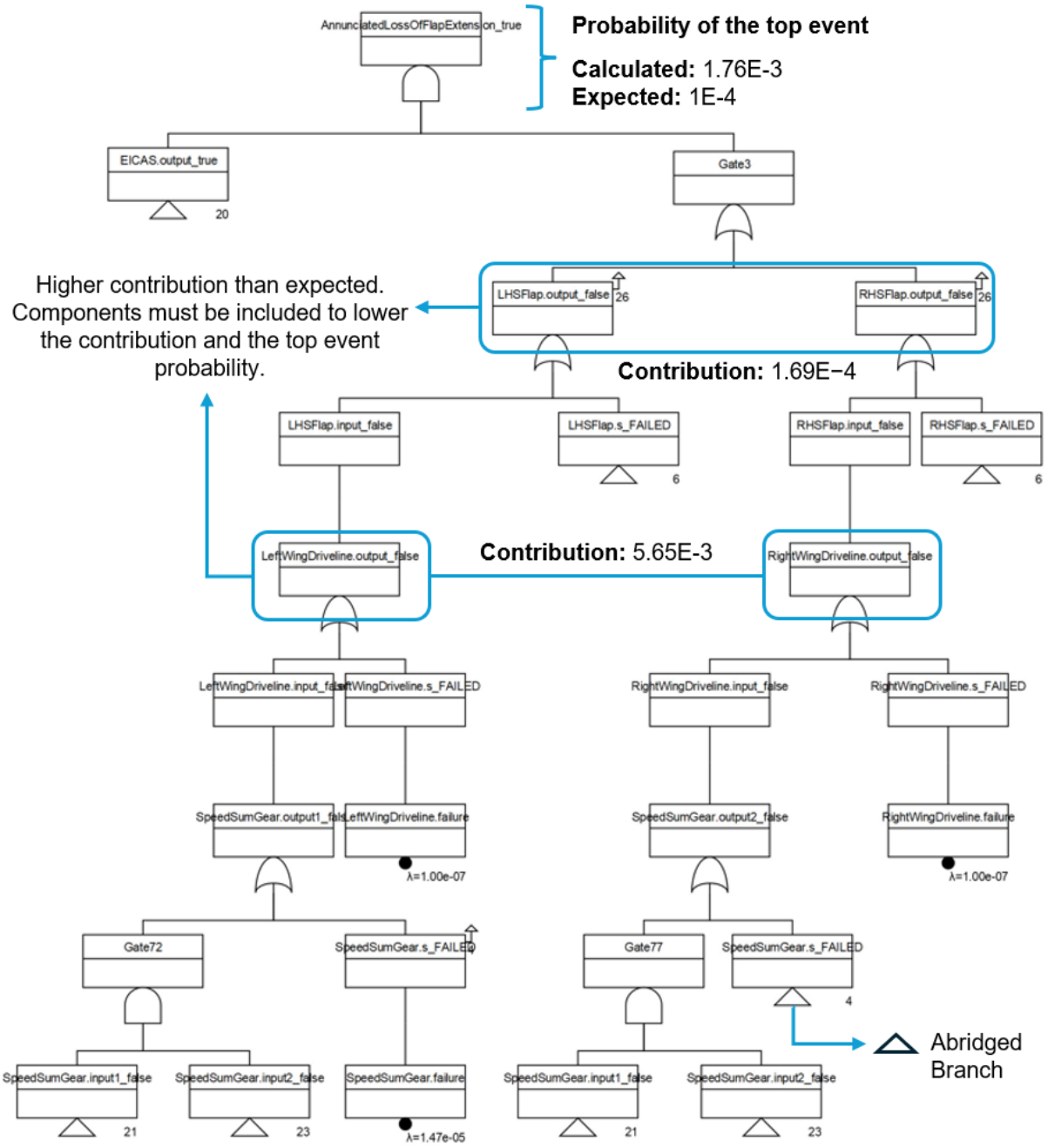

Figure 4.23 - Fault Tree of Annunciated Loss of Flap Extension at L1

The system model at L2 has all relevant details and specifications, including complete information on the parameters necessary for quantitative analysis, redundancy, dependencies, system configurations, and component failure rates. Therefore, the emphasis switches to quantitative analyses. Figure 4.24 shows after introducing each flap panel and driveline element, their contributions are lowered. As a result, the calculated probability of the top event indicates that the system architecture is considered to be safe for this particular failure scenario.

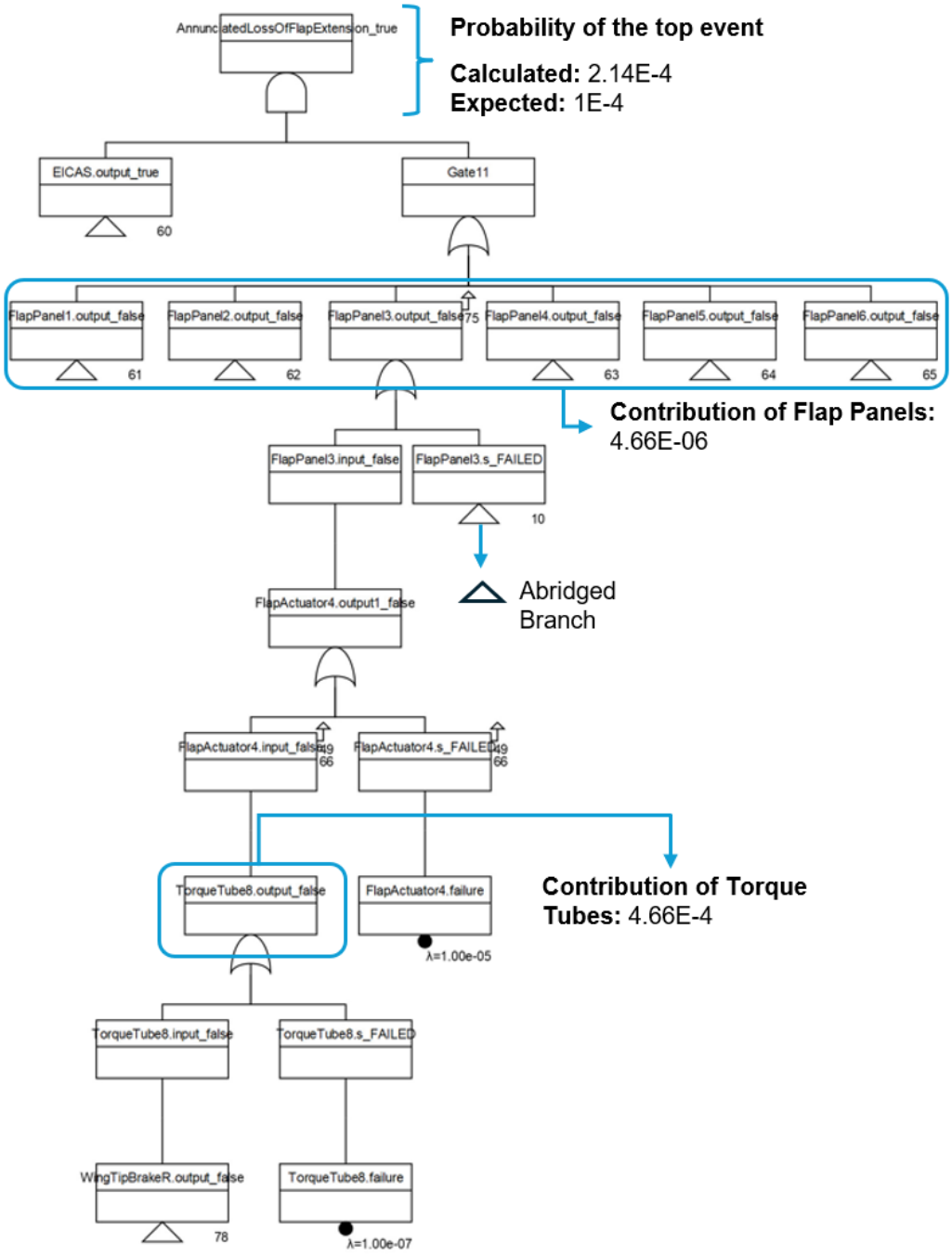

Figure 4.24 - Fault Tree of Annunciated Loss of Flap Extension at L2

By introducing the new flap panel event redundancies (*FlapPanel1.failure* to *FlapPanel6.failure*) with contributions lower than the original *LHSFlap.failure* and *RHSFlap.failure*, the top event probability is expected to drop. This is because the system now requires multiple independent failures to occur simultaneously for the top event to happen, which is less likely than a single failure of *LHSFlap.failure* or *RHSFlap.failure*. The same logic applies to introducing torque tube and bevel gear events for *LeftWingDriveline.failure* and *RightWingDriveline.failure*.

Table 4.6 shows the results of quantitative analyses done at L2 per flight hour. The number of minimal cutsets is consistent for the failure scenarios of annunciated loss of flaps extension/retraction and unannunciated loss of flaps extension/retraction, with both scenarios yielding 47. In the case of the annunciated failure scenario, the minimal cutsets represent failures directly causing loss of the flap extension/retraction. Conversely, for the unannunciated scenario, the minimal cutsets encompass failures in the flap extension/retraction mechanism and the EICAS display system. Considering the assumptions made, the results expected match the results obtained.

Table 4.6 - Quantitative Probabilities per Flight Hour and Minimal Cutsets for L2

Failure Scenario	Gravity	Quantitative Probability Expected (per FH)	Quantitative Probability Calculated (per FH)	Number of Minimal Cutsets
Unannunciated loss of flaps extension/retraction	Catastrophic	1E-9	2.14E-9	47
Annunciated loss of flaps extension/retraction	Minor	1E-4	2.14E-4	47
Flap panel disconnection	Catastrophic	1E-9	6E-9	6

4.5 Implementation

In this section, the implementation of the thesis' methodology is detailed using a specific example: the *flap ball-screw actuator 4*, located at the tip of the right-wing driveline. The implementation shown in Figure 4.25 comprises four steps. Each step represents a stage in the methodology process:

- Step 1: Capella Model
  The process begins with systems engineers constructing a Capella model. Figure 4.25 captures the entire L2-level logical architecture of the system. Within this model, *flap ball-screw actuator 4* is highlighted and shown separately. The figure depicts the command connections of the actuator between *Torque Tube 8* and *Flap Panel 3* by illustrating red-colored component exchanges that capture the flow variables of the actuator.
- Step 2: PVMT Integration
  PVMT integration is the second step of the implementation process, where the incorporation of safety artifacts into the flap ball-screw actuator 4 takes place. Both system engineers and safety analysts should work in the Capella environment for system development and tracing of safety properties. This step involves several integration points that help to build a safety model in AltaRica:

- Failure Rate: A generic flap actuator failure rate, 1.0E-5 per flight hour [97], is added as a safety artifact into the *flap ball-screw actuator 4* logical component of L2.
- State Variables Specification: Within PVMT, nominal and loss state variables for the actuator are entered.
- Transfer Functions: Each state variable is accompanied by a corresponding transfer function that details the components' behavior in response to varying inputs. Therefore, a transfer function for the nominal and another one for the loss state are embedded in the component.
- Flow Variables: Component exchanges edited by PVMT give information on flow variables with their direction, as explained in the previous step.

For the embedding process of the additional safety properties into the Capella system model and PVMT user guide, refer to Appendix B.

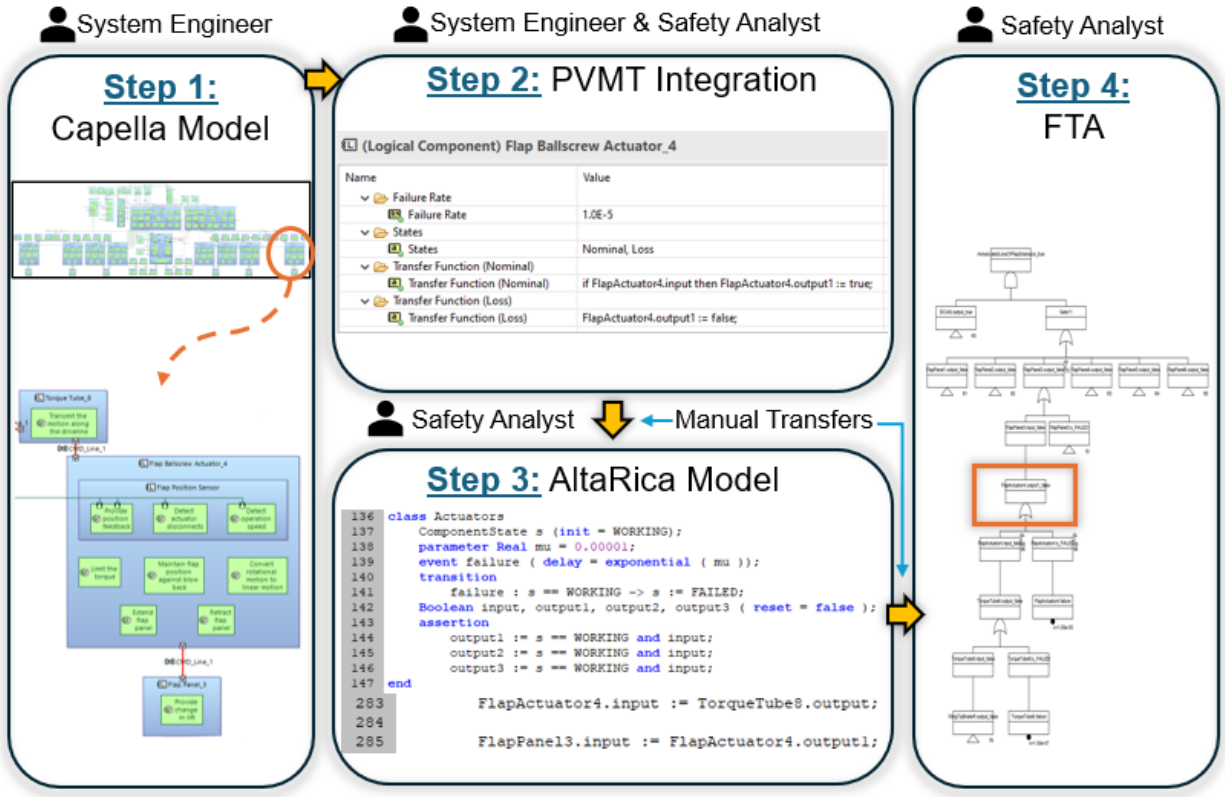

Figure 4.25 - Flap Ballscrew Actuator Example of Methodology Implementation

➢ Step 3: AltaRica Model
In this step, the figure highlights the actuator class and its assertions within the AltaRica model. The parts of the coding depicted in the figure can be coded with the system architecture specification by safety analysts since PVMT helps to add the additional safety properties mentioned above.

- Step 4: FTA
  The final step is safety analysts performing FTAs depending on the failure case inspected with the FTA tool available in AltaRica 3.0 and the generation of fault tree diagrams with the Arbre Analyst software. The fault tree provides a visual representation of potential failure scenarios and their associated causes. Figure 4.25 shows the *annunciated loss of the flap extension* failure case and highlights the *flap ball-screw actuator 4* failure as an intermediate event. The actuator appears on the fault tree since it is connected to *flap panel 3*, and loss of flap panels is required for the failure case to occur. The actuator continues the tree by sending its output to *torque tube 8*. Eventually, the tree ends with basic events.

4.6 Modeling and Integration Summary

The presented flap system test case validates the methodology introduced in Chapter 3. This test case focused on enhancing the system architecture specification process to support automated FTA through the integration of MBSE and MBSA. The selected test case is developed using the Capella workbench and analyzed with AltaRica 3.0.

The key outcomes of the case study can be listed as follows:

- System Model Development Methodology: Extending ARCADIA's logical architecture levels proved effective in representing the system architecture with the required granularity for MBSA to perform automated FTAs.
- Enhanced System Specification: Using Capella's PVMT add-on allowed for the integration of safety properties into the system model. This facilitated an architecture specification that can be used as a source to create safety models.
- Automated FTA Implementation: The case study successfully demonstrated the use of AltaRica 3.0 to create safety models. These models enabled automated FTA, identifying minimal cutsets and calculating failure rates.

Each logical level is enriched by integrating safety artifacts with the help of PVMT. L0 has its failure chains, which are unique to this level, to support safety assessment by displaying potential failure connections but lacks precise failure rates since it provides a high-level architecture with no logical systems other than the flap system. L1 implements precise failure rates for the parts of the flap system that have seen enough breakdowns, while L2 has all the failure rates for each component. All levels include component exchanges edited with PVMT to show flow variables and their directions, state variables for defining different states a component takes, and transfer functions for the behavior of the components.

AltaRica 3.0 is used for safety analyses. Three distinct failure scenarios are inspected through FTAs: Annunciated loss of flaps extension/retraction, unannunciated loss of flaps extension/retraction, and flap panel disconnection. To confirm the validity of the safety models, minimal cutsets of each level are examined by the industry partners. Also, a quantitative FTA validates if the architecture developed satisfies the expected failure rates of the failure cases presented. AltaRica safety models with fault tree images for three failure scenarios are presented in Appendix C.

The test case study presented several challenges and limitations. The flap system involves many components and sensors, making the safety assessment difficult since the AltaRica language does not give detailed error messages, especially when closing the feedback loops in the safety models. Also, at the levels that involve many system model elements, L2 and PA, Capella diagrams make

the individual elements difficult to distinguish from others. Refer to Appendix A for Capella system model diagrams at each logical level.

In addition to the flap system test case, the methodology introduced in Chapter 3 can potentially be applied to more complex systems. In more complex contexts, the methodology offers, in principle, scalability and adaptability to accommodate larger systems. The extension of ARCADIA's logical architecture levels, as demonstrated in the test case, can effectively capture the system architecture with the required granularity for MBSA. This enables the integration of safety properties into the different phases of the system model using PVMT, facilitating an architecture specification that serves as a foundation for creating safety models. Since the safety artifacts integrated into the system model are aligned with the MBSA inputs specified by ARP4761, using different MBSA languages to create safety models is also possible by changing the syntax of transfer functions according to the language used.

5. Conclusion

This thesis outlines an approach to improve model-based system architecture by integrating safety elements into the system development process, thereby supporting model-based safety assessments. The method consists of successive stages, each playing a role in refining and improving the system's model.

MBSE brings various benefits to the aircraft development process. A major advantage is that the utilization of model elements is possible in every stage of the development process. Therefore, the presented framework and specifications can continue to later stages of development, where manufacturer suppliers and subcontractors create systems that align with aircraft and system requirements. Moreover, MBSE tools such as Capella and their add-ons feature various viewpoints that can be tailored depending on the engineering discipline for system architecture, making it suitable for various domains, including model-based safety assessment. Also, the three-stage logical level approach presented in this thesis ensures that the different levels of detail are displayed for system elements. Capella diagrams that show a breakdown of components and functions can support the presented approach for further emphasis on granularity levels.

5.1 Summary of Contributions

This thesis contributes to the integration of safety analysis in MBSE in the following ways: (1) methodology development, (2) implementation in Capella, and (3) test case application.

This thesis presents a systematic methodology that maps the ARP4761 safety process to three stages of system model development: System Analysis, Logical Architecture, and Physical Architecture. Using the extended logical architecture levels (L0, L1, and L2) from Tabesh [38], this thesis adds the appropriate safety analysis steps. At L0, system functions are tagged with FHA results to identify top-level events for FTAs. L1 introduces partial redundancy information and a breakdown of system elements, representing a top-down aircraft development approach. The final logical phase, L2, includes all redundancy information, detailed architecture, and system elements, allowing for quantitative FTAs to validate FHA results.

The main contribution of this thesis is to integrate artifacts required for safety analysis into the Capella system model using the PVMT add-on. This integration incorporates safety properties needed for analyses into system model elements. The following properties have been identified as essential: failure rates, state variables, and transfer functions In addition, component exchanges edited with PVMT represent flow variables addressing failure connections between system components. Also, functional chains have been proposed to identify failure relationships across functions, supporting comprehensive safety assessments and helping engineers understand the potential impacts of component failures on other functions.

The methodology presented in this thesis is validated using a flap system as a test case. Through iterative processes, the application of the methodology is demonstrated in improving system models, incorporating safety features, and performing FTAs for different failure scenarios. The integration of fault tree analyses with the development process allows for the evaluation of potential safety concerns related to the system architecture. The use of AltaRica for safety analysis provides a formal representation of system dynamics and state changes, facilitating the automated creation of safety artifacts, increasing efficiency, and reducing development time.

5.2 Discussion of Limitations

Several challenges and drawbacks have been faced while creating different viewpoints and diagrams for the proposed method. One challenge is the representation of complex architectures. At L2 and PA levels, the architectural diagrams have many elements that make the diagrams big in size, and consequently, individual elements become very difficult to read. Although this comprehensive representation allows suppliers to examine the architecture with every component and their interconnections in detail, it is not ideal for high-level stakeholders and system development teams. Also, the test case architecture layout illustrates the real-life placement of system components (right and left wings), but there is no means in Capella to show the location of systems and their components referencing the aircraft. Furthermore, Capella diagrams do not support encapsulation. In order to reach a subsystem from a main architectural diagram, another diagram needs to be created in Capella, unlike SysML diagrams. Another challenge has occurred while performing FTAs with AltaRica regarding errors. Many errors, whether they are syntax-related or not, do not have a detailed description of them. Thus, developing safety models with Altarica, especially for new users, can become time-consuming. Also, a challenge for safety modeling is feedback loops. Although the AltaRica 3.0 version addresses the issues of feedback loops in safety models, the complexity of the test case resulted in not closing the loop in this thesis..

5.3 Future Work

An important area for further development is the automation of the transition between Capella and AltaRica. Safety properties added using the PVMT add-on in Capella can be extracted and utilized to enable the automatic creation of safety models within AltaRica. This automation would facilitate the creation of safety models and could be combined with the already automated FTA, thus significantly reducing the gap between MBSE and MBSA.

Following the automation improvements, reducing the number of assumptions required in performing FTAs. The current models include several simplifying assumptions to define the scope of this thesis. Future work should aim to increase the level of detail in MBSA by closing the loop on control of aircraft systems, thereby refining the safety models without compromising the overarching goal of enhancing MBSE.

Another important step is integrating other types of safety analyses into Capella. This would provide a more holistic safety assessment process. The current work can be extended by incorporating FMEA at the PA stage. Therefore, future efforts should aim to capture FHA, FTA, and FMEA within a single system model.

Bibliography

[1] International Civil Aviation Organization, “Future of Aviation”, Apr. 22, 2024. https://www.icao.int/Meetings/FutureOfAviation/Pages/default.aspx (accessed Apr. 22, 2024).

[2] A. Macintosh and L. Wallace, “International aviation emissions to 2025: Can emissions be stabilised without restricting demand?”, vol. 37, no. 1, Jan. 2009, doi: 10.1016/J.ENPOL.2008.08.029.

[3] International Energy Agency, “Aviation”, 2022. https://www.iea.org/energy-system/transport/aviation.

[4] European Commission. Directorate General for Research and Innovation. and European Commission. Directorate General for Mobility and Transport., Flightpath 2050 :Europe’s vision for aviation : maintaining global leadership and serving society’s needs. LU: Publications Office, 2012. doi: 10.2777/15458.

[5] H.-H. Altfeld, Commercial Aircraft Projects: Managing the Development of Highly Complex Products. Routledge, 2010. doi: 10.4324/9781315572833.

[6] S. Gradel, B. Aigner, and E. Stumpf, “Model-based safety assessment for conceptual aircraft systems design,” CEAS Aeronautical Journal, vol. 13, no. 1. Springer Science and Business Media LLC, pp. 281–294, Nov. 23, 2021. doi: 10.1007/s13272-021-00562-2.

[7] A. K. Jeyaraj, N. Tabesh, and S. Liscouët-Hanke, “Connecting Model-based Systems Engineering and Multidisciplinary Design Analysis and Optimization for Aircraft Systems Architecting,” AIAA AVIATION 2021 FORUM. American Institute of Aeronautics and Astronautics, Jul. 28, 2021. doi: 10.2514/6.2021-3077.

[8] L. R. Jenkinson, P. Simpkin, and D. Rhodes, Civil Jet Aircraft Design. Oxford England: Butterworth-Heinemann, 2003.

[9] C. S. Tang, J. D. Zimmerman, and J. I. Nelson, “Managing New Product Development and Supply Chain Risks: The Boeing 787 Case,” Supply Chain Forum: An International Journal, vol. 10, no. 2. Informa UK Limited, pp. 74–86, Jan. 2009. doi: 10.1080/16258312.2009.11517219.

[10] I. Dörfler and O. Baumann, “Learning from a Drastic Failure: The Case of the Airbus A380 Program,” Industry and Innovation, vol. 21, no. 3. Informa UK Limited, pp. 197–214, Apr. 03, 2014. doi: 10.1080/13662716.2014.910891.

[11] SAE International, “ARP4754A: Development of Civil Aircraft and Systems,” 2011.

[12] J. Estefan, "Survey of model-based systems engineering (MBSE) methodologies," Int. Council Syst. Eng., San Diego, CA, USA, Jan. 2008.

[13] A. L. Ramos, J. V. Ferreira, and J. Barcelo, “Model-Based Systems Engineering: An Emerging Approach for Modern Systems,” IEEE Transactions on Systems, Man, and

Cybernetics, Part C (Applications and Reviews), vol. 42, no. 1. Institute of Electrical and Electronics Engineers (IEEE), pp. 101–111, Jan. 2012. doi: 10.1109/tsmcc.2011.2106495.

[14] L. Grunske and B. Kaiser, “Automatic generation of analyzable failure propagation models from component-level failure annotations,” Fifth International Conference on Quality Software (QSIC’05). IEEE, 2005. doi: 10.1109/qsic.2005.16.

[15] A. Joshi, S. Vestaland P. Binns, “Automatic Generation of Static Fault Trees from AADL Models”, Apr. 2007. [Online]. Available: https://hdl.handle.net/11299/217313

[16] T. Prosvirnova et al., “The AltaRica 3.0 Project for Model-Based Safety Assessment,” IFAC Proceedings Volumes, vol. 46, no. 22. Elsevier BV, pp. 127–132, 2013. doi: 10.3182/20130904-3-uk-4041.00028.

[17] F. Mhenni, “Safety analysis integration in a systems engineering approach for mechatronic systems design”, 2014.

[18] INCOSE Systems engineering handbook: a guide for system life cycle processes and activities, 4th Edition, John Wiley & Sons, I., Hoboken., 2015.

[19] A. M. Madni and M. Sievers, “Model-based systems engineering: Motivation, current status, and research opportunities,” Systems Engineering, vol. 21, no. 3. Wiley, pp. 172–190, May 2018. doi: 10.1002/sys.21438.

[20] INCOSE, "Systems Engineering Vision 2020," San Diego, CA, USA, Sep. 2007.

[21] K. A. Odukoya, R. I. Whitfield, L. Hay, N. Harrison, and M. Robb, “An Architectural Description For The Application Of Mbse In Complex Systems,” 2021 IEEE International Symposium on Systems Engineering (ISSE). IEEE, Sep. 13, 2021. doi: 10.1109/isse51541.2021.9582510.

[22] B. Beihoff et al., "A World in Motion – Systems Engineering Vision 2025," 2014.

[23] N. A. Tepper, “Exploring the use of Model-Based Systems Engineering (MBSE) to develop systems architectures in naval ship design”, 2010.

[24] J. M. Borky and T. H. Bradley, Effective Model-Based Systems Engineering. Springer International Publishing, 2019. doi: 10.1007/978-3-319-95669-5.

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

[26] F. Patou, M. Dimaki, A. Maier, W. E. Svendsen, and J. Madsen, “Model-based systems engineering for life-sciences instrumentation development,” Systems Engineering, vol. 22, no. 2. Wiley, pp. 98–113, Mar. 14, 2018. doi: 10.1002/sys.21429.

[27] A. Fisher et al., “3.1.1 Model Lifecycle Management for MBSE,” INCOSE International Symposium, vol. 24, no. 1. Wiley, pp. 207–229, Jul. 2014. doi: 10.1002/j.2334-5837.2014.tb03145.x.

[28] B. A. Morris, D. Harvey, K. P. Robinson, and S. C. Cook, “Issues in Conceptual Design and MBSE Successes: Insights from the Model-Based Conceptual Design Surveys,” INCOSE International Symposium, vol. 26, no. 1. Wiley, pp. 269–282, Jul. 2016. doi: 10.1002/j.2334-5837.2016.00159.x.

[29] G. F. Dubos, D. P. Coren, A. Kerzhner, S. H. Chung, and J.-F. Castet, “Modeling of the flight system design in the early formulation of the Europa Project,” 2016 IEEE Aerospace Conference. IEEE, Mar. 2016. doi: 10.1109/aero.2016.7500604.

[30] P. A. Jansma and R. M. Jones, “Advancing the Practice of Systems Engineering at JPL,” 2006 IEEE Aerospace Conference. IEEE. doi: 10.1109/aero.2006.1656171.

[31] R. Malone, B. Friedland, J. Herrold, and D. Fogarty, “Insights from Large Scale Model Based Systems Engineering at Boeing,” INCOSE International Symposium, vol. 26, no. 1. Wiley, pp. 542–555, Jul. 2016. doi: 10.1002/j.2334-5837.2016.00177.x.

[32] R. Cloutier, "Model Based Systems Engineering Survey," Presented at the University of South Alabama, AL, USA, Dec. 2018; 2019.

[33] E. T. McDermott, N. Hutchison, A. Salado, K. Henderson, and M. Clifford, "Benchmarking the Benefits and Current Maturity of Model-Based Systems Engineering across the Enterprise: Results of the MBSE Maturity Survey," Systems Engineering Research Center (SERC), Hoboken, NJ, USA, 2020.

[34] S. Liscouët-Hanke, A. K. Jeyaraj. “A Model-Based Systems Engineering Approach for Efficient Flight Control System Architecture Variants Modelling in Conceptual Design.”In Proceedings of the International Conference on Recent Advances in Aerospace Actuation Systems and Components, Toulouse, France, 30 May–1 June 2018; pp. 34–41.

[35] A. K. Jeyaraj, “A Model-Based Systems Engineering Approach for Efficient System Architecture Representation in Conceptual Design: A Case Study for Flight Control Systems,” Master’s Thesis, Concordia University, Montreal, QC, Canada, 2019.

[36] S. Liscouët-Hanke, H. Jahanara, and J.-L. Bauduin, “A Model-Based Systems Engineering Approach for the Efficient Specification of Test Rig Architectures for Flight Control Computers,” IEEE Systems Journal, vol. 14, no. 4. Institute of Electrical and Electronics Engineers (IEEE), pp. 5441–5450, Dec. 2020. doi: 10.1109/jsyst.2020.2970545.

[37] P. George Mathew, S. Liscouët-Hanke, and Y. Le Masson, “Model-Based Systems Engineering Methodology for Implementing Networked Aircraft Control System on Integrated Modular Avionics – Environmental Control System Case Study,” SAE Technical Paper Series. SAE International, Oct. 30, 2018. doi: 10.4271/2018-01-1943.

[38] N. Tabesh, "A Model-Based System Engineering Approach to Support System Architecting Activities in Early Aircraft Design," Master's Thesis, Concordia University, Montreal, QC, Canada, 2023.

[39] J. Ma, G. Wang, J. Lu, H. Vangheluwe, D. Kiritsis, and Y. Yan, “Systematic Literature Review of MBSE Tool-Chains,” Applied Sciences, vol. 12, no. 7. MDPI AG, p. 3431, Mar. 28, 2022. doi: 10.3390/app12073431.

[40] J. D’Ambrosio and G. Soremekun, “Systems engineering challenges and MBSE opportunities for automotive system design,” 2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC). IEEE, Oct. 2017. doi: 10.1109/smc.2017.8122925.

[41] S. Gérard and B. Selic, “The UML – MARTE Standardized Profile,” IFAC Proceedings Volumes, vol. 41, no. 2. Elsevier BV, pp. 6909–6913, 2008. doi: 10.3182/20080706-5-kr-1001.01171.

[42] M. Hause, “4.5.2 Model- Based System of Systems Engineering with UPDM,” INCOSE International Symposium, vol. 20, no. 1. Wiley, pp. 580–594, Jul. 2010. doi: 10.1002/j.2334-5837.2010.tb01090.x.

[43] A. T. Morris and J. C. Breidenthal, “The necessity of functional analysis for space exploration programs,” 2011 IEEE/AIAA 30th Digital Avionics Systems Conference. IEEE, Oct. 2011. doi: 10.1109/dasc.2011.6096136.

[44] “Types of Models - SEBoK.” Accessed: Apr. 25, 2024. [Online]. Available: https://www.sebokwiki.org/wiki/Types_of_Models#Model_Classification

[45] “What is UML | Unified Modeling Language.” https://www.uml.org/what-is-uml.htm (accessed Apr 25, 2024).

[46] “SysML Open Source Project - What is SysML? Who created SysML?.” Eclipse Foundation, [Online], Available: https://sysml.org/

[47] J.-L. Voirin, “Motivations, Background and Introduction to Arcadia,” Model-Based System and Architecture Engineering with the Arcadia Method. Elsevier, pp. 3–14, 2018. doi: 10.1016/b978-1-78548-169-7.50001-9.

[48] J.-L. Voirin, “Modelling Languages for Functional Analysis Put to the Test of Real Life,” Complex Systems Design & Management. Springer Berlin Heidelberg, pp. 139–150, 2013. doi: 10.1007/978-3-642-34404-6_9.

[49] S. Bonnet, J. Voirin, V. Normand, and D. Exertier, “Implementing the MBSE Cultural Change: Organization, Coaching and Lessons Learned,” INCOSE International Symposium, vol. 25, no. 1. Wiley, pp. 508–523, Oct. 2015. doi: 10.1002/j.2334-5837.2015.00078.x.

[50] J. Voirin, S. Bonnet, D. Exertier, and V. Normand, “Simplifying (and enriching) SysML to perform functional analysis and model instances,” INCOSE International Symposium, vol. 26, no. 1. Wiley, pp. 253–268, Jul. 2016. doi: 10.1002/j.2334-5837.2016.00158.x.

[51] J. Voirin, “9.1.1 Method and Tools for constrained System Architecting,” INCOSE International Symposium, vol. 18, no. 1. Wiley, pp. 981–995, Jun. 2008. doi: 10.1002/j.2334-5837.2008.tb00857.x.

[52] “Capella MBSE Tool - Arcadia,” Apr. 29, 2024. https://mbse-capella.org/arcadia.html (accessed Apr. 29, 2024).

[53] P. Roques, “Systems Architecture Modeling with the Arcadia Method: A Practical Guide to Capella,” London: ISTE Press, 2018.

[54] O. Lisagor, J. A. McDermid, and D. J. Pumfrey, "Towards a Practical Process for Automated Safety Analysis," 2006.

[55] A. A. Abdellatif and F. Holzapfel, “Model Based Safety Analysis (MBSA) Tool for Avionics Systems Evaluation,” 2020 AIAA/IEEE 39th Digital Avionics Systems Conference (DASC). IEEE, Oct. 11, 2020. doi: 10.1109/dasc50938.2020.9256578.

[56] O. Lisagor, T. Kelly, and R. Niu, “Model-based safety assessment: Review of the discipline and its challenges,” The Proceedings of 2011 9th International Conference on Reliability, Maintainability and Safety. IEEE, Jun. 2011. doi: 10.1109/icrms.2011.5979344.

[57] M. Machin, E. Saez, P. Virelizier, and X. de Bossoreille, “Modeling Functional Allocation in AltaRica to Support MBSE/MBSA Consistency,” Model-Based Safety and Assessment. Springer International Publishing, pp. 3–17, 2019. doi: 10.1007/978-3-030-32872-6_1.

[58] H. Mortada, T. Prosvirnova, and A. Rauzy, “Safety Assessment of an Electrical System with AltaRica 3.0,” Model-Based Safety and Assessment. Springer International Publishing, pp. 181–194, 2014. doi: 10.1007/978-3-319-12214-4_14.

[59] P. Bieber, C. Bougnol, C. Castel, J.-P. H. Christophe Kehren, S. Metge, and C. Seguin, “Safety Assessment with Altarica,” Building the Information Society. Springer US, pp. 505–510. doi: 10.1007/978-1-4020-8157-6_45.

[60] S. Kabir, K. Aslansefat, I. Sorokos, Y. Papadopoulos, and Y. Gheraibia, “A Conceptual Framework to Incorporate Complex Basic Events in HiP-HOPS,” Model-Based Safety and Assessment. Springer International Publishing, pp. 109–124, 2019. doi: 10.1007/978-3-030-32872-6_8.

[61] P. H. Feiler and A. Rugina, “Dependability Modeling with the Architecture Analysis & Design Language (AADL),” Carnegie Mellon University, 2007, doi: 10.1184/R1/6572996.V1.

[62] P. H. Feiler and D. P. Gluch, “Model-Based Engineering with AADL: An Introduction to the SAE Architecture Analysis & Design Language.” Addison-Wesley Professional, 2012.

[63] O. Akerlund et al., "ISAAC, a framework for integrated safety analysis of functional, geometrical and human aspects," 2007.

[64] SAE International, “ARP4761A: Guidelines for Conducting the Safety Assessment Process on Civil Aircraft, Systems, and Equipment,” 2023.

[65] O. Lisagor, “Failure logic modelling: a pragmatic approach,” Ph.D. Thesis, 2010.

[66] A. Baklouti, N. Nguyen, F. Mhenni, J.-Y. Choley, and A. Mlika, “Improved Safety Analysis Integration in a Systems Engineering Approach,” Applied Sciences, vol. 9, no. 6. MDPI AG, p. 1246, Mar. 25, 2019. doi: 10.3390/app9061246.

[67] N. G. Leveson, “Safety Analysis in Early Concept Development and Requirements Generation,” INCOSE International Symposium, vol. 28, no. 1. Wiley, pp. 441–455, Jul. 2018. doi: 10.1002/j.2334-5837.2018.00492.x.

[68] E. Tranøy and G. Muller, “7.1.1 Reduction of Late Design Changes Through Early Phase Need Analysis,” INCOSE International Symposium, vol. 24, no. 1. Wiley, pp. 570–582, Jul. 2014. doi: 10.1002/j.2334-5837.2014.tb03168.x.

[69] J. F. W. Peeters, R. J. I. Basten, and T. Tinga, “Improving failure analysis efficiency by combining FTA and FMEA in a recursive manner,” Reliability Engineering & System Safety, vol. 172. Elsevier BV, pp. 36–44, Apr. 2018. doi: 10.1016/j.ress.2017.11.024.

[70] N. Yakymets, H. Jaber, and A. Lanusse, "Model-Based System Engineering for Fault Tree Generation and Analysis," in MODELSWARD 2013 - Proceedings of the 1st International Conference on Model-Driven Engineering and Software Development.

[71] P. David, V. Idasiak, and F. Kratz, “Reliability study of complex physical systems using SysML,” Reliability Engineering & System Safety, vol. 95, no. 4. Elsevier BV, pp. 431–450, Apr. 2010. doi: 10.1016/j.ress.2009.11.015.

[72] F. Mhenni, N. Nguyen, and J.-Y. Choley, “SafeSysE: A Safety Analysis Integration in Systems Engineering Approach,” IEEE Systems Journal, vol. 12, no. 1. IEEE, pp. 161–172, Mar. 2018. doi: 10.1109/jsyst.2016.2547460.

[73] P. Helle, “Automatic SysML-based safety analysis,” Proceedings of the 5th International Workshop on Model Based Architecting and Construction of Embedded Systems. ACM, Sep. 30, 2012. doi: 10.1145/2432631.2432635.

[74] Y. Papadopoulos and J. A. McDermid, “Hierarchically Performed Hazard Origin and Propagation Studies,” Computer Safety, Reliability and Security. Springer Berlin Heidelberg, pp. 139–152, 1999. doi: 10.1007/3-540-48249-0_13.

[75] B. M, Bozzano, Villafiorita, A, Akerlund, O, Akerlund, P, "ESACS: an integrated methodology for design and safety analysis of complex systems," in ESREL, Edinburgh, Scotland, 2000.

[76] “FMECA and FTA software - Safety Architect”, Apr. 29, 2024. https://www.all4tec.com/en/safety-architect-fmeca-fta-sofware/ (accessed Apr. 29, 2024).

[77] M. Sango, F. Vallée, A.-C. Vié, J.-L. Voirin, X. Leroux, and V. Normand, “MBSE and MBSA with Capella and Safety Architect Tools,” Complex Systems Design & Management. Springer International Publishing, pp. 239–239, Dec. 09, 2016. doi: 10.1007/978-3-319-49103-5_22.

[78] J. Dumont, F. Sadmi, and F. Vallée, "CONSISTENT SAFETY ANALYSES IN MODEL-BASED SYSTEM ENGINEERING: CONCEPTS AND TOOLS," Embedded Real-Time Software and Systems (ERTS2012), Toulouse, France, Feb. 2012.

[79] G. Point and A.B. Rauzy, "AltaRica: Constraint automata as a description language," 1999.

[80] A. Arnold, G. Point, A. Griffault, and A. Rauzy, “The AltaRica Formalism for Describing Concurrent Systems,” Fundamenta Informaticae, vol. 40, no. 2,3. IOS Press, pp. 109–124, 1999. doi: 10.3233/fi-1999-402302.

[81] M. Bozzano et al., “Safety assessment of AltaRica models via symbolic model checking,” Science of Computer Programming, vol. 98. Elsevier BV, pp. 464–483, Feb. 2015. doi: 10.1016/j.scico.2014.06.003.

[82] M. Batteux, T. Prosvirnova, and A. Rauzy, "AltaRica 3.0: language specification," AltaRica Association, 2017.

[83] Binns, P., Englehart, M., Jackson, M., Vestal, S., “Domain-specific software architectures for guidance, navigation and control,” International Journal of Software Engineering and Knowledge Engineering, vol. 06, no. 02. World Scientific Pub Co Pte Lt, pp. 201–227, Jun. 1996. doi: 10.1142/s0218194096000107.

[84] S. Vestal, "MetaH Support for Real-Time Multi-Processor Avionics," in Proceedings of the 1997 Joint Workshop on Parallel and Distributed Real-Time Systems (WPDRTS / OORTS '97) (WPDRTS '97), IEEE Computer Society, USA, 1997, pp. 11.

[85] P. H. Feiler, B. A. Lewis, and S. Vestal, “The SAE Architecture Analysis & Design Language (AADL) a standard for engineering performance critical systems,” 2006 IEEE Conference on Computer Aided Control System Design, 2006 IEEE International Conference on Control Applications, 2006 IEEE International Symposium on Intelligent Control. IEEE, Oct. 2006. doi: 10.1109/cacsd-cca-isic.2006.4776814.

[86] B. A. Lewis and P. H. Feiler, "Multi-Dimensional Model Based Engineering for Performance Critical Computer Systems Using the AADL," 2009.

[87] Y. Papadopoulos, "Safety-Directed System Monitoring Using Safety Cases," Ph.D. thesis, Dept. Computer Science, The University of York, 2000.

[88] Y. Papadopoulos, J. McDermid, R. Sasse, and G. Heiner, “Analysis and synthesis of the behaviour of complex programmable electronic systems in conditions of failure,” Reliability Engineering & System Safety, vol. 71, no. 3. Elsevier BV, pp. 229–247, Mar. 2001. doi: 10.1016/s0951-8320(00)00076-4.

[89] T. Prosvirnova et al., “Strategies for Modelling Failure Propagation in Dynamic Systems with AltaRica,” Model-Based Safety and Assessment. Springer International Publishing, pp. 101–115, 2022. doi: 10.1007/978-3-031-15842-1_8.

[90] A. B. Rauzy, “Guarded transition systems: A new states/events formalism for reliability studies,” Proceedings of the Institution of Mechanical Engineers, Part O: Journal of Risk and

Reliability, vol. 222, no. 4. SAGE Publications, pp. 495–505, Dec. 01, 2008. doi: 10.1243/1748006xjrr177.

[91] P. Roques, "MBSE with the ARCADIA Method and the Capella Tool," in 8th European Congress on Embedded Real Time Software and Systems (ERTS), Jan. 2016, Toulouse, France.

[92] “Capella MBSE Tool - Add-Ons,” mbse-capella.org. https://mbse-capella.org/addons.html (accessed May 05, 2024).

[93] Bombardier Inc., “Bombardier Global 5000 Flight Crew Operating Manual Vol. 2, Rev. 12 (Airplane General),” 2006. [Online]. Available: https://www.smartcockpit.com/my-aircraft/bombardier-global-5000/

[94] Bombardier Inc., “Bombardier Global 5000 Flight Crew Operating Manual Vol. 2, Rev. 2A (Flight Controls),” 2005. [Online]. Available: https://www.smartcockpit.com/my-aircraft/bombardier-global-5000/

[95] Bombardier Inc., “Bombardier Global 5000 Flight Crew Operating Manual Vol. 2, Rev. 2A (Automatic Flight Control System),” 2005. [Online]. Available: https://www.smartcockpit.com/my-aircraft/bombardier-global-5000/

[96] W. Denson, G. Chandler, W. Crowell, and R. Wanner, “Nonelectronic Parts Reliability Data 1991,” Defense Technical Information Center, May 1991. doi: 10.21236/ada242083.

[97] J.-C. Maré, Aerospace Actuators 1: Needs, Reliability and Hydraulic power solutions. 2016. [Online]. Available: https://hal.archives-ouvertes.fr/hal-02055655

[98] S. Liscouët-Hanke, B. R. Mohan, P. Jeyarajan Nelson, C. Lavoie, and S. Dufresne, “Evaluating a Model-Based Systems Engineering approach for the conceptual design of advanced aircraft high-lift system architectures,” Canadian Aeronautics and Space Institute AERO 2017, 2017.

Appendix A

This section of the Appendix provides an overview of L2-level Capella diagrams.

Figure A.1 illustrates the logical architecture diagram of the L2 phase with section numbers. In the following figures, sections are displayed separately to show a clear picture of the architectural diagram. Chapter 3 details the differences between L2 and PA-level diagrams. Since the layouts of these two levels are the same, the PA level is not shown in the Appendix.

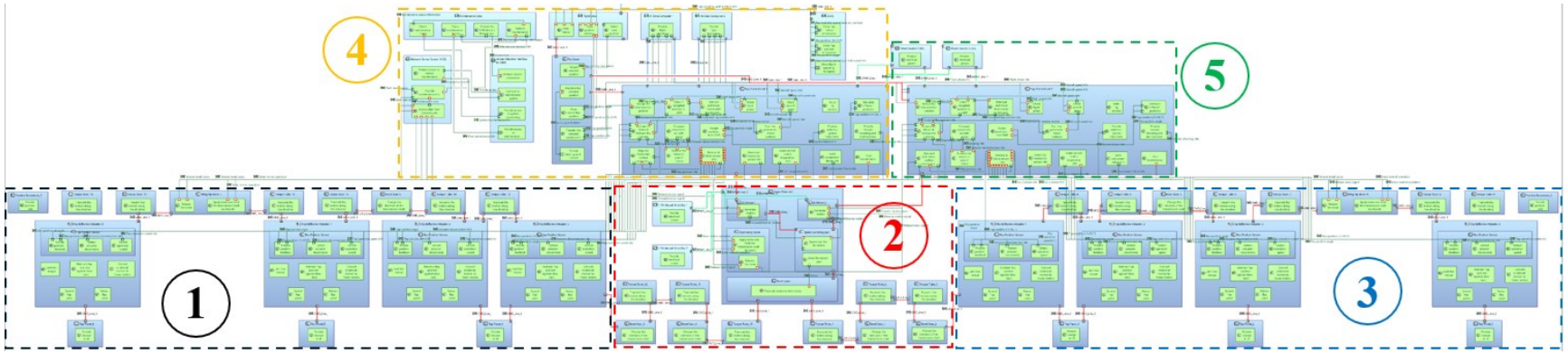

Figure A.1 - Logical Architecture Diagram [LAB] of L2

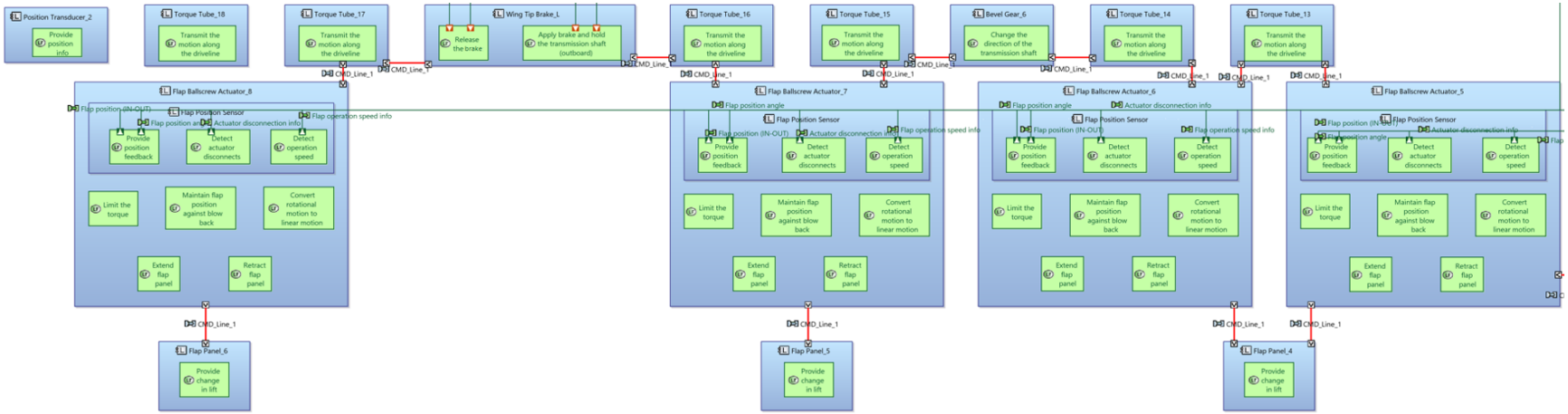

Figure A.2 - Section 1 of Logical Architecture Diagram [LAB] of L2

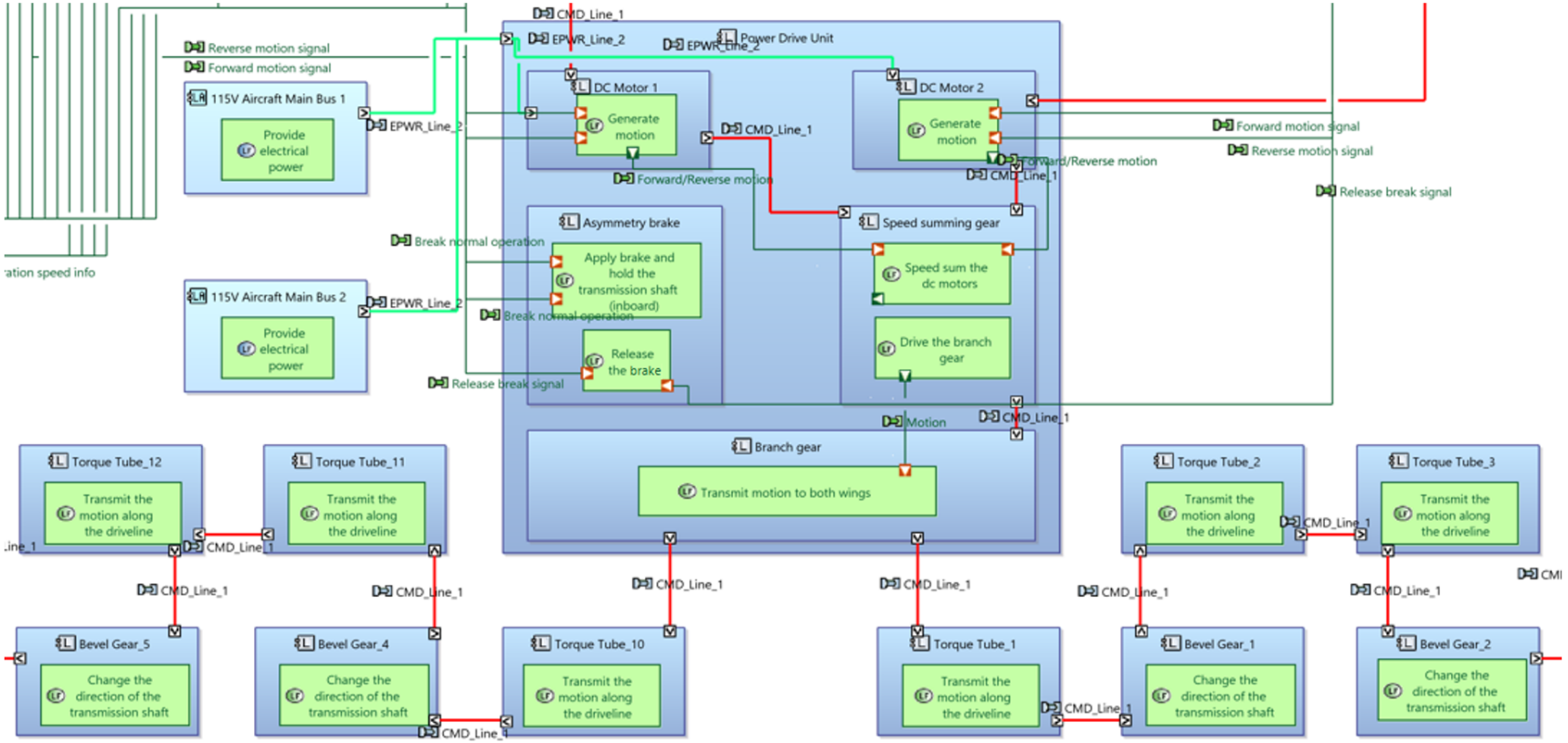

Figure A.3 - Section 2 of Logical Architecture Diagram [LAB] of L2

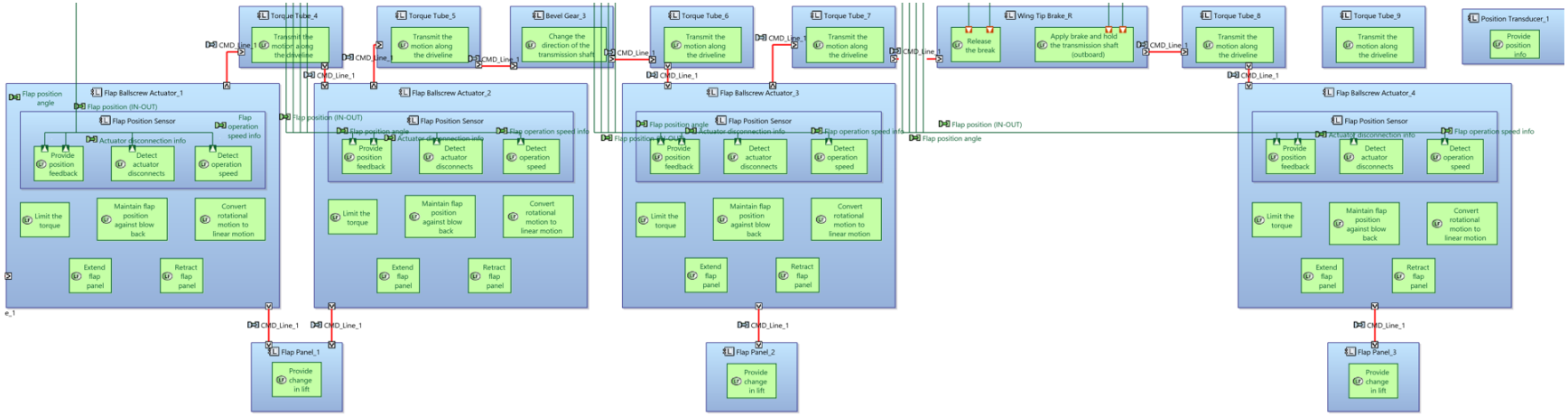

Figure A.4 - Section 3 of Logical Architecture Diagram [LAB] of L2

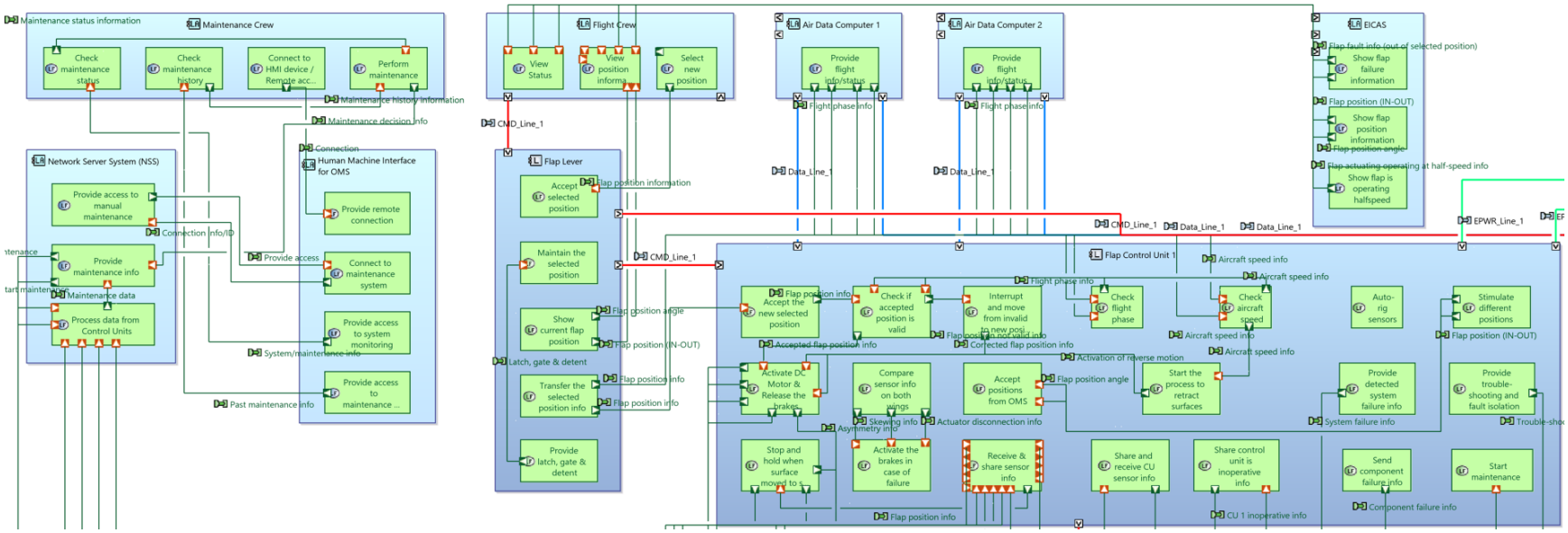

Figure A.5 - Section 4 of Logical Architecture Diagram [LAB] of L2

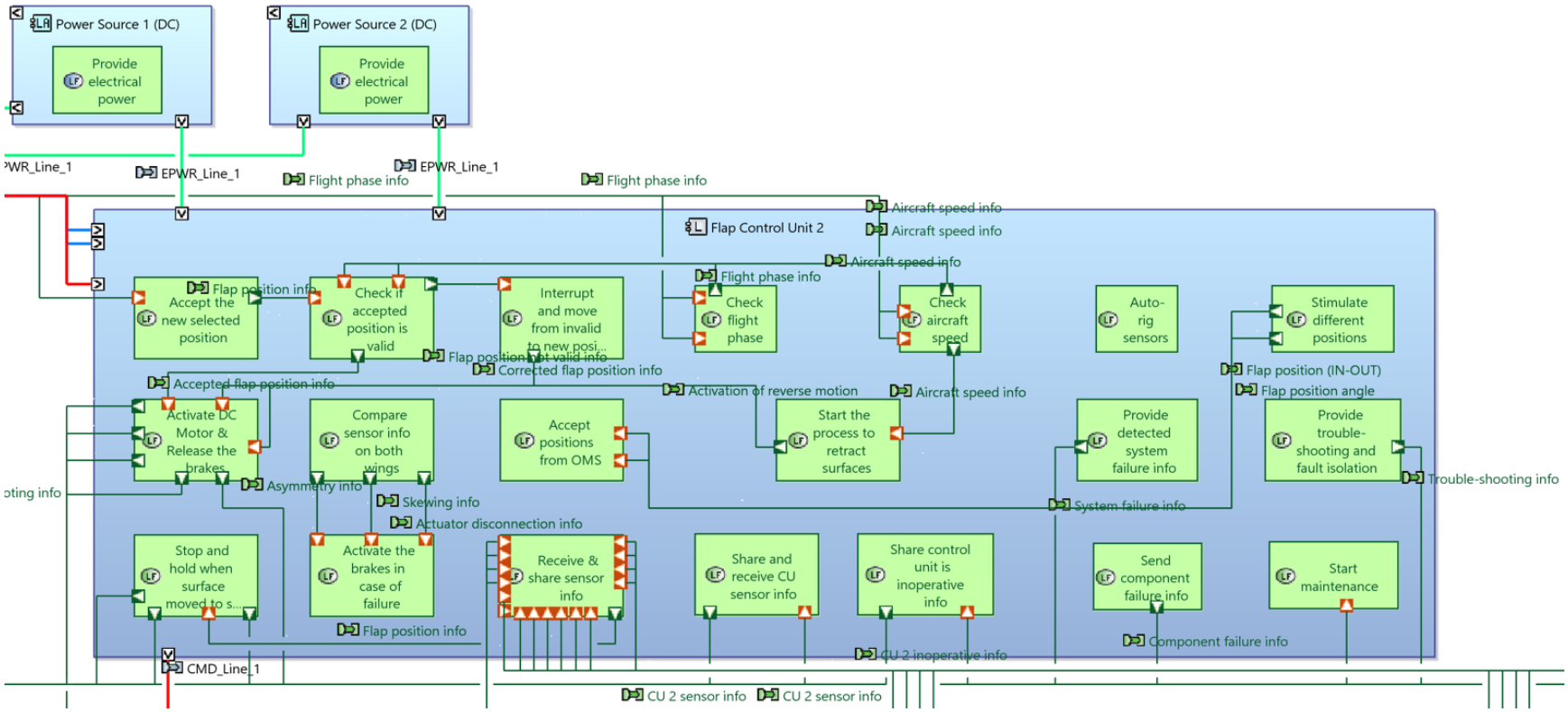

Figure A.6 - Section 5 of Logical Architecture Diagram [LAB] of L2

Appendix B

Appendix B presents a guide on how to enrich the system model with the PVMT add-on. Figure B.1 shows the first step, which is how to add a PVMT viewpoint to a Capella Project.

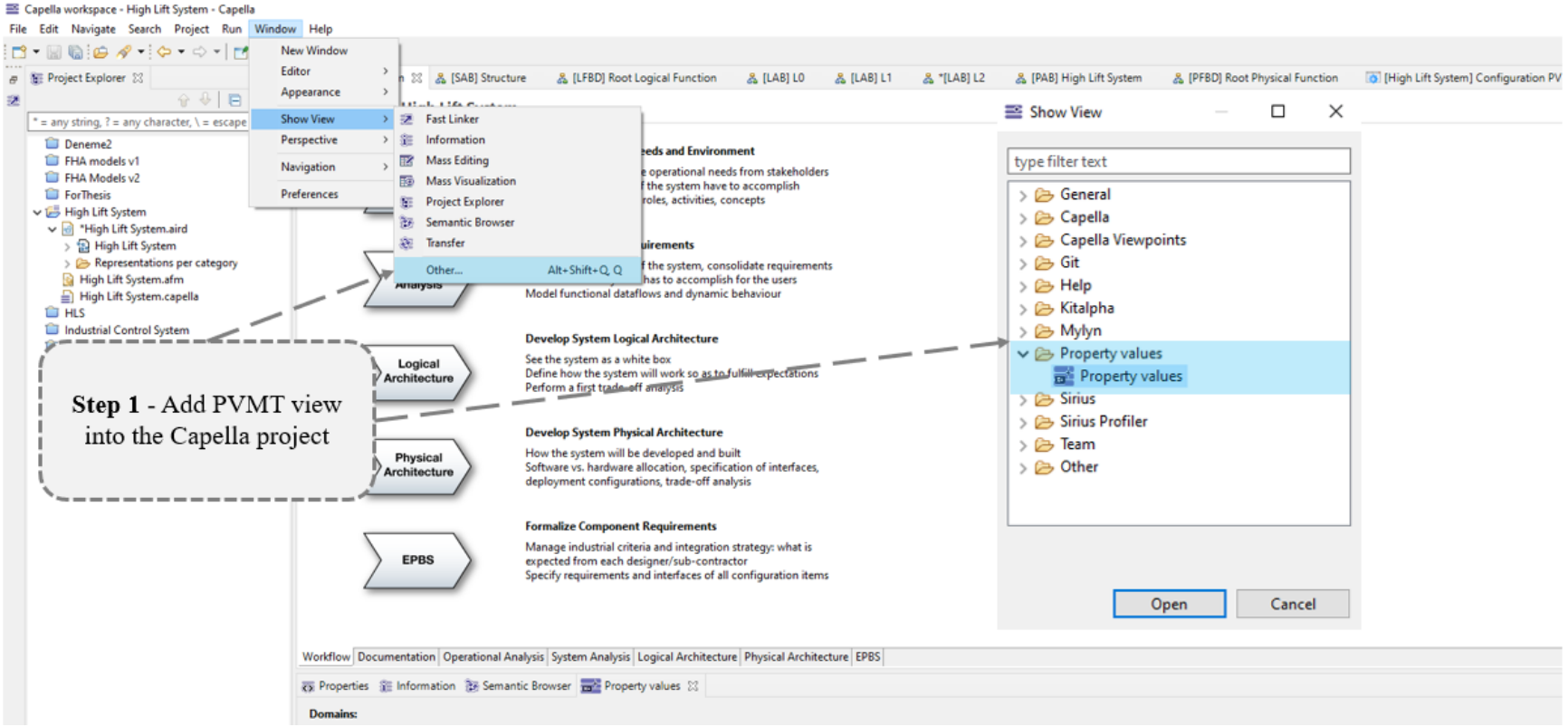

Figure B.1 - Adding PVMT Viewpoint to a Capella Project

Figure B.2 depicts how to activate the definition editor, where additional properties are defined.

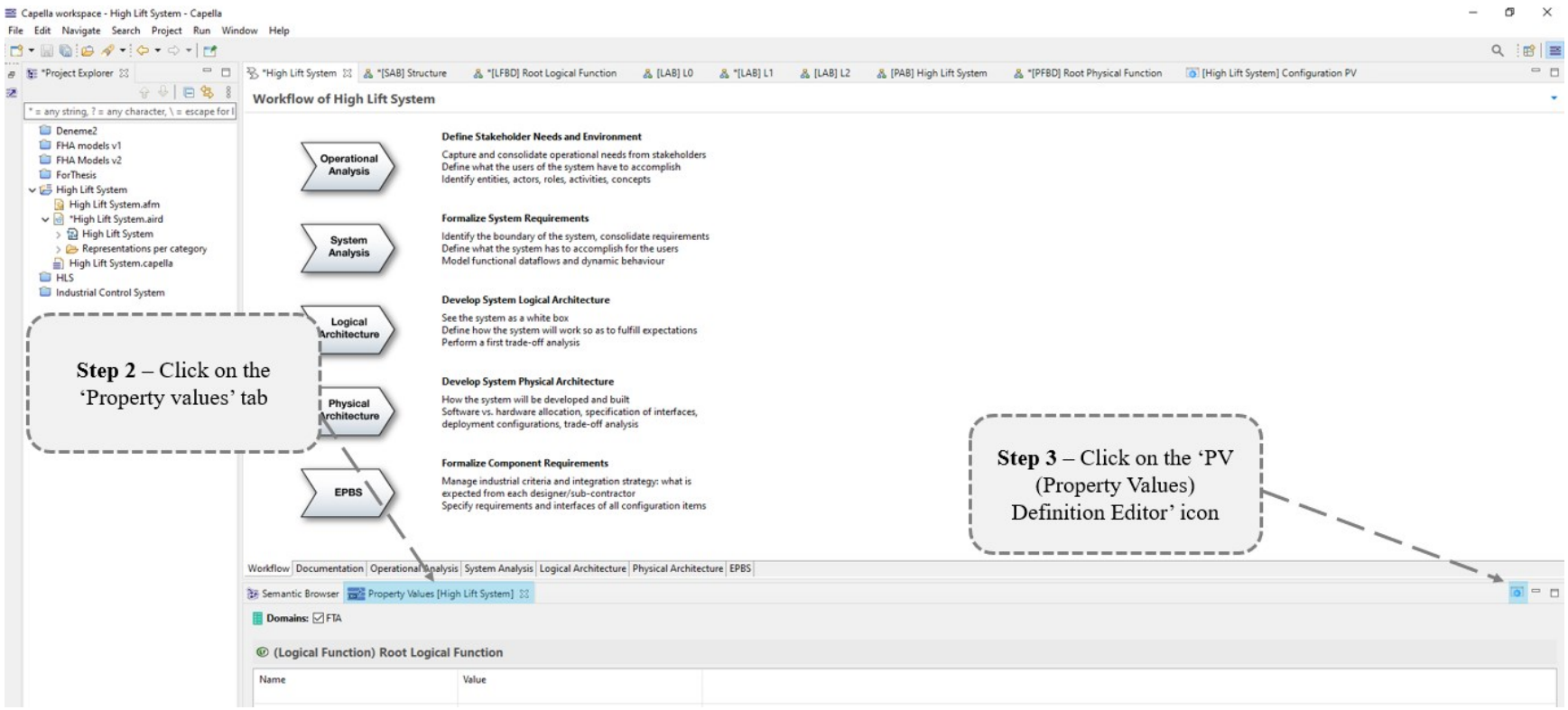

Figure B.2 - Activating PVMT and Definition Editor View

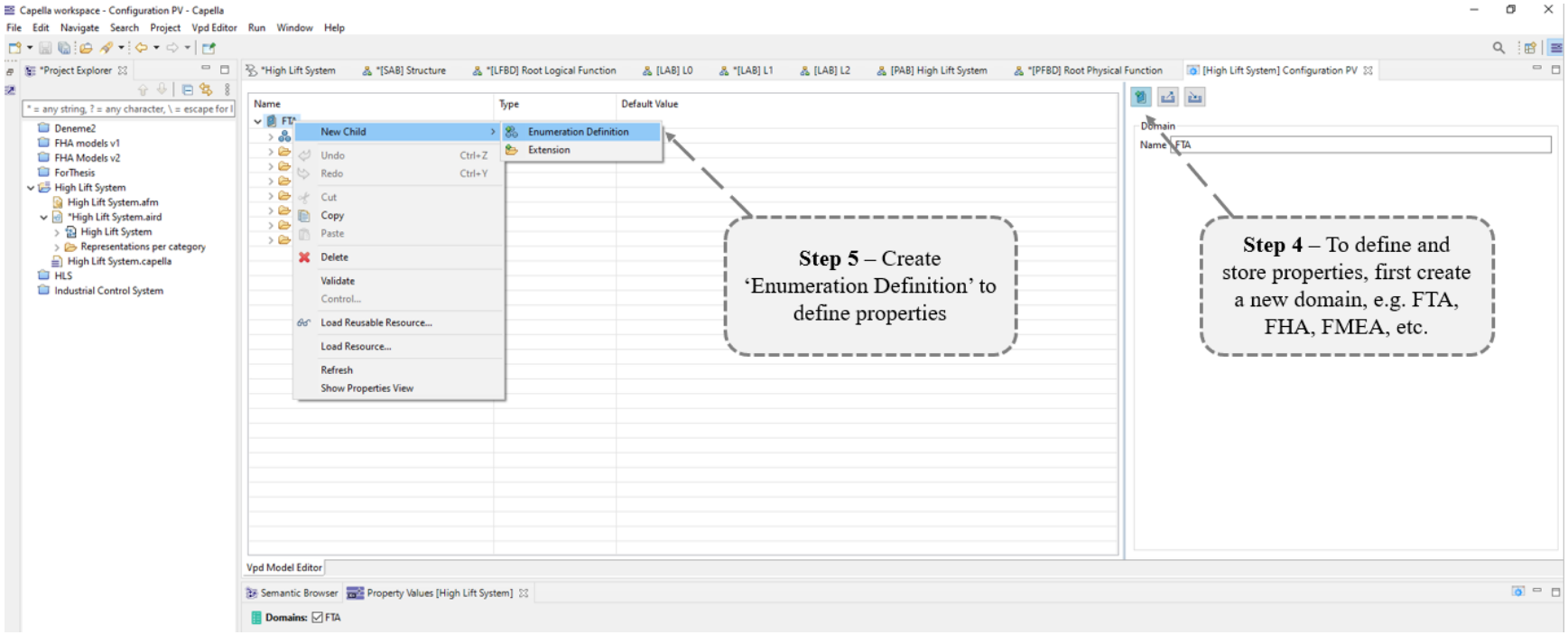

Figure B.3 - Creating Property Domains and Enumeration Definitions

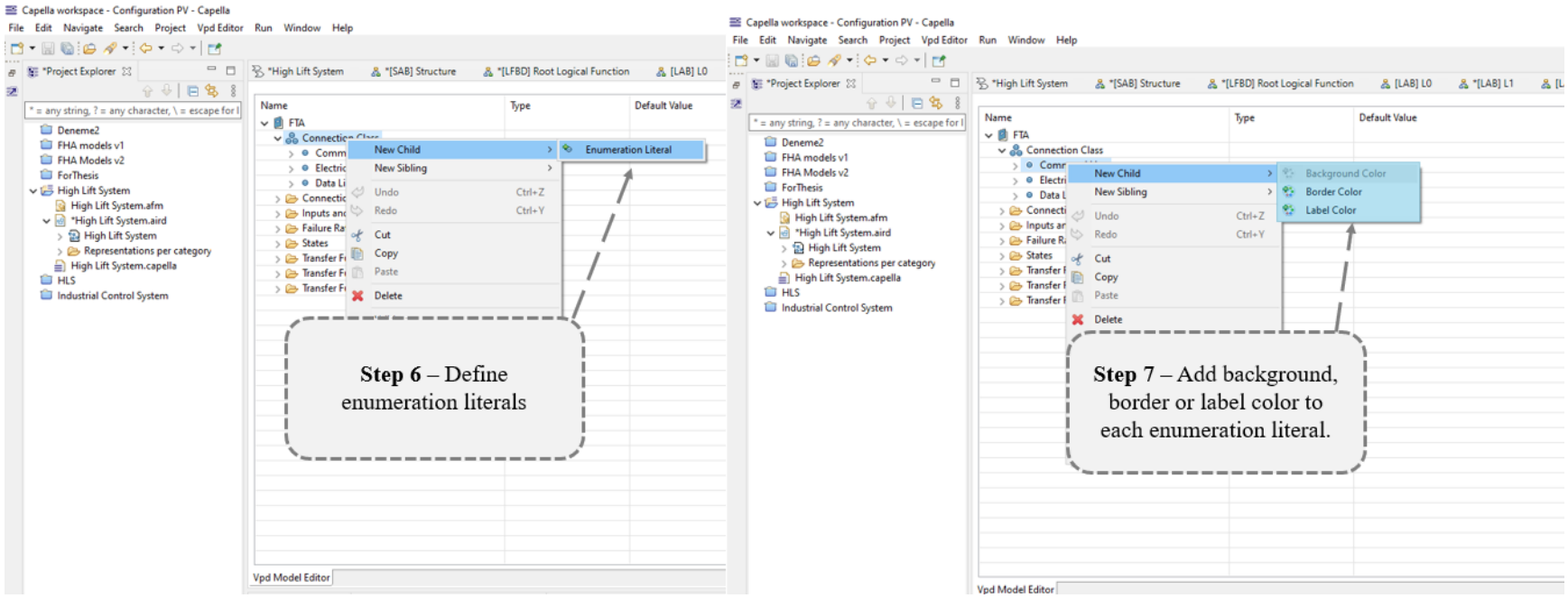

Figure B.4 - Creating Enumeration Literals

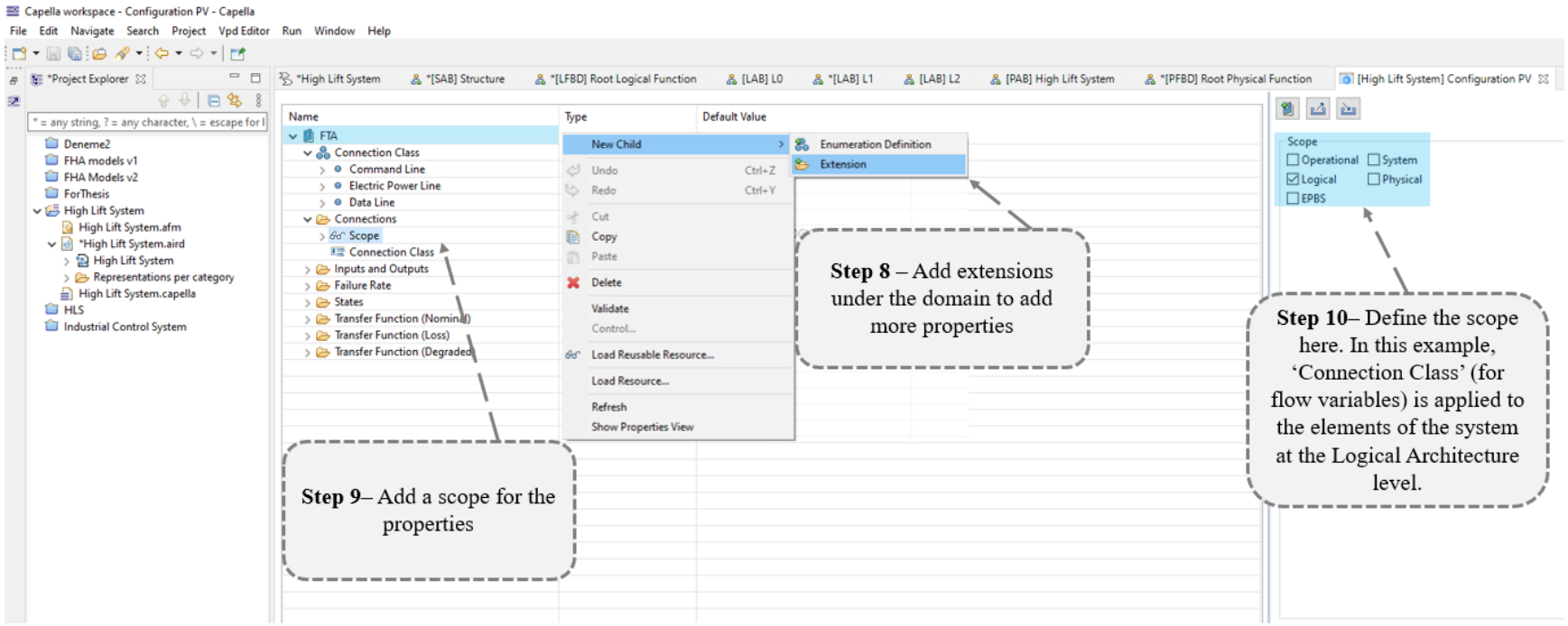

Figure B.5 - Adding Scope to the Property Extension

Figure B.6 displays different property types (string, integer, float, boolean, and enumeration) and how to define them. It also shows how to set a default value and descriptions for the enumeration properties.

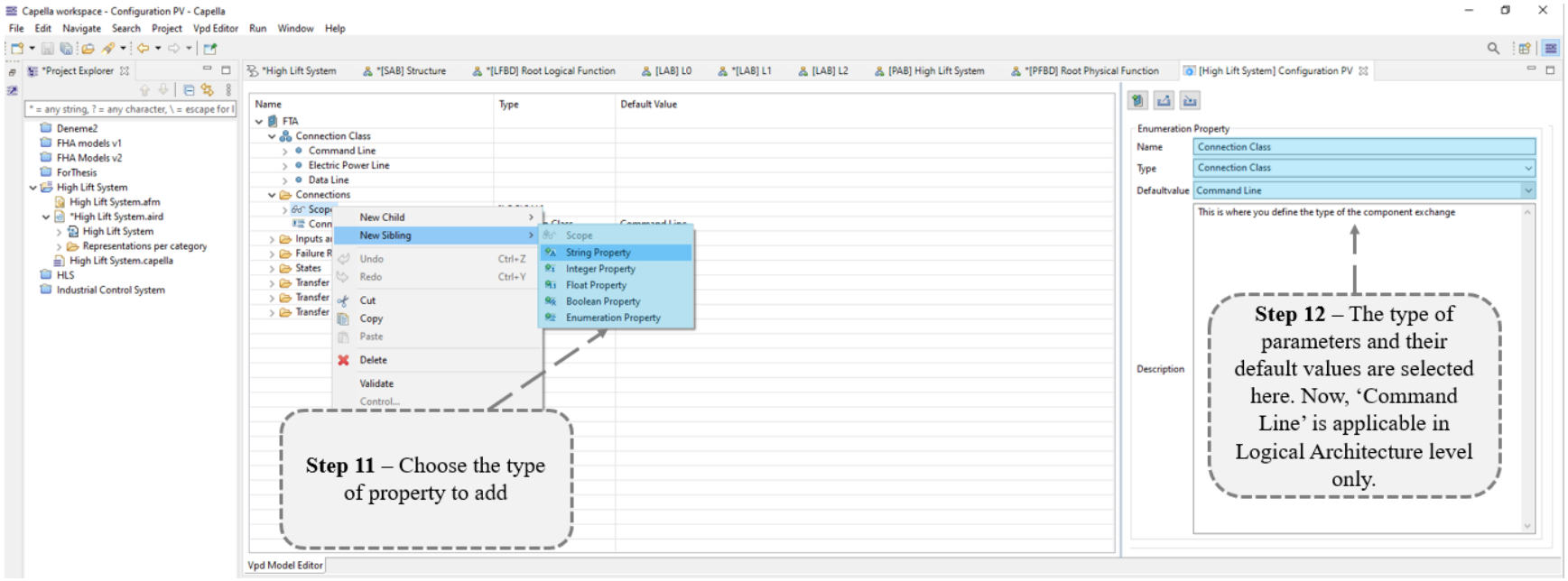

Figure B.6 - Creation of Different Property Types

Figure B.7 - The FTA Viewpoint Created using PVMT to Enhance the Capella System Model

Appendix C

This section of the Appendix shows a guide on how to create safety models and perform FTAs with AltaRica 3.0.

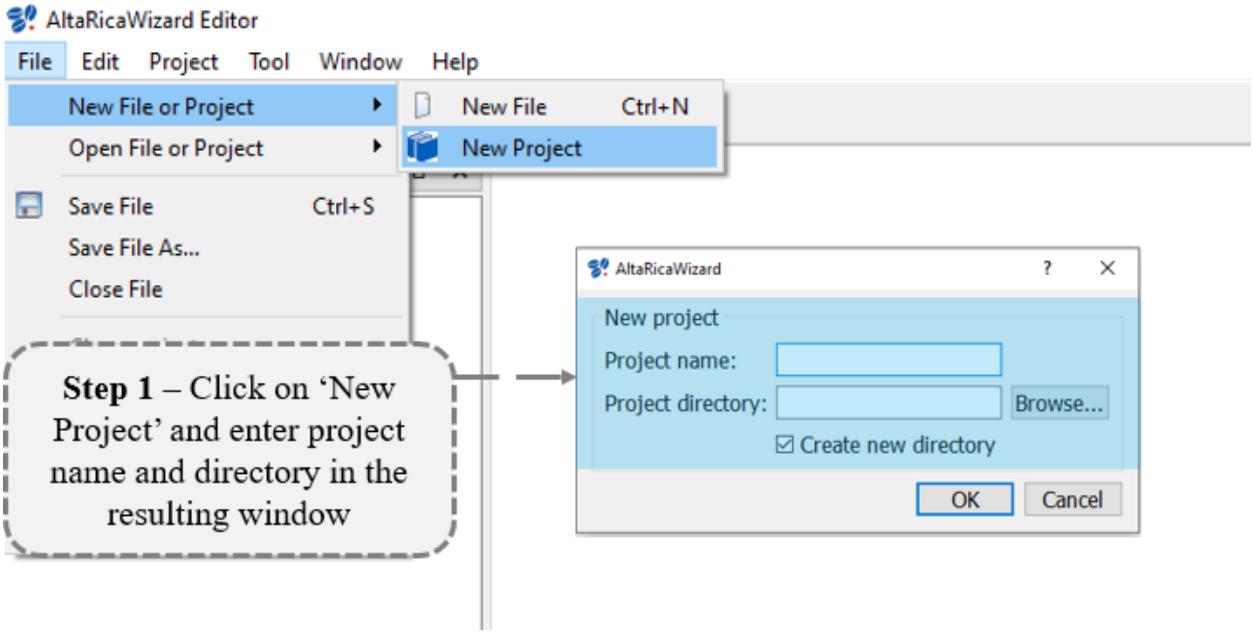

Figure C.1 - AltaRica 3.0 Starting a New Project

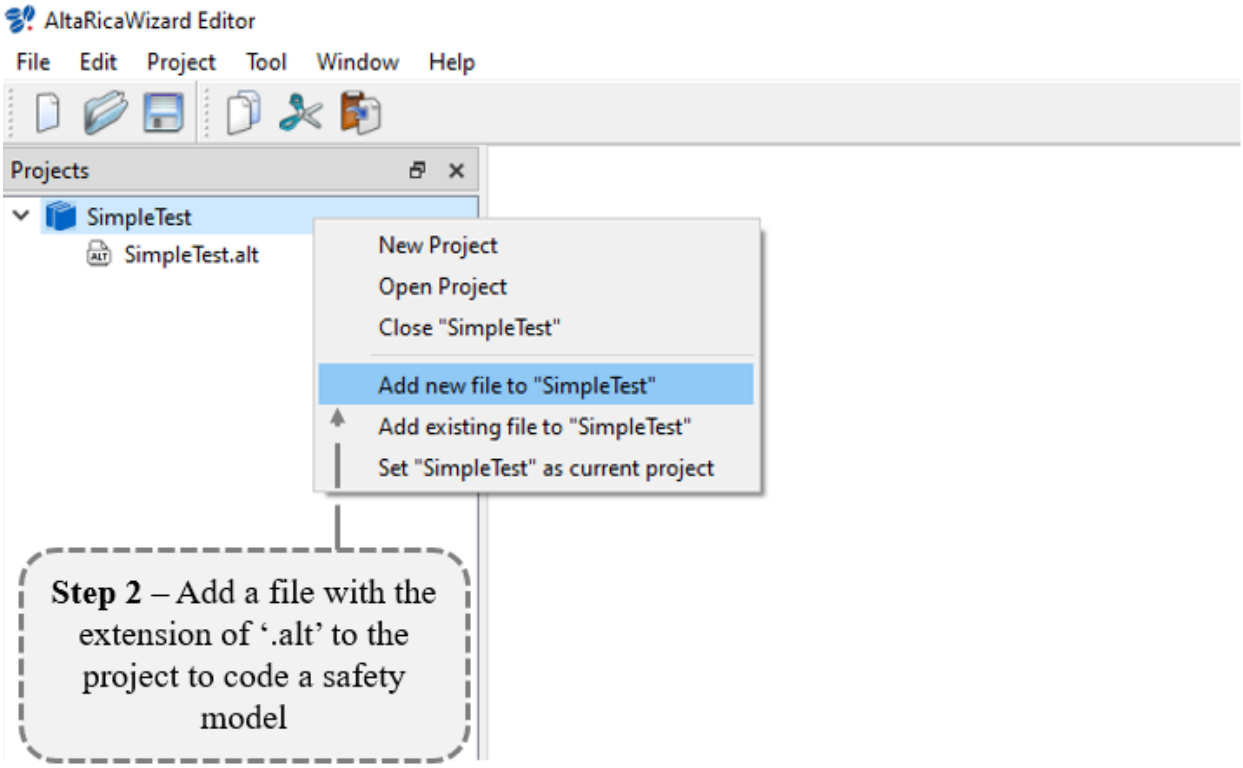

Figure C.2 - Adding '.alt' AltaRica file to the Existing Project

After the second step, displayed in Figure C.2, the user should code the safety model into the '.alt' file. Figure C.3 shows how to start the flattening process, which is the first step of all AltaRica analysis tools. Flattening compiles the AltaRica model into a guarded transition system.

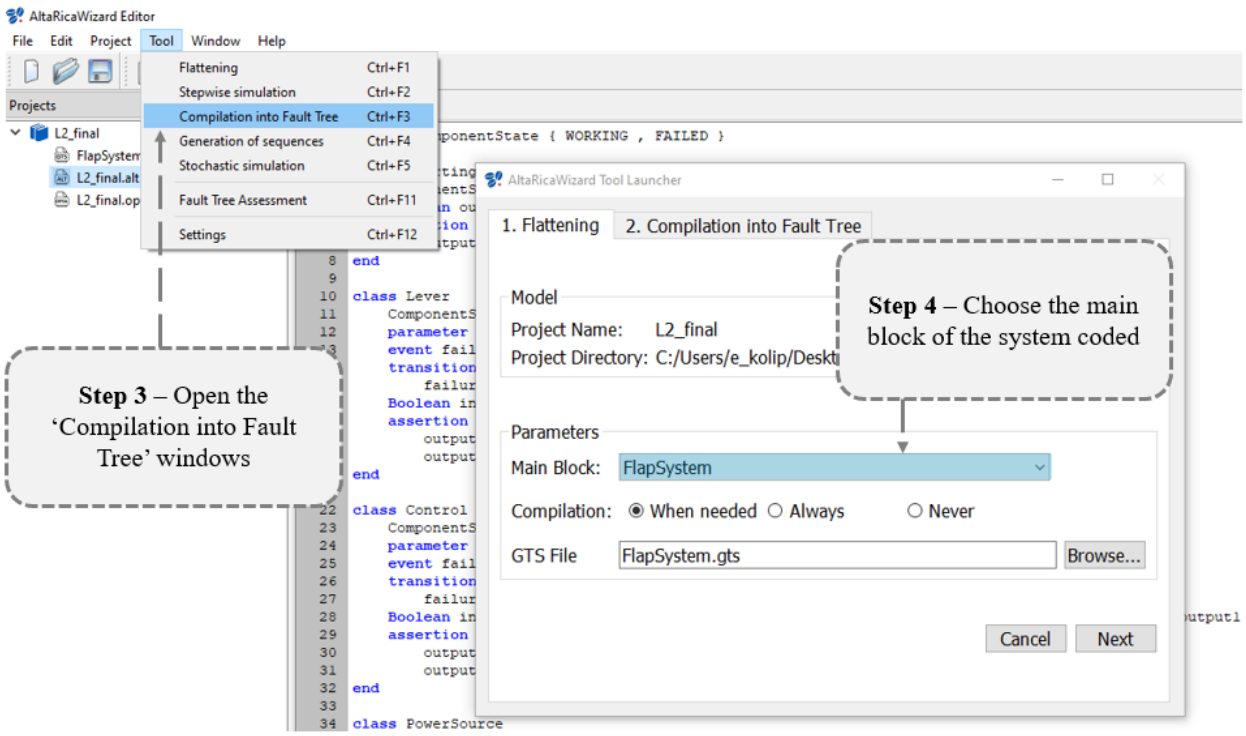

Figure C.3 - Flattening Process of AltaRica 3.0

AltaRica safety models can be utilized to assess different safety objectives or generate different fault trees with several observers. Hence, the AltaRica language allows users to code a safety model and compile it into fault trees rather than designing fault trees directly. The compilation process is explained in Figure C.4.

This thesis uses the Arbre Analyst tool to visualize fault trees. Importing AltaRica safety models into the Arbre Analyst tool is explained in Figure C.5.

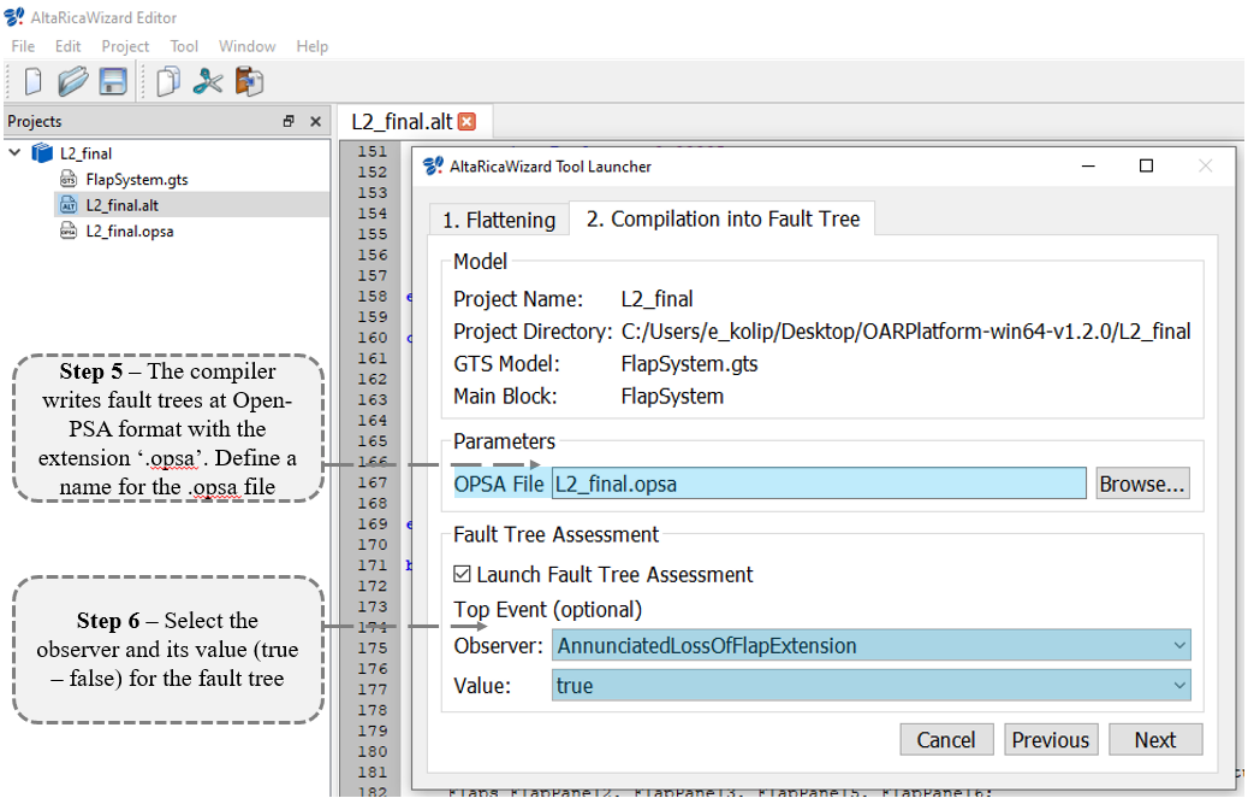

Figure C.4 - Compilation into Fault Tree

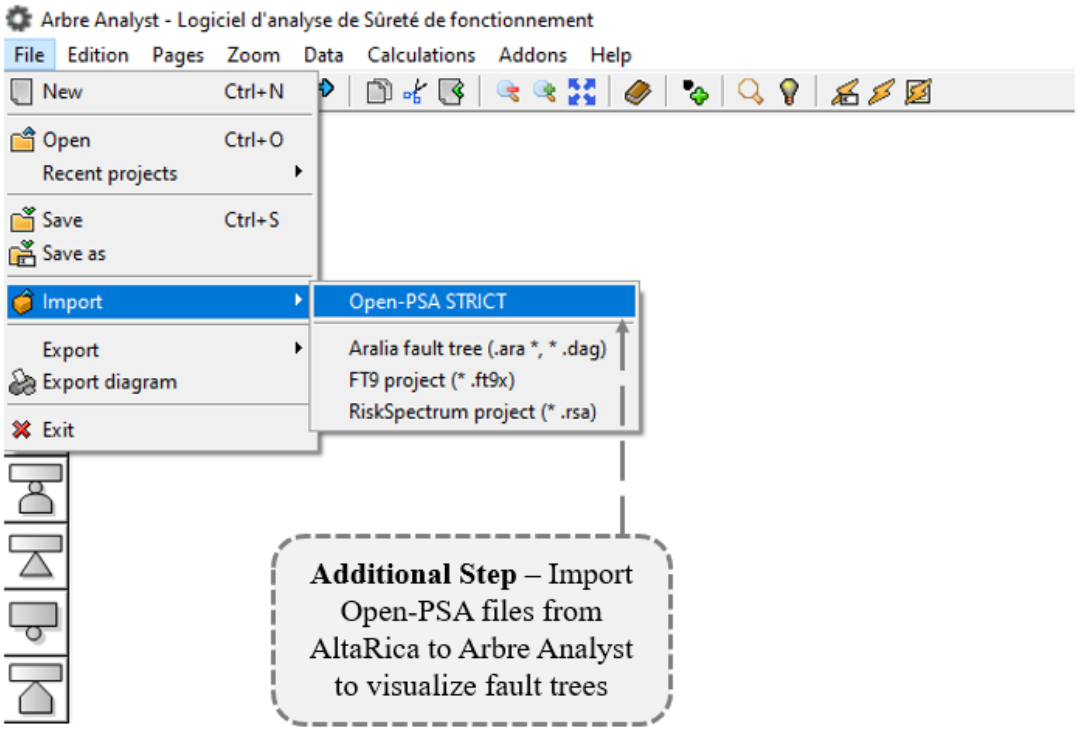

Figure C.5 - Importing '.opsa' Files to Arbre Analyst to Visualize Fault Trees

Appendix D

Appendix D shows parts of AltaRica safety model '.alt' files, which present coding in AltaRica syntax. Figures in this section of Appendix D show the flap system block of L0, L1, and L2 levels in AltaRica safety models. The displayed blocks involve class names with initiated components out of them. The blocks also contain observers for each level and assertions between components.

```
block FlapSystem
        StartingPoint FlightCrew;
        Lever FlapLever;
        EPWER ElectricalPowerSource;
        Control ControlSystem;
        Actuation ActuationSystem;
        Flaps LHSFlap, RHSFlap;
        Sources AirDataComputer, MonitoringSystem;
        Endings EICAS;
        observer Boolean AnnunciatedLossOfFlapExtension = EICAS.output and (not LHSFlap.output or not
RHSFlap.output);
        observer Boolean UnannunciatedLossOfFlapExtension = EICAS.s == FAILED and (not
LHSFlap.output or not RHSFlap.output);
        observer Boolean AnyFlapDisconnection = LHSFlap.s == FAILED or RHSFlap.s == FAILED;
        assertion
                FlapLever.input := FlightCrew.output;
                ControlSystem.input1 := FlapLever.output;
                ControlSystem.input2 := ElectricalPowerSource.output1;
                ControlSystem.input3 := AirDataComputer.output;
                ControlSystem.input4 := MonitoringSystem.output;
                EICAS.input := ControlSystem.output2;
                ActuationSystem.input1 := ControlSystem.output1;
                ActuationSystem.input2 := ElectricalPowerSource.output2;
                LHSFlap.input := ActuationSystem.output1;
                RHSFlap.input := ActuationSystem.output2;
end
```

Figure D.1 - Flap System Block of L0 in AltaRica Safety Model

```
block FlapSystem
        StartingPoint FlightCrew;
        Lever FlapLever;
        PowerSource DCPowerSource1, DCPowerSource2, mainBUS1, mainBUS2;
        Control ControlUnit1, ControlUnit2;
        Motors DCMotor1, DCMotor2;
        Data AirDataComputer1, AirDataComputer2, PositionTransducer, FlapPositionSensor;
        Endings EICAS;
        Drivelines LeftWingDriveline, RightWingDriveline;
        Gear SpeedSumGear;
        Flaps LHSFlap, RHSFlap;
        observer Boolean AnnunciatedLossOfFlapExtension = EICAS.output and (not LHSFlap.output or not
RHSFlap.output);
        observer Boolean UnannunciatedLossOfFlapExtension = EICAS.s == FAILED and (not
LHSFlap.output or not RHSFlap.output);
        observer Boolean AnyFlapDisconnection = LHSFlap.s == FAILED or RHSFlap.s == FAILED;
        assertion
                FlapLever.input := FlightCrew.output;
                ControlUnit1.input1 := FlapLever.output1;
                ControlUnit1.input2 := DCPowerSource1.output1;
                ControlUnit1.input3 := DCPowerSource2.output1;
                ControlUnit1.input4 := AirDataComputer1.output1;
                ControlUnit1.input5 := AirDataComputer2.output1;
                ControlUnit1.input6 := PositionTransducer.output1;
                ControlUnit1.input7 := FlapPositionSensor.output1;
                ControlUnit2.input1 := FlapLever.output2;
                ControlUnit2.input2 := DCPowerSource1.output2;
                ControlUnit2.input3 := DCPowerSource2.output2;
                ControlUnit2.input4 := AirDataComputer1.output2;
                ControlUnit2.input5 := AirDataComputer2.output2;
                ControlUnit2.input6 := PositionTransducer.output2;
                ControlUnit2.input7 := FlapPositionSensor.output2;
                EICAS.input1 := ControlUnit1.output2;
                EICAS.input2 := ControlUnit2.output2;
                DCMotor1.input1 := ControlUnit1.output1;
                DCMotor1.input2 := mainBUS1.output1;
                DCMotor1.input3 := mainBUS2.output1;
                DCMotor2.input1 := ControlUnit2.output1;
                DCMotor2.input2 := mainBUS1.output2;
                DCMotor2.input3 := mainBUS2.output2;
                SpeedSumGear.input1 := DCMotor1.output;
                SpeedSumGear.input2 := DCMotor2.output;
                LeftWingDriveline.input := SpeedSumGear.output1;
                RightWingDriveline.input := SpeedSumGear.output2;
                LHSFlap.input := LeftWingDriveline.output;
                RHSFlap.input := RightWingDriveline.output;
```

Figure D.2 - Flap System Block of L1 in AltaRica Safety Model

```
block FlapSystem
	StartingPoint FlightCrew;
	Lever FlapLever;
	PowerSource DCPowerSource1, DCPowerSource2, mainBUS1, mainBUS2;
	Control ControlUnit1, ControlUnit2;
	Motors DCMotor1, DCMotor2;
	Data AirDataComputer1, AirDataComputer2;
	Endings EICAS;
	Gear SpeedSumGear;
	PduExit BranchGear;
	Actuators FlapActuator1, FlapActuator2, FlapActuator3, FlapActuator4, FlapActuator5,
FlapActuator6, FlapActuator7, FlapActuator8;
	Flaps FlapPanel2, FlapPanel3, FlapPanel5, FlapPanel6;
	InboardFlaps FlapPanel1, FlapPanel4;
	Drivelines TorqueTube1, TorqueTube2, TorqueTube3, TorqueTube4, TorqueTube5, TorqueTube6,
TorqueTube7, TorqueTube8, TorqueTube9, TorqueTube10, TorqueTube11, TorqueTube12, TorqueTube13,
TorqueTube14, TorqueTube15, TorqueTube16, TorqueTube17, TorqueTube18, BevelGear1, BevelGear2,
BevelGear3, BevelGear4, BevelGear5, BevelGear6;
	Brakes WingTipBrakeL, WingTipBrakeR;
	Sensors FlapPositionSensor1, FlapPositionSensor2, FlapPositionSensor3, FlapPositionSensor4,
FlapPositionSensor5, FlapPositionSensor6, FlapPositionSensor7, FlapPositionSensor8, PositionTransducer1,
PositionTransducer2;

	observer Boolean AnnunciatedLossOfFlapExtension = EICAS.output and (not FlapPanel1.output or
not FlapPanel2.output or not FlapPanel3.output or not FlapPanel4.output or not FlapPanel5.output or not
FlapPanel6.output);
	observer Boolean UnannunciatedLossOfFlapExtension = EICAS.s == FAILED and (not
FlapPanel1.output or not FlapPanel2.output or not FlapPanel3.output or not FlapPanel4.output or not
FlapPanel5.output or not FlapPanel6.output);
	observer Boolean AnyFlapDisconnection = FlapPanel1.s == FAILED or FlapPanel2.s == FAILED or
FlapPanel3.s == FAILED or FlapPanel4.s == FAILED or FlapPanel5.s == FAILED or FlapPanel6.s ==
FAILED;
	assertion
		FlapLever.input := FlightCrew.output;
```

Figure D.3 - Flap System Block of L2 in AltaRica Safety Model - Part I

```
                    ControlUnit1.input1 := FlapLever.output1;
                    ControlUnit1.input2 := DCPowerSource1.output1;
                    ControlUnit1.input3 := DCPowerSource2.output1;
                    ControlUnit1.input4 := AirDataComputer1.output1;
                    ControlUnit1.input5 := AirDataComputer2.output1;
                    ControlUnit1.input6 := PositionTransducer2.output;
                    ControlUnit1.input7 := FlapPositionSensor5.output;
                    ControlUnit1.input8 := FlapPositionSensor6.output;
                    ControlUnit1.input9 := FlapPositionSensor7.output;
                    ControlUnit1.input10 := FlapPositionSensor8.output;
                    ControlUnit2.input1 := FlapLever.output2;
                    ControlUnit2.input2 := DCPowerSource1.output2;
                    ControlUnit2.input3 := DCPowerSource2.output2;
                    ControlUnit2.input4 := AirDataComputer1.output2;
                    ControlUnit2.input5 := AirDataComputer2.output2;
                    ControlUnit2.input6 := PositionTransducer1.output;
                    ControlUnit2.input7 := FlapPositionSensor1.output;
                    ControlUnit2.input8 := FlapPositionSensor2.output;
                    ControlUnit2.input9 := FlapPositionSensor3.output;
                    ControlUnit2.input10 := FlapPositionSensor4.output;
                    EICAS.input1 := ControlUnit1.output2;
                    EICAS.input2 := ControlUnit2.output2;
                    DCMotor1.input1 := ControlUnit1.output1;
                    DCMotor1.input2 := mainBUS1.output1;
                    DCMotor1.input3 := mainBUS2.output1;
                    DCMotor2.input1 := ControlUnit2.output1;
                    DCMotor2.input2 := mainBUS1.output2;
                    DCMotor2.input3 := mainBUS2.output2;
                    SpeedSumGear.input1 := DCMotor1.output;
                    SpeedSumGear.input2 := DCMotor2.output;
                    BranchGear.input := SpeedSumGear.output;
                    TorqueTube1.input := BranchGear.output1;
                    TorqueTube10.input := BranchGear.output2;
// For Position Transducers
                    TorqueTube9.input := FlapActuator4.output3;
                    TorqueTube18.input := FlapActuator8.output3;
                    PositionTransducer1.input := TorqueTube9.output;
                    PositionTransducer2.input := TorqueTube18.output;
// For Flap Position Sensors
                    FlapPositionSensor1.input := FlapActuator1.output3;
                    FlapPositionSensor2.input := FlapActuator2.output3;
                    FlapPositionSensor3.input := FlapActuator3.output3;
                    FlapPositionSensor4.input := FlapActuator4.output2;
                    FlapPositionSensor5.input := FlapActuator5.output3;
                    FlapPositionSensor6.input := FlapActuator6.output3;
                    FlapPositionSensor7.input := FlapActuator7.output3;
                    FlapPositionSensor8.input := FlapActuator8.output2;
```

Figure D.4 - Flap System Block of L2 in AltaRica Safety Model - Part II

```
// Right hand side
                BevelGear1.input := TorqueTube1.output;
                TorqueTube2.input := BevelGear1.output;
                TorqueTube3.input := TorqueTube2.output;
                BevelGear2.input := TorqueTube3.output;
                FlapActuator1.input := BevelGear2.output;
                TorqueTube4.input := FlapActuator1.output1;
                FlapPanel1.input1 := FlapActuator1.output2;
                FlapActuator2.input := TorqueTube4.output;
                TorqueTube5.input := FlapActuator2.output1;
                FlapPanel1.input2 := FlapActuator2.output2;
                BevelGear3.input := TorqueTube5.output;
                TorqueTube6.input := BevelGear3.output;
                FlapActuator3.input := TorqueTube6.output;
                TorqueTube7.input := FlapActuator3.output1;
                FlapPanel2.input := FlapActuator3.output2;
                WingTipBrakeR.input := TorqueTube7.output;
                TorqueTube8.input := WingTipBrakeR.output;
                FlapActuator4.input := TorqueTube8.output;
                FlapPanel3.input := FlapActuator4.output1;
// Left hand side
                BevelGear4.input := TorqueTube10.output;
                TorqueTube11.input := BevelGear4.output;
                TorqueTube12.input := TorqueTube11.output;
                BevelGear5.input := TorqueTube12.output;
                FlapActuator5.input := BevelGear5.output;
                TorqueTube13.input := FlapActuator5.output1;
                FlapPanel4.input1 := FlapActuator5.output2;
                FlapActuator6.input := TorqueTube13.output;
                TorqueTube14.input := FlapActuator6.output1;
                FlapPanel4.input2 := FlapActuator6.output2;
                BevelGear6.input := TorqueTube14.output;
                TorqueTube15.input := BevelGear6.output;
                FlapActuator7.input := TorqueTube15.output;
                TorqueTube16.input := FlapActuator7.output1;
                FlapPanel5.input := FlapActuator7.output2;
                WingTipBrakeL.input := TorqueTube16.output;
                TorqueTube17.input := WingTipBrakeL.output;
                FlapActuator8.input := TorqueTube17.output;
                FlapPanel6.input := FlapActuator8.output1;
```

Figure D.5 - Flap System Block of L2 in AltaRica Safety Model - Part III

Appendix E

Appendix E presents the following figures that are visualized fault trees used for FTA in AltaRica. *Unannunciated loss of flap extension* and *flap panel disconnection* failure scenarios are depicted in the figures. *Annunciated/Unannunciated loss of flap retraction* cases resemble the same fault tree structure as the extension cases. Therefore, they are not shown in this section.

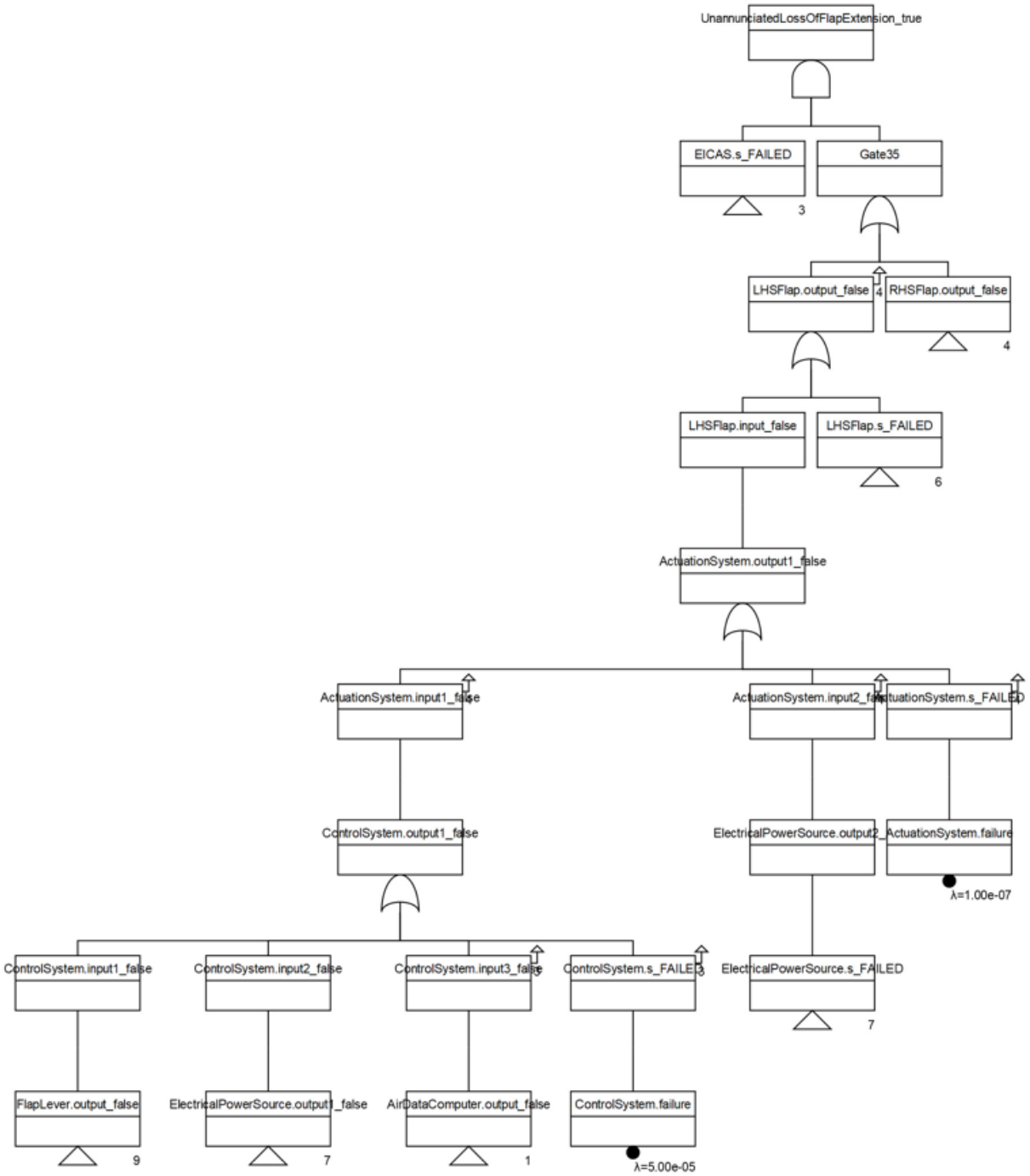

Figure E.1 - Fault Tree of Unannunciated Loss of Flap Extension at L0

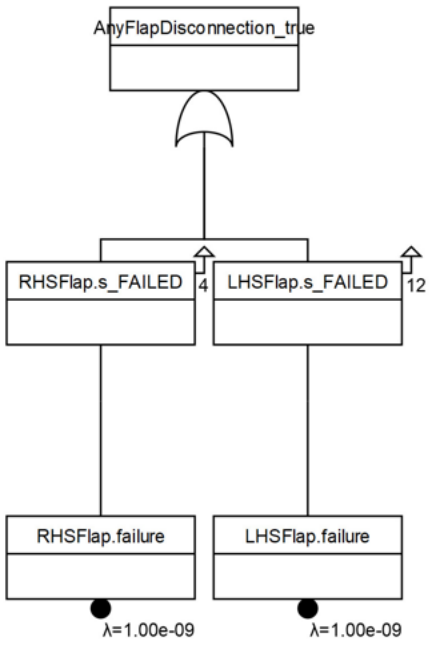

Figure E.2 - Fault Tree of Flap Panel Disconnection at L0

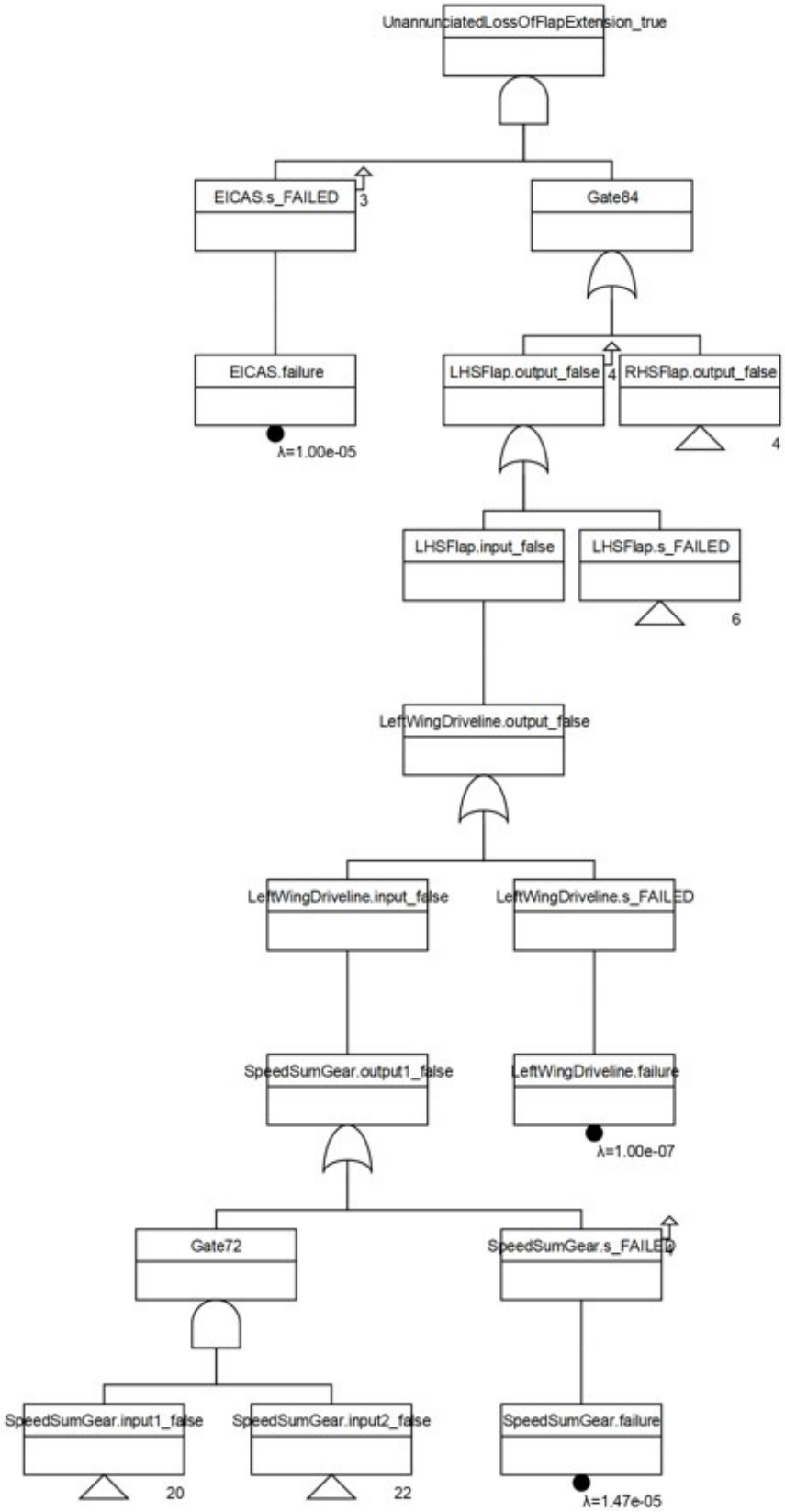

Figure E.3 - Fault Tree of Unannunciated Loss of Flap Extension at L1

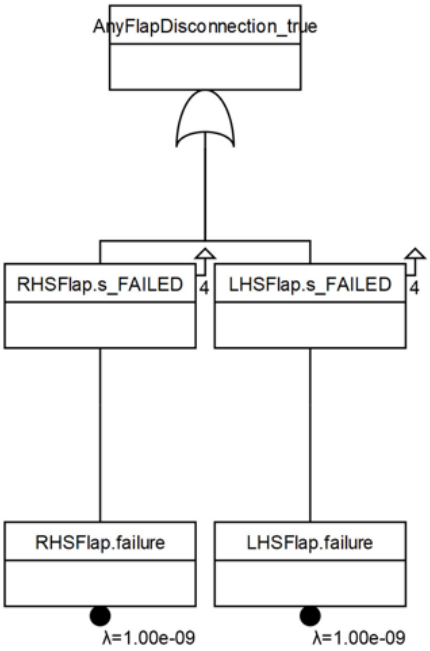

Figure E.4 - Fault Tree of Flap Panel Disconnection at L1

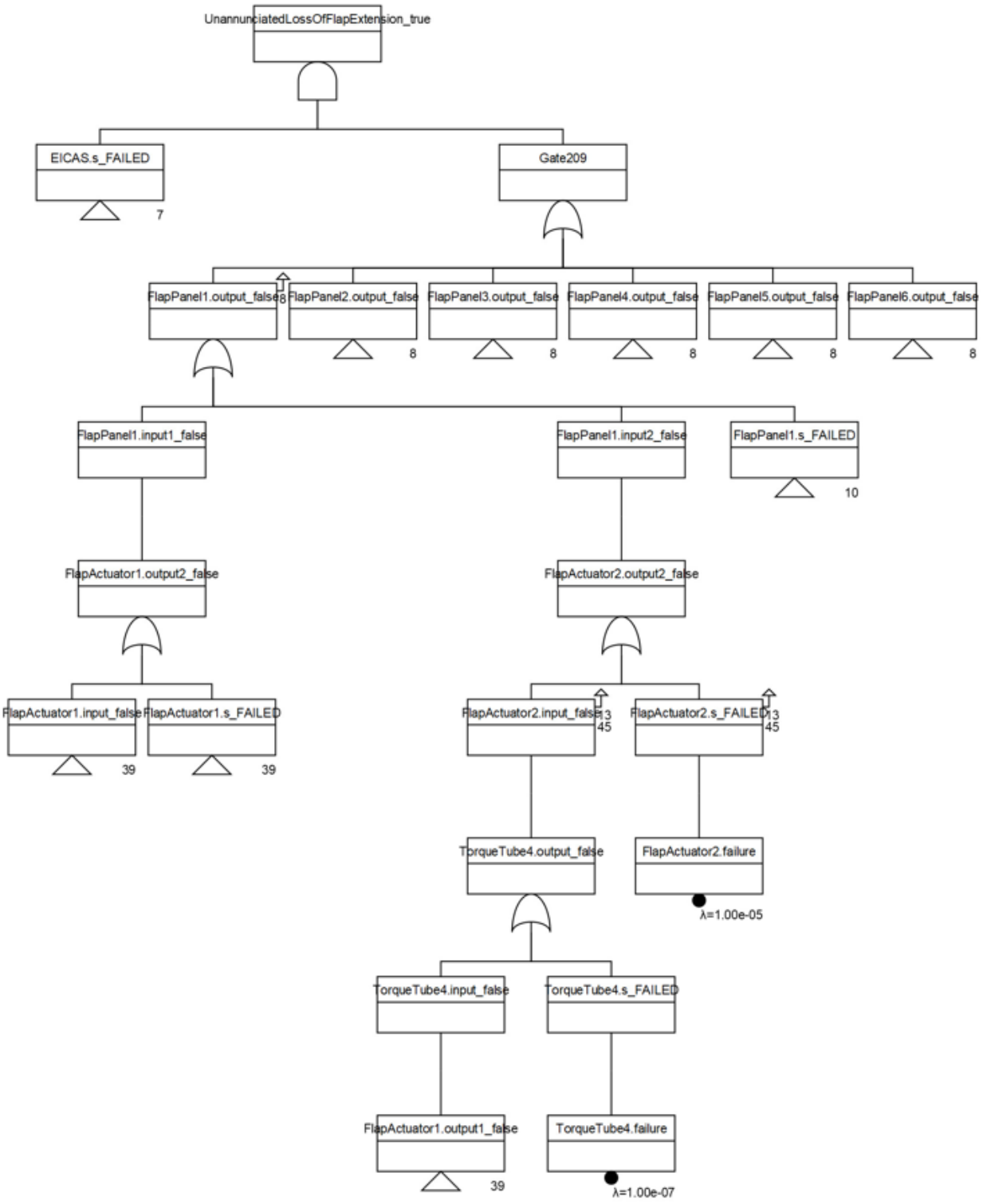

Figure E.5 - Fault Tree of Unannunciated Loss of Flap Extension at L2

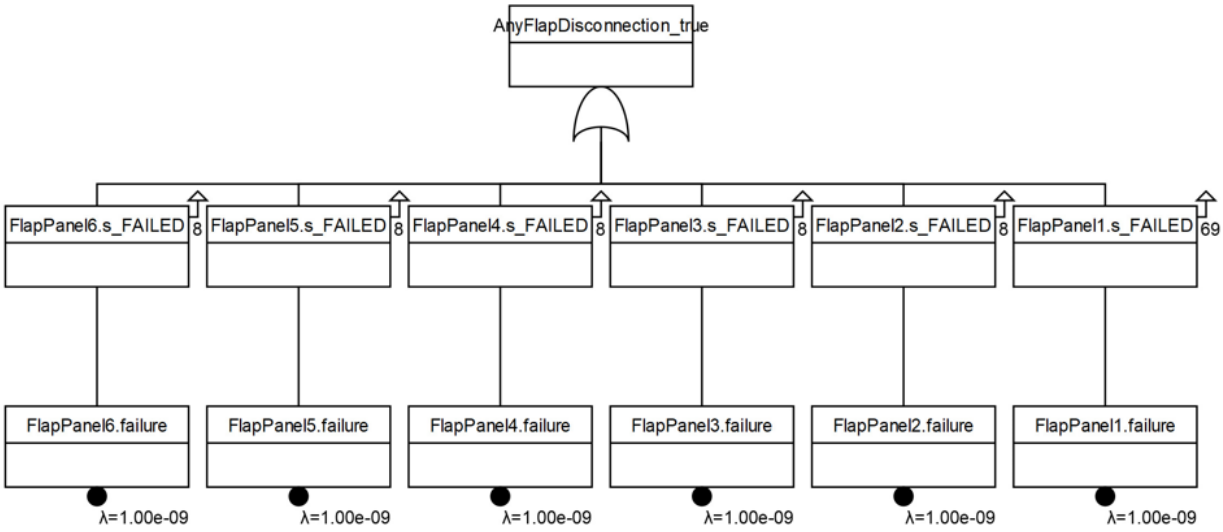

Figure E.6 - Fault Tree of Flap Panel Disconnection at L2