CRANFIELD UNIVERSITY

SERGIO JIMENO ALTELARREA

BUILDING SAFETY INTO THE CONCEPTUAL DESIGN OF COMPLEX SYSTEMS. AN AIRCRAFT SYSTEMS PERSPECTIVE.

SCHOOL OF AEROSPACE, TRANSPORT AND MANUFACTURING
Aerospace Engineering

PhD
Academic Year: 2017–2021

Supervisors: Prof. Marin D. Guenov, Dr Atif Riaz
4th June 2021

CRANFIELD UNIVERSITY

SCHOOL OF AEROSPACE, TRANSPORT AND MANUFACTURING
Aerospace Engineering

PhD

Academic Year: 2017–2021

SERGIO JIMENO ALTELARREA

Building safety into the conceptual design of complex systems. An aircraft systems perspective.

Supervisors: Prof. Marin D. Guenov, Dr Atif Riaz
4th June 2021

Abstract

Safety is a critical consideration during the design of an aircraft, as it constrains how primary functions of the system can be achieved. It is essential to include safety considerations from early design stages to avoid low-performance solutions or high costs associated with the substantial redesign that is commonly required when the system is found not to be safe at late stages of the design. Additionally, safety is a crucial element in the certification process of aircraft, which requires compliance with safety requirements to be demonstrated.

Existing methods for safety assessment are limited in their ability to inform architectural decisions from early design stages. Current techniques often require large amounts of manual work and are not well integrated with other system engineering tools, which translates into increased time to synthesise and analyse architectures, thus reducing the number of alternative architectures that can be studied. This lack of timely safety assessment also results in a situation where safety models evolve at a different pace and become outdated with respect to the architecture definition, which limits their ability to provide valuable feedback.

Within this context, the aim is to improve the efficiency and effectiveness of design for safety as an integral part of the systems architecting process. Three objectives are proposed to achieve the stated aim: automate and integrate the hazard assessment process with the systems architecting process; facilitate the interactive introduction of safety principles; and enable a faster assessment of safety and performance of architectures. The scope is restricted to the earlier (conceptual) design stages, the use of model-based systems engineering for systems architecting (RFLP paradigm) and steady-state models for rapid analysis.

Regarding the first objective, an enabler to support the generation of safety requirements through hazard assessment was created. The enabler integrates the RFLP architecting process with the System-Theoretic Process Analysis to ensure consistency of the safety assessment and derived safety requirements more efficiently.

Concerning the second objective, interactive enablers were developed to support the designer when synthesizing architectures featuring a combination of safety principles such as physical redundancy, functional redundancy, and containment. To ensure consistency and reduce the required amount of work for adding safety, these methods leverage the ability to trace dependencies within the logical view and between the RFLP domains of the architecture.

As required by the third objective, methods were developed to automate substantial parts of the creation process of analysis models. In particular, the methods enable rapid obtention of models for Fault Tree Analysis and subsystem sizing considering advanced contextual information such as mission, environment, and system configurations.

To evaluate this research, the methods were implemented into AirCADia Architect, an object-oriented architecting tool. The methods were verified and evaluated through their applications to two aircraft-related use cases. The first use case involves the wheel brake systems and the second one involves several subsystems. The results of this study were presented to a group of design specialists from a major airframe manufacturer for evaluation. The experts concluded that the proposed framework allows architects to define and analyse safe architectures faster, thus enabling a more effective and efficient design space exploration during conceptual design.

Keywords: Design for Safety; Aircraft Conceptual Design; Model-Based Systems Engineering; Systems-Theoretic Process Analysis (STPA); Safety Principles; Fault Tree Analysis (FTA); Aircraft Systems Sizing.

Contents

List of Figures

List of Tables

List of Abbreviations

AADL	Architecture Analysis & Design Language
APU	Auxiliary Power Unit
ASA	Aircraft Safety Assessment
BFS	Breadth-First Search
BSCU	Brake System Control Unit
CAB	Cabin
CAST	Causal Analysis Based on Systems Theory
CCA	Common Cause Analysis
CMD	Command
CS	Certification Specifications
DFS	Depth-First Search
DMM	Domain Mapping Matrix
DRM	Design Research Methodology
DSM	Design Structure Matrix
ECS	Environmental Control System
ELAC	Elevator Aileron Computer
ENG	Engine
EPS	Electrical Power System
ETA	Event Tree Analysis
FAC	Flight Augmentation Computer
FAR	Federal Aviation Regulations

FCS	Flight Control System
FHA	Functional Hazard Assessment
FMEA	Failure Modes and Effects Analysis
FMECA	Failure Modes, Effects, and Criticality Analysis
FMES	Fault Modes and Effects Summary
FRAM	Functional Resonance Analysis Methods
FTA	Fault Tree Analysis
HAZOP	Hazard and Operability
HYD	Hydraulic System
IBD	Internal Block Diagram
IDE	Integrated Development Environment
MDM	Multiple Domain Matrix
MOCUS	Method Of Cut Sets
MON	Monitor
PASA	Preliminary Aircraft Safety Assessment
PNE	Pneumatic System
PSSA	Preliminary System Safety Assessment
PSU	Power Supply Unit
RBD	Reliability Block Diagrams
RCA	Root Cause Analysis
RCM	Reliability Configuration Model
RFLP	Requirements, Functional, Logical and Physical
SAE	Society of Automotive Engineers
SAHRA	STPA based Hazard and Risk Analysis
SATM	School of Aerospace, Transport and Manufacturing
SEC	Spoiler Elevator Computer
SSA	System Safety Assessment
STAMP	Systems-Theoretic Accident Model and Processes

STPA	System Theoretic Process Analysis
SysML	Systems Modeling Language
UI	User Interface
UML	Unified Modeling Language
WBS	Wheel Brake System
XSTAMPP	Extensible STAMP Platform

Acknowledgements

This has been a long and intense journey from the beginning of my PhD to this moment when my thesis is complete. I want to express my gratitude to the many people that have helped me as I would not have been able to make it on my own.

First, I would like to thank my supervisors Professor Marin D. Guenov and Dr Atif Riaz for their guidance and patience throughout this research, and for their constructive criticism, which was not always easy to assimilate but it has definitely improved my research. I also extend my gratitude to Dr Arturo Molina-Cristóbal, who was my supervisor during the first part of my PhD.

I would also like to thank the past and present colleagues from our research group for the good moments we spent and their feedback, which has helped me grow as a researcher. Thank you, Xin Chen, Stevan van Heerden, Yogesh Bile, Soufiane El Fassi and Gabriele Mura.

A special thank you to the design specialists from Airbus UK who helped me to evaluate my research.

I also need to express my gratitude to Teresa Townsend for helping me to see things from a different angle and understand myself a little bit better. I would also like to thank Petra Briggs for her efficient support on the more administrative side and to the rest of Cranfield staff that make possible our research.

Last but not least, I would like to thank my family for their support and encouragement during my PhD years. I need to mention especially the love and support from my partner Kexin Zhou, who always stood by my side even in the most challenging parts of the journey. I would not have come this far without you.

Chapter 1

Introduction

1.1 The Importance of Safety at Conceptual Design

Design can be defined as the set of activities that ‘develop a product from a need, product idea or technology to the full documentation needed to realise the product and to fulfil the perceived needs of the user and other stakeholders’ [1]. One particularly important kind of need is safety, which aims to limit the undesired behaviours of the system that can result in catastrophic effects such as injuries, loss of life and environmental damage, provided that there is no ill intent from the users of the product. Safety limits the way products can be designed, as it excludes those designs that might result in any catastrophic effects but that might be otherwise successful in fulfilling the rests of the need.

Aircraft are a representative example of potentially unsafe products. Due to the physical characteristics of aircraft and their operation, which involves flying at great speed and altitude, accidents that cause many fatalities are unfortunately still possible. Thus, aircraft design must improve the ability to transport passengers and cargo more economically while doing so in a safe manner. Regulatory agencies such as the Federal Aviation Administration in the United States and the European Aviation Safety Agency in the European Union establish the minimum safety requirements aircraft must meet to obtain a certificate of airworthiness. Safety efforts have succeeded in reducing the number of fatalities involving passenger and cargo operations of large aeroplanes worldwide from a yearly average of 1500 fatalities in the 1970s to a yearly average below 500 fatalities in the last decade [2] — while the number of revenue passenger kilometres[*] increased tenfold in the same period [3]. However, there is still room for improvement. The use of innovative solutions such as novel configurations and increasingly more electric aircraft introduce new challenges. This is because much of the past experience regarding safety might not be applicable to the new designs.

Regarding aircraft design, the conceptual stage is considered a critical design phase as decisions in this phase can have a great impact in later design stages. Product cost is, to a great extent, 'committed early in the design process and spent late in the process' [4, p. 5]. For example, unsound decisions regarding safety might be discovered at later design stages where lack of design freedom might require that safety is achieved at the expense of performance, or to require substantial redesign, with its associated high costs. Furthermore, safety problems might remain undetected until aircraft enter into service and be discovered later when accidents happen. This is the case of the Boeing 737 MAX airliners [5] that were grounded worldwide for over a year due to problems with their Maneuvering Characteristics Augmentation system, which resulted in 346 fatalities in the crashes of Lion Air Flight 610 in October 2018 [6] and Ethiopian Airlines Flight 302 in March 2019 [7]. To avoid these kinds of undesired situations, it is fundamental to have the best possible support for architecting safety along with the remaining aspects of the aircraft as early as possible in the design.

[*]A revenue passenger kilometre is a measure of airline traffic; it is calculated as the number of revenue passengers multiplied by the number of kilometres flown.

1.2 Motivation

The research presented in this thesis is motivated by the limitations of existing support methods for safety architecting, which affect their ability to inform architectural decisions from the early design stages. These limitations, which are presented in more detail next, are used to formulate the aim and objectives of the research in Section 1.3.

1.2.1 Limited Support for Hazard Assessment and Safety Analysis

Hazard assessment and safety analysis techniques generally require specific models, which highlight particular aspects of the system, to be created. These models are subsequently analysed using different techniques, obtaining the desired safety information as a result. For instance, fault tree analysis requires modelling a specific fault of the system via a fault tree, which can be analysed qualitatively and quantitatively to obtain safety and reliability metrics. Current support for hazard assessment and safety analysis techniques often require large amounts of manual work to create the necessary models. Safety experts are also generally required during the model creation process. Even in the cases where significant parts of the creation process are automated, both the information required to create the models and the resulting models themselves are not well integrated with the rest of the system engineering tools.

The lack of adequate support translates into increased time to analyse architectures and obtain their relevant safety characteristics, which limits the number of architectures that can be studied in a given period of time, and potentially reduces the quality of the design space exploration. Long analysis times also affect other kinds of studies, such as determining the impact on safety that a modification of a baseline design has. The necessity of expert cooperation together with the lack of integration with system engineering tools and the difficulty of obtaining timely safety assessment increase the likelihood of safety models evolving at a different pace than the rest of architecting activities. This way, safety considerations can become outdated with respect to the architecture definition, which limits their ability to inform the design.

1.2.2 Lack of Support for Safety Architecting

Once safety requirements are established (e.g. from hazard assessment results or imposed by a certification authority), new architectures need to be proposed or the existing ones need to be modified to comply with such requirements. There exist several design patterns, referred to as safety principles, that can be applied to architectures to improve their safety characteristics and therefore comply with the stated safety requirements. However, no support methods for architecting safety principles that are relevant within the scope of this research were found. This lack of support increases the necessary time to generate candidate architectures or modify existing designs, leading to problems such as those stated in Section 1.2.1, which limit the ability to explore the design space.

1.3 Aims and Objectives

Taking into account the discussion and motivation provided in this chapter, the proposed aim of this research is:

> *To improve the efficiency and effectiveness of design for safety as an integral part of the systems architecting process.*

In order to achieve the stated aim, the following objectives are proposed:

1. Automate and integrate the hazard assessment process with the systems architecting process to allow a seamless definition of safety requirements.

2. Develop interactive methods to support the introduction of safety principles during architecture definition.

3. Automate the creation of system safety and performance models, enabling swift assessment of the candidate architectures.

The first and third objectives are proposed to overcome the limitations concerning the inadequate support for hazard assessment and safety analysis above. The lack of support for safety architecting motivates the third objective of this research.

1.4 Scope

Civil aircraft system architectures are a central element of this research. However, the proposed methods are not limited to this particular kind of system, which broadens the scope of the research. The focus on civil aircraft system architectures guided the literature review. It also determined which kind of systems were utilised in the use cases developed to evaluate the research. At the same time, great care was taken not to limit unnecessarily the scope of applicability of the proposed methods. The methods are expected to be applicable or easily adaptable to other types of aircraft (such as general aviation or military aeroplanes) and complex systems other than aircraft — as long as the systems can be modelled according to or be converted to the RFLP framework used to model architectures in this research, which assumes that systems architecting is distributed over the requirement, functional and logical domains (see Chapter 4 for more details).

The scope is restricted to the earlier (conceptual) design stages because it is fundamental to consider safety as early as possible in the design process. Better safety-related decisions at early stages are expected to improve system safety and mitigate the negative impact on the rest of aircraft properties, such as performance. In consonance with early design stages, the systems engineering approach employed combines a model-based approach (see Section 2.2.1) with the RFLP paradigm (see Section 2.2.2). Only steady-state models for rapid analysis were considered for sizing and determining the performance of the systems. Better safety-related decisions at early stages are expected to improve system safety and mitigate the negative impact that safety might have on other aircraft properties such as performance.

Finally, it must be noted that the research presented in this thesis is not a study to assess the safety of a particular architecture or determine which safety strategies are better. Rather, it is intended to provide the tools to enable designers to answer these questions. Therefore, the data and models used for the demonstration of the methods are meant to be realistic and of appropriate fidelity for industrial evaluation, but are not necessarily real.

1.5 Research Methodology

Blessing and Chakrabarti [1] propose a design research methodology (DRM) that aims to make design research more effective and efficient through more concrete objectives such as providing a framework for design research, helping to identify promising research areas and adequate methods, improving the rigour of the research and providing a more solid line of argumentation. As shown in Figure 1.1 DRM consists of four stages: *Research Clarification*, *Descriptive Study I*, *Prescriptive Study* and *Descriptive Study II*.

1.5.1 Research Clarification

The *Research Clarification* stage should provide an initial picture of the existing situation and the situation to be achieved at the end of the research. The aim and objectives of the research, its scope, questions, hypotheses and areas to be reviewed need to be identified. The outcomes from this stage are the existing

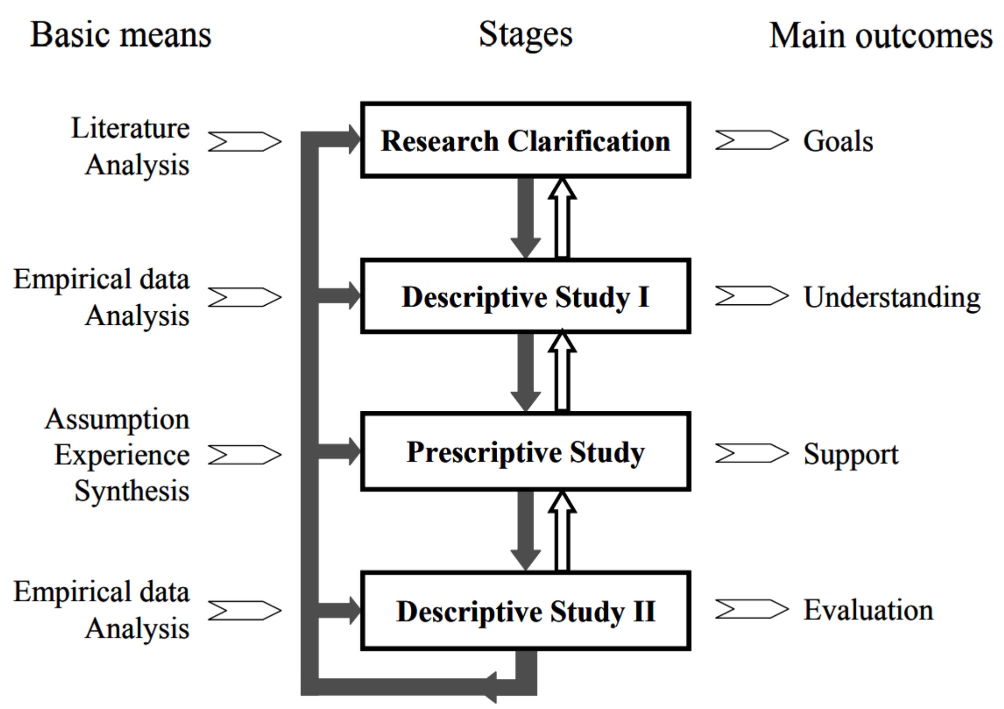


Figure 1.1: Overview of the research methodology (From [1])

situation, which corresponds to the motivation of the research, the aim and objectives provided in this chapter, and some of the ideas present at the beginning of Chapter 3.

1.5.2 Descriptive Study I

The *Descriptive Study I* stage consists of reviewing the literature, undertaking empirical research or applying reasoning to increase the understanding of design and how to achieve the stated aim. Literature review (see Chapter 3) constitutes the main activity performed during this stage of the research. The review resulted in a deeper understanding of the safety assessment process and its relation with the rest of the system development activities. Current methods for hazard assessment, safety architecting and safety and performance assessment were reviewed, and their limitations were identified.

1.5.3 Prescriptive Study

The *Prescriptive Study* develops the intended support, which is expected to address the limitations identified during the *Descriptive Study I* and help to achieve the desired situation as stated in the *Research Clarification*. Additionally, the support should be implemented to a level of detail that is sufficient to evaluate the proposed approach. In this research, methods for automating and integrating activities of the hazard assessment, safety architecting and safety and performance assessment processes are developed (see Chapter 5). The methods require the architectures to be modelled according to the RFLP framework presented in Chapter 4. The RFLP framework and methods are implemented in AirCADia Architect, a prototype software tool that is presented in Section 6.2.

1.5.4 Descriptive Study II

The *Descriptive Study II* stage focuses on the evaluation of support. According to the DRM, there are three types of evaluation.

- **Support Evaluation:** verifies that the developed support fulfils its requirements.
- **Application Evaluation:** determines the usability of the support (regarding its intended task) and whether it impacts in the desired way those factors of the situation that can be influenced.
- **Success Evaluation:** identify whether the support is useful to achieve the desired situation, considering possible side effects.

Two use cases are developed for evaluation purposes in this thesis. The methods are tested by applying them to the use cases and observing their ability to influence satisfactorily the targeted factors. One of the use cases was presented to a group of experts who provided feedback regarding the industrial usefulness of the methods.

1.5.5 Classification of this Work

Depending on the level of detail in which the stages are undertaken, design research can be classified into seven possible types. Studies are said to be review-based when they consist only of the review of the literature. By contrast, comprehensive studies include additional results produced by the researcher. All DRM stages can be review-based except the *Descriptive Study II*, which always includes original results by the researcher. The stages of *Prescriptive Study* and *Descriptive Study II* can also be categorised as initial if these studies only involve the first few steps to show the consequences of the results.

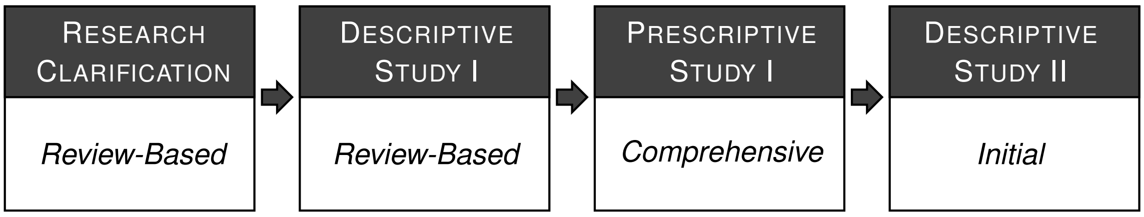


Figure 1.2: Stages of the presented research

In this research, only the *Prescriptive Study I* is conducted at a comprehensive level of detail, as shown in Figure 1.2. This situation corresponds to a 'Type 3' study according to DRM terminology.

1.6 Thesis structure

Figure 1.3 shows the main chapters of the thesis and, when applicable, relates their content to the research methodology steps discussed in the previous section.

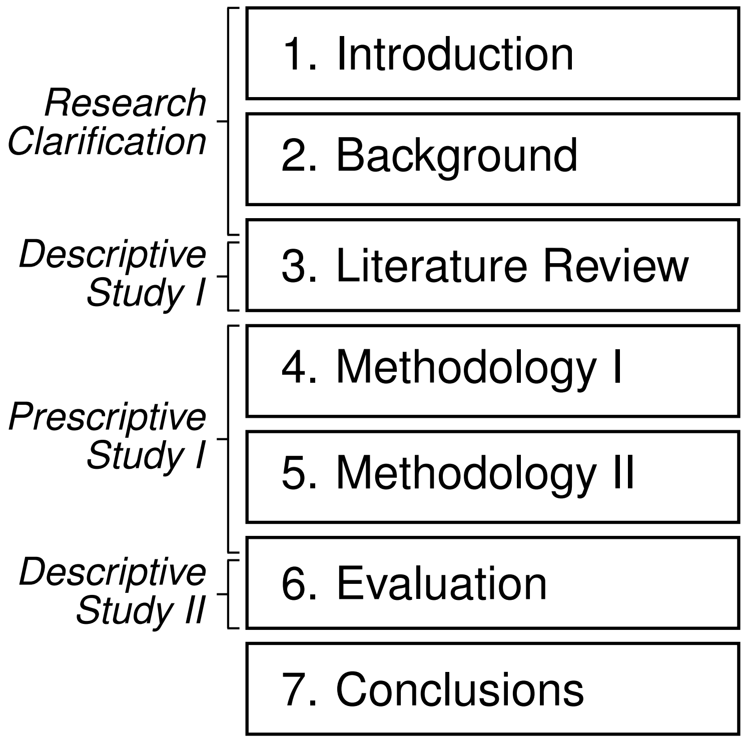


Figure 1.3: Structure of the thesis

This chapter introduces the general area and the motivation of the research, provides the aim and objectives and introduces the research methodology. Then, Chapter 2 presents an overview of background concepts such as systems engineering and functional modelling. Chapter 3 *Literature Review* discusses the concepts of safety, reliability and resilience and how they are interrelated, states the current approaches for safety assessment and identifies research gaps in the topics of hazard assessment, safety architecting and safety and performance analysis. The chapter also reviews the most relevant methods in the literature regarding such topics and identifies their limitations. The literature review corresponds to the *Descriptive Study I* and *Research Clarification* stages of the DRM.

Chapter 4 *Methodology I* proposes an RFLP framework that is used to model architectures and enables the application of the methods developed in this research. Chapter 5 *Methodology II* proposes the methods that automate and facilitate the integration of the safety assessment and system development processes. A method to automate the creation of models used by a hazard assessment technique called STPA is presented, as well as a method that automates the creation of fault trees. Support for the architecting of three safety design principles is also developed. The last contribution of the methodology chapter is a method that speeds up the sizing process considering the aircraft mission and failure conditions. Chapters 4 and 5 correspond to the *Prescriptive Study*. This stage also includes the development of AirCADia Architect, the software tool used to demonstrate the proposed support, which is presented at the beginning of Chapter 6.

The remainder of Chapter 6 demonstrates the application of the developed methods by applying them to two uses cases. Additionally, one of the uses cases was presented to a group of experts who provided feedback regarding the potential benefits that would be obtained by the application of the methods in current industrial practice. This chapter corresponds to the *Descriptive Study II*.

Finally, the conclusions of the thesis are presented in Chapter 7, which begins with a summary of the research. The contributions of the research and possibilities for future research are discussed next. The conclusions chapter is followed by Appendices A to D, which support and supplement the information provided in the main body of the thesis. Appendix E lists the publications that have resulted from this research.

Chapter 2

Background

2.1 Introduction

This chapter begins with a brief overview of the topics of systems engineering and, in particular, model-based systems engineering and the RFLP paradigm. These elements represent the foundation of the RFLP framework developed in Section 4. Then, the concept of functional basis, on which the functional modelling in this research is based, is presented in Section 2.3. Section 2.4 introduces the fundamental concepts of graph theory. This section also presents graph traversals in more detail, which are fundamental for the developed methods, and the minimum cut set problem, which is used for the containment enabler developed in Section 5.3.3. Section 2.5 introduces fault trees, which are created automatically by the methods presented in Section 5.4.1. Finally, Section 2.6 describes the characteristics of computational workflows such as the ones created by the sizing method proposed in Section 5.5.

2.2 Systems Engineering

Systems engineering is defined by the International Council on Systems Engineering as:

> *'A transdisciplinary and integrative approach to enable the successful realization, use, and retirement of engineered systems, using systems principles and concepts, and scientific, technological, and management methods [8]'*

Engineered systems are designed to interact with an anticipated operational environment to achieve their intended purposes while complying with applicable constraints [8]. Systems engineering is transdisciplinary and integrative as it requires the knowledge and methods of various disciplines to work together towards a common end (e.g. the design of the system). Defining required functionality, starting early in the development cycle, and generating and evaluating alternative solution concepts and architectures are the most relevant aspects of systems engineering within this research.

2.2.1 Model-Based Systems Engineering

Model-Based Systems Engineering consists in the formalized application of modelling to support system engineering activities; it begins in the conceptual design phase and continues throughout the rest of the design phases [9]. This model-centric approach is expected to replace the traditional document-centric approach, as it is thought to enhance productivity and quality, reduce risk, and improve communications among development teams. Modelling languages such as SysML [10] and Modelica [11], and modelling paradigms such as RFLP provide a foundation to fully specify and analyze systems.

2.2.2 RFLP Paradigm

The RFLP paradigm assumes that systems architecting is distributed over four notional domains: Requirements, Functional, Logical and Physical [12]. RFLP, which stands for Requirements engineering, Functional design, Logical design, and Physical design, is based on the German guideline *VDI 2206* [13]. According to the RFLP framework proposed by Guenov et al. [14], which is the one used in this research, each one of the views of the architecture serves a different purpose:

- **Requirements View:** displays all the architecture requirements, which are initially gathered from the stakeholder needs via requirements elicitation. Two types of requirements, functional and performance, are considered in References [15] and [14]. Requirements can be decomposed hierarchically and mapped to the functions of the system that are proposed to satisfy the requirements.

- **Functional View:** contains all the architecture functions, defined as what the system (or parts of the system) must do to meet the requirements. Like requirements, functions can also be decomposed hierarchically. Functions can be linked to requirements and components in the logical view. Two kinds of relations, function satisfaction and function derivation, exist between functions and solutions.

- **Logical View:** contains the components of the architecture, which represent the solutions selected to implement the functions. Components can also be structured hierarchically and interconnected through ports, which represent the exchange of energy, material and signal flows [14].

- **Physical View:** represents the geometry of the system, including the size and layout of the subsystems. Logical components have their corresponding physical representation in this view.

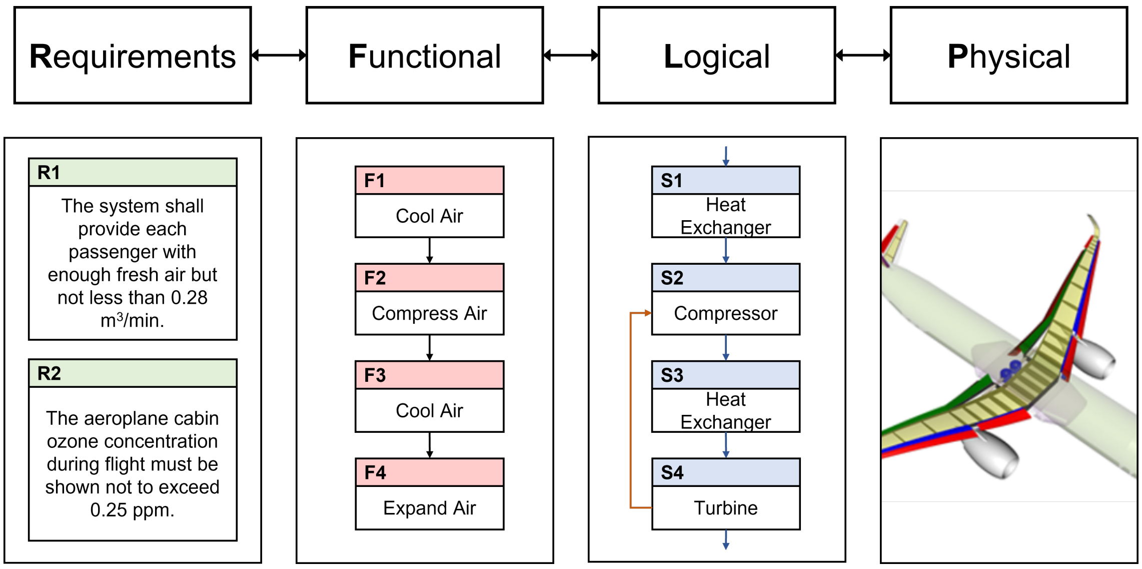

Requirements
Functional
Logical
Physical
R1
The system shall provide each passenger with enough fresh air but not less than 0.28 m³/min.
R2
The aeroplane cabin ozone concentration during flight must be shown not to exceed 0.25 ppm.
F1
Cool Air
F2
Compress Air
F3
Cool Air
F4
Expand Air
S1
Heat Exchanger
S2
Compressor
S3
Heat Exchanger
S4
Turbine

Figure 2.1: RFL and computational domains

The works by Bile [16] and Bile et al. [15] extend the RFLP paradigm with a computational domain. In these works, steady-state computational models are associated with logical components. The computational domain automates many of the processes required to orchestrate the models into computational sizing workflows. Traceability between the RFLP and computational domains is proposed by Guenov et al. [14], which facilitates a more effective and interactive design process. An overview of the RFLP and computational domains is presented in Figure 2.1, the physical view is not considered in the figure.

2.2.3 The Unified Modeling Language UML

The Unified Modeling Language [17] is a language whose purpose is to specify, visualize, and document models of software systems, including their structure and design, in a way that the system requirements can be met. UML models architectures by using thirteen types of diagrams, divided into three categories:

1. **Structure Diagrams** comprise six diagrams that represent the static application structure.

2. **Behaviour Diagrams** consist of three diagrams that represent general types

of behaviour.

3. **Interaction Diagrams** include four diagrams that represent different aspects of interactions.

The relevance of UML in this thesis is twofold. First, it serves as the foundation for SysML, a modelling language used for systems engineering and employed by some of the methods reviewed during the literature survey. Second, UML is used to describe the object model that serves as the foundation for the RFLP framework used in this research. In particular, this thesis employs a kind of structure diagram called Class Diagram.

Class Diagrams

Class diagrams describe the structure of a system with the core elements class, attribute, method, association, and composition [18]. These main elements are defined as follows:

1. **Classes** are collections of attributes and methods that determine the state and the behaviour of its instances. Classes are connected to each other by associations and inheritance. Classes are identified by their name [18].
2. **Attributes** define the stare of a class instance. Attributes are described by their name and type [18].
3. **Methods** store the functionality of a class. A method consists of a signature and a body (implementation) [18].
4. **Association** are binary relation between classes that provide structural information [18].
5. **Aggregation** is a form of association that indicates that one class owns an object of another class that can exist independently of the owner. It is indicated by an empty diamond at the owner end of the association.

6. **Composition** is a form of association that indicates that one class owns an object of another class that cannot exist independently of the owner. It is indicated by a filled diamond at the owner end of the association.

7. **inheritance** is a mechanism that allows the creation of more specialised sub-classes from an existing class. It is indicated by an empty arrow pointing at the less specialised class.

Subfigure 2.2a shows an example of the notation used to define a class in a class diagram, including its attributes and methods. Subfigures 2.2b, 2.2c and 2.2d present the notation of composition, aggregation and inheritance relations respectively.

2.3 Functional Modelling

Functional modelling is the process of creating the functional model of a system. The functional model is a description of the system in terms of the elementary functions that are required to achieve its overall function or purpose [19].

2.3.1 Functional Basis

The functional basis is a formal function representation developed to support functional modelling [20]. The functional basis is the result of reconciling and extending the previous efforts to create such a basis, namely Little, Wood and McAdams [21], Szykman, Racz and Sriram [22], and Stone and Wood [19]. The basis is expected to describe the electro-mechanical design space completely; it does so by proposing two vocabularies that can be combined to describe a function:

- **Flow vocabulary:** represents the quantities that are the inputs and outputs of functions [19]. The most common flow quantities are energy, matter and

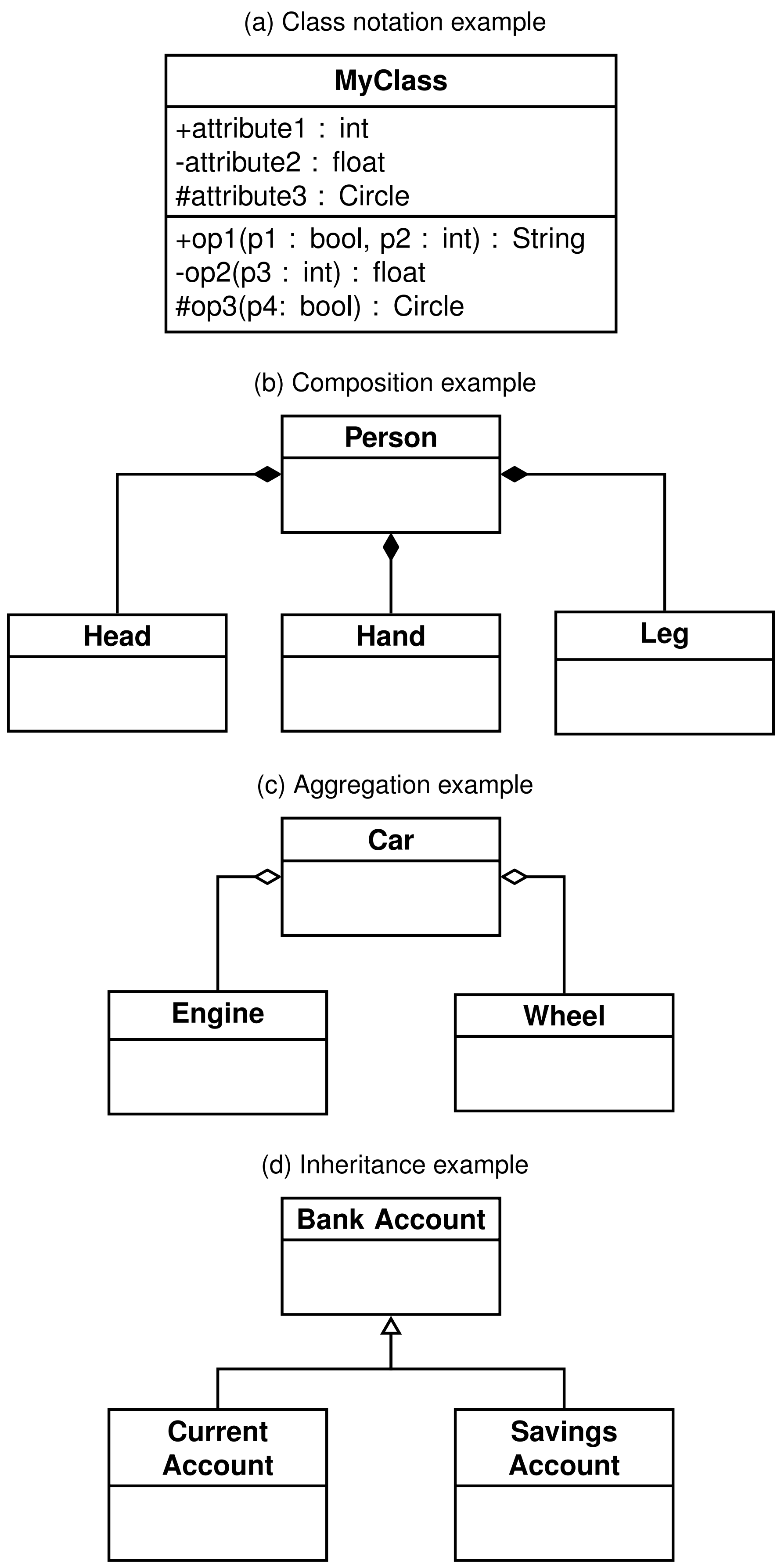


Figure 2.2: UML class diagram examples

information (signal) [23].

- **Function vocabulary:** describes the operations on a particular set of flows to be performed by a device or artefact [19]. Branch, channel, connect, control, convert and provision are some examples of terms in the function vocabulary.

Flow and function vocabularies are generally employed to respectively fill the verb and object parts of the standard verb-object function format proposed by Pahl et al. [23]. However, the basis is not necessarily restricted to the verb-object formulation and can be used with other function formats.

The terms from both vocabularies are organised hierarchically in three levels of increasing specificity. The top level is the class or primary level, which is followed by the secondary and tertiary levels [20]. The part of the flow hierarchy under the *Energy* class flow is supplemented with information about power conjugate complements, which are part of bond graphs [19].

Bond graphs a system modelling technique that considers systems as collections of subsystems. Subsystems can be interconnected through ports, which are the places at which power can flow between subsystems [24]. A bond graph consists of subsystems linked together by lines representing power bonds, which determine that the value of the power conjugates at both ends of the bond must be equal. Power conjugates consist of pairs of variables, effort and flow, whose product equals to the instantaneous power transmitted by the flow. *Force-Velocity* and *Torque-Angular Velocity* are examples of power conjugate complements.

Functional modelling in this research is done via the functional basis. Functional modelling is required to model the functions in the architecture, and therefore is fundamental for those methods that work with the functional view of the architecture. The functional basis is also employed to model the exchange of energy, material and signal flow among solutions. In particular, power conjugates are fundamental to model the energy flow exchanges and create the computational workflows (see Section 2.6) that model the architected systems.

2.4 Graph Theory

Graph theory is a central aspect of the methods developed in this research. First, this section introduces the concept of a graph. Then, graph traversals are presented and finally, the minimum cut problem is introduced.

2.4.1 Graphs

A graph G is a pair (V,E), where V is a finite set and E is a binary relation on V [25, p. 1168]. The set V contains the vertices of G and the set E contains the edges of G. A vertex v is said to be adjacent to a vertex u if the edge (u,v) belongs to $G=(V,E)$. When this relation is symmetrical, i.e. the existence of edge (u,v) implies the existence of (v,u), a graph is undirected, otherwise, a graph is said to be directed. A directed graph is shown in figure 2.3, where the circles represent the vertices of the graph and the arrows connecting the circles correspond to the edges.

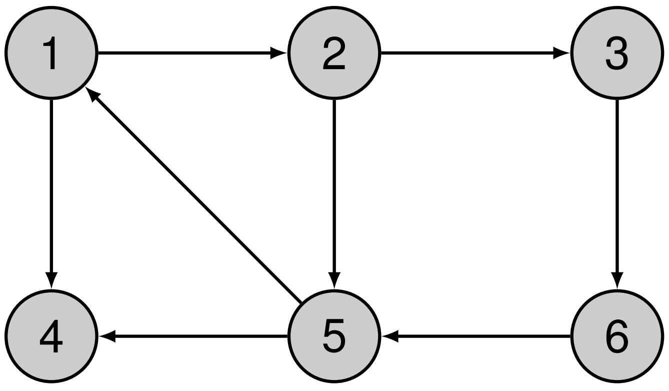


Figure 2.3: Example of a directed graph

Graphs can be used to model certain aspects of the architecture. For example, the exchange of flow among components might be modelled with a graph where vertices represent the components and edges represent the exchange of flow between two components. Furthermore, the direction of the exchange might be indicated by using a directed graph.

2.4.2 Graph Traversals

Graph traversals are the most common operations done with graphs in the current research. Different graphs model different relations among elements, either within the same view of the architecture or across views. A graph is traversed each time a method needs to know which elements are related to another one.

Given a graph $G = (V,E)$ and a source vertex s, a traversal algorithm systematically explores the edges of G to discover every vertex that is reachable from s [25, p. 594]. The order in which the vertices are explored varies depending on the algorithm. The two simplest ways to traverse a graph are Breadth-First Search (BFS) and Depth-First Search (DFS). The differences between them are:

- **Breadth-First Search:** expands the frontier between discovered and undiscovered vertices uniformly across the breadth of the frontier [25, p. 594]. The algorithm discovers all vertices at distance k from s before discovering any vertices at distance $k+1$. A BFS from vertex 1 in the graph from Figure 2.3 would discover vertices 2 and 4 first as they are at distance one from vertex 1. The traversal would discovered vertices 3 and 5 later as they are at distance two from vertex 1. Finally, vertex 6, which is at distance 3, would be discovered.

- **Depth-First Search:** explores edges leaving the most recently discovered vertex v [25, p. 603]. Once all of v's edges have been explored, the algorithm proceeds to explore edges out of the vertex from which v was discovered. A DFS from vertex 1 in the graph from Figure 2.3, unlike a BFS, might discover vertices 2, 3, 5 and 6 before vertex 4, and vertices 3 and 6 before 5.

Mathematical proofs, pseudocode, and information regarding the applications of BFS and DFS can be found in Reference [25]. The results of the traversal depend on which elements of the architecture are added to the traversed graph and how the elements are connected. The order of the results depends on the traversal algorithm. If a particular order is required by any of the developed methods, the order is explicitly indicated. In other cases, either traversal algorithm can be employed.

2.4.3 Minimum Cut Problem

The understanding of the minimum cut problem is necessary for the method developed in Section 5.3.3. The concepts of flow in directed graphs and cuts of a graph are presented before introducing the problem and an algorithm for its resolution.

Flows in Directed Graphs

Given a directed graph $G = (V, E)$, each edge (u, v) can be associated with a non-negative real number $c(u, v)$, which represents the capacity of the edge [26]. In flow problems, the capacity represents the maximal amount per unit time of some quantity that can flow from u to v. A further requirement is that if E contains an edge (u, v) then there should not exist an edge (v, u) in the reverse direction [25, p. 709].

Flow graphs are modelled with the help of two special vertices: the source s and the sink t [25, p. 709]. All the flow is generated at s and eventually reaches t. If the system to be modelled has multiples sources or sinks, a dummy source connected to all sources or a dummy sink connected to all sinks is added to the graph. The flow through an edge (u, v) is represented by the flow function $f : V \times V \to \mathbb{R}$. As explained in Reference [25, p. 709], this function satisfies the capacity constraint for all pair of vertices $u, v \in V$.

$$0 \leq f(u,v) \leq c(u,v)$$

and the flow conservation condition for all vertices $u \in V \setminus s,t$

$$\sum_{v \in V} f(v,u) = \sum_{v \in V} f(u,v)$$

Figure 2.4 shows a flow graph where each edge is labelled with the actual flow $f(u,v)$ divided by the capacity $c(u,v)$.

Maximum Flow

Flow graphs are commonly employed in maximum-flow problems. Given a flow graph G with source s and sink t, a maximum-flow problem intends to find a flow of maximum value [25, p. 710]. The maximum flow, which originates at s and finishes at t, must satisfy the capacity constraints and respect the flow conservation rule.

Minimum Cut

A cut is a subset of edges that, if removed, separates G into two disjoint sets as one set of vertices cannot be reached from the other [26]. The cuts of interest in flow graphs, such as the one displayed in Figure 2.4, are those that partition V into S and $T = V \setminus S$ such that $s \in S$ and $t \in T$ [25, p. 720].

A minimum cut is a cut whose capacity is minimum over all cuts of the network [25, p. 721]. The capacity c is defined as the sum of the capacities from all edges that cross from S to T:

$$c(S,T) = \sum_{u \in S} \sum_{v \in T} c(u,v)$$

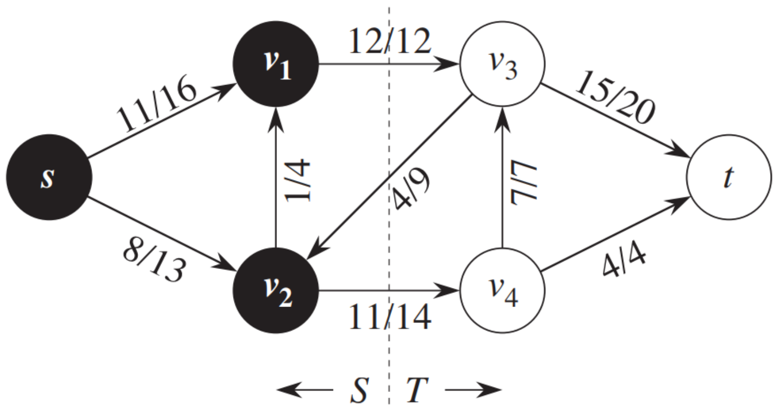


Figure 2.4: Example of a flow graph and a cut (From [25, p. 721])

The minimum cut is not necessarily unique. The minimum cut problem consists in finding a minimum cut.

Algorithms for Solving the Minimum Cut

A common approach for solving the minimum cut problem is to use algorithms to obtain the maximum flow and identify the minimum cut in a subsequent step. This is possible thanks to the max-flow min-cut theorem, which states that the value of a maximum flow is equal to the capacity of a minimum cut [25, p. 723]. A proof of the theorem can be found in Reference [25, pp. 723–724]. Given a maximum flow, the edges that belong to the minimum cut are those whose flow equal to their capacity.

The selected algorithm for computing the maximum flow in this research is the Edmonds–Karp algorithm [27]. This algorithm is simple to implement and runs with an asymptotic complexity of $O(|V||E|^2)$.

2.5 Fault Tree Analysis

Fault Tree Analysis FTA, as described in the *NUREG-0492* [28], is an inductive safety analysis technique that explains a particular system failure mode, called the top event, in terms of lower-level events. These lower-level events are combined using logical gates. Logical gates indicate how the lower-level events (inputs to the gate) produce the higher-level event (output of the gate). According to Ruijters and Stoelinga [29], the most common types of logical gates are:

- **AND:** the output event occurs when all input events happen.
- **OR:** the output event when any of the input events occur.
- K **out of** N**:** any combination that includes k of the N input events makes the output event happen.

Dynamic fault trees, which account for the temporal sequence of failures, add three new kinds of gates:

- **PAND:** stands for priority AND. This kind of gate is like an AND gate but required the input events to occur in a specified order.
- **FDEP:** emits a dummy output that never happens. However, when the first input of the gate occurs, it triggers the failure of the rest of the inputs.
- **SPARE:** represents a component that is substituted by any of the available spares when it fails. The spares are inactive until needed.

Qualitative analysis of fault trees often focuses on obtaining the minimal sets of components that can together cause the system to fail, known as minimal cut sets [29]. The minimal cut set can be used to obtain the probability of failure of the top event of the tree, which can be expressed as a function of the probability of failure of the individual components. The probability of failure of a basic event is generally modelled with an inverse exponential distribution [29]

$$\mathrm{P_f}(t) = 1 - \exp(-\lambda t)$$

which depends on the constant failure rate λ. Importance measures, which indicate what parts of a system are the biggest contributors to the failures, are another fundamental aspect of quantitative analysis. Dutuit and Rauzy [30] analyze the mathematical formulation and physical interpretation of six importance measure. The most significant two are presented next:

- **Fussell-Vesley importance factor:** measures the fraction of the system unavailability that involves the occurrence of basic event A.

  $$FV = \frac{\mathrm{P_f}(top\,|A = 1)}{\mathrm{P_f}(top)}$$

  where $\mathrm{P_f}(top|A = 1)$ is the probability of the top event given that event A does occur ($\mathrm{P_f}(A) = 1$).

- **Birnbaum importance factor:** indicates the conditional probability of the top event not happening given that the failure of A is fixed.

  $$Birnbaum = \mathrm{P_f}(top\,|A = 1) - \mathrm{P_f}(top\,|A = 0)$$

2.6 Computational Workflows

This section briefly introduces computational workflow, as understood within the context of this research. Workflows are the central element of the RFLP computational domain proposed in references [15, 16]. A computational workflow is an ordered set of computational models [31]. The models are executed according to their order, which is obtained so that every model only depends on the outputs of previous models. Figure 2.5 displays the graphical representation of a computational workflow.

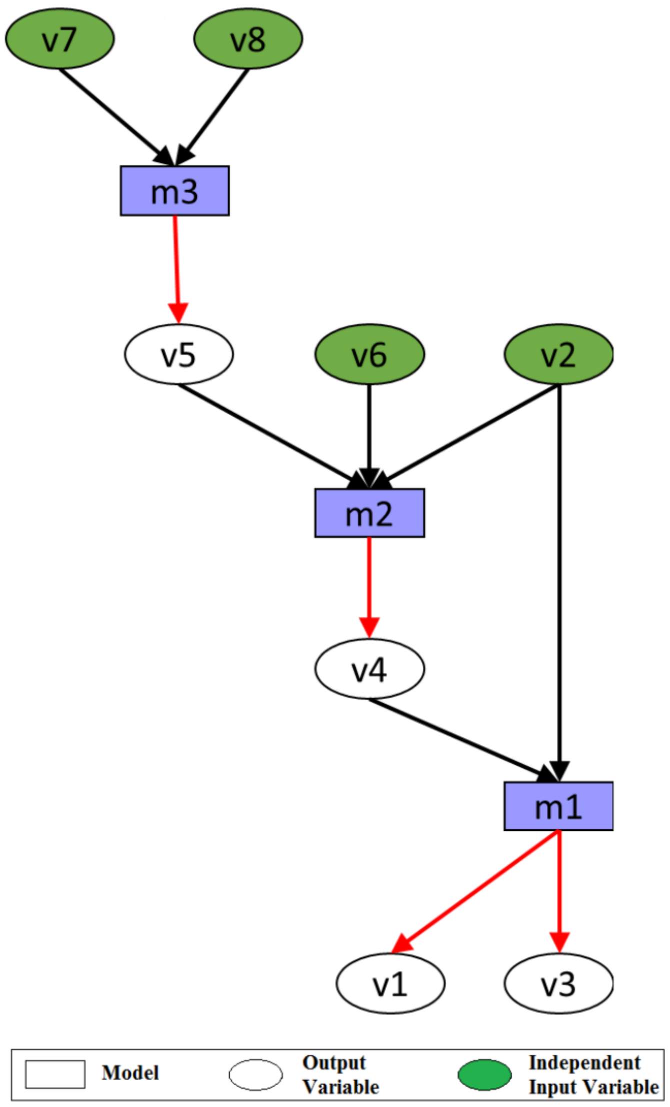


Figure 2.5: Example of a workflow (From [16])

Computational models are executable pieces of computer code that describes part of the physical behaviour and other relevant characteristics (e.g. weight or cost) of a solution in the architecture [31]. A logical component can have more than one model associated with it. Models relate input variables with outputs variables, which are updated when the model is executed according to the value of the inputs.

Models have a default direction of execution, which determines the default inputs and outputs of a model. For instance, a computational model that calculates a force using the following equation $F = ma$ has m and a as default inputs and F as default output. If any other combination of variables were known (e.g. m and F), the model could still be used to compute the remaining variable (a). However, since models are treated as black boxes, this would require the use of numerical methods to guess the value of the unknown default input (a) that provides the known original output (F) after the model is executed. A model is said to be reversed when the set of actual inputs is different from the set of default inputs.

Workflow models are scheduled according to the method proposed by Bile [16, pp. 98–130]. This method tries to match models with their inputs, which are the variable whose value is known or computed by previous models in the workflow. The method tries to assign the right number of inputs to each model and minimise the number of reversed models. A model is said to be overdetermined when it has too many inputs. In the general case of overdetermined models, the values of their inputs are in conflict and a solution cannot be found. A model is underdetermined when the number of known inputs is too low and some other inputs need to be estimated to obtain output values.

Chapter 3

Literature Review

3.1 Introduction

The research presented in this thesis involves concepts from various academic fields. The most relevant fields included safety, reliability and resilience, aircraft conceptual design and systems engineering, and graph theory, amongst others. This literature review intends both to introduce the reader to the concepts involved and to highlight the limitations and research gaps in the existing approaches to design for safety.

The review starts with a discussion of the concept of safety in Section 3.2, which is compared to the often associated concept of reliability, and the increasingly popular concept of resilience. Section 3.3 introduces the safety assessment process — relating the generic concept of safety to the particular task of aircraft design — and identifies two main kinds of activities: hazard assessment and safety analysis. Safety must be explicitly architected to obtain safe designs, which has an impact on performance that must be taken into account. Section 3.4 covers the accident models that provide the foundation for hazard and safety assessment methods such as those presented in Section 3.5. This is followed by a review of existing computational support tools for STPA, a hazard assessment method, whose limitations are identified in Section 3.6. Section 3.7 discusses the drawbacks of current methods for automating fault tree analysis, a particular kind of safety analysis technique. In Section 3.8, the architecting principles that can be implemented into aircraft designs to improve their safety are discussed. Finally, Section 3.9 presents the weaknesses of current sizing and performance methods to support the determination of the impact of safety on performance. The conclusions from the literature review are summarised in Section 3.10.

3.2 Safety, Reliability and Resilience

Safety is the central topic of this research. In practice, safety is often associated with other terms such as reliability, which is perhaps the most commonly related topic. Another important concept, especially in recent year, is resilience. This section presents definitions elaborated from those found in the literature (see Appendix A), ensuring their relevance within the scope of this thesis. The relationship between each pair of terms is also discussed.

3.2.1 Safety

There are many publications related to safety but not many of them define what safety is. Five definitions were found in the literature that was reviewed to clarify safety and related concepts; they are presented in Appendix A.1. Definition 2 can barely be considered a definition as it only says it is a system property and the rest of the quote describes what safety encompasses but not what safety is. The remaining definitions present a high degree of agreement on the fact that the focus of safety is human life and property damage. The proposed definition is as follows:

Safety: *ability of a system not to cause, under given conditions,* *critical or catastrophic events. Catastrophic event are defined as those that can cause death, injury, occupational illness, damage to or loss of equipment or property, or damage to the environment*

3.2.2 Resilience

Resilience is an increasingly popular term in the literature, the reviews by Hosseini, Barker and Ramirez-Marquez [32] and Patriarca et al. [33] clearly show the growth of the number of publications from 2000 to 2015. Resilience is being discussed in a variety of domains such as the organizational, social, economic and engineering domains [32]. There is variability with respect to definitions of resilience both between domains and within each domain. Henry and Ramirez-Marquez [34] point out the lack of standardization and rigour when quantitatively defining resilience. This adds a significant amount of ambiguity to the term and makes it difficult to understand the term precisely.

The focus of this research is on the engineering domain and, in particular, on the conceptual design of aircraft system architectures. The goal of the resilience literature review was to obtain a definition of resilience that conforms to existing publications but that it is also relevant when considering the scope of the research. The most salient definitions of resilience found in the literature have been compiled and are presented in Appendix A.2. Table 3.1 shows a summary of these 27 definitions. The table indicates with a cross 'X' whether a particular definition (or reference from which the definition is quoted) highlights one of seven selected relevant features. The four first features refer to the phases of resilience:

- **Anticipate — Avoid — Plan:** this feature refers to the actions that are taken to avoid a disruption completely before the system encounters it.
- **Absorb — Withstand — Survive:** once the disruption happens, this feature refers to the ability to limit its impact on the system.

- **Reconfigure — Adapt — React:** this refers to the activities that change the system after a disruption to facilitate the recovery of its functionality.
- **Recover:** this feature refers to the partial or complete recovery of the lost functionality or even to the improvement of its functionality after the disruption is encountered, the system survives and is reconfigured.

The last three features refer not only to definitions of resilience but also to metrics or analyses found in the documents containing the definitions:

- **Time:** it refers to either the necessity of a rapid recovery expressed in definitions or the inclusion of the time dimension in metrics and analyses.
- **Cost:** similar to time, it implies either the requirement of resilience being cost-effective in the definitions or the explicit consideration of cost in metrics and analyses.
- **Ilities:** this feature relates to the inclusion of other ilities in the literature, either to define resilience, complement it or compare it to other capabilities. These capabilities include reliability, robustness, flexibility, adaptability, reconfigurability, agility and survivability.

The last column of Table 3.1 contains the total amount of times each feature appears, which provides an idea of how frequent each feature appears in the literature. This information is also expressed more visual way, a bar chart, in Figure 3.1.

Based on the reviewed definitions, it is possible to conclude that the two most relevant phases are survival and recovery. Reconfiguration of the system is the preferred way to enable recovery after a disruption is encountered and survived; this feature is present in roughly half of the definitions. The anticipation phase is not deemed relevant in most of the references. The temporal scale of resilience is also made evident by its frequency of appearance. Although mentioned fewer

Table 3.1: Summary of resilience definitions

	Anticipate	Absorb	Reconfigure	Recover	Time	Cost	Ilities
1. Sitterle et al. [35]			x			x	
2. Patriarca et al. [33]			x	x		x	x
3. Tierney and Bruneau [36]		x	x		x		x
4. Hollnagel [37]	x		x				x
5. Woods [38]		x	x	x	x		x
6. Westrum [39]	x	x	x				
7. Madni and Jackson [40]	x	x	x	x		x	x
8. Francis and Bekera [41]	x	x	x	x	x	x	x
9. Enos [42]			x	x	x		x
10. Neches and Madni [43]			x		x	x	x
11. Neches and Madni [43]		x				x	x
12. Chalupnik, Wynn and Clarkson [44]		x		x		x	x
13. Hosseini, Barker and Ramirez-Marquez [32]				x	x		x
14. Henry and Ramirez-Marquez [34]				x	x	x	x
15. Tran et al. [45]	x	x	x	x	x		x
16. Uday, Chandrahasa and Marais [46]		x	x	x	x		x
17. Uday, Chandrahasa and Marais [46]		x		x	x		x
18. Farid [47]		x		x			x
19. Ayyub [48]	x	x	x	x	x	x	x
20. Jackson and Ferris [49]	x	x	x	x	x		x
21. Uday and Marais [50]		x	x	x	x	x	x
22. Jackson [51]		x		x	x	x	
23. Martin-Breen and Anderies [52]		x		x	x		
24. Whitson and Ramirez-Marquez [53]		x		x	x		x
25. Haimes [54]		x		x	x	x	
26. Jackson [55]		x	x	x		x	x
27. Burch [56]		x			x	x	x
28. Youn, Hu and Wang [57]	x	x		x		x	x
Total	**8**	**21**	**16**	**21**	**18**	**15**	**23**

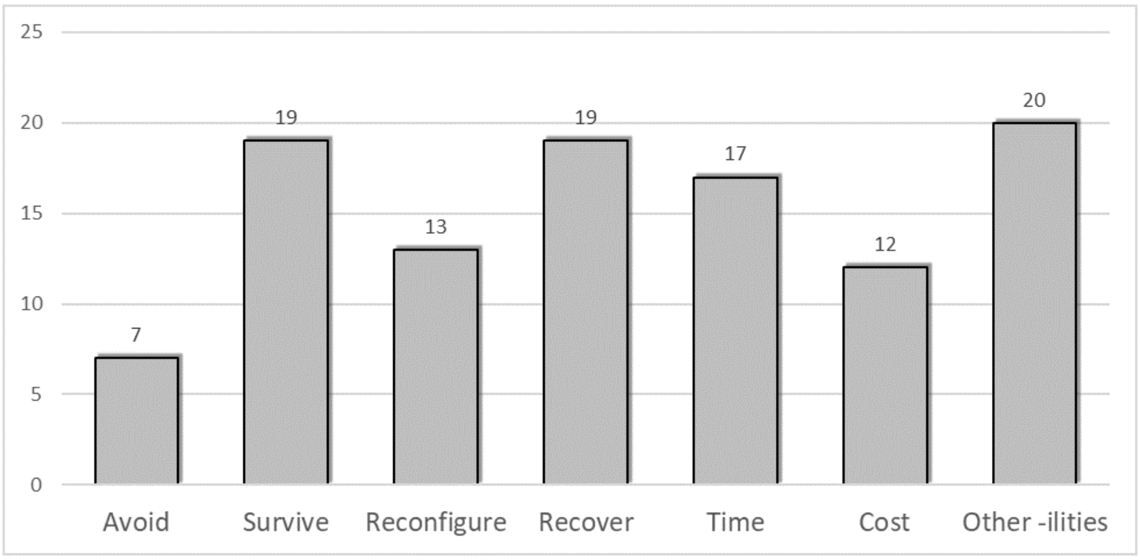


Figure 3.1: Frequency of resilience features in definitions

times, cost is also considered relevant to resilience but mainly as something that will require trade-offs in a context of limited resources more than a requirement of resilience itself. Finally, the most common of the features is the relation of the term resilience with other ilities, including safety and reliability. With all of the above considered, the following definition of resilience is proposed

Resilience: *ability of a system to maintain its functionality in the face of disruptive events. It is a dynamic capability, enabled by three sequential phases:*

1. *Survival phase: where the system survives the disruption and the objective is to limit its impact as much as possible.*
2. *Reconfiguration phase: where the necessary changes, if any, happen in the system to transition to the last phase.*
3. *Recovery phase: where functionality is recovered to the highest possible level.*

3.2.3 Reliability

Like safety, reliability is discussed in many references, but not many of them define the term. To obtain a good definition, as in the case of resilience, a table (see Table 3.2) has been elaborated with the most relevant features of the reliability definitions. These definitions are presented in Appendix A.3. In this case, the five features characterising the definitions are:

- **Low probability of failure:** indicates that the document defines reliability as a probabilistic quantity. The lower the probability of failure the higher the reliability.
- **Ability, not probability:** definitions marked with this do not constrain reliability to a probability.
- **Expected conditions:** this feature indicates that reliability is measured against disruptions expected during the design of the system.
- **Unexpected conditions:** implies that a reliable system should cope with unexpected disruptions.
- **Time period:** it shows that a definition considered that reliability is measured with respect to a predefined time period.

In Table 3.2, we can observe a greater agreement. Most authors that provided a definition, consider reliability as a probabilistic quantity, although two of them do not restrict reliability in this way. The fact that reliability deals with expected conditions is unanimous amongst the definitions despite one author also considering unexpected conditions. The fact that reliability is measured with respect to a time period presents a great consensus as well. Consequently, the proposed reliability definition is:

Table 3.2: Summary of reliability definitions

	Low probability of failure	Ability, not probability	Expected conditions	Unexpected conditions	Time period
1. Patriarca et al. [33]	x				
2. Madni and Jackson [40]		x	x		x
3. Francis and Bekera [41]					
4. Enos [42]	x		x		x
5. Chalupnik et al. [44]	x		x	x	
6. Hosseini et al. [32]					
7. Tran et al. [45]					
8. Uday and Marais [50]		x	x		x
9. Ayyub [48]					
10. Jackson and Ferris [49]					
11. Whitson and Ramirez-Marquez [53]	x		x		x
12. Jackson [55]					
13. Burch [56]					
14. ARP 4761 [58]	x		x		x
15. Leveson [59]	x		x		x
Total	**6**	**2**	**7**	**1**	**6**

Reliability: *ability of the system to perform its required functions under expected conditions for a specified period of time, generally expressed as a probability.*

3.2.4 Relation amongst Safety, Reliability and Resilience

Safety vs Reliability

According to Leveson [59], safety is often confused with reliability. If safety were equivalent to reliability, then increasing system or component reliability would necessarily increase system safety. However, this is not always true as explained next. Lack of reliability is caused by component failures, and a component is said to have failed when it is not capable to perform its intended function Leveson [59]. Therefore it is possible to have components with low reliability (e.g. the flight entertainment system in an airliner), but with little or no impact on safety as long as their failure does not lead to loss of human life, injury, property damage and so forth. Conversely, increasing the reliability of such component would not have any significant effect on safety. Systems with human operators that act unreliably (e.g. not according to procedures) but that are able to prevent accidents due to their ability to deviate from established procedures are another example of safe but unreliable systems.

The other possible case where safety and reliability are different is when systems are reliable; the probability of failure of the components is very low. But the interaction amongst system component leads to an accident. An example of this is the Mars Polar Lander accident, which crashed into the surface of Mars as its software shut down the descent engines because the noise from the landing legs was misinterpreted as a landing indication [59, 60].

The conclusion, is that reliability and safety are not equivalent nor is one a subset of the other [61]. This is shown graphically in Figure 3.2.

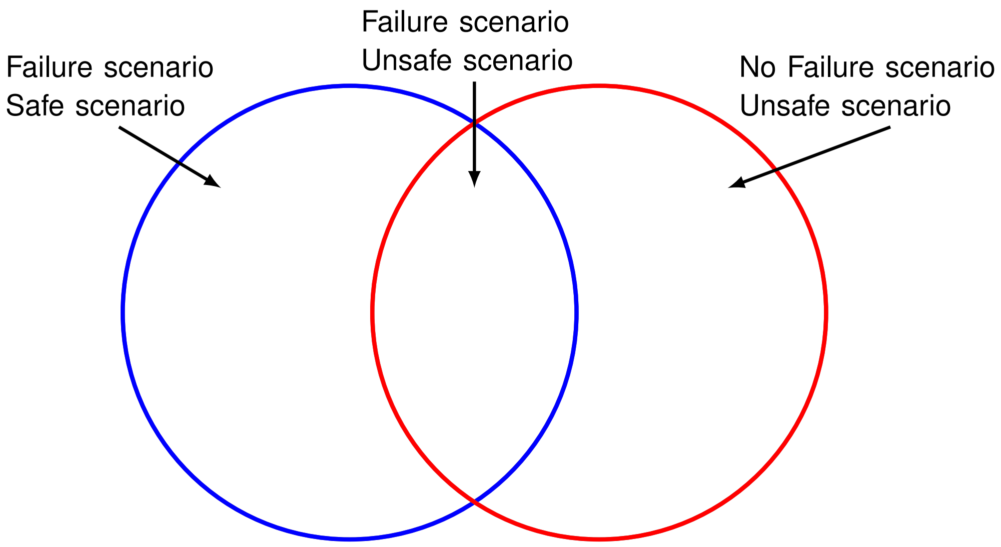


Figure 3.2: Unsafe vs unreliable scenarios (Adapted from [61])

Hollnagel and Sundström [62] discuss the relationship between safety and reliability in terms of system complexity. Probabilistic safety assessment is considered to be reasonable only for technical systems but inappropriate when considering human performance, such as in the case of systems with human or organisational functions. Probabilistic safety assessment understands safety as a probability of failure, equating safety and reliability. It can be concluded that the overlap of the two circles in Figure 3.2 is expected to be big for simple technical systems or the technical parts of more complex systems, and become smaller as more complexity is included in the analysis.

Resilience vs Reliability

The relation between resilience and reliability is established from the definitions and comments extracted from the references containing the definitions. Both the comments about the relation and reliability definitions are presented in Appendix A.3. The four most frequent kinds of relations found in the literature, as shown in Table 3.3, are:

- **Different conditions:** the main distinction between reliability and resilience

Table 3.3: Summary of reliability vs resilience in the literature

	Different conditions	Enables resilience	Anticipation	Recovery
1. Patriarca et al. [33]				x
2. Madni and Jackson [40]	x		x	
3. Francis and Bekera [41]		x		
4. Enos [42]		x		
5. Chalupnik et al. [44]	x			
6. Hosseini et al. [32]		x		
7. Tran et al. [45]	x			x
8. Uday and Marais [50]		x		
9. Ayyub [48]		x		
10. Jackson and Ferris [49]	x	x		
11. Whitson and Ramirez-Marquez [53]	x			
12. Jackson [55]	x	x		
13. Burch [56]	x			
14. ARP 4761 [58]				
15. Leveson [59]				
Total	**7**	**7**	**1**	**2**

is the type of conditions considered, either unexpected or external disruptions.

- **Enables resilience:** the literature marked with this feature considers reliability as an enabler of resilience or any of its parts or phases.
- **Anticipation:** indicates that resilience is different from reliability because it considers an anticipation phase.
- **Recovery:** specifies that resilience is different from reliability as it considers a recovery phase.

It seems that the goal of both resilience and reliability is similar: to maintain functionality in as many situations as possible and as long as possible. In this respect, resilience can be seen as a more advanced kind of reliability, which includes more phases (such as anticipation and recovery), considers a wider spectrum of disruptions and utilises new methods. In line with this view, many authors consider that reliability is one of the system properties that enables resilience, specifically the survival phase. Uday and Marais [50] bring about a valuable discussion of how greater complexities motivated the existence of resilience as something more and more different to reliability. For them, the main motivation of resilience is to overcome the limiting hypothesis of classic reliability methods. Table 3.4 summarises this idea, showing whether classic reliability methods (column *Rel.*) are applicable or, on the contrary, more advanced resilience methods are required (column *Res.*) depending on the complexity of the system.

Safety vs Resilience

The relation between safety and resilience is determined from the literature. Appendix A.2 contains definitions of safety and relevant comments from the literature about how resilience and safety are related. Table 3.5 shows which reference

Table 3.4: Relation between reliability, resilience and complexity

Level	**Reliability or resilience**	**Rel.**	**Res.**
Parts	Classic reliability methods are applicable.	x	
Components	Probability of failure determined by its constituent components.	x	
Simple Systems	No scope for reconfiguration.	x	
Complex systems	Classic reliability methods are applicable in some cases. Greater scope for reconfiguration enables resilience.	x	x
Systems of systems Human intensive systems	Classic reliability methods too limited by their hypothesis. Resilience is necessary to overcome these limitations.		x

highlights which aspect of the relationship between both terms. The most commented aspects are as follows:

- **Resilience as a new safety paradigm:** this safety paradigm focuses on systems copying with complexity and that need to balance performance with safety. It is a common view in the reviewed literature.
- **Resilience implies safety:** every author agrees on the fact that a resilient system can be generally considered safe, except Uday and Marais [50] who argue that when the focus is solely on performance resilience does not necessarily guarantee safety.
- **Resilience overcoming traditional safety limitations:** such as excessive hindsight and dependency on the calculation of failure probabilities or not considering the interaction of components.
- **Safety does not enable resilience:** Francis and Bekera [41] argue that

safety is not enough to enable resilience as it can be obtained in exchange for performance, but resilience looks to maintain a high level of performance.

- **Safety enables resilience:** Uday and Marais [50] argue that safety enables the survivability aspect of resilience, minimizing human loss and this is enough when the focus is not on performance.
- **Unexpected disruptions:** resilience is expected by some authors to be able to cope with unexpected disruptions while safety is not.
- **Anticipation and recovery:** safety does not traditionally consider anticipation to threats and recovery mechanism after the effect of disruptions.

Table 3.5: Summary of safety definitions

	Paradigm for safety	Resilience implies safety	Safety does not enable resilience	Safety enables resilience	Beyond tradition	Unpredicted disruptions	Anticipate	Recover
1. Patriarca et al. [33]	x	x						
2. Hosseini et al. [32]	x	x			x	x	x	x
3. Uday et al. [46]		x	x			x	x	x
4. Farid [47]				x				
5. Whitson et al. [53]		x					x	x
6. Burch [56]	x	x						
7. Richards et al. [63]		x			x		x	
Total	**3**	**6**	**1**	**1**	**2**	**2**	**4**	**3**

As shown in Table 3.5, resilience is frequently considered as a new safety paradigm, motivated by the growing complexity of systems and the desire of overcoming traditional safety limitations, related to traditional reliability methods. This is analogous to the view of resilience as a kind of advanced reliability. Resilience is often considered as implying safety, although Uday and Marais [50] argue that this is not true when the focus of resilience is solely on performance. The belief of resilience implying safety seems to be rooted in the confusion between reliability and safety analysed earlier in this section. Under this view, the concept of safety itself is flawed and needs to be improved by a new concept called resilience. However, the problem seems to be on the traditional reliability methods, which are not equivalent to safety.

All things considered and within the scope of this research, it is possible to conclude that rather than bringing a radically new capability to systems, resilience is an extension to reliability that incorporates the dynamic nature of systems and acknowledges their inherent complexity. Resilience can contribute to safety — in similar ways to reliability and also new dynamic ways involving reconfiguration of the system — and also to the performance of functions without an effect on safety. Therefore, resilience methods and design principles (see Section 3.8) have the potential to contribute to architecting safer systems. However, it is also necessary to understand that, as in the case of reliability, safety might require other kinds of considerations that are not necessarily included in resilience.

3.2.5 Discussion

A review of the existing literature has been used to proposed definitions for safety, reliability and resilience. Whereas the terms of safety and resilience seem to be well-established with little variability according to their definitions, the situation regarding resilience is the opposite. The most common aspects of resilience definitions were extracted and a definition that is relevant for this research was proposed.

Regarding the relationships between terms, it was observed that there is confusion between safety and reliability affecting especially the most traditional approaches. It was shown that, although related, safety and reliability are different and none of them contains the other. This confusion seems to persist in some views of resilience, which is sometimes seen as advanced reliability or advanced safety. In this research, it is proposed to consider resilience as a kind of advanced reliability that accounts for more complex, dynamic systems. Safety is the main topic of this research; therefore it is important to understand how reliability and resilience contribute to it. It is deducted that design principles for reliability and resilience can contribute to safety when they cover failure scenarios that affect safety (as they might result in a catastrophic event) but also that there are aspects of safety not covered by these principles, such as when unsafe scenarios arise due to wrong interactions rather than component failure.

3.3 Safety, Systems Design and Certification

After the rather theoretical discussion of safety and related concepts in Section 3.2, this section focuses on how safety affects the design process of complex technical systems in general and aircraft and their certification in particular. The main safety-related activities are identified as well as the various safety methods available in the literature.

A central document for designing safe aircraft systems is the ARP 4754A *Guidelines for Development of Civil Aircraft and Systems* [64]. The guidelines provided in the document were developed in the context of the CS-25 [65] (or its equivalent FAR Part 25 [66]). The document addresses the development cycle for aircraft and systems that implement aircraft functions. This kind of systems involves significant interaction with other systems within a larger integrated environment. A top-down level iterative approach for the development of the systems is assumed. Other significant parts of the development of safe aircraft, such as safety assessment or software development, revolve around the ARP 4754A but are covered in detail in other documents. Figure 3.3 displays all the guideline documents and their interrelations.

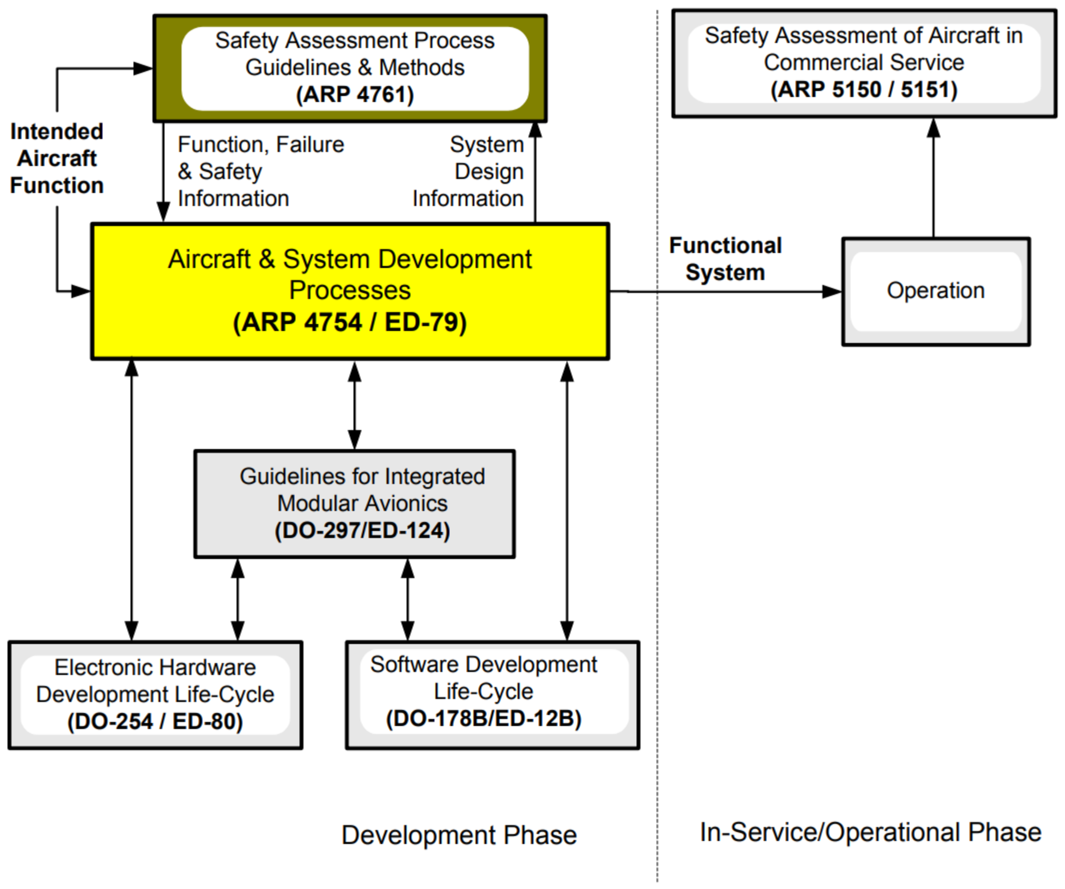

Safety Assessment Process Guidelines & Methods (ARP 4761)
Safety Assessment of Aircraft in Commercial Service (ARP 5150 / 5151)
Intended Aircraft Function
Function, Failure & Safety Information
System Design Information
Aircraft & System Development Processes (ARP 4754 / ED-79)
Functional System
Operation
Guidelines for Integrated Modular Avionics (DO-297/ED-124)
Electronic Hardware Development Life-Cycle (DO-254 / ED-80)
Software Development Life-Cycle (DO-178B/ED-12B)
Development Phase
In-Service/Operational Phase

Figure 3.3: Guideline documents for the design of safe aircraft (From [64])

The development assurance process is responsible to guarantee the safety of the system, minimising failure conditions with an appropriate level of rigour. It includes validation and verification of the system requirements. ARP 4754A suggests using the classification of failure conditions provided by the safety assessment process to determine the development assurance level that corresponds to the various items and hierarchical levels of the design.

ARP 4754A understands safety requirements as a hierarchical structure derived from either functional requirements at the various hierarchical levels (aircraft, system or item), the functional decomposition and allocation process, or common cause analyses. Safety requirements inform the various design stages, contributing to the implementation of a safe system. Furthermore, compliance with safety requirements — in particular those directly imposed by the certification authority or derived from the authority's regulations — is a fundamental part of the certification process.

3.3.1 The Safety Assessment Process

ARP 4754A [64] defines the safety assessment process as the process used by a company to show compliance with certification requirements (such as the CS 25 [65]). Four primary safety assessment processes, explained in detail in the ARP 4761 [58], are proposed:

- **Functional Hazard Assessment FHA:** examines aircraft and system functions to identify potential functional failures and classifies the associated hazards.
- **Preliminary Aircraft Safety Assessment PASA / Preliminary System Safety Assessment PSSA:** establish safety requirements and provide a preliminary indication of the ability to meet those safety requirements.
- **Aircraft Safety Assessment ASA / System Safety Assessment SSA:** verifies that the safety requirements established by the PASA / PSSA are met.
- **Common Cause Analysis CCA**: establishes and verifies physical and functional separation, isolation and independence requirements.

Figure 3.4 shows an overview of the safety assessment process and how its various parts interact with each other and with other system development activities. Although both safety and development processes are highly interdependent, as suggested by the many links that exist between them, safety assessment often

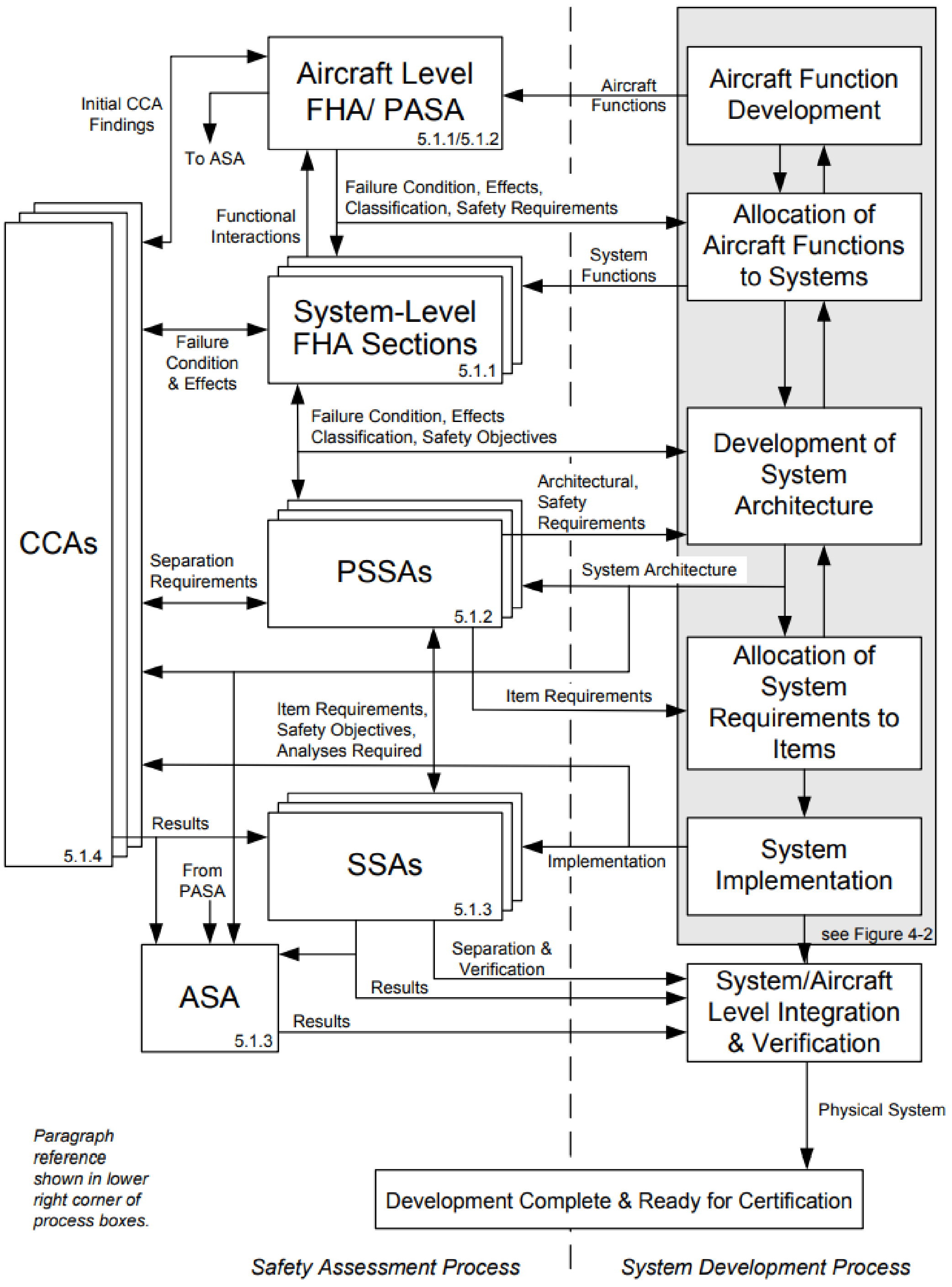

Aircraft Level FHA/ PASA 5.1.1/5.1.2
Initial CCA Findings
To ASA
Aircraft Functions
Aircraft Function Development
Failure Condition, Effects, Classification, Safety Requirements
Functional Interactions
Allocation of Aircraft Functions to Systems
System Functions
System-Level FHA Sections 5.1.1
Failure Condition & Effects
Failure Condition, Effects Classification, Safety Objectives
Development of System Architecture
Architectural, Safety Requirements
CCAs 5.1.4
Separation Requirements
PSSAs 5.1.2
System Architecture
Allocation of System Requirements to Items
Item Requirements
Item Requirements, Safety Objectives, Analyses Required
Results
SSAs 5.1.3
Implementation
System Implementation
From PASA
see Figure 4-2
ASA 5.1.3
Separation & Verification
Results
Results
System/Aircraft Level Integration & Verification
Physical System
Paragraph reference shown in lower right corner of process boxes.
Development Complete & Ready for Certification
Safety Assessment Process
System Development Process

Figure 3.4: Safety assessment process overview (From [64])

lags behind other aspects of the design [67], and sometimes takes place later on in the product development process after the design is finalized [68]. Moreover, many of the methods are generally performed manually, which brings unnecessary subjectivity and lack of consistency to the analyses [69] and increase the probability of mistakes being committed [70]. Lack of consistency between analysis at different design stages or different design abstraction levels has also been found [71].

The above-presented limitations of current safety practice motivate the search for safety methods to be automated and tightly integrated with the system development process. The various safety methods found in the literature can be classified depending on to which part of the safety assessment process contribute:

- **Hazard Assessment Methods:** are those methods that support the hazard assessment process, where the system is examined to identify safety-related risks [72]. This definition, which corresponds to the FHA, is extended in this research to include any method that can help to derive safety requirements, such as those used during PASA or PSSA. Ericson [73] estimates that there are over 100 different hazard analysis techniques (some of them represent only minor variations), and new techniques constantly appear.

- **Safety Analysis Methods:** are defined as those that can be used to verify the safety requirements of the system by analysing its architecture. Sometimes, depending on how it is used, a method can be used both for hazard assessment and safety analysis. For instance, a fault tree analysis (see Section 3.5.1) can be used to establish reliability requirements for the system components or to determine the probability of failure provided the reliability of the individual components is known. Analysis methods are commonly employed during ASA, SSA or CCA.

Table 3.6 shows an overview of the methods that are analysed in more detail in Section 3.5, classified into hazard assessment and hazard analysis methods.

3.3.2 Accident Causation Models

Accident causation models are a fundamental part of safety assessment, and more specifically of hazard assessment. Accident causation models are models that explain how accidents happen and therefore determine how the accidents are investigated, how the risk associated with existing products is assessed, and how safer systems are designed [59]. As Leveson [59] discusses, all models are abstractions that focus on what is considered to be relevant and overlook the rest. This choice, sometimes arbitrary, is critical in determining the usefulness and accuracy of the model in predicting future events.

Hollnagel [74] classifies accident causation models in three types according to their principles:

- **Simple linear models** are based on direct causality and, therefore, focus on finding specific causes and cause–effect links. The recommendations derived from these models are to eliminate causes and cause–effect links.
- **Complex linear models** explore latent conditions and hidden dependencies and intend to find combinations of unsafe acts and latent conditions. Their recommendations usually are to strengthen barriers and defences.
- **Non-linear, systemic models** are based on dynamic couplings and functional resonance (non-linear effects). Analysis derived from this kind of models consist of finding tight couplings and complex interactions. The recommendations focus on how to monitor and manage the variability in system performance.

These kinds of models, although not mutually exclusive, differ in terms of the situations for which they are effective [37]. Simpler models can be applied to simple systems, but fail to explain accident where deficiencies in the organization, management or safety culture play a fundamental role [59]. Another fundamental difference is the time scale of the processes for which they are suitable. At the simple end, a of cause-effect links — e.g. from the failure of a component until an aircraft crashes as a result of that failure — describes events that occur in seconds or minutes. However, more complex accidents — e.g. one accident involving an airline's inadequate maintenance practices due to financial pressure — are explained by decisions happening during months or years. Table 3.6 relates the accident models, which are treated in more detail in Section 3.5, with the methods that they support.

Table 3.6: Overview of safety items included in the literature review

	Safety Methods (3.5)	
Accident Models (3.4)	**Hazard Assessment**	**Safety Analysis**
Domino Models (3.4.1)	Fault Tree Analysis (3.5.1)	Fault Tree Analysis (3.5.1)
Swiss Cheese (3.4.2)	Reliability Block Diagrams (3.5.2)	Reliability Block Diagrams (3.5.2)
		Markov Analysis (3.5.3)
		Event Tree Analysis (3.5.4)
	Failure Modes and Effects Analysis (3.5.4)	
	Hazard and Operability (3.5.6)	
Systems-Theoretic Accident Model and Processes (3.4.3)	System-Theoretic Process Analysis (3.5.7)	
Rasmussen's Risk Management Framework (3.4.4)	AcciMap (3.5.8)	
Functional Resonance Analysis Method (3.5.9)	Functional Resonance Analysis Method (3.5.9)	

3.3.3 Discussion

Safety assessment for aircraft systems architectures mostly relies on processes such as the one presented in the ARP 4754A. Although the interdependency of the safety assessment and system development processes is considered in the standard, the current safety practice still presents many integration problems. It is proposed to automate and integrate safety methods with other development activities. Two main kinds of methods (reviewed in Section 3.5) are considered: hazard assessment methods to derive safety requirements and safety analysis methods to verify such requirements. Accident causation models are fundamental to understand the validity of the various safety methods. Accident models are reviewed in Section 3.4.

3.4 Accident Causation Models

3.4.1 Domino models

The Domino Model, published by Heinrich [75] in 1931, is one of the first general accident models. The original model, as depicted in Figure 3.5, consists of five pieces so that if one falls it knocks down the following one. It was later extended to include management factors by Bird and Loftus [76]. Hollnagel [74] argues that nowadays, the domino pieces do not possess a predefined meaning and just represents something that can fail. Independently of the precise meaning of each piece, the representation of accidents as predictable sequences of events is the common element of all domino models.

3.4.2 Swiss Cheese

Three main ideas influence Reason's Swiss Cheese Accident Model [78]. The first idea is the resident pathogens metaphor, which was developed by Reason

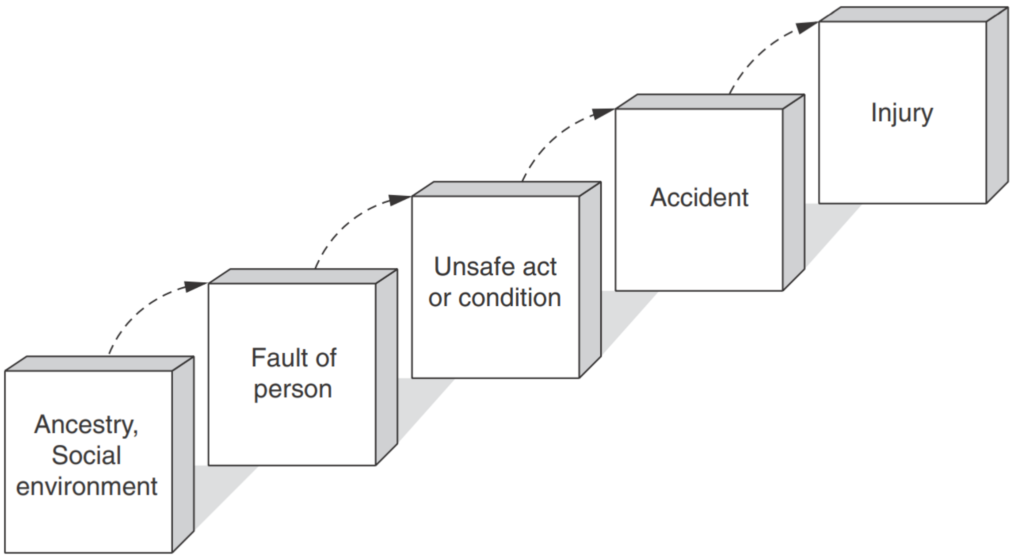


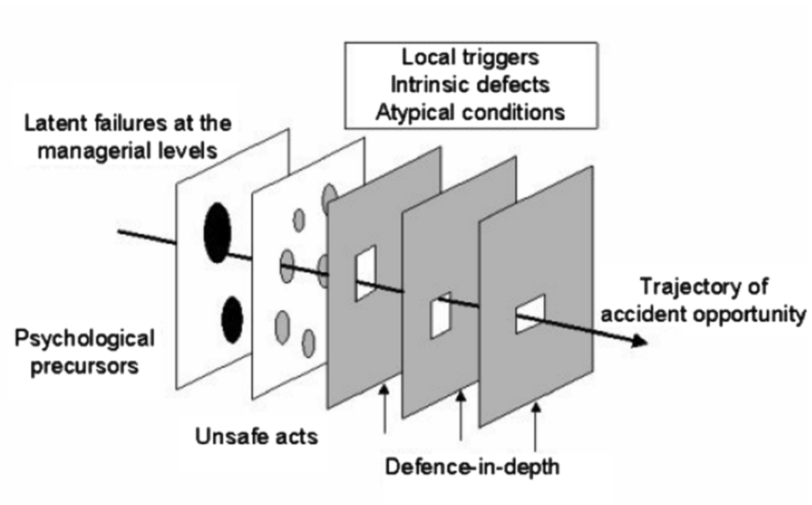
Figure 3.5: Domino accident model (From [59])

and postulates that accidents happen due to a combination of active errors and latent conditions. The second influence is the idea of defence in depth, which postulates that any productive organisation is constituted by political decision-makers, a managerial chain, preconditions (operators, equipment, future plans, maintenance, etc.), productive activities, and defences. The first two concepts materialised into the Organisational Accident Model by Reason [77] (published in 1990) shown in Figure 3.6a. Ten years later, Reason [79] adopted the third idea, the Swiss cheese analogy itself, which changed the way the model is presented, as shown in Figure 3.6b.

3.4.3 Systems-Theoretic Accident Model and Processes

Leveson's [80, 81] Systems-Theoretic Accident Model and Processes STAMP considers that accidents occur when external disturbances, component failures, or dysfunctional interactions are not adequately handled by the control system. Therefore, preventing accidents requires designing a control structure capable to enforce the necessary constraints. STAMP views systems as dynamic processes,

(a) Organisational accident model (From [77])



(b) Swiss cheese accident model (From [79])

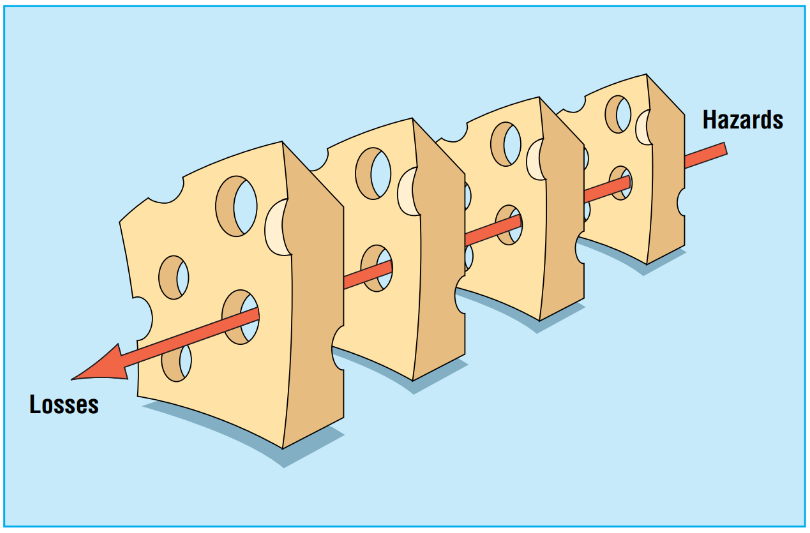


Figure 3.6: Reason's accident models

kept in a state of dynamic equilibrium by feedback loops of information and control. Due to this dynamic nature, safety management cannot be understood only in terms of preventing component failure but as a continuous control task. The basic concepts in STAMP are:

- **Constraints**, which shape the behaviour at each level of a system. Control laws impose relationships between the values of system variables to enforce the desired emergent properties such as safety.
- **Control loops and process models**. Systems are modelled as a hierarchy of control or adaptive feedback mechanisms, the control loops. Figure 3.7 exemplifies a typical control loop. Loops consist of a controller with a set of goals (enforce constraints). Controllers can affect the state of the system (actuators) based on their own model of the system (process model) which utilises information about the state of the system (sensor).
- **Levels of control**, which facilitate the modelling of complex organization or industries. Each level imposes constraints on the level beneath, based on feedback from lower levels about the effectiveness of imposed constraints.

3.4.4 Rasmussen's Risk Management Framework

The risk management framework proposed by Rasmussen [82] and Rasmussen and Svedung [83] understands accidents as a result of the drift of the system towards the boundary of acceptable performance. Once this boundary is crossed, a normal variation in somebody's behaviour can release an accident. The gradient toward least effort (reducing workload) and management pressure towards efficiency are the main factors that push the system towards the boundary. Figure 3.8 provides a visualisation of this process.

Within the framework, risk management becomes a control problem focused on maintaining the process within the boundaries. Systems are analysed in terms

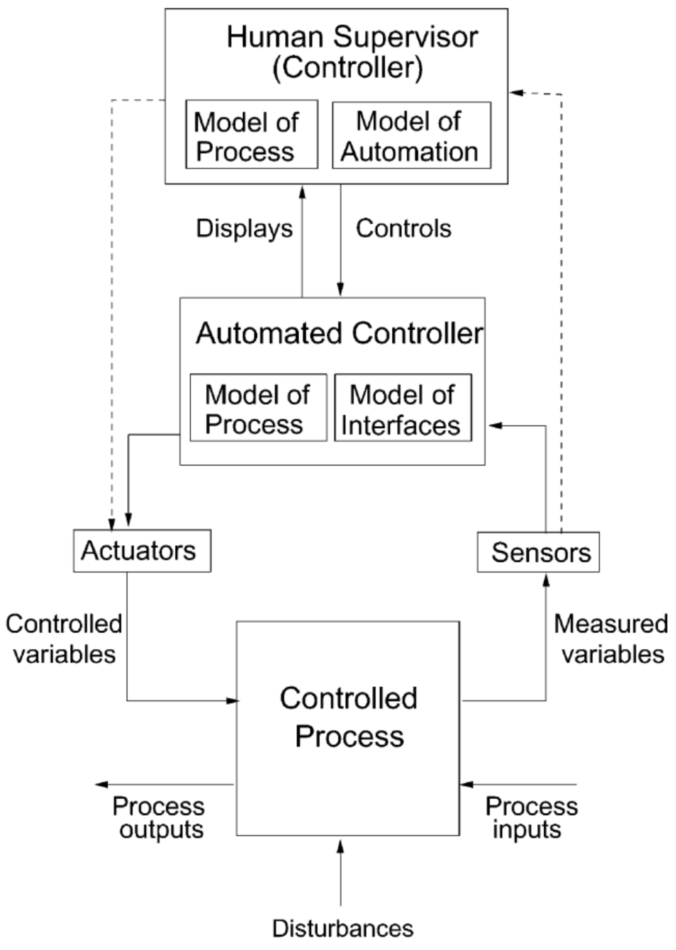


Figure 3.7: Typical STAMP accident model (From [80])

of the relational structure of the workspace, the objectives and constraints of decision-makers, and the boundaries of acceptable performance [84]. Increasing the margin from normal operation to the loss-of-control boundary, and increasing the awareness of the boundary are identified as possible means of improving safety. However, they are considered weaker than the preferred strategy, consisting of making the boundaries explicit and developing coping skills at the boundary.

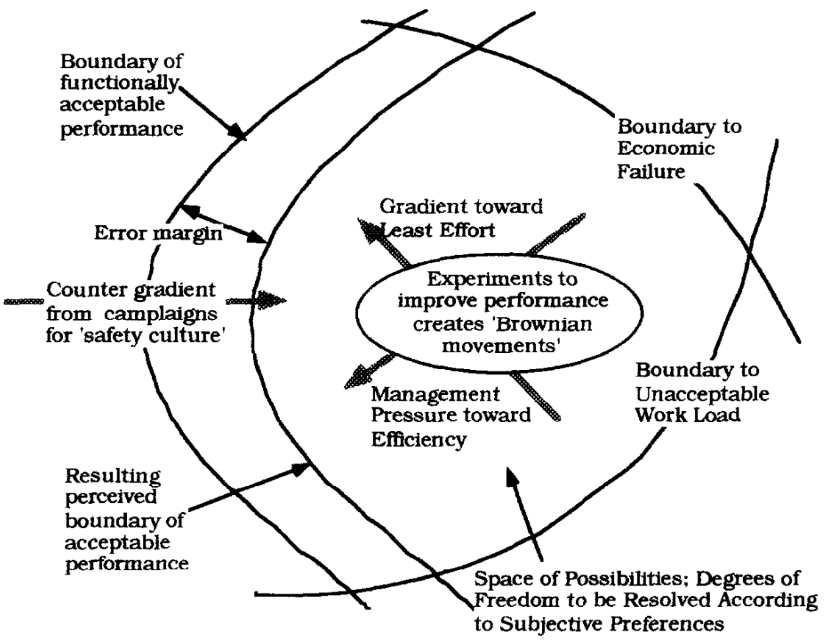

Boundary of functionally acceptable performance
Boundary to Economic Failure
Gradient toward Least Effort
Error margin
Counter gradient from camplaigns for 'safety culture'
Experiments to improve performance creates 'Brownian movements'
Boundary to Unacceptable Work Load
Management Pressure toward Efficiency
Resulting perceived boundary of acceptable performance
Space of Possibilities; Degrees of Freedom to be Resolved According to Subjective Preferences

Figure 3.8: Visualisation of system drift (From [82])

3.5 Safety Methods

3.5.1 Fault Tree Analysis

Fault Tree Analysis FTA, as described in the *NUREG-0492* [28], is an inductive safety analysis technique that explains a particular system failure mode, called the top event, in terms of lower-level events. These lower-level events are combined using logical gates. Logical gates indicate how the lower-level events (inputs to the gate) produce the higher-level event (output of the gate). For example, an *AND* gate requires every one of its input to be failed to produce a failed output, whereas an *OR* gate will produce a failed output as long as any of its inputs is failed. The tree is developed until the failure of a single component or events that cannot be developed in more detail are reached. Figure 3.9 portrays an example of a fault tree.

Fault trees can be analysed qualitatively or quantitatively. Qualitative analysis often focuses on obtaining the minimal sets of components that can together cause the system to fail, known as minimal cut sets [29]. Quantitative analysis

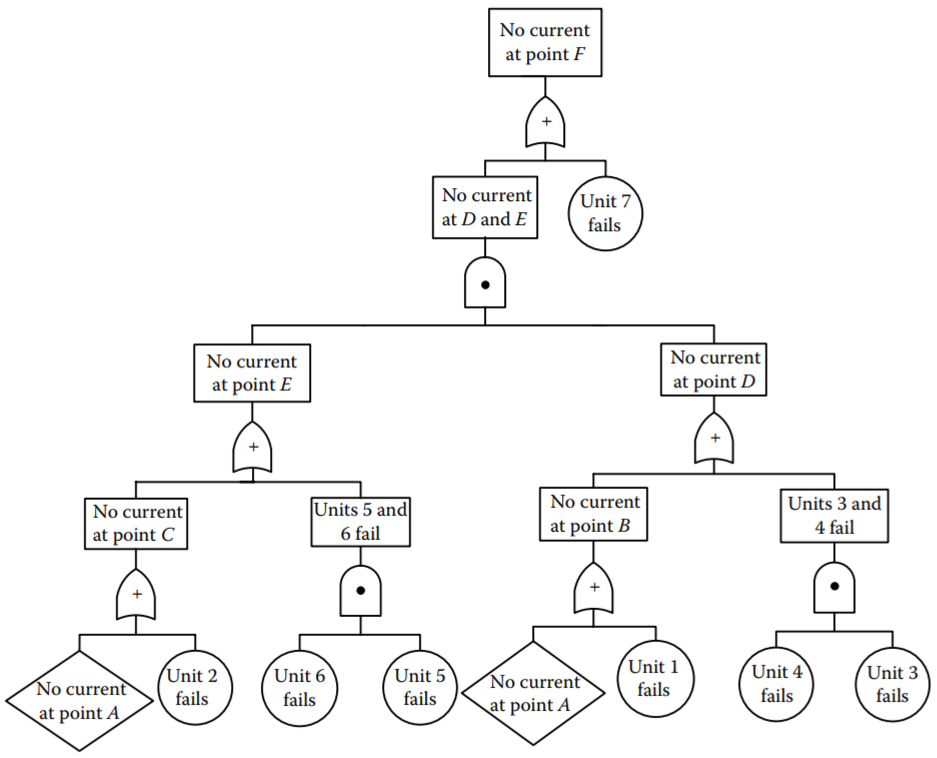


Figure 3.9: Example of a fault tree (From [85])

can provide several reliability-related metrics (such as reliability, availability or mean time to failure), as well as importance measures indicating which parts of a system are the biggest contributors to the failure. Fault trees have been extended to include temporal sequence information, as in the case of Dynamic fault trees, and other considerations such as fuzzy probabilities or repairability. For a more detailed overview of existing kinds of fault trees, analyses and tools, the reader is directed to the review paper by Ruijters and Stoelinga [29].

3.5.2 Reliability Block Diagrams

Reliability Block Diagrams RBD, also know as Dependence Diagrams (e.g in ARP 4761 [58]), are networks that describe the function of the system in terms of the logical connections of the functioning components needed to fulfil the function [86]. When every component in a set of components is required for the system to be functioning, the components in the set are connected in series. A set of components are connected in parallel when any of them suffices for the system to functioning. An example of RBD is shown in Figure 3.10.

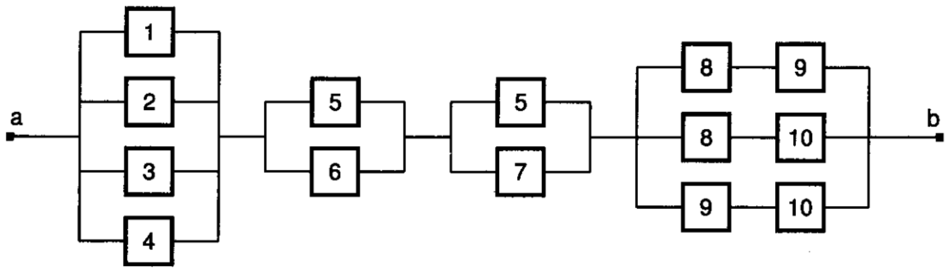


Figure 3.10: Example of a reliability block diagram (From [86])

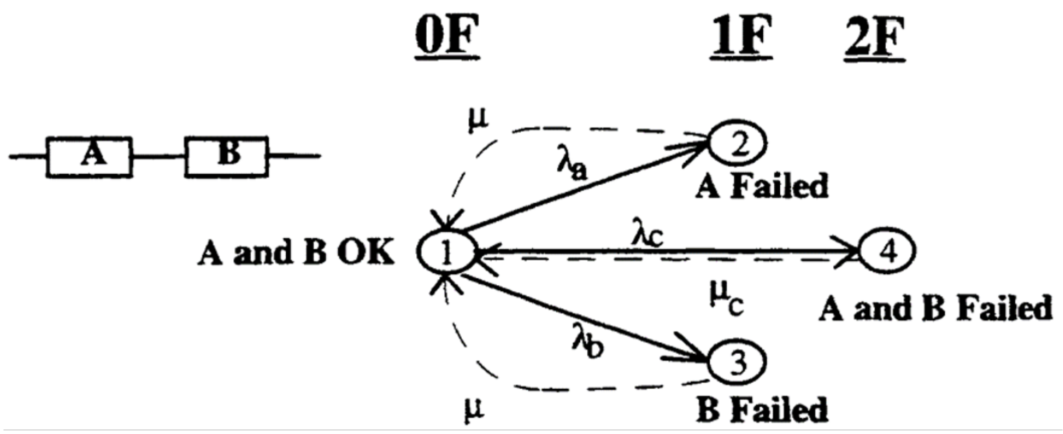
RBD are analogous to fault trees only containing *AND* gates (parallel connections) and *OR* gates (series connections), with the exception that RBD omit the intermediate events that describe the output of the fault tree's logical gates. Due to this analogy, the kind of qualitatively or quantitatively information obtained from analysis RBDs is similar to that obtained by FTA, being considered alternative methods in ARP 4761 [58].

As with FTA, more complex extensions of RBD have been developed. This is the case of Dynamic Reliability Block Diagrams [87, 88], which uses a states-events machine to model the dynamics state of components and their interdependencies (e.g. an event in one component disable or enables another one). These additional capabilities allow in turn the modelling of more complex redundancy and load sharing schemes.

3.5.3 Markov Analysis

Markov analysis models systems via Markov chains, which describe the systems in terms of several states and transitions between states [86]. Transitions from one state to another occur according to know probabilities. The Markov technique is proposed by ARP 4761 [58] as an alternate technique to FTA and RBDs, and it is claimed to be most appropriate for fault-tolerant systems, with monitoring and reconfiguration capabilities.

An example of a Markov model of a repairable series system is presented in Figure 3.11. There are four possible states, which are represented by the nodes in the figure. The states represent all possible combinations of components A and B being 'OK' or 'Failed'. The edges between state nodes model transition between nodes. The solid links correspond to the failure of components and the dashed links to the repair of components. The probability of failure events is λ_A for A, λ_B for B, and λ_C for both. A component is repaired with probability μ; both of them are repaired with probability μ_C.



$$\frac{dP_1(t)}{dt} = -(\lambda_a + \lambda_b + \lambda_c)P_1(t) + \mu(P_2(t) + P_3(t)) + \mu_c P_4(t)$$

$$\frac{dP_2(t)}{dt} = \lambda_a P_1(t) - \mu P_2(t)$$

$$\frac{dP_3(t)}{dt} = \lambda_b P_1(t) - \mu P_3(t)$$

$$\frac{dP_4(t)}{dt} = \lambda_c P_1(t) - \mu_c P_4(t)$$

Figure 3.11: Example of a Markov model (From [58])

Markov analysis assumes that the probability of transition from one state to another does not depend on the global time; it only depends on the interval of time available for transition [86]. A system of equations describing the probability of each state of the system at a point in time can be formulated and solved to obtain the probability of being in a particular state at a given time and several reliability metrics such as system availability or mean time between failures.

3.5.4 Event Tree Analysis

Event Tree Analysis ETA is a hazard analysis technique for identifying and evaluating the sequence of events in a potential accident scenario [73]. ETA utilizes a visual logic tree structure known as an event tree. An event tree starts from an initiating event. Different branches represent the success or failure of the safety methods established to prevent accidents. This way, different outcomes can be studied along with their probabilities. Figure 3.12 shows an example of an event tree.

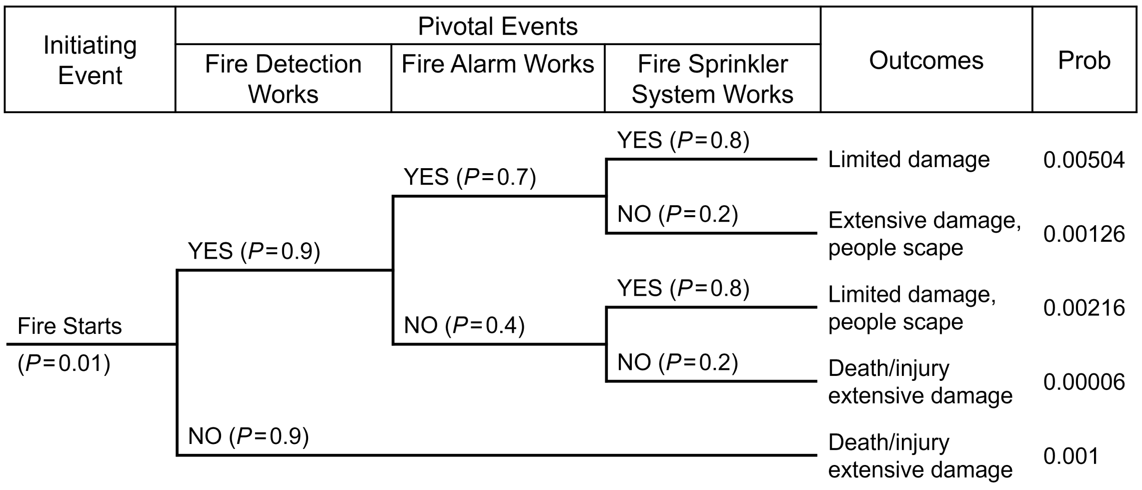


Figure 3.12: Example of an event tree (Adapted from [73])

3.5.5 Failure Modes and Effects Analysis

Failure Mode and Effects Analysis FMEA is a safety analysis procedure by which each potential failure mode in a system is analyzed to determine the results on the system [89]. FMEA is therefore an inductive technique, as the reasoning goes from a particular fault to a general effect of the system [28]. FMEAs are usually performed with the help of worksheets such as the one shown in Figure 3.13.

Results from one or various FMEAs can be grouped according to the effect that failures produce, which results in a Failure Modes and Effects Summary [58]. It is also possible to complement an FMEA with a criticality analysis, which establishes the severity classification of each failure mode [89]. This combination is known as Failure Mode, Effects, and Criticality Analysis FMECA.

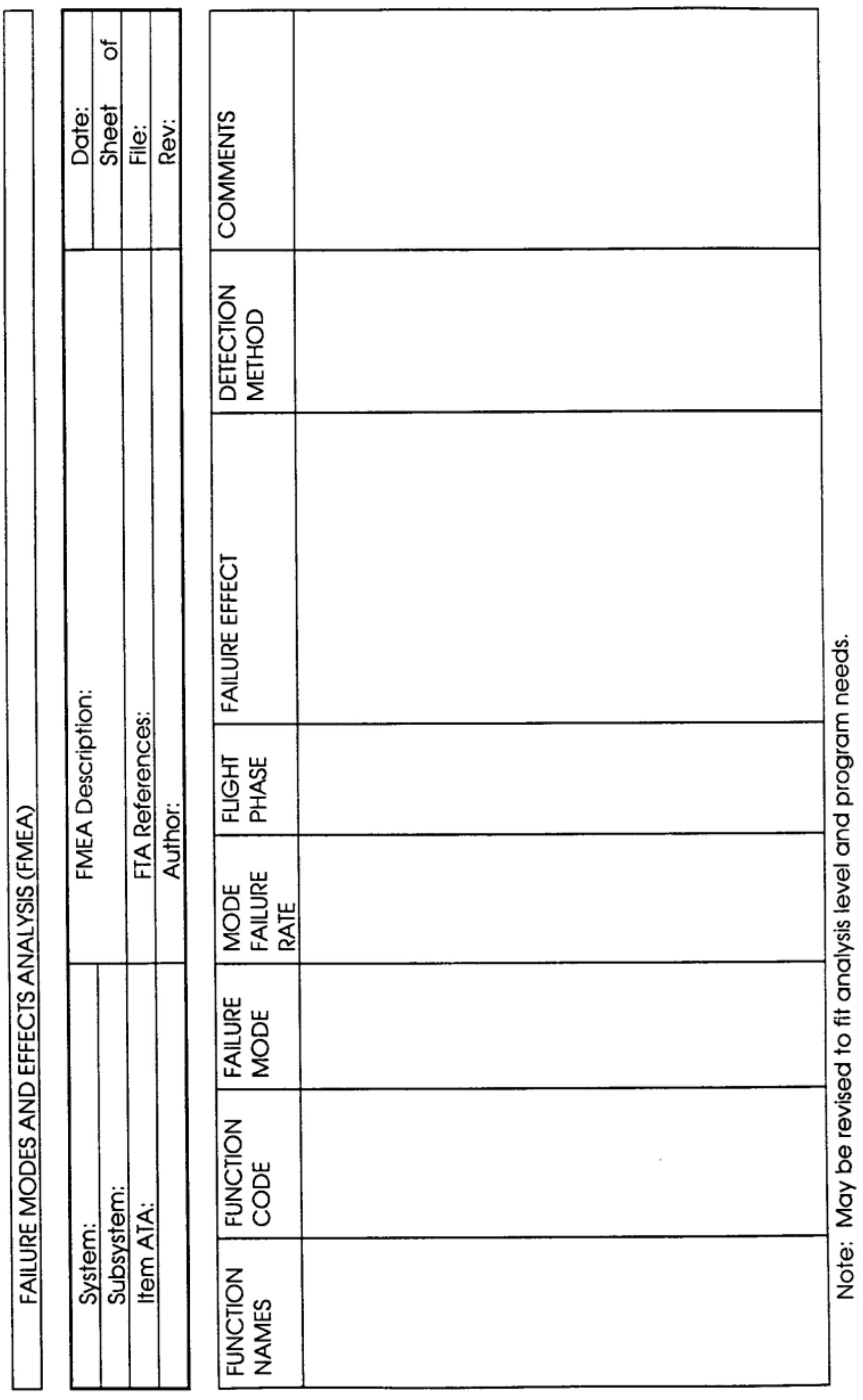
3.5.6 Hazard and Operability

A Hazard and Operability HAZOP study aims to identify how a process may deviate from its design intent, via the application of a formal, systematic critical examination of the process and the engineering intentions of facilities [90]. The method combines guide-words (such as no, more, less) with process parameters (e.g. temperature, flow, pressure) to determine deviations from normal operation that could result in a hazard. The possible causes and their consequences of the deviations are determined along with safeguard or recommendations to prevent or mitigate the hazardous situation.

The concept of a HAZOP study first appeared related to process in facilities that manage highly hazardous materials and its purpose was to eliminate sources leading to major accidents. However, it has been extended to cater for other types of facilities or systems such as software and programmable systems [91] among others.

3.5.7 System-Theoretic Process Analysis

Alternative hazard analysis methods such as the System Theoretic Process Analysis STPA [59] have been proposed. Leveson et al. [61] criticise the traditional hazard analysis methods described in ARP 4761 as they question their effect-



FAILURE MODES AND EFFECTS ANALYSIS (FMEA)

System:	FMEA Description:	Date:
Subsystem:		Sheet of
Item ATA:	FTA References:	File:
	Author:	Rev:

FUNCTION NAMES	FUNCTION CODE	FAILURE MODE	MODE FAILURE RATE	FLIGHT PHASE	FAILURE EFFECT	DETECTION METHOD	COMMENTS

Note: May be revised to fit analysis level and program needs.

Figure 3.13: Example of an FMEA worksheet (From [58])

iveness on software-intensive systems where accidents may result not only from component failures but also from unsafe interactions among the components. To overcome these limitations, STPA is based on systems theory rather than reliability theory and includes additional causes for accidents such as system design errors, human error (in more detail than just random failure) and various types of systemic accident causes.

STPA-based safety assessment [59, 92] consist of four steps:

1. **Define the purpose of the analysis:** in this step, the losses, system-level hazards and system-level constraints are identified. Losses refer to anything of value to stakeholders, such as human life or injury, property damage or environmental pollution. A hazard is a system state that, under a particular set of worst-case environmental conditions, will lead to a loss. System-level constraints specify what the system must do to prevent hazards.

2. **Model the control structure:** a hierarchical control structure is composed of feedback control loops including controllers, control actions, feedback signals, controlled processes and other inputs and outputs. Loops are ordered by decreasing level of control authority. Figure 3.14 shows a generic simple hierarchical control structure.

3. **Identify unsafe control actions:** the control structure is examined to determine the unsafe control actions and controller constraints are defined to prevent them. A control action is unsafe if, in a particular context and worst-case environment, will lead to a hazard. Control actions can be unsafe because they are not provided when needed or provided when not needed, as well as if their timing (too early or too late or out of order) and duration (too short or too long) is not right.

4. **Identify loss scenarios:** a loss scenario describes the causal factors that can lead to the identified unsafe control actions. Loss scenarios can be

caused by unsafe controller behaviour or inadequate feedback and other inputs to the controller. Safe control actions that are not executed or executed improperly, due to failures in the control path (actuators) or controlled process, also represent loss scenarios.

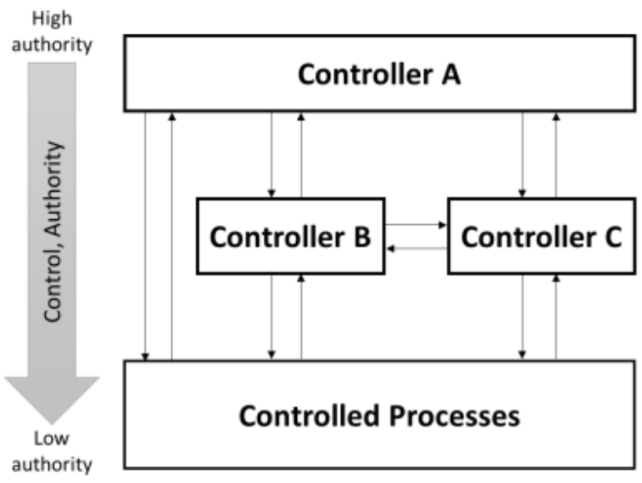


Figure 3.14: Safety assessment process overview (From [64])

3.5.8 AcciMap

AcciMap [84] provides a graphical representation of accidents that is based on Rasmussen's Risk Management Framework (see Section 3.4.4). AcciMap is intended to structure the analysis and to identify the interactions that lead to accidents in socio-technical systems. *AcciMap* consists of a graph that represents a particular accident scenario. It represents the causal flow of events, supplemented by a representation of the planning, management, and regulatory bodies. Other graphical representations such as the *Generic AcciMap*, the *ActorMap* or the *InfoFlowMap* complement *AcciMap* when analysing risk in socio-technical systems.

AcciMap and *Generic AcciMap* are structured in six levels as shown in Figure 3.15. The bottom represents the context of the accident. The immediate

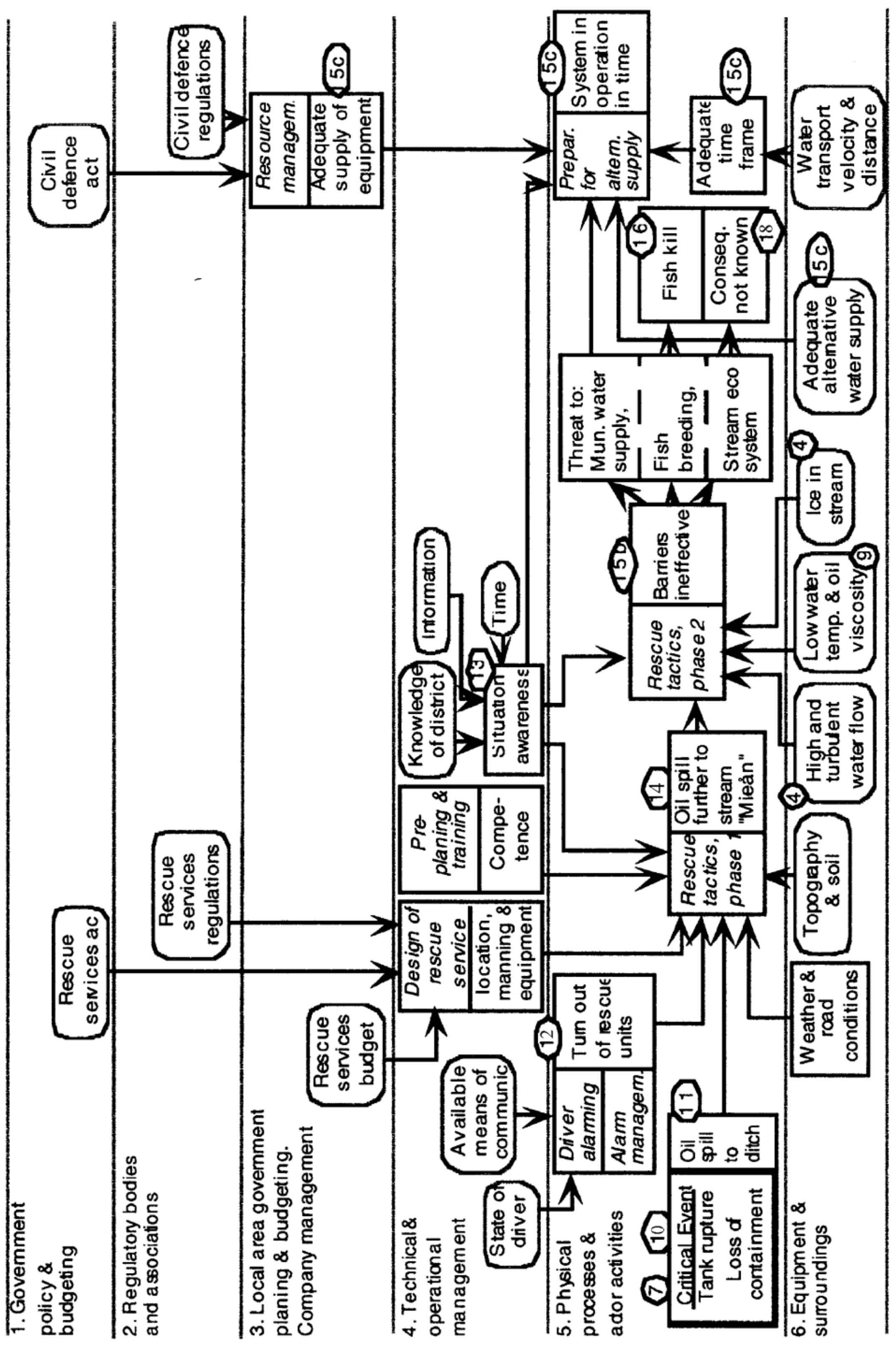

1. Government policy & budgeting
2. Regulatory bodies and associations
3. Local area government planing & budgeting. Company management
4. Technical & operational management
5. Physical processes & actor activities
6. Equipment & surroundings
Civil defence act
Civil defence regulations
Resource managem.
Adequate supply of equipment
5c
Rescue services act
Rescue services regulations
Rescue services budget
Design of rescue service
location, manning & equipment
Pre-planing & training
Competence
Knowledge of district
Information
Time
Situation awareness
13
Available means of communic.
Driver alarming
Alarm managem.
Turn out of rescue units
12
State of driver
Critical Event
Tank rupture
Loss of containment
7
10
Oil spill to ditch
11
Rescue tactics, phase 1
Oil spill further to stream "Mieån"
14
Rescue tactics, phase 2
Barriers ineffective
15b
Threat to: Mun. water supply, Fish breeding, Stream eco system
Fish kill
16
Conseq. not known
18
Prepar. for altern. supply
System in operation in time
15c
Adequate time frame
15c
Weather & road conditions
Topography & soil
High and turbulent water flow
4
Low water temp. & oil viscosity
9
Ice in stream
4
Adequate alternative water supply
5c
Water transport velocity & distance

Figure 3.15: Example of a *Generic AcciMap* (From [84])

higher level represents the accident processes, the causal and functional relations. The levels above represent all decision-makers that have the potential to influence the accident. While the recommendations from *AcciMap* might be biased toward the particular accident analysed, the *Generic AcciMap* is better suited to generate recommendations as it is based on the normal, causal flow of activities.

3.5.9 Functional Resonance Analysis Method

The Functional Resonance Analysis Method FRAM [74] has been proposed to overcome the limitations of linear thinking and simple cause-relation analysis. These limitations may not be significant in technological systems, such as the ones considered in this research, but are considered to be crucial in socio-technological systems (including human and organizational factors). FRAM is built upon four principles:

- Failures and successes have an equivalent origin.
- The everyday performance of socio-technical systems is constantly adjusted to match the conditions.
- Many of the outcomes are emergent, meaning that cannot be explained only by decomposition and causality.
- Relationships among the functions are situation-specific and potentially non-linear due to feedback and resonance phenomena.

The application of the method, both when used to explain and accident or anticipate risks, consist of four steps:

1. Identify the functions that are required for everyday work to succeed, describing how things are done in detail. The functions constitute the FRAM model.

2. Characterise the variability of the functions both in general and under specific conditions (instantiations of the model).

3. Examine instantiations of the model to understand how function variabilities may become coupled and lead to unexpected outcomes.

4. Propose ways to manage uncontrolled performance variability. The suggested approach is to monitor the performance and dampen uncontrolled variability when required.

3.5.10 Discussion

Several safety methods and their underlying accident models are reviewed in this section and Section 3.4. The most traditional methods such as FTA, FMEA or Markov analysis are still among the most popular. Whereas these methods have been successful in improving the safety of technical systems, especially by improving their reliability, they are limited by the accident models on which they rely. This makes them inadequate for modelling the safety of certain aspects of the system such as those involving software or human behaviour or to consider system failure caused by unsafe interaction of components where none of the components involved has failed. Novel accident models such as STAMP or FRAM and safety methods such as STPA or AcciMap have been developed to explain these more complex accidents and help to develop safer systems.

This research focuses on improving the integration between safety and other development activities and accelerating the safety assessment process by automating safety methods. For this purpose, one hazard assessment method and one safety analysis method are selected as candidates for automation. STPA is chosen due to its potential to consider more kinds of hazards than traditional methods, combined with the similarity of STPA control loops and hierarchical structures with logical models of systems architectures, which is seen as an opportunity to develop novel methods automating parts of STPA analysis.

Regarding safety analysis, FTA is the selected method. The applicability of FTA is more limited than that of STPA due to its limitations, which are the ones common to traditional probabilistic safety methods. Despite its more limited applicability, FTA is still relevant within the scope of this research, which is mainly related to the most technical parts of aircraft. The apparent availability of the information required for creating fault trees within logical views of systems architectures is the main reason why FTA is chosen over similar methods.

3.6 Computation Support for STPA

Section 3.5 discusses several safety methods and selected STPA as the best candidate among existing hazard assessment methods to which provide computational support. In this section, existing STPA tools and computational methods are reviewed, and their limitations are identified.

3.6.1 Existing STPA Tools

XSTAMPP

A-STPA

A-STPA [93] is a computational tool for supporting STPA analysis. The tool was developed in 2014 to improve the current STPA practice at that time, which relied on paper, word documents and drawing software. The tool provides a user interface (see 3.16) to edit STPA analysis data, draw the control structure diagram, edit tables such as the control actions table or the unsafe control action table.

XSTAMPP [94] (eXtensible STAMP Platform) is an open-source platform for safety engineering developed at the University of Stuttgart. It was designed specially to encourage the widespread adoption and use of STAMP-based method-

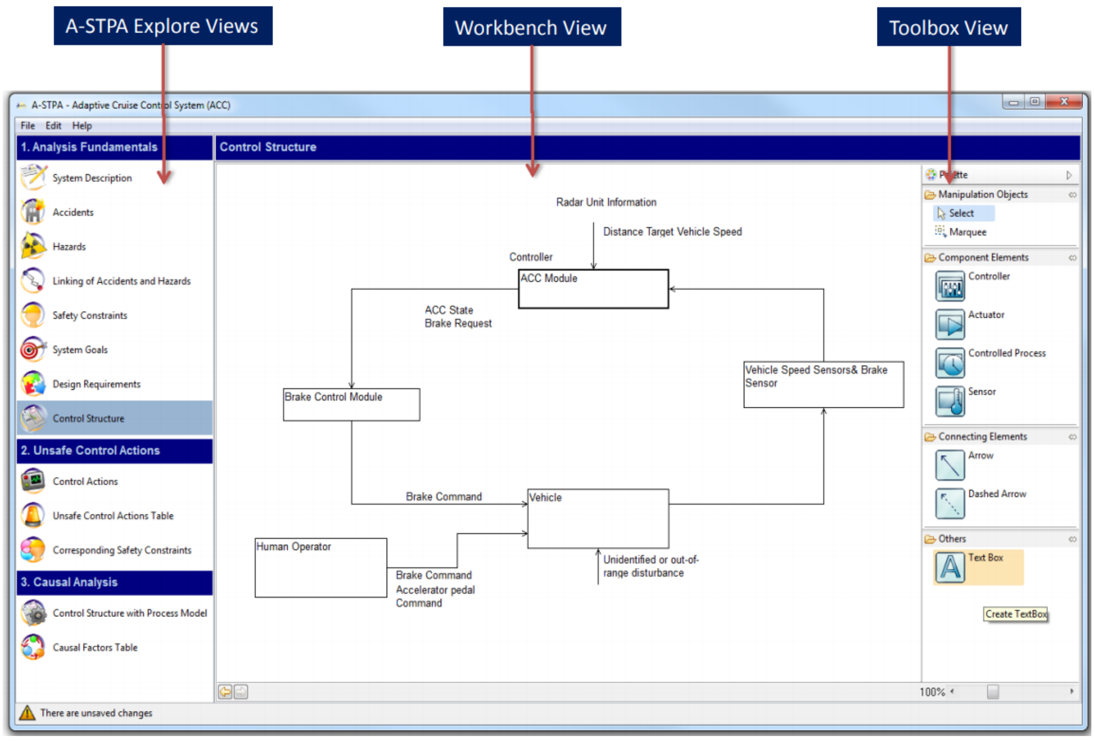


Figure 3.16: Control loop in A-STPA UI (From [93])

ologies (such as STPA and CAST*) by safety analysts in different applications areas [95]. XSTAMPP includes seven plug-ins for Eclipse IDE [96] and provides a user interface (see Figure 3.17) that allows the editing of STPA and CAST projects. XSTAMPP was developed to overcome the limitations of A-STPA tool.

SAHRA

SAHRA [97, 98] is an integrated software tool for STPA developed by Zurich University of Applied Sciences. It considers the hierarchical control structures as just ‘another view’ of systems, which are described by using different UML/SysML views within Sparx Systems Enterprise Architect [99]. Instead of using the common tabular format, SAHRA introduces mind maps (such as the one in Figure 3.18) as a means of visually representing and editing STPA Step 1. Mind maps are used to build a directed graph that enables traceability analysis between

*CAST, which is based on STAMP, is a technique to analyze accident causality from a systems perspective.

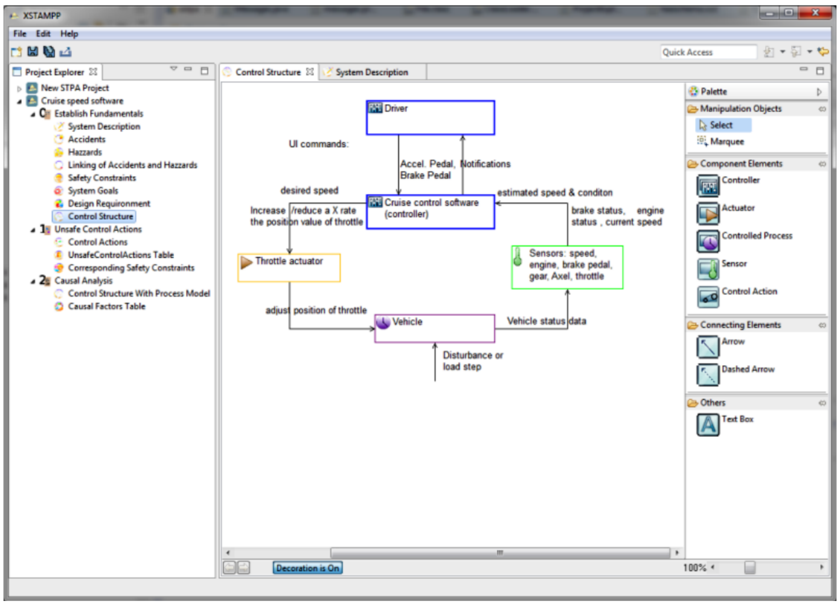


Figure 3.17: XSTAMPP Workbench UI (From [95])

different kind of elements such as losses, hazards and unsafe control actions among others.

STPA solution in Risk Management Studio

STPA Solution in Risk Management Studio [100, 101] is a tool that allows STPA hazard analysis. In particular, it supports the graphical modelling of the control structure, identification of losses hazards and constraints, and determination of control actions and loss scenarios. Risk Management Studio supports linking losses, hazard and constraints and facilitates the work of the safety analyst by using the control structure (see Figure 3.19) to pre-populate the lists of control actions and loss scenarios to be assessed.

STAMP Workbench

STAMP Workbench [102] support four STPA steps. Step 0 is the preparation step, which consists of determining the preconditions, identifying accidents, hazards and safety constraints, and drawing the control structure. Step 1 focuses on extracting unsafe control actions. Hazard Causal Factors (loss scenarios) are

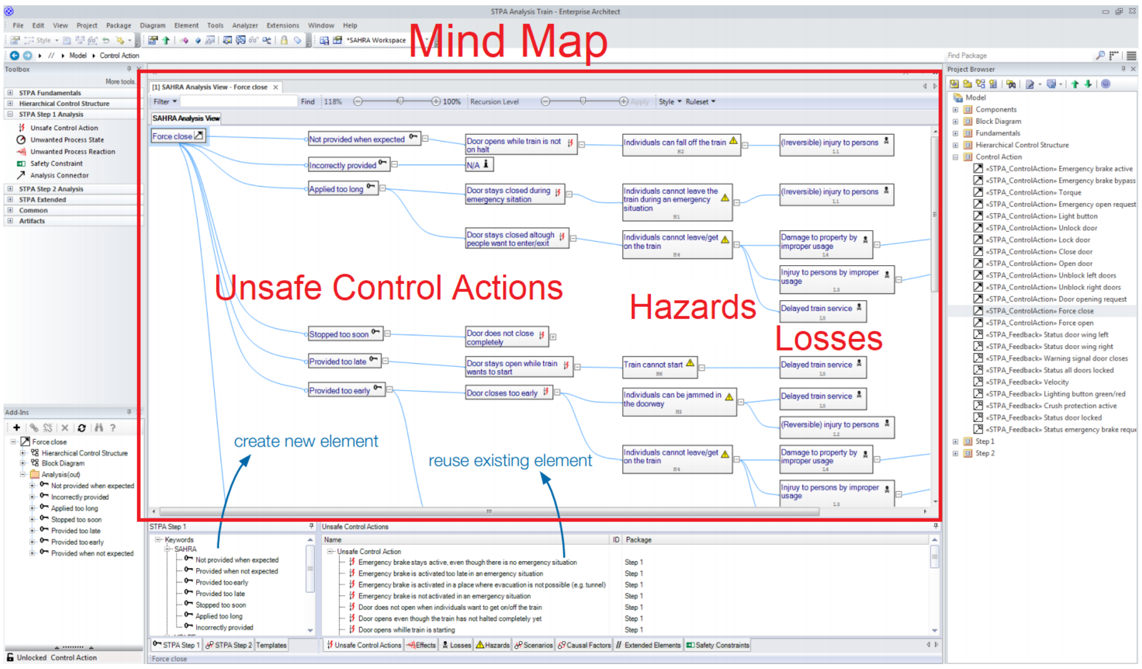


Figure 3.18: SAHRA UI displaying a mind map (From [97])

determined in Step 2. Finally, Step 3 derives countermeasures from the identified causal factors.

Astah System Safety

Astah System Safety [103] support STPA tables for preconditions, accidents, hazards, safety constraints, unsafe control actions, loss scenarios and countermeasures. It also allows the construction of hierarchical control structure diagrams (see Figure 3.20) and more detailed control loop diagrams.

CAIRIS

CAIRIS [104] is an open-source platform for building security and usability into software. The platform supports STPA by providing automatic traceability between STPA elements, automatic generation of visual models and documentation, and reasoning support to help identify and validate casual scenarios. CAIRIS works by mapping STPA elements to their own types of elements. For instance, losses

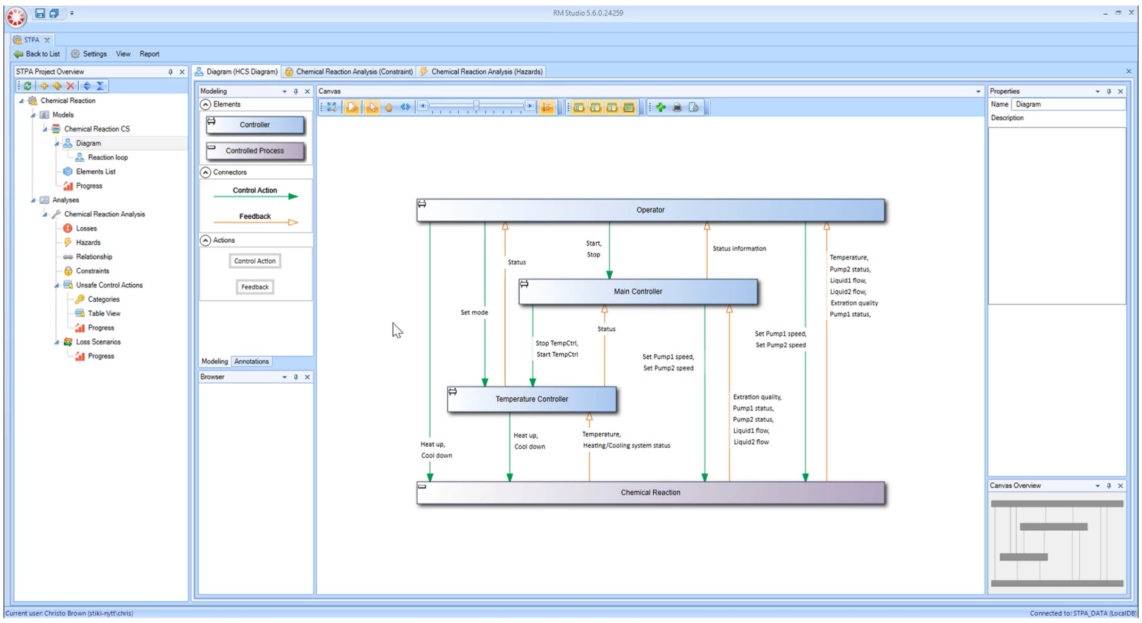


Figure 3.19: STPA control structure in Risk Management Studio (From [100])

are mapped to obstacles, constraints to goals, and the control structure is modelled via a data flow diagram.

An STPA Tool

An STPA tool [105] can automatically generate a context table (used to determine Unsafe Control Actions), identify conflicts in the table and generate requirements such as the one in Figure 3.21. The tool is based on the general structure for Unsafe Control Actions and context modelling via Process Model Variables, which are presented in detail in Section 3.6.2.

SafetyHAT

The transportation systems Safety Hazard Analysis Tool (SafetyHAT) [106] is a software tool that facilitates STPA hazard analysis through a wizard like the one in Figure 3.22. This tool is tailored for transportation systems, including transportation-oriented guide phrases and causal factors. The most salient claims of SafetyHAT are its ability to organize and manage a large quantity of data thanks

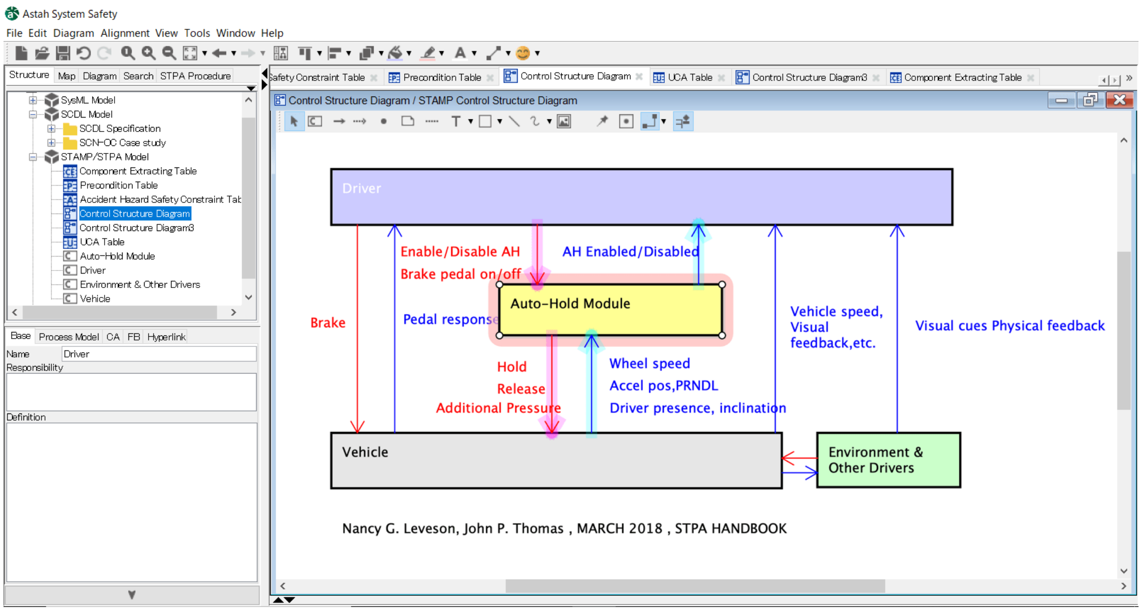

Astah System Safety
File Edit Diagram Alignment View Tools Window Help
Structure Map Diagram Search STPA Procedure
SysML Model
SCDL Model
SCDL Specification
SCN-OC Case study
STAMP/STPA Model
Component Extracting Table
Precondition Table
Accident Hazard Safety Constraint Tab
Control Structure Diagram
Control Structure Diagram3
UCA Table
Auto-Hold Module
Driver
Environment & Other Drivers
Vehicle
Base Process Model CA FB Hyperlink
Name Driver
Responsibility
Definition
Safety Constraint Table
Precondition Table
Control Structure Diagram
UCA Table
Control Structure Diagram3
Component Extracting Table
Control Structure Diagram / STAMP Control Structure Diagram
Driver
Enable/Disable AH
Brake pedal on/off
AH Enabled/Disabled
Brake
Pedal response
Auto-Hold Module
Vehicle speed, Visual feedback,etc.
Visual cues Physical feedback
Hold
Release
Additional Pressure
Wheel speed
Accel pos,PRNDL
Driver presence, inclination
Vehicle
Environment & Other Drivers
Nancy G. Leveson, John P. Thomas , MARCH 2018 , STPA HANDBOOK

Figure 3.20: Astah System Safety UI (From [103])

to using a relational database, to provide traceability from system-level hazards to component level causal factors and to generate auditable documentation.

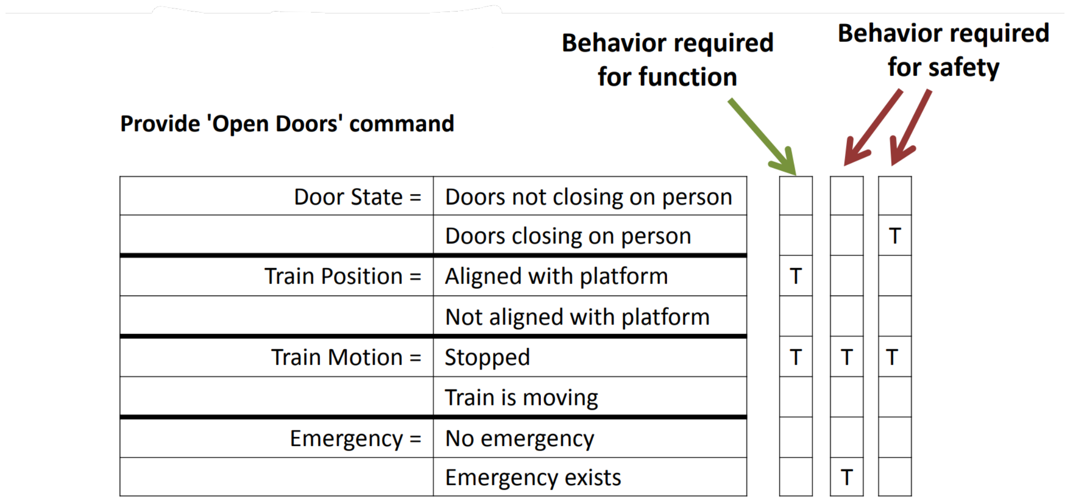
Simulink-STPA tool

Abdellatif and Holzapfel [107] propose to use Simulink models to automatically determine the components of the control hierarchy in STPA analysis, the linkage of the components remains a manual task. A tool (for which no name is provided) was developed to provide this capability. The tool provides automatic verification of some physical aspects of the system. It checks for missing links (control or feedback) between controllers and controlled processes, and sensor compatibility, emitting results such as those in Figure 3.23.

3.6.2 Existing STPA Methods

Automated Generation of Formal Model-Based Safety Requirements

Thomas [108] and Thomas and Leveson [109] propose a method to generate formal model-based safety requirements within STPA hazard assessment. A gen-


Behavior required for function
Behavior required for safety

Provide 'Open Doors' command

Door State =	Doors not closing on person			
	Doors closing on person			T
Train Position =	Aligned with platform	T		
	Not aligned with platform			
Train Motion =	Stopped	T	T	T
	Train is moving			
Emergency =	No emergency			
	Emergency exists		T	

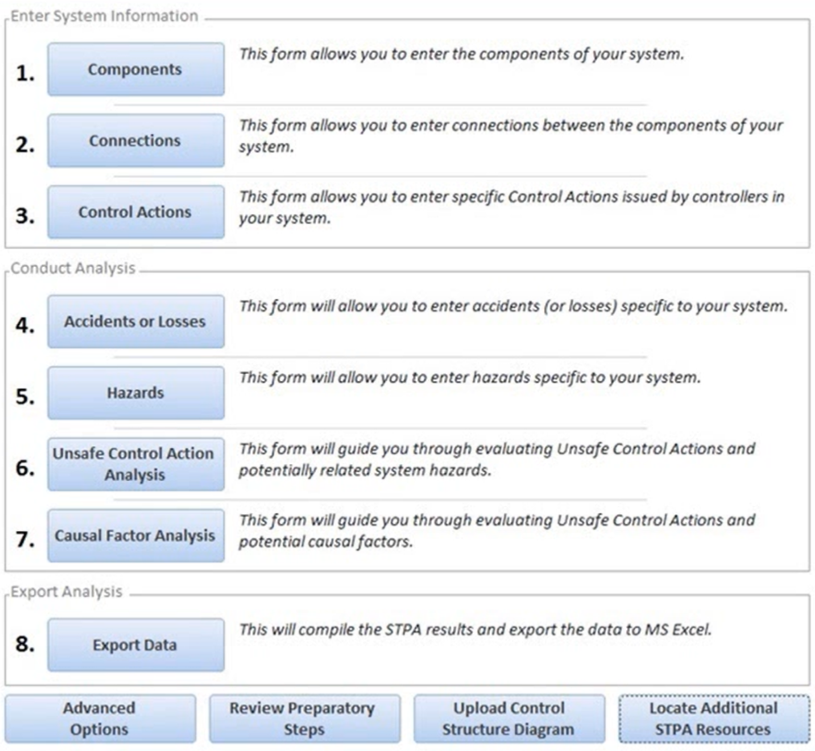
Figure 3.21: Controller requirement in An STPA Tool (From [105])

eral structure for unsafe control actions (designated as hazardous control actions in the references) is identified. Unsafe control actions are expressed as a combination of four terms

- **Controller:** part of the system that can issue control actions.
- **Type of control action:** either *Provided* or *Not Provided*.
- **Control action:** command that is (or is not) output by the controller.
- **Context:** conditions in the system and environment that make an action (or inaction) hazardous.

According to this structure, the identification of a hazardous control action requires the determination of potentially hazardous contexts for the action. A context is the combination of one or more conditions. A condition is a variable value pair where:

- **Variable:** attribute of the system or environment with more than one possible value.
- **Value:** particular values adopted for the variable.


Enter System Information

1. Components — *This form allows you to enter the components of your system.*
2. Connections — *This form allows you to enter connections between the components of your system.*
3. Control Actions — *This form allows you to enter specific Control Actions issued by controllers in your system.*

Conduct Analysis

4. Accidents or Losses — *This form will allow you to enter accidents (or losses) specific to your system.*
5. Hazards — *This form will allow you to enter hazards specific to your system.*
6. Unsafe Control Action Analysis — *This form will guide you through evaluating Unsafe Control Actions and potentially related system hazards.*
7. Causal Factor Analysis — *This form will guide you through evaluating Unsafe Control Actions and potential causal factors.*

Export Analysis

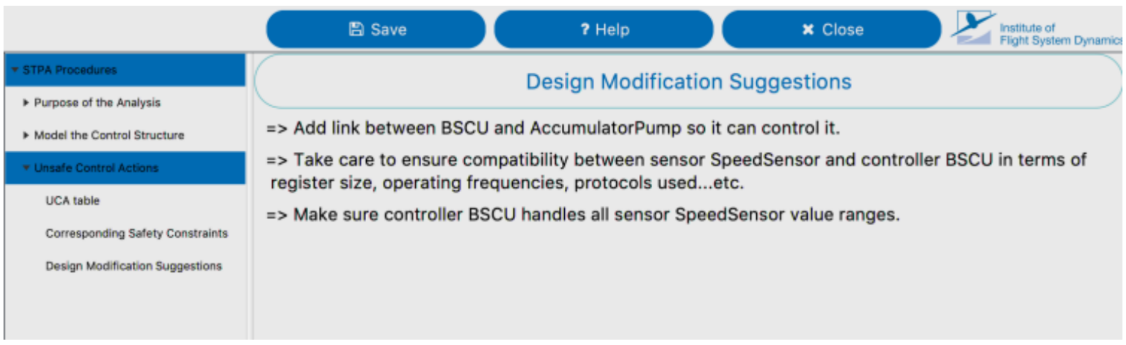
8. Export Data — *This will compile the STPA results and export the data to MS Excel.*

Advanced Options
Review Preparatory Steps
Upload Control Structure Diagram
Locate Additional STPA Resources

Figure 3.22: SafetyHAT wizard UI (From [106])

A list of potential hazardous control actions can be automatically generated by combining items from the sets of controllers, types of action, control actions and contexts. Whether a combination is hazardous needs to be determined for each combination — which will require more or less manual input depending on the availability of behavioural models.

Requirements, which are expressed as a formal SpecTRM-RL [110] specification such as the one in Figure 3.21, can be automatically generated by computing


Save
? Help
Close
Institute of Flight System Dynamics
STPA Procedures
Purpose of the Analysis
Model the Control Structure
Unsafe Control Actions
UCA table
Corresponding Safety Constraints
Design Modification Suggestions

Design Modification Suggestions

=> Add link between BSCU and AccumulatorPump so it can control it.

=> Take care to ensure compatibility between sensor SpeedSensor and controller BSCU in terms of register size, operating frequencies, protocols used...etc.

=> Make sure controller BSCU handles all sensor SpeedSensor value ranges.

Figure 3.23: Design modification suggestion (From [107])

the combinations of controller, control action and context that prevent hazardous behaviour. Consistency can also be determined by examining two cases. In the first case, lack of consistency is found when it is hazardous both to provide and not provide a particular action within the same context. The second case refers to actions that need to be provided to fulfil the functions of the system but result in a hazard.

3.6.3 Discussion

In this section, existing STPA computational support tools and methods have been presented. All tools, except for *An STPA Tool*, provide a graphical user interface to manually edit the various tables and control diagrams that are usually generated during STPA hazard assessment. The most common automation feature is to allow the linking of different elements such as losses, hazards and so on, thus enabling automatic traceability between the elements. *SAHRA*, *Risk Management Studio* and *CAIRIS* explicitly support this feature.

A greater degree of automation is provided by the method for automated generation of formal model-based safety requirements [108, 109], which is implemented only in *An STPA Tool.* The tool linking Simulink to STPA [107] is the only one to enable automatic determination of the components in the STPA control hierarchy and partial verification of the design.

In general, the degree of automation provided by the existing tools is low, which can impact the total time required for STPA analysis and hinder the ability of the architect to keep it up to date and inform the design. Furthermore, except in the case where Simulink is used to model the architecture, the consistency between STPA analysis and other architectural models is entirely dependent on the ability of the analysts to interpret such architectural models, if existing.

3.7 Automation of Fault Tree Analysis

Fault tree analysis (see Section 3.5.1) was selected as the best safety analysis method to be automated from the review of safety methods presented in Section 3.5. Fault tree analysis requires creating the fault tree and then analysing it through various quantitative and qualitative techniques. Creation of the fault tree is the process that requires the greatest amount of manual work by safety experts, which motivates this research to focus on the automation of the tree creation process. Additionally, the need for tighter integration of safety and system development activities was also identified in Section 3.3. This section introduces existing methods for automated fault tree creation and identifies their limitations.

3.7.1 Existing Methods

Fault Tree Creation from SysML

Mhenni, Nguyen and Choley [68] propose to use SysML to integrate safety analysis within a systems engineering approach. Fault trees can be generated from SysML's Internal Block Diagrams at any point in the design process maintaining consistency between the system model and the safety analysis. Internal Block diagrams are a kind of structural diagram that describes the internal structure of a system in terms of its parts, ports and connectors [111]. The emphasis of the diagram is placed on the logical relationships between elements. Internal block diagrams describe the way that the various parts are connected through ports, interfaces, and connectors and the items that flow between parts.

The automated generation process consists of traversing the diagram identifying known patterns that can be translated to combinations of fault tree gates and events. The traversal starts at an external output port, advances through edges created with direction opposite to the flow declared in the diagram, and stops at diagram input ports.

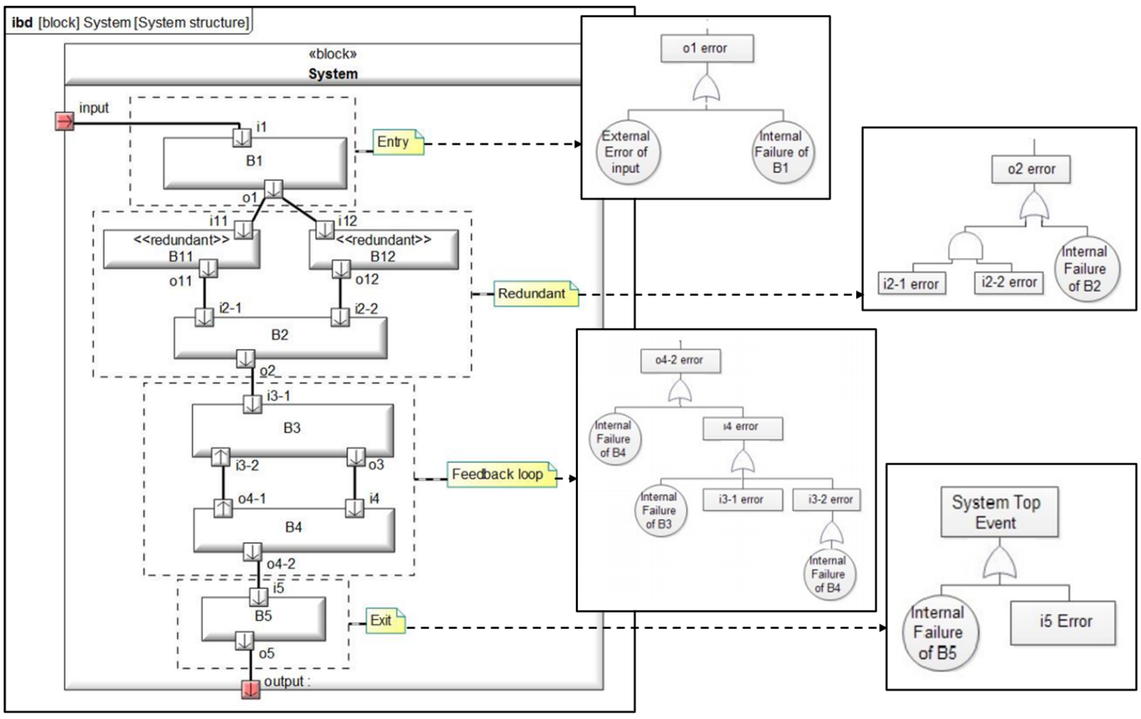


Figure 3.24: IBD Patterns and FTA transformations (Adapted from [68])

As shown in Figure 3.24, four different patterns are considered.

- **Entry pattern:** the part has at least one input port connected to an external input. It is transformed into an *OR* gate with events representing the external failure, the internal failure of the part and other input flows.
- **Exit pattern:** the part has at least one output port connected to an external output. It results in an *OR* gate whose operands are the failure of part and other input flows.
- **Feedback pattern:** occurs when a part that has already been visited is encountered. This pattern is transformed into various or gates (as in the example in 3.24) where inputs from already visited components are not developed further.
- **Redundant pattern:** consists of a part that receives flow coming from redundant blocks providing the same function. It is translated into an *OR* gate

with the part failure and an *AND* gate representing the redundancy as inputs.

Fault Tree Creation from DSMs

The methodology of Structural Complexity Management [112] provides the foundation for the automated fault tree creation process developed by Roth, Wolf and Lindemann [113]. The methodology handles complex system and structural dependencies by combining Design Structure Matrices (DSM) and Domain Mapping Matrices (DMM). Structural Complexity Management is adapted to the task of creating fault trees, resulting in a procedure that consists of six steps grouped in four phases:

- **System Definition Phase**
  1. **Setting up the Multiple Domain Matrix (MDM):** the MDM includes the definition of the domains and their relevant relations. In the example shown in Figure 3.25, the elements correspond to the row and column headers (function, flow/state, failure and so on) and the relations are indicated by the elements inside the matrix (fulfils, produces/influences and so forth).
- **Information Acquisition Phase**
  2. **Modelling the System Structure and its Failures:** by recording the elements (function and flow/state in the example) together with their direct dependencies (DMMs such as *FuFl* and *FlFu* in Figure 3.25)
- **Deduction of Indirect Dependencies Phase**
  3. **Generating a Failure Network:** calculate the DSM modelling dependencies between failures (*FaFa* in Figure 3.25) via matrix multiplication.

- **Structure Analysis Phase**

   4. **Generating Fault Trees:** by traversing *FaFa* from each identified top event. The proposed approach only generates *OR* gates. *AND* gates modelling redundancy must be included manually.

   5. **Identifying Minimal Cut Sets:** by recursively replacing gates by their own influences (primary failures or other logic gates).

   6. **Evaluation and Visualization:** estimating the impact of individual failures and their degree of occurrence.

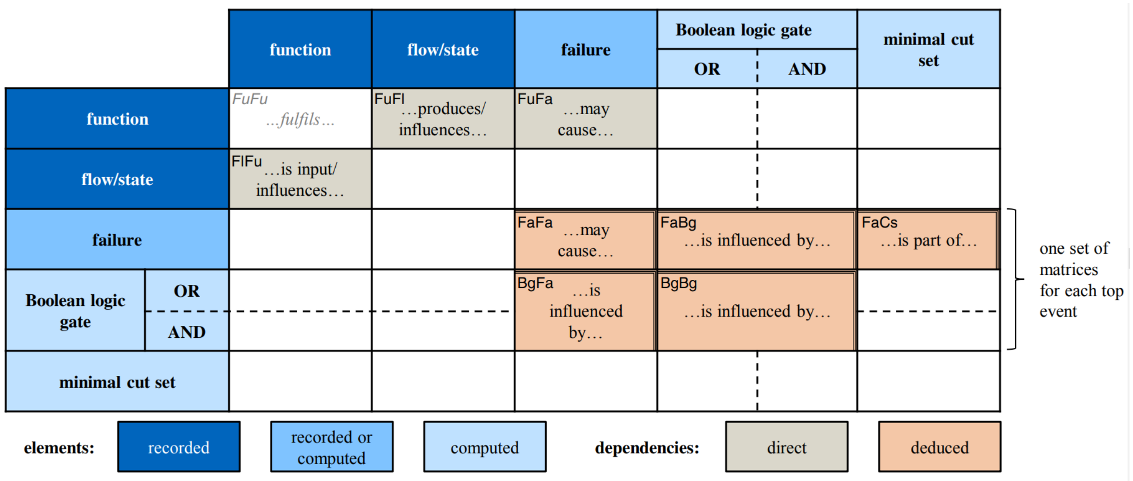


Figure 3.25: Example of Multiple Domain Matrix (From [113])

Fault Tree Creation from Maude Language and SysML

Xiang et al. [114] propose to first develop a reliability configuration model (RCM) containing the system configuration information needed for reliability analysis and then generate static fault trees from the RCM specifications. The models are specified using an executable algebraic formal specification language called Maude [115]. Two kinds of dynamic fault tree gates, *Functional Dependency* and *Priority AND* are converted to a combination of static standard *AND* and *OR* gates, enabling the modelling of more complex systems. Extra SysML stereotypes are defined to enable transformation from SysML to the RCM, which is capable to produce the desired fault trees.

Hierarchically Performed Hazard Origin and Propagation Studies

Papadopoulos et al. [71] develop a method called Hierarchically Performed Hazard Origin and Propagation Studies (HiP-HOPS), which enables the assessment of systems from the top functional level to the lower levels of hardware and software implementation. The method is founded on well-established techniques such as FMEA and FTA, which are modified, automated and integrated.

The operation of a system is described at each hierarchical level by flow diagrams, where components exchange material, energy or data. A tool named *Safety Argument Manager*, shown in Figure 3.26, is created for this purpose. Fault trees are modelled by applying the following principle: a component's output failure is the result of either internal component malfunctions or deviations of the component inputs. Component malfunction or deviation of component inputs information is generated during an FMEA that must be completed prior to the FTA. The synthesis algorithm proceeds recursively in two dimensions:

- **Vertically:** express system failure as a result of component failures.
- **Horizontally:** translates output failures to a combination of component malfunctions and input deviations.

Fault Tree Creation from AADL

The Architecture Analysis & Design Language (AADL) is a modelling language that supports early analyses of a system's architecture [116]. The language describes systems architectures in terms of distinct components (such as software, computational hardware, and system components) and their interactions. The

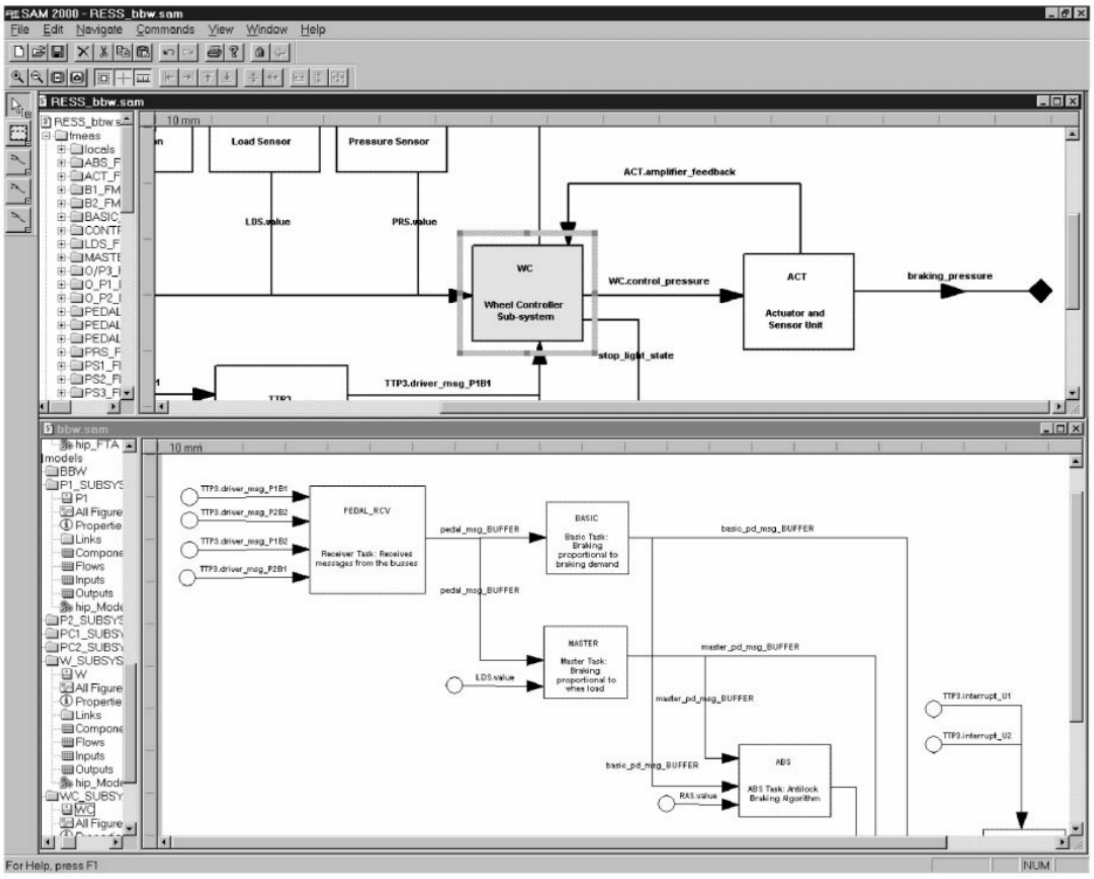


Figure 3.26: Safety Argument Manager, a hierarchical modelling tool (From [71])

AADL is claimed to be especially effective to model complex real-time embedded systems.

The Error-Model Annex of the SAE AADL enables annotation of system and software architectures (modelled in AADL) with hazard, fault propagation, failure modes and effects, and compositional fault behaviour information [67]. Delange and Feiler [67] extend the language with safety semantics and a fault propagation ontology that supports the modelling of error behaviours.

Fault propagation is modelled via errors that originate at source components and propagate through path components and connections between components until a sink component is reached. Errors follow an ontology including omission and commission, timing, value, replication and concurrency errors. The error behaviour of components is modelled using state machines (see Figure 3.27), where different states determine different error propagation behaviours. Transitions to other states are triggered by either component-specific events or failure propagation from other components. For composite components, error behaviour is defined in terms of the behaviour of the subcomponents.

```
error behavior Simple
events
   failure : error event;
   recov : error event;
states
   Operational: initial state;
   Failed: state;
transitions
   t1 : Operational -[failure]-> Failed;
   t2 : Failed -[recov]-> Operational;
end behavior;
```

Figure 3.27: Example of AADL error behaviour state machine (From [67])

The information provided by the Error-Model Annex can be utilised to automatically generate various ARP 4761 safety assessment methods [117], for example, generating fault trees from AADL models. The method by Feiler and Delange [118] creates the fault tree starting from a selected failure of interest, which becomes the top event of the tree. Propagation paths and error flows are traversed backwards creating *OR* or *AND* gates depending on the structure of the system and provided error model information.

Joshi, Vestal and Binns [69] propose a method that first generates a directed graph from the supplied AADL error propagation information, and then uses a recursive algorithm to create the fault tree. The algorithm starts with a top event corresponding to one of the declared system hazards and then generates and optimises intermediate gates by removing redundant operators, collapse gates or sharing subtrees.

Fault Tree Creation from Altarica

Altarica is a formal language that describes systems as a hierarchy of components [119]. The nodes in the hierarchy that contain sub-nodes represent the system or subsystems [120]. Leaf nodes, which contain no sub-nodes, represent components that cannot be decomposed.

Li and Li [120] use Altarica models to automate the generation of fault trees. Leaf components are defined by the combination of:

1. **States, flows and events.** States are internal variables, flows correspond to the inputs and outputs of the system and events represent the trigger of transitions from one state to another.

2. **Transitions:** including the initial state and transition guards, the logical conditions for a transition to occur.

3. **Assertions:** that determine the value of outputs based on component state and inputs.

Composite components are described in terms of their children.

For leaf components, the top event of the corresponding tree represents an output variable. The bottom events represent input and state variables and are linked together with gates whose type is determined by the assertions of the component. For composite components, the top event of the corresponding tree represents a subcomponent connected to an output variable. The fault tree for the subcomponent is generated following the procedure for leaf components. If a bottom event of this tree corresponds to an output provided by sibling subcomponents, the process is repeated — taking the output as the top event of a subtree — until all bottom events correspond to input and internal state variables of the composite component.

Fault Tree Creation from Simulink

Papadopoulos and Maruhn [91] develop an automatic fault tree creation method using Simulink [121] models and HAZOP. A computer HAZOP study is used to determine the specific failure modes of each component's output (e.g. service provision, timing and domain-value failures), which input combinations or component malfunctions lead to each failure mode, and the associated probability of failure. The proposed algorithm generates fault trees by traversing the hierarchical model of the system backwards with respect to the failure propagation direction. It also considers dependencies between parent/child components.

Latif-Shabgahi and Tajarrod [122] propose a different method to automatically elaborate fault trees from Simulink [121] models. The first step of the method determines the system topology, considering components with more than one output as multifunctional components and handling them as N (the number of outputs) virtual single-function components. The second step consists in creating the extended model, which classifies the components according to their impact regarding the top event, and identifies groups of components that perform a particular task and which groups are redundant. Finally, the various groups of components are translated into fault tree gates and events.

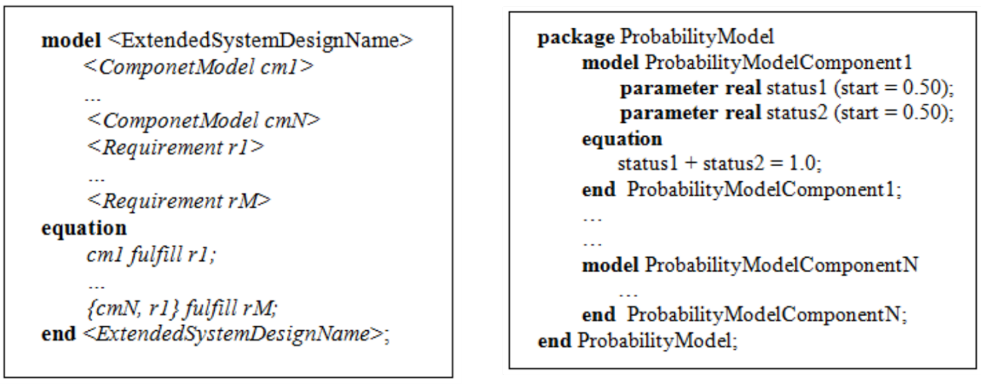
Fault Tree Creation from Modelica

Schallert [123] proposes a method to obtain FTA results, such as minimal cut or path sets from Modelica models, whose equations need to be extended to model failure behaviour. The results are obtained by simulation of the system under each possible failure combinations. As the number of combinations grows exponentially with the number of component inputs, mitigation strategies are suggested. The Electrical Network Architecture Design Optimisation Tool, a tool that enables the application of the proposed method to electric systems, is developed.

Tundis et al. [124] develop a method to generate fault trees from Modelica models, where the basic events correspond to the probability of a failure of a component. The information required is extracted from two sources (see Figure 3.28):

1. **Fulfil relationships:** an extension of Modelica, normally used for tracing requirements, which provides information about the structure of the tree.

2. **Probability models:** list all possible states in which a component can be and the characteristics associated with each state. They provide probability values for each state employed in FTA.



```
model <ExtendedSystemDesignName>
    <ComponetModel cm1>
    ...
    <ComponetModel cmN>
    <Requirement r1>
    ...
    <Requirement rM>
equation
    cm1 fulfill r1;
    ...
    {cmN, r1} fulfill rM;
end <ExtendedSystemDesignName>;
```

```
package ProbabilityModel
    model ProbabilityModelComponent1
        parameter real status1 (start = 0.50);
        parameter real status2 (start = 0.50);
    equation
        status1 + status2 = 1.0;
    end  ProbabilityModelComponent1;
    ...
    ...
    model ProbabilityModelComponentN
        ...
    end  ProbabilityModelComponentN;
end ProbabilityModel;
```

Figure 3.28: Example of a fulfil relationship and a probability model (From [124])

3.7.2 Discussion

Several methods for automating the creation of fault trees are reviewed in this section. Most methods operate similarly, starting from a top event and traversing the system until all relevant parts are included in the tree. To improve the quality of the results, the fault tree creation algorithms are supplemented with additional information such as redundant groups, fault modes and effect, states and transitions and so forth. The method by Schallert [123] is an exception, as it uses simulation under different failure scenarios to determine FTA results, skipping the step of creating the fault tree. The most relevant differences between methods stem from the language or formalism required to express the inputs to the fault tree creation algorithms, and the level of detail required — which can be related to the design phase where the algorithms are expected to be more useful.

The (flow) connections between components at the same level are the most common source of information to construct the tree and seem to be considered either explicitly or implicitly in every method. The methods proposed in references [67, 71, 91, 120] also include hierarchical information such as expressing the state of composite components in terms of their children state. Additionally, more specific safety and reliability information is supplemented to the fault tree creation process in various formats, namely Maude language [114], AADL Error-Annex [67, 69, 118], Altarica [120], expert judgement from a HAZOP analysis [91] or various extensions to Modelica [123, 124]. The level of detail employed by the methods correlates with the amount of information required, e.g. methods that require state and transition information are more detailed than methods that required connectivity only.

The main limitation of the reviewed methods is that they require the architectural information to be translated to various safety-specific languages or tools such as Altarica and the Safety Argument Manager. These tools, except for the AADL, which supports architecture definition, are not capable of providing support for other system development activities such as architecture definition or sizing and do not seem to be easy to integrate with other existing support tools capable of providing it. The methods relying on Simulink or Modelica mitigate the sizing part of the problem as, although the focus of these languages is simulation rather than sizing, they can be adapted for this task. The approaches based on SysML, AADL and DSMs provide better support for architecture definition but lack support for sizing.

3.8 Safety Architecting

The degree of safety resulting from an architecture depends on various aspects. The type of components used, different materials or manufacturing techniques can lead to different probabilities of failure; the number of components and how the components are connected, different kinds and degrees of redundancy can be achieved depending on this factor as well as the possibility of dynamic reconfiguration of the system; the control algorithms, process models, software and human-machine interfaces are some of the most relevant elements impacting safety. Safety and resilience literature contains information about how to make systems safer acting in one or more of the aspects enumerated above. This thesis refers to these strategies as safety principles, a term that is defined below.

> ***Safety principle:*** *a general design strategy that is expected to result in a safer system and that can be applied to the designs of diverse systems, e.g. it is not tied to a particular project or kind of system.*

This section presents various safety principles from different domains and discusses their applicability within the scope of the current research, resulting in the selection of the three most relevant principles.

3.8.1 Review of Safety Principles

Burch [56] proposes to increase the resilience of space systems by means of two general mechanisms: elemental protection and distribution of system capability. The former consists of providing the susceptible components with built-in protection against specific threats such as solar flares or space debris. This mechanism can be generalized to other kinds of systems by considering the relevant threats for each context. The second mechanism, distribution of system capability, consist of partitioning the total system capability among several system elements, increasing mission capability beyond minimum performance requirements to improve resilience. Burch argues that distribution is an attractive means of increasing resilience as this principle is threat agnostic. Four methods of distributing system capability are proposed.

- **Division:** distribution of the total capability, uniformly or not, among several elements.
- **Proliferation:** adding one or more elements of the same size to provide oversupply.
- **Disaggregation:** separation of functions, reducing the use of multifunctional components allocating functions into discrete sub-elements.
- **Fractionation:** is defined as the segmentation of system capability by sub-functions among multiple elements.[†]

Jackson and Ferris [49] proposed a set of resilience principles and subprinciples applicable to the design of engineered systems. The origin of these principles can be traced back to the works of Madni and Jackson [40] and Jackson [55, pp. 159–180] where many of them are presented. The principles are presented in a non-hierarchical manner and, for this research, risk and culture heuristics have been left out of scope. The applicability of such a set to the commercial aircraft domain is discussed in [51]. Uday and Marais [50] select the most relevant resilience heuristics within the context of systems of systems and relate them to resilient attributes such as impact reduction, survivability, recoverability, and total time to recover. The set is eventually updated in [125], with some principles being renamed and the addition of some subprinciples. A comparison of these works can be found in Table 3.7, together with principles proposed by other authors that will be introduced later in this section.

[†]This definition is to some extent is unclear and, unfortunately, Burch [56] does not provide further information. It is interpreted in this thesis that it implies some kind of functional redundancy.

The descriptions of the principles followed by the descriptions of their corresponding subprinciples are presented next. The definitions are based on Jackson and Ferris [125]. The name of the principle (or subprinciple) is shown in bold font. It is followed by the reference to the original source that inspired the principle. The text after the colon corresponds to the description of the principle.

- **Absorption**, Woods [126]: the system shall be capable of withstanding the design level disruption.
  - **Margin**, Woods [126]: the design level shall be increased to allow for an increase in the disruption.
  - **Hardening**, Richards [127]: the system shall be resistant to deformation.
  - **Context Spanning**, Jackson [55]: the system shall be designed for the maximum and most likely disruption levels.
  - **Limit Degradation**, Jackson and Ferris [49]: the system shall not be allowed to degrade due to ageing or poor maintenance.
- **Restructuring**, Woods [126]: the system shall be capable of restructuring itself.
  - **Authority escalation**, Maxwell and Emerson [128]: authority shall escalate in accordance with the severity of the crisis.
  - **Regroup**[‡]: the system shall restructure itself after an encounter with a threat.
- **Reparability**, Richards [127]: the system shall be capable of repairing itself. Repairability also appears as the ability of a system to being brought up to partial or full functionality over a specified period of time and in a specified environment (Jackson and Ferris [49]).

[‡]Jackson and Ferris [125] claim the source of this principle to be a comment made by A. Raveh in 2008 during tutorial on resilience in Utrecht, The Netherlands.

- **Drift Correction**, Woods [126]: the system shall perform corrective action when approaching the boundary of resilience.
  - **Detection**, Jackson and Ferris [49]: the system shall be capable of detecting an approaching threat.
  - **Corrective Action**, Jackson and Ferris [49]: the system shall be capable of performing a corrective action following a detection.
  - **Independent Review**, Haddon-Cave et al. [129]: the system shall be capable of detecting faults that may result in a disruption at a later time.
- **Cross-scale Interaction**, Woods [126]: all nodes of a system should be capable of communicating, cooperating, and collaborating with every other node.
  - **Knowledge Between Nodes**, Billings [130]: states that all nodes of a system should be capable of knowing what all the other nodes are doing.
  - **Human Monitoring**, Billings [130]: automated systems should understand the intent of the human operator.
  - **Automated System Monitoring**, Billings [130]: the human should understand the intent of the automated system.
  - **Intent Awareness**, Billings [130]: all the nodes of a system should understand the intent of the other nodes.
  - **Informed Operator**, Billings [130]: the human should be informed as to all aspects of an automated system.
  - **Internode Impediment**, Jackson [55]: there should be no administrative or technical obstacle to the interactions among elements of a system.

- **Functional Redundancy**, Leveson [131]: there should be two or more independent and physically different ways to perform a critical task.
- **Physical Redundancy**, Leveson [131]: the system should possess two or more independent and identical legs to perform critical tasks.
- **Layered Defence**, Reason [132]: the system should have more than one way to address a vulnerability.
- **Neutral State**, Madni and Jackson [40]: human agents should delay in taking action to make a more reasoned judgement. This is enabled by the ability of the system to be in a neutral state following a disruption (Madni and Jackson [40]).
- **Human in the loop**, Madni and Jackson [40]: there should always be human in the system when there is a need for human cognition.
  - **Automated function**, Billings [130]: humans are preferred to perform a function rather than automated systems when conditions are acceptable.
  - **Reduce Human Error**, Billings [130], Reason [77]: standard strategies should be used to reduce human error.
  - **Human in Control**, Billings [130]: humans should have final decision-making authority unless conditions preclude it.
- **Complexity Avoidance**, Neches and Madni [43], Perrow [133]: the system should not be more complex than necessary.
  - **Reduce Variability**[§]: the relationship between the elements of the system should be as stable as possible.

[§]The source of this principle is an email communication between J. Marczyk and the authors Jackson and Ferris [125] in 2012 in Como, Italy.

- **Reduce Hidden Interactions**, Leveson [131], Perrow [133]: potentially harmful interactions between elements of the system should be reduced.
- **Modularity**, Madni and Jackson [40], Perrow [133]: the functionality of a system should be distributed through various nodes.
- **Loose Coupling**, Perrow [133]: the system should have the capability of limiting cascading failures by intentional delays at the nodes.
  - **Containment**, Jackson and Ferris [49]: the system will assure that failures cannot propagate from node to node.

The work by Madni and Jackson [40] proposes some additional heuristics:

- **Human Backup**, Madni and Jackson [40]: humans should be able to back up automation when context changes in a way to which automation is not sensitive provided there is sufficient time.
- **Predictability**, Madni and Jackson [40]: automated systems should behave in predictable ways to assure trust and not evoke frequent human override.
- **Graceful degradation**, Madni and Jackson [40]: the performance of the system should degrade gradually when the unexpected occurs.
- **Inspectability**, Madni and Jackson [40]: the system should allow for human intervention without making unsubstantiated assumptions.
- **Learning – Adaptation**, Madni and Jackson [40]: continually acquiring new knowledge from the environment to reconfigure, reoptimize and grow.

In a different work, Jackson [55] proposes a similar set of principles. In this case, adaptability is not considered as a heuristic but as an umbrella term encompassing the heuristics presented above. This work [55] also includes a new principle and borrows five more from Richards et al. [63]:

- **Organizational Planning Heuristic**, Jackson [55]: signs should be noticed that call into question organizational models, plans, and routine.
- **Mobility**: the system should be able to avoid a threat by moving
- **Prevention**: the system should be able to suppress future potential disruptions.
- **Retaliation**: the system should be able to retaliate to a threat.
- **Concealment**: the system should attempt to conceal itself against potential threats.
- **Deterrence**: the system should attempt to deter hostile threats from attacking.

Some of these principles imply causes of failure due to active and hostile intent, which are causes that are out of the scope of this research. The principles are presented here for completeness, but they will be discarded during the discussion of the most relevant principles (see Section 3.8.2).

Richards et al. [134] extend the set design principles shown above. Although the authors refer to survivability, the definition of the term used by the authors is close to the definition of resilience used in this research. Therefore, the principles are considered relevant for this literature review. Presented below is the complete list of principles in Reference [134], which includes descriptions only for the items that have not appeared yet in this section:

- **Prevention**.
- **Mobility**.
- **Concealment**.
- **Deterrence**.

- **Preemption**: suppression of an imminent disturbance.
- **Avoidance**: manoeuvrability away from an ongoing disturbance.
- **Hardness**.
- **Redundancy**.
- **Margin**.
- **Heterogeneity**.
- **Distribution**.
- **Failure mode reduction**: elimination of system hazards through intrinsic design.
- **Fail-safe**: prevention or delay of degradation via physics of incipient failure.
- **Evolution**: alteration of system elements to reduce disturbance effectiveness.
- **Containment**.
- **Replacement**: substitution of system elements to improve value delivery.
- **Repair**.

Pumpuni-Lenss, Blackburn and Garstenauer [135] propose a series of attributes of resilience, classified according to the four components of resilience proposed by the (Department of Defense of the United States of America, 2011). These attributes are avoidance, robustness, reconstitution and recovery. As shown in Table 3.7, this work is similar to the work by Richards et al. [134] — at least at a superficial level, since definitions for the terms are not provided. Ouyang, Dueñas-Osorio and Min [136] compile a series of strategies to improve infrastructure system resilience according to three stages of resilience, which are based on resistant, absorptive, and restorative capacities. More detailed descriptions of these strategies are also missing in this case.

Other authors, propose improving system resilience through the implementation of other -ilities rather than by including a particular design principle. For instance, Enos [42] propose improving adaptability, extensibility, flexibility, repairability and versatility as a means of achieving resilience. This view is in agreement with that of Uday and Marais [50], who propose the improvement of flexibility, robustness, and adaptability of the constituent elements of a system of systems. Yodo and Wang [137] review various resilience metrics and conclude that resilience is achieved via reliability, survivability, and recoverability.

Table 3.7: Comparison of safety principles

Madni and Jackson [40]	**Jackson [55]**	**Jackson and Ferris [49]**	**Jackson [51]**	**Jackson and Ferris [125]**	**Uday and Marais [50]**	**Richards et al. [134]**	**Pumpuni-Lenss, Blackburn and Garstenauer [135]**	**Ouyang, Dueñas-Osorio and Min [136]**
	Absorption	Absorption	Absorption	Absorption				
		— Limit degradation	— Limit degradation	— Limit degradation				
	Margin	— Margin	— Margin	— Margin		Margin	Over capacity. Excess margin	
Context spanning	Context spanning	— Context spanning		— Context spanning				
	Hardness			— Hardening		Hardness	High damage thresholds	Harden
Phys. Redundancy	Phys. Redundancy	Phys. Redundancy	Phys. Redundancy	Phys. Redundancy	Physical Redundancy	Redundancy	Active / Passive Redundancy	Redundancy
Funct. Redundancy	Funct. Redundancy	Funct. Redundancy	Funct. Redundancy	Funct. Redundancy	Funct. Redundancy	Heterogeneity		
		Layered Defence	Layered Defence	Layered Defence	Layered Defence			
Human in the Loop	Human in the Loop	Human in the Loop	Human in the Loop	Human in the Loop	Human in the Loop			
		— Human in control		— Human in control				
		— Reduce Human Error	— Reduce Human Error	— Reduce Human Error				
	Automatic Function Heuristic	— Automated Function	— Automated Function	— Automated Function				
Complexity avoidance	Complexity avoidance. Simplicity	Reduce Complexity	Reduce Complexity	Complexity avoidance				
				— Reduce variability				
Reorganization	Reorganization	Reorganization	Reorganization	Restructuring				Adjust topology
	Regroup	— Regroup		— Regroup				
		— Auth. Escalation		— Auth. Escalation				
		Repairability		Repairability	Repairability		Self-healing	Self-healing
	Repairability		Repairability			Repair	Repair	Recovery strategies

Madni and Jackson [40]	**Jackson [55]**	**Jackson and Ferris [49]**	**Jackson [51]**	**Jackson and Ferris [125]**	**Uday and Marais [50]**	**Richards et al. [134]**	**Pumpuni-Lenss, Blackburn and Garstenauer [135]**	**Ouyang, Dueñas-Osorio and Min [136]**
		Localized Capacity	Localized Capacity	Modularity	Localized Capacity			
	Loose Coupling	Loose Coupling	Loose Coupling	Loose Coupling		Distribution		
		— Containment		— Containment		Containment		
Drift correction	Drift correction	Drift correction	Drift Correction	Drift correction	Drift correction			Sense, monitor, update
		— Independent review		Independent review				
				— Detection				
				— Corrective action				
Neutral State	Neutral State	Neutral State	Neutral State	Neutral State			Reset. Restart	
		Inter-Node Interaction	Inter-Node Interaction	Cross-scale Interaction	Inter-Node Interaction/Improved communication			Efficient communication
	Inter-element impediment	— Inter-Node Impediment	— Inter-Node Impediment	— Inter-Node Impediment				
	Informed operator	— Informed Operator	— Informed Operator	— Informed Operator				
	Knowledge between nodes	— Knowledge between Nodes	— Knowledge between Nodes	— Knowledge between Nodes				
	Human Monitoring	— Human Monitoring	— Human Monitoring	— Human Monitoring				
	Automated System Monitoring	— Automated System Monitoring	— Automated System Monitoring	— Automated System Monitoring				
Intent Awareness	Intent Awareness			— Intent awareness				
	Hidden Interaction	Reduce Hidden Interactions	Reduce Hidden Interactions	Reduce Hidden Interactions				

Madni and Jackson [40]	**Jackson [55]**	**Jackson and Ferris [49]**	**Jackson [51]**	**Jackson and Ferris [125]**	**Uday and Marais [50]**	**Richards et al. [134]**	**Pumpuni-Lenss, Blackburn and Garstenauer [135]**	**Ouyang, Dueñas-Osorio and Min [136]**
Human Backup	Human Backup							
Predictability	Predictability							
Graceful degradation	Graceful degradation							
Inspectability	Inspectability							
Learning. Adaptation								Learn and improve
	Organizational Planning							
	Mobility					Mobility	Mobility	
	Prevention					Prevention	Neutralization	
	Retaliation							
	Concealment					Concealment	Covertness	
	Deterrence					Deterrence	Deterrence	
						Pre-emption	neutralization	
						Avoidance	Manoeuvrability	
							Countermeasures	
					System-level Properties (ilities)			
						Failure Mode Reduction		
						Fail-safe		
						Evolution		
						Replacement	Replace	
							Rebuild	

3.8.2 Discussion of the Most Relevant Principles

The principles presented above are generic in nature and their applicability spans several kinds of systems and design stages. However, this research is focused on aircraft system architectures at early design stages such as conceptual design. Therefore, it is necessary to critically evaluate the appropriateness of each heuristic within this context. The criteria to downselect the set of principles is presented throughout this section.

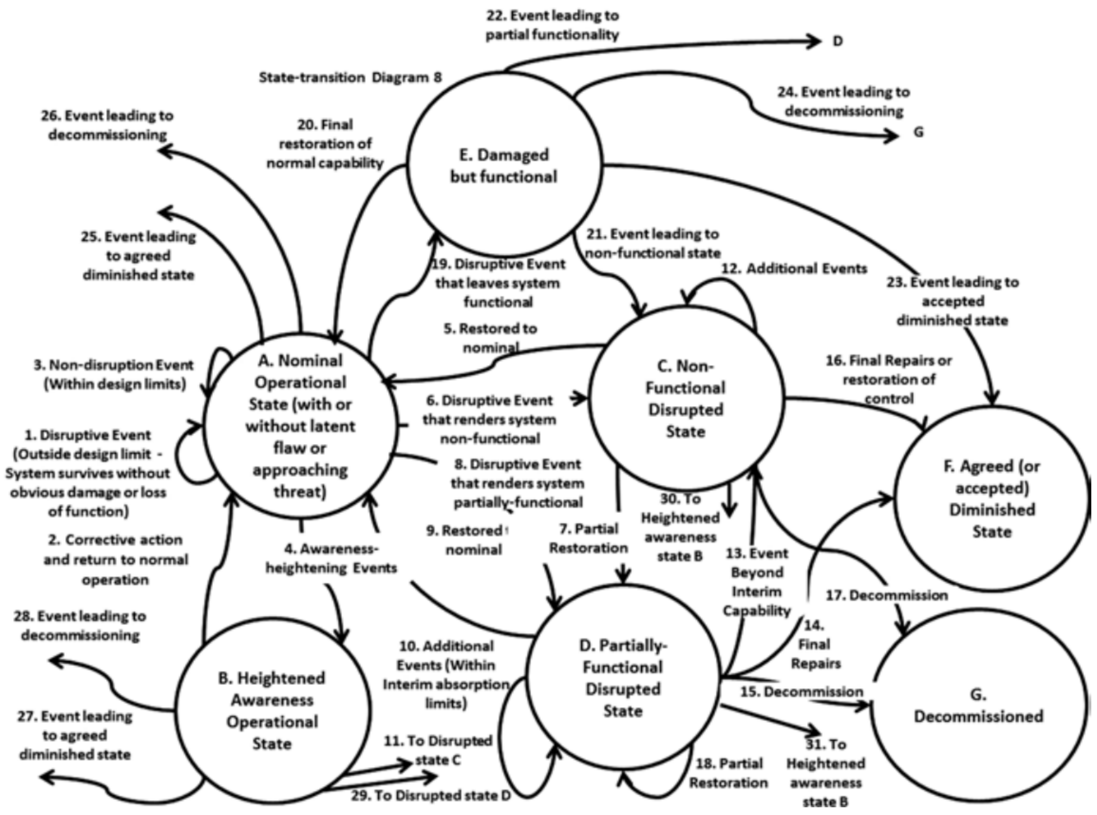

State-transition Diagram 8
22. Event leading to partial functionality
D
24. Event leading to decommissioning
G
26. Event leading to decommissioning
20. Final restoration of normal capability
E. Damaged but functional
25. Event leading to agreed diminished state
21. Event leading to non-functional state
19. Disruptive Event that leaves system functional
12. Additional Events
23. Event leading to accepted diminished state
5. Restored to nominal
3. Non-disruption Event (Within design limits)
A. Nominal Operational State (with or without latent flaw or approaching threat)
C. Non-Functional Disrupted State
16. Final Repairs or restoration of control
6. Disruptive Event that renders system non-functional
1. Disruptive Event (Outside design limit - System survives without obvious damage or loss of function)
8. Disruptive Event that renders system partially-functional
F. Agreed (or accepted) Diminished State
30. To Heightened awareness state B
9. Restored to nominal
7. Partial Restoration
2. Corrective action and return to normal operation
4. Awareness-heightening Events
13. Event Beyond Interim Capability
17. Decommission
28. Event leading to decommissioning
14. Final Repairs
10. Additional Events (Within Interim absorption limits)
D. Partially-Functional Disrupted State
B. Heightened Awareness Operational State
15. Decommission
G. Decommissioned
27. Event leading to agreed diminished state
11. To Disrupted state C
31. To Heightened awareness state B
18. Partial Restoration
29. To Disrupted state D

Figure 3.29: State transition diagram (From [125])

The work by Jackson and Ferris [125] describes the system during operation by means of a state diagram and matches a set of resilience principles with state transitions in the diagram. The diagram, shown in Figure 3.29, intends to be generic and the authors claim that it includes all the states and transitions that could be meaningful in any system. Whether all states are relevant and what conditions trigger the transition from one state to another need to be determined for a particular system, which explains the lack of information regarding transitioning conditions presented in the diagram.

The possible states of a system range from nominal operation state to decommissioned, including several intermediate states. Transitions represent disruptions, repairs and other kinds of events. Some of the states (e.g. non-functional, diminished and decommissioned states) describe situations that are out of the scope of this research. Therefore, safety principles that are associated with a transition from or to the non-relevant states can be excluded.

Jackson [51, pp. 229–242] discusses several design rules within the context of commercial aviation. The information provided helps to relate the abstract principles to a more concrete kind of design, commercial aircraft designs. However, the analysis presented by Jackson includes consideration for later design or operational phases, which are not the subject of this thesis. Principles such as *Repairability*, *Drift Correction*, *Neutral State* and *Human in the Loop* are not considered further as their correct application requires modelling of human behaviour, control strategies, or repair strategies, which is out of the scope of this research.

Other principles such as *Human Backup*, *Predictability*, *Inspectability* and *Organizational Planning and Learning* are discarded because they are strongly coupled with human behaviour and automation. The principles of *Mobility*, *Prevention*, *Retaliation*, *Concealment*, *Deterrence*, *Pre-emption*, and *Countermeasures* are most applicable under a military context, where failures are due to active and hostile intent, and thus also excluded; the focus is on civil aviation. The heuristic of *Avoidance* is mainly related to the manoeuvrability of the aircraft, which is mostly determined by parts of the design that are out of scope in this research such as the aerodynamic design and airframe configuration of the aircraft.

Improving the design of individual components or modifying their layout and interrelations are central to the *Fail Safe* and *Failure Mode Reduction* principles.

The design of individual components is out of scope, which leads to both principles being discarded. *Evolution*, *Replacement* and *Rebuild* focus mainly on the ability of the system to provide desirable attributes (e.g. safety or performance) as the design evolves over time. However, this research is not focused on the evolvability of the design.

After applying all the considerations discussed in the paragraphs above, the following reduced set of safety principles (and subprinciples) is obtained:

- **Absorption.**
  - Margin.
  - Hardening.
  - Context spanning.
- **Physical Redundancy.**
- **Functional redundancy.**
- **Loose coupling.**
  - Containment.
- **Complexity avoidance.**
  - Reduce variability.
- **Hidden Interaction Avoidance.**
- **Layered Defence.**
- **Graceful Degradation.**
- **Restructuring.**
  - Regroup.

- **Modularity**

However, there are still too many principles in this list to implement architecting support for each one of them in the time available for the research. To further reduce the number of principles, three additional criteria were employed:

1. **Interdependence of principles:** as noted by Jackson and Ferris [49], not all principles can be applied independently. The correct application of many of them requires implementing two or more at the same time. Combining the interdependency information provided in [49] with the information presented in this section, the hierarchy presented in Figure 3.30 was elaborated. Leaf principles (rightmost node at each level) are the most important ones as they enable the rest of the principles.

2. **Existing RFLP architecture representation:** the ability to model the relevant details of the system so that support can be provided to the application of a principle was considered. The RFLP model with computational extensions considered [14, 15] is able to model the principles of *Modularity*, *Containment*, *Physical Redundancy*, and *Functional Redundancy*. By creating computational workflows for the system and varying component parameters, *Margin*, *Context Spanning*, and *Hardening* can be considered.

3. **Existing support:** whereas for principles such as *Margin*, *Context Spanning*, and *Hardening* or *Modularity* there is a considerable amount of literature; no existing methods were found to support architects to implement *Containment*, *Physical Redundancy*, or *Functional Redundancy*.

With the above criteria in mind, *Physical Redundancy*, *Functional Redundancy* and *Containment* were selected as the most promising principles. There seem to be no existing methods to facilitate the inclusion of any of these methods in the design published in the literature except a publication [138] by the author of this research.

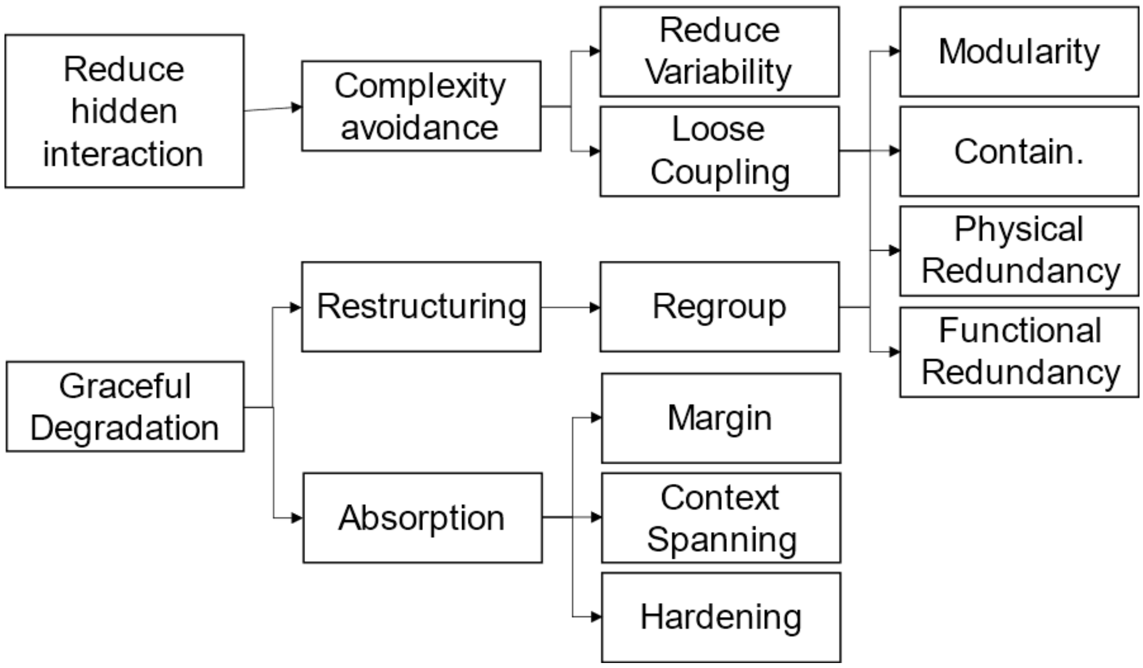


Figure 3.30: Interdependency of safety principles

3.9 Sizing and Performance Assessment

Section 3.8 shows how architecting safer systems, particularly by implementing the three safety principles selected in this thesis, requires substantial changes in the architecture. New components are added in the process and new connections are established, which will most likely impact important performance aspects such as the weight of the aircraft. Additionally, the demand of a component that fulfils a function might be shared across several components, depending on the possible failure conditions considered.

Determining the precise impact on performance of these changes requires sizing the system under different scenarios. The greater the number of safety options being considered the more frequent the sizing process needs to be executed. Due to this coupling between safety and performance, a fast sizing process becomes crucial to study different candidate architectures from both a performance and a safety point of view. This section presents existing methods for automating part of the sizing and performance assessment processes.

3.9.1 Existing Methods

Simulation framework by Liscouët-Hanke et al.

Liscouët-Hanke [139] and Liscouët-Hanke, Maré and Pufe [140] propose a methodology where the architecture is composed of power system modules representing the various aircraft systems. A power system module, as depicted in Figure 3.31a is influenced by aircraft and system parameters can be run in two modes (see Figure 3.31b):

- **Sizing mode:** computes sizing characteristics (e.g. diameters, sizes or masses) based on one or more sizing scenarios including different phases and operation modes (normal, degraded or failure mode).
- **Performance mode:** obtains off-design energy flows and drag values required for calculating aircraft level performance for the whole mission profile.

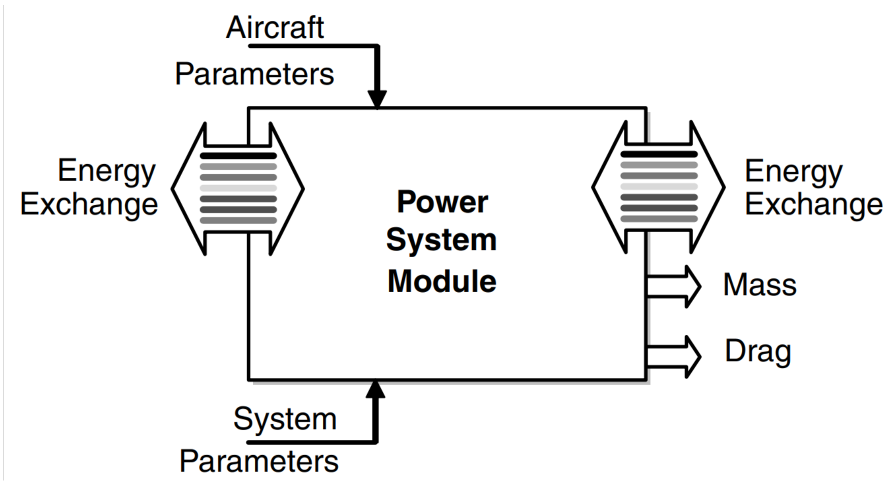
The power modules are orchestrated manually depending on whether they are power consumers, distributors or generators. Each subsystem in the workflow can be replaced by one that can provide the same function, which allows studying various technology choices.

Coordinated Optimization Method by de Tenorio et al.

The work by de Tenorio [141] and de Tenorio et al. [142, 143] employs a functional approach to select subsystem solutions, first from aircraft level requirements and then from induced requirements derived by the selected solutions. The execution of subsystem models is scheduled according to the functional flows between components, for example, if a pump provides energy to an actuator the actuator is sized first.

The subsystems are then sized to satisfy requirements throughout the mission, which is divided into several phases with flight conditions (e.g. ISA) and

(a) General power system module



(b) Framework overview

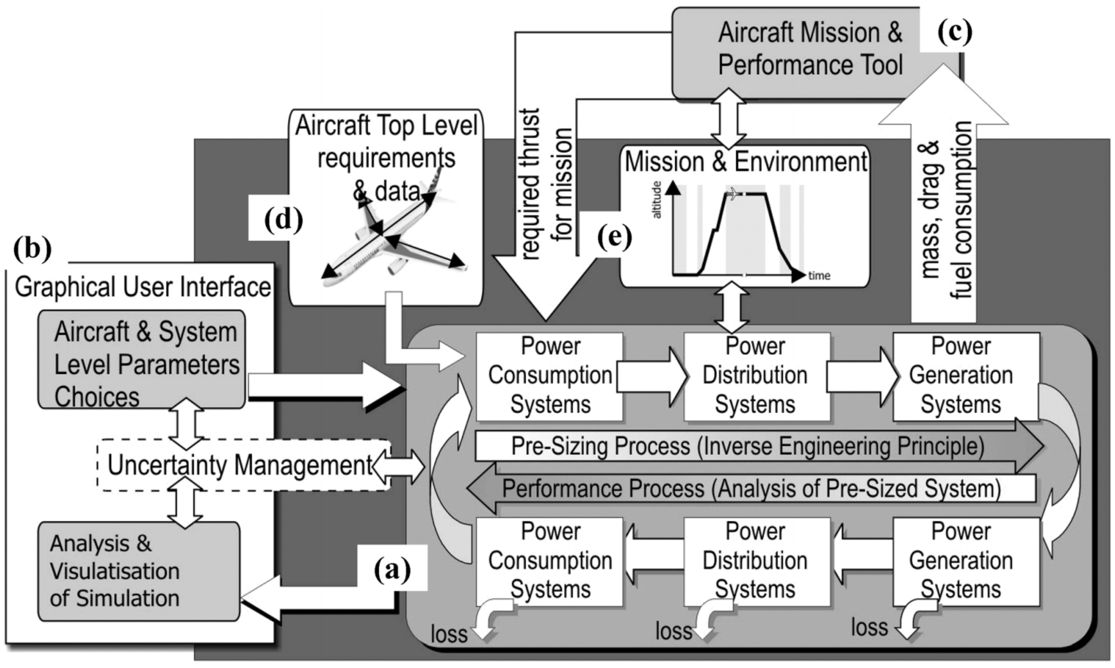

Aircraft Mission & Performance Tool
(c)
Aircraft Top Level requirements & data
(d)
required thrust for mission
(e)
Mission & Environment
altitude
time
mass, drag & fuel consumption
(b)
Graphical User Interface
Aircraft & System Level Parameters Choices
Uncertainty Management
Analysis & Visulatisation of Simulation
(a)
Power Consumption Systems
Power Distribution Systems
Power Generation Systems
Pre-Sizing Process (Inverse Engineering Principle)
Performance Process (Analysis of Pre-Sized System)
Power Consumption Systems
Power Distribution Systems
Power Generation Systems
loss
loss
loss

Figure 3.31: Simulation framework by Liscouët-Hanke et al. (From [140])

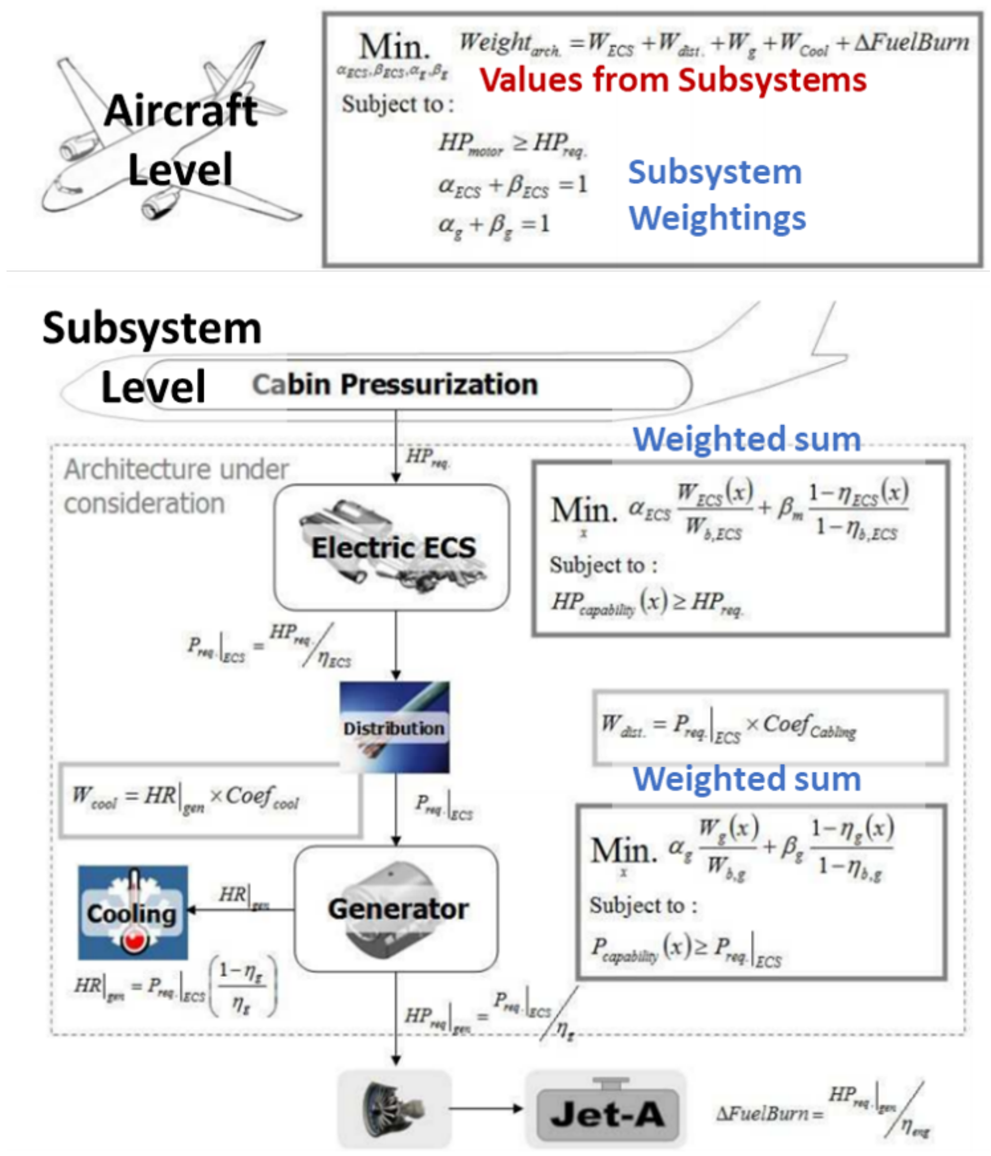

Aircraft Level
$\text{Min.}_{\alpha_{ECS},\beta_{ECS},\alpha_g,\beta_g}\ Weight_{arch.} = W_{ECS} + W_{dist.} + W_g + W_{Cool} + \Delta FuelBurn$
Values from Subsystems
Subject to :
$HP_{motor} \geq HP_{req.}$
$\alpha_{ECS} + \beta_{ECS} = 1$
$\alpha_g + \beta_g = 1$
Subsystem Weightings
Subsystem Level
Cabin Pressurization
HP_{req}
Architecture under consideration
Electric ECS
Weighted sum
$\text{Min.}_x\ \alpha_{ECS}\frac{W_{ECS}(x)}{W_{b,ECS}} + \beta_m\frac{1-\eta_{ECS}(x)}{1-\eta_{b,ECS}}$
Subject to :
$HP_{capability}(x) \geq HP_{req}$
$P_{req}|_{ECS} = HP_{req}/\eta_{ECS}$
Distribution
$W_{dist.} = P_{req}|_{ECS} \times Coef_{Cabling}$
Weighted sum
$W_{cool} = HR|_{gen} \times Coef_{cool}$
$P_{req}|_{ECS}$
$\text{Min.}_x\ \alpha_g\frac{W_g(x)}{W_{b,g}} + \beta_g\frac{1-\eta_g(x)}{1-\eta_{b,g}}$
Subject to :
$P_{capability}(x) \geq P_{req}|_{ECS}$
Cooling
$HR|_{gen}$
Generator
$HR|_{gen} = P_{req}|_{ECS}\left(\frac{1-\eta_g}{\eta_g}\right)$
$HP_{req}|_{gen} = P_{req}|_{ECS}/\eta_g$
Jet-A
$\Delta FuelBurn = HP_{req}|_{gen}/\eta_{eng}$

Figure 3.32: Example of sizing optimization problem (Adapted from [143])

architecture configurations associated with each one of them. Subsystems are sized via optimization, where the objective function equals to the weighted sum of several subsystem attributes. Varying the values of the weighting factors steer the development of the subsystem towards different objectives such as lighter subsystems or more energy-efficient ones. Aircraft level sizes and performance are computed based on subsystem sizing results. The optimization of aircraft performance consists of obtaining the combination of subsystem weighting factors that optimizes the utility of the aircraft. These factors, in turn, determine the subsystem attributes that lead to an aircraft optimum. The approach is illustrated in Figure 3.32.

Integrated sizing approach by Chakraborty et al.

The integrated sizing approach presented in Chakraborty [144], Chakraborty et al. [145] and Chakraborty and Mavris [146] allows exploring the design space generated by different subsystem choices, such as conventional or more electric systems. Subsystems are modelled using steady-state equations and empirical relations such as those captured in data tables, which provide the most relevant sizing parameter for a particular subsystem. For each candidate subsystems architecture, consisting of different choices for each aircraft subsystem, the models are orchestrated following power flow considerations. Powers consuming subsystems are linked to their respective distribution elements, which in turn are connected to the power generation and distribution devices. Sizes are obtained by considering a few points of the mission profile, which vary from subsystem to subsystem. The sizing points are selected a priori, generally based on results from previous works.

This sizing methodology also includes the aircraft level. As shown in Figure 3.33 the values obtained in mission analysis influence the sizing of the subsystems. In turn, subsystems influence the aircraft level via weight and drag contributions, and resizing rules applied to keep capabilities such as payload and range constant. The process needs to be iterated until convergence is achieved.

Automated Aircraft Systems Architecture Analysis by Judt and Lawson

The approach by Judt and Lawson [147, 148] utilizes a methodology that generates the full enumeration of architectures from a given function tree (see Figure 3.34). This function tree starts with the aircraft-level functions (boundary functions). From each function, one or more branches appear representing candidate

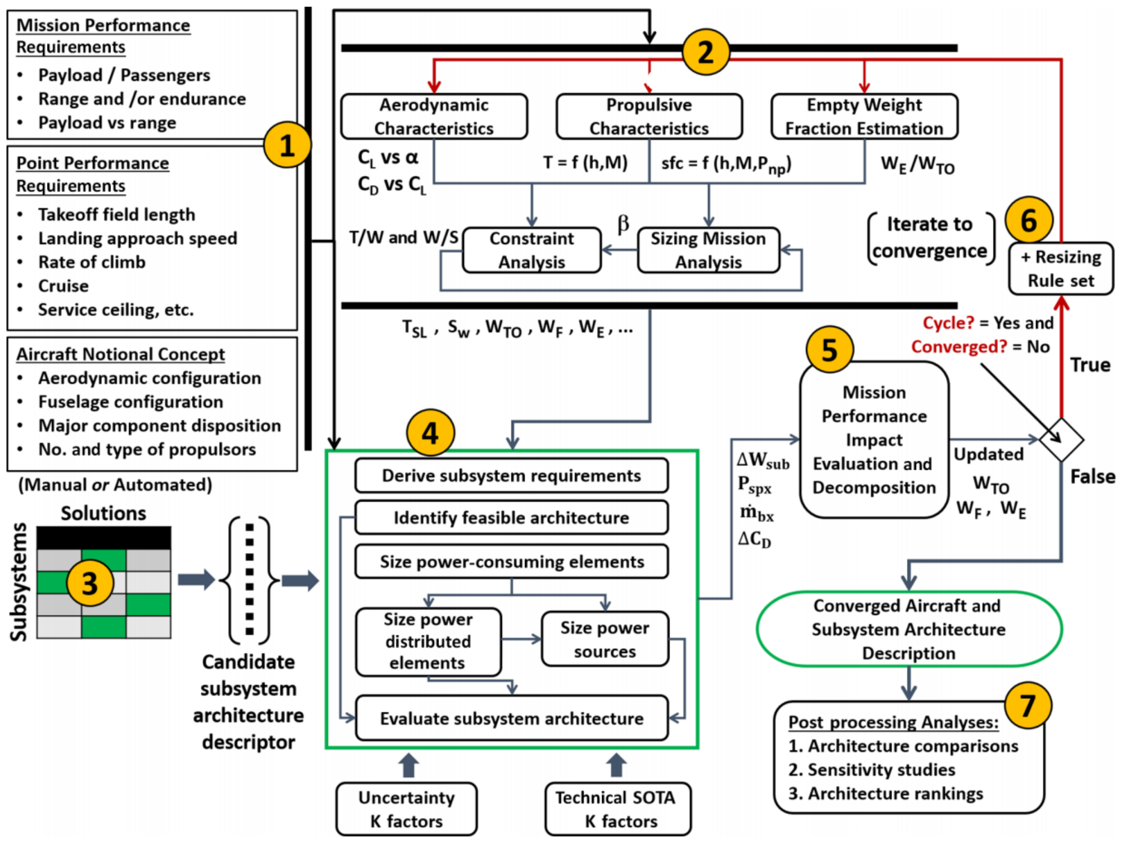

Mission Performance Requirements
• Payload / Passengers
• Range and /or endurance
• Payload vs range
Point Performance Requirements
• Takeoff field length
• Landing approach speed
• Rate of climb
• Cruise
• Service ceiling, etc.
Aircraft Notional Concept
• Aerodynamic configuration
• Fuselage configuration
• Major component disposition
• No. and type of propulsors
(Manual or Automated)
1
2
Aerodynamic Characteristics
Propulsive Characteristics
Empty Weight Fraction Estimation
C_L vs α
C_D vs C_L
T = f (h,M)
sfc = f (h,M,P_{np})
W_E/W_{TO}
T/W and W/S
Constraint Analysis
β
Sizing Mission Analysis
Iterate to convergence
6
+ Resizing Rule set
T_{SL}, S_W, W_{TO}, W_F, W_E, …
Cycle? = Yes and Converged? = No
True
5
Mission Performance Impact Evaluation and Decomposition
Updated W_{TO} W_F, W_E
False
Solutions
Subsystems
3
Candidate subsystem architecture descriptor
4
Derive subsystem requirements
Identify feasible architecture
Size power-consuming elements
Size power distributed elements
Size power sources
Evaluate subsystem architecture
ΔW_{sub}
P_{spx}
$\dot{m}_{bx}$
ΔC_D
Converged Aircraft and Subsystem Architecture Description
7
Post processing Analyses:
1. Architecture comparisons
2. Sensitivity studies
3. Architecture rankings
Uncertainty K factors
Technical SOTA K factors

Figure 3.33: Overview of the approach (From [146])

solutions to satisfy each function. The candidate solution might in turn induce more functions, which can be satisfied by various solutions and so forth.

The best architectures according to established criteria can be obtained in two different ways. The first methods consist in enumerating and sizing all architectures and then ranking them according to the criteria. The second method generates only the architectures required by an ant colony algorithm mixed with a genetic algorithm; this significantly reduces the number of architectures evaluated, although the obtention of the global optimum is not guaranteed. The order for subsystem execution depends on the interfaces between subsystems, which are determined manually and reviewed continuously until a sufficient level of confidence is obtained. Sizing of subsystems is based on a representative mission plus specific mission failure scenarios that affect the backup components that are active in such situations.

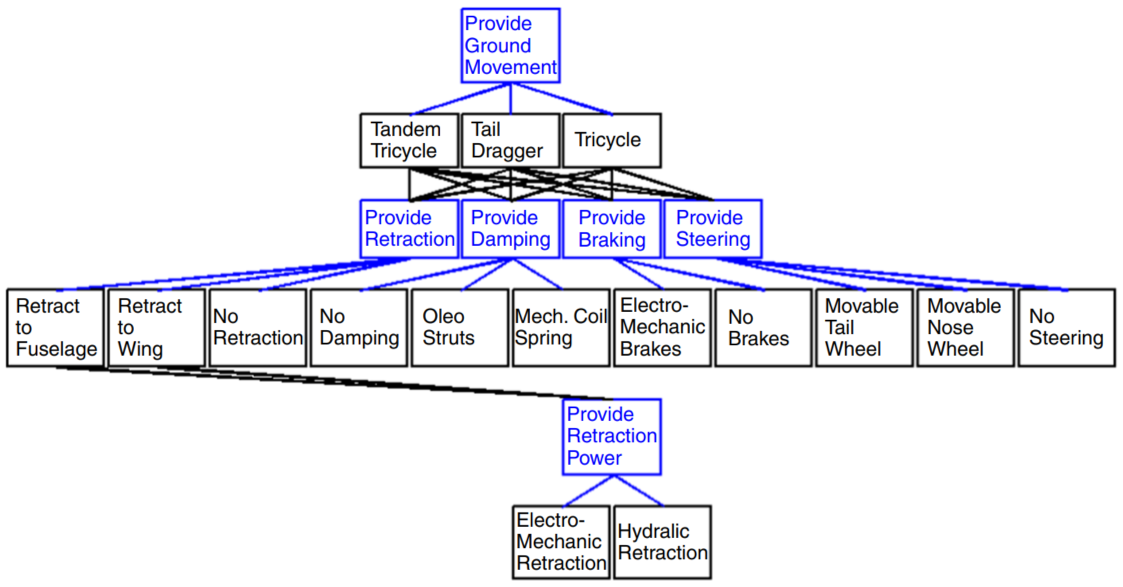


Figure 3.34: Example of a function tree (From [148])

RFLP Airframe System Sizing by Bile et al.

Bile [16] and Bile et al. [15] propose a framework for the definition and automated sizing of airframe systems defined by using the RFLP paradigm. Architecture definition occurs simultaneously in the functional and logical via functional-logical zigzagging [12]. This zigzagging happens when a component — included to fulfil a function — derives a new function to be fulfilled. The new function might be fulfilled by another component inducing new functions and so on.

Subsystem sizing is computed by a computational workflow that is generated from the logical view of the architecture, which includes a hierarchy of components connected through flow relations. At the bottom of the hierarchy, for each subsystem, the computational models associated with the subsystem components are compiled and connection models are generated according to flow connections. The models are scheduled within a computational workflow (see Figure 3.35) through a maximum matching algorithm that uses the model's input-output relations. Sizing is performed based on a single user-defined sizing point. At higher levels, the subsystem workflows are scheduled according to source-

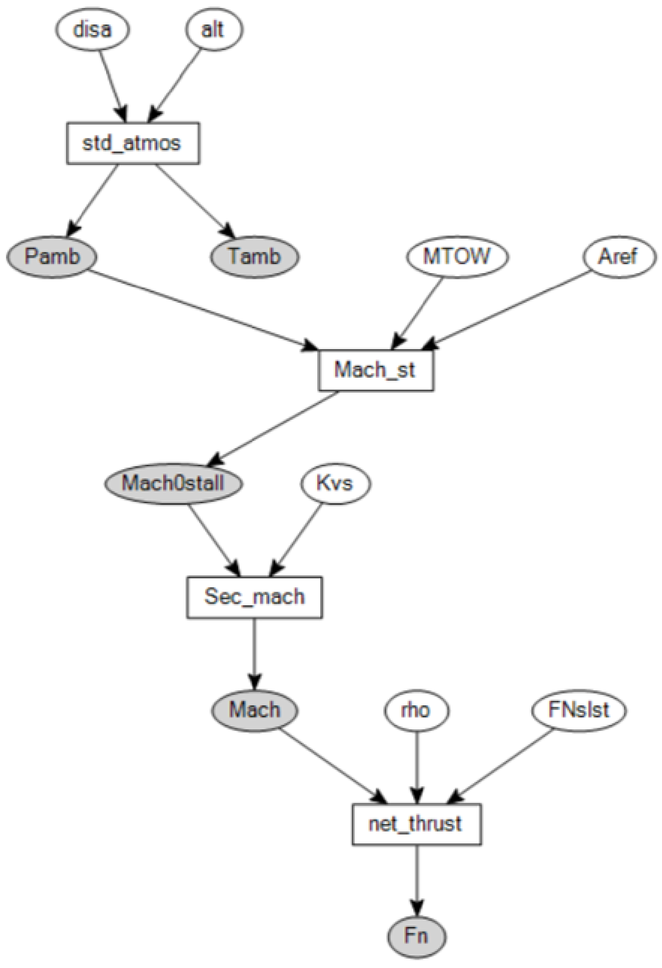


Figure 3.35: Example of a computational workflow (From [16])

sink relations, and value discrepancies at the interface between subsystems are minimised via optimisation. Magnitudes such as weight, drag or power consumption are aggregated up to aircraft level so that they can be considered for aircraft sizing.

3.9.2 Discussion

Various automated sizing approaches are reviewed in this section. All of them require the manual definition of architectures from a functional perspective, which facilitates the consideration of different options (e.g. regarding the technology for a particular subsystem). However, the level of detail varies from a set of predefined functions with various candidate solutions for each one of them in References [139, 140, 144–146] to function trees [141–143, 147, 148] and functional-logical zigzagging [15, 16]. A limitation of all methods except the one using RFLP is that alternative solutions are considered at a subsystem level. This limits the ability to consider changes at a component level, as they require the manual modification of subsystem models.

The number of sizing points considered ranges from a single point in References [15, 16] to several points in the rest. The method presented in References [144–146] relies primarily on past experience for selecting the points, whereas the rest of the methods explore various flight phases, flight conditions and configurations in a more systematic way. However, even the most flexible methods consider only a small number of scenarios, which might lead to incorrect sizing when considering novel configuration whose behaviour is not well understood.

3.10 Conclusions

This literature review intended to introduce the reader to the concepts involved in this research and to highlight the limitations of the existing safety-related methods. The following conclusions were drawn from the review:

- Reliability is a term often related to and sometimes mistaken with safety. However, although reliability failures can affect safety, this is not always the case. Unreliability is not necessarily unsafe, as some component failures do not have a significant impact on safety. Unsafe scenarios can arise without unreliability, such as in the accidents that are caused by interactions of components where none of them has failed. Resilience, an increasingly popular concept in literature, does not have a clear unique definition. In general, resilience is adopted to overcome the limitations of traditional safety approaches while considering systems with growing complexity and to study situations where safety is achieved dynamically (via reconfiguration of the system) rather than with a static configuration. In the case of aircraft early design (and systems of similar complexity), the use of the term resilience is deemed unnecessary. The reason is that the ultimate goal of resilience is generally the same as that of safety for the kind of systems and design stages within the scope of this research. However, some findings from resilience literature, such as the architecting principles reviewed in Section 3.8 are still considered to be relevant.

- The study of safety applied to the process of aircraft and system development highlighted the importance of the safety assessment process, used to show compliance with certification requirements. There are two main kinds of activities in the safety assessment process, namely hazard assessment (creation and verification of safety requirements) and safety analysis (validation of the system against safety requirements). As a result of safety assessment, two activities become necessary on the system development side: architecting of safety in the design and determination of the impact of safety in performance. Supporting all these activities in an integrated, timely manner is fundamental to increase the number of alternative architectures that can be studied, and therefore reduce the risk of redesign due to problems found in the latter design stages. It is also important to improve the consistency of the results from safety methods at different design stages and abstraction levels, which was found to be low due to the prominence of manual work required for safety-related activities.

- A fundamental part of hazard assessment and safety analysis methods are the accident models that underpin them. The validity of a method is determined by the combination of its underlying accident model and the nature of the system that is being analysed (e.g. degree of complexity or human involvement). After reviewing the most popular methods, the STPA hazard assessment method and FTA safety analysis method were identified as some of the most prominent approaches and good candidates to be supported via computational tools such as the one developed in this research.

- Several computational tools that support STPA were reviewed. Only one of the tools was found to support the automated creation of hierarchical control structures from the architecture definition. However, this tool does not support the creation of more detailed control loops. None of them supports the automated creation of both kinds of models. Furthermore, none of them was integrated nor easily integrable with other parts of the architecture definition such as requirements or functions.
- Various methods for automating the creation of fault trees were found. To define the inputs of the methods, the approaches require the architectural information to be translated to various specific languages. None of the languages was found to be suitable to provide or integrate easily with other tools that provide sizing and performance information.
- Several safety principles, many of them coming from the related field of resilience, were reviewed. Three of them, namely physical redundancy, functional redundancy, and containment were identified as the most applicable within the scope of this research, as their inclusion within the architecture is amenable to receive computation support.
- The available methods to support the sizing of complex systems present two kinds of limitations. Some of the methods require extensive manual setup or expert knowledge, hindering the ability to provide results swiftly after the architecture definition is modified. Others, although more automated, are too prescriptive in terms of which systems and components can be considered.

One method free of both limitations was found, however, its application is limited to a single flight condition, which is not generally enough to determine the most demanding conditions for a system.

In the next two chapters, it is demonstrated how the limitations presented above are addressed. Chapter 4 proposes an improved framework for RFLP architecture representation. This representation serves as the starting point for the methods developed by the author, which are introduced in Chapter 5. The methods cover STPA hazard assessment, FTA safety analysis, architecting of safety principles and sizing and performance of the resulting designs.

Chapter 4

Methodology I — Improved RFLP Architecture Representation

4.1 Introduction

This chapter is the first one of the two methodology chapters included in this thesis, which describe a novel architecting framework for designing safe systems. Chapter 5 presents the methods and algorithms in the framework, which intend to fulfil the objectives of this research (see Section 1.3). The methods require that existing RFLP frameworks are enhanced with the appropriate kind of elements and relationships. This chapter introduces such an improved RFLP representation. The presented elements and relationships are generic as they are not tied to any particular implementation. However, this chapter also suggests a concrete object model with the appropriate classes and associations, which make possible the realisation of design for safety methods.

The chapter is structured as follows: first, an overview of the framework is provided in Section 4.2, in which the main elements of the framework and their relationships are discussed. This is followed by a description of the data-objects used to represent RFLP definitions of architectures. Section 4.3 presents the relations that are common to objects belonging to two or more views of the architecture. The views and objects that compose them are detailed in Section 4.4. The relations involving elements from more than one view are presented in Section 4.5. Finally, Section 4.6 introduces template libraries, which are proposed to reduce the redefinition of elements and the reuse of past knowledge.

4.2 Description of the Architecting Framework

As shown in the literature review (see Chapter 3), safety assessment activities are done in parallel to other system development activities, including the definition of the architecture itself. There are many kinds of assessment activities required to successfully develop a system, nevertheless, this research focuses on three of them: STPA hazard assessment, fault tree analysis, and subsystems sizing. These activities, which relate to Objective 1 and Objective 3, correspond to the three boxes at the bottom of Figure 4.1.

As discussed in Section 2.2, the approach for systems engineering employed in this thesis is model-based. Consequently, a common model for the architecture is used by all the developed methods. As shown at the top of Figure 4.1, this research uses an RFLP representation of the architecture (the physical view is omitted as it is out of scope) and focuses on the architecting of safety principles, particularly physical redundancy, functional redundancy and containment. The architecting support for safety principles corresponds to Objective 2 of the research.

Both sides, architecture definition and analysis, advance iteratively. The architecture definition is the starting point for hazard assessment and other architecture analyses. In turn, the results from the analyses — safety requirements in the case of hazard assessment and various qualitative and quantitative metric in the case of other analysis — guide the definition of the architecture.

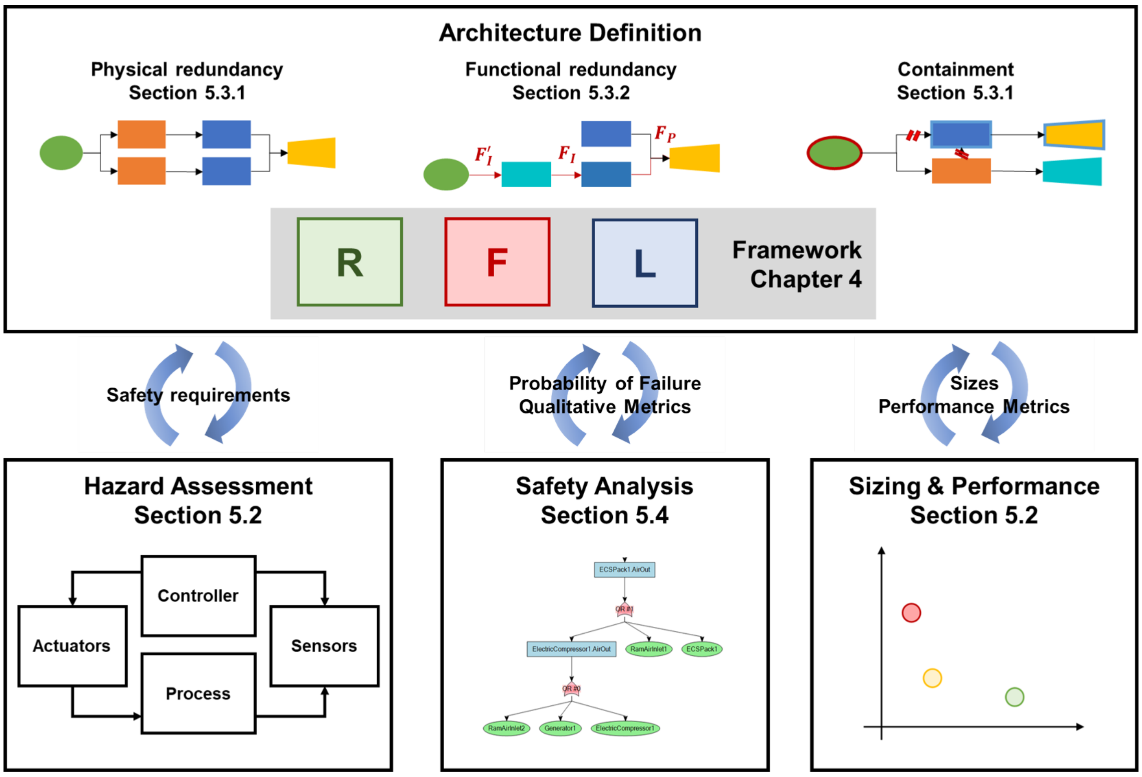


Figure 4.1: Overview of architecting activities

4.2.1 Improved RFLP Architecture Representation

The RFLP approach used in this research is based on the one proposed by Guenov et al. [14], which was developed to support interactive architecting within RFLP domains, is used as the foundation for the developed methods. However, the framework as proposed is not able to provide the required information for the methods proposed in this thesis. Therefore, changes are made to the framework to overcome this limitation. These changes are described throughout this section.

4.3 Common Relations

The views of the architecture considered in this research are composed of requirements, functions and solutions, which are referred to collectively as elements. Changes to the original RFLP framework affect the various classes that are used to represent these kinds of elements. New members are added to existing classes to include new pieces of information as required by the novel architecting enablers. Existing classes are also specialized to better represent the diversity of RFLP elements considered by the methods. In this section, the characteristics that are common to all or several elements are discussed first, followed by the description of the particular elements belonging to each of the RFLP views.

All RFLP elements of the architecture possess some common characteristics that are featured in the base class *ArchitecturalElement*. The salient attributes of this class, omitting implementation details, are presented in table 4.1. Figure 4.2 shows the inheritance hierarchy down to the base classes for requirements, functions and solutions.

Table 4.1: Description of ArchitecturalElement attributes.

Attribute Name	**Description**
Name	Word or brief sentence used to refer to a particular element, e.g. ‘Compress Air’ (function) or ‘Turbine’ (solution).

4.3.1 Hierarchical Relations

Decomposition (excluding iterative function-solution decomposition, also known as ‘zigzagging’) and aggregation relations [14] are common to requirements, functions and solutions. These relations enable a hierarchical representation of the views that contain these elements. The hierarchies can be modelled as a tree — a directed acyclic graph. The class *HierarchicalElement* is used as a based class for the architectural elements which form hierarchies as described. In turn, this class is derived from *ArchitecturalElement* because it is intended to model only elements that belong to the architecture. An example of a hierarchical view (a functional hierarchical view in this case) is presented in Figure 4.3. The parent and children of function *Control Mass Flow*, which is highlighted in red, are labelled accordingly

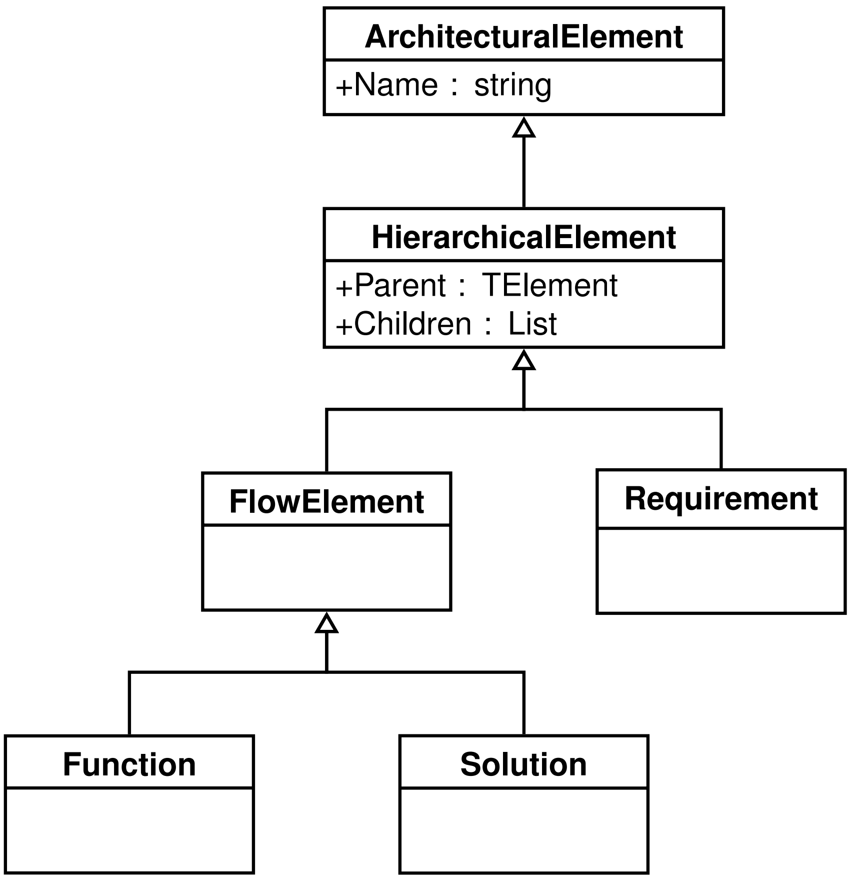


Figure 4.2: Class diagram for fundamental RFLP classes

Table 4.2 introduces the main attributes of *HierarchicalElement*. All parent and children elements are of the same kind of hierarchical element, e.g. if the element is a function, its parent and children will be functions as well. A parent element can be decomposed into their children and, conversely, all children of an element can be aggregated into that element.

Table 4.2: Description of HierarchicalElement attributes.

Attribute Name	**Description**
Parent	Element immediately higher in the hierarchy formed by decomposition and aggregation relations.
Children	Elements immediately lower in the hierarchy formed by decomposition and aggregation relations.

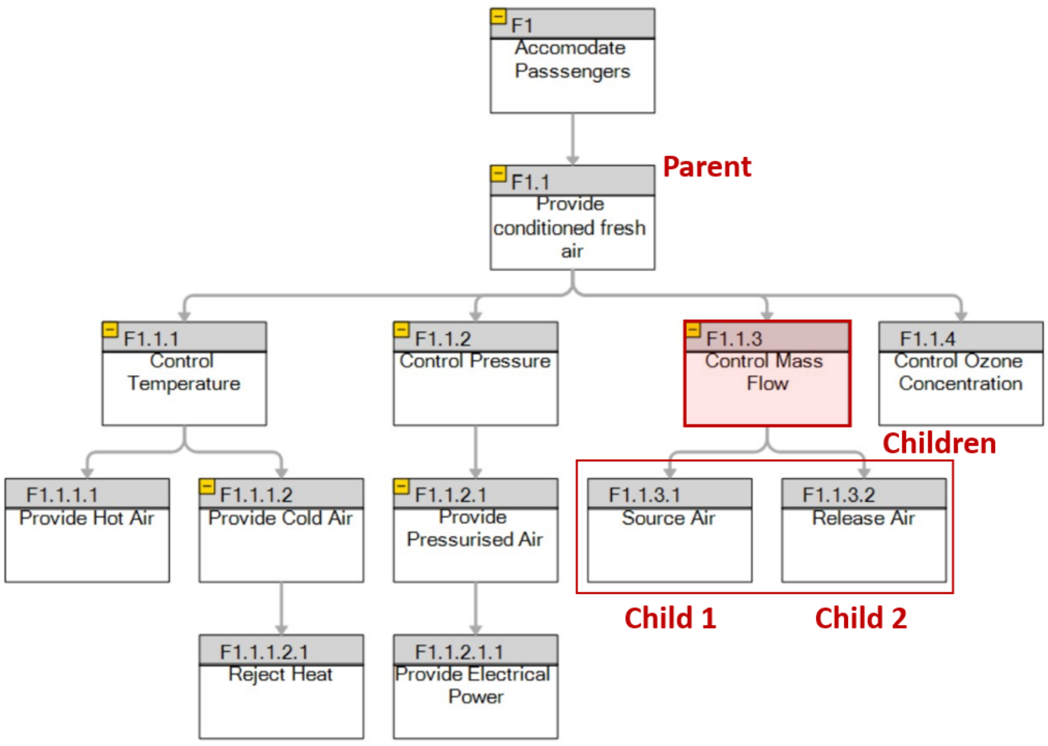


Figure 4.3: Example of a functional hierarchical view (Adapted from [138])

Top Element

Requirements and functional views do not always present a single top-level requirement/function. This means that instead of dealing with a single tree, the hierarchical relations are represented by a forest. To simplify the development of algorithms, by avoiding the special case of the forest, an artificial top-level element is added for all views. In the case of the logical view, the top element might be interpreted as the top-level system (e.g. the aircraft) but regarding the rest of the views, the interpretation is not straightforward beyond conveniently combining all trees.

4.3.2 Flow Relations

Flow relations arise because functions and logical components of the architecture exchange flows of energy, material or signal. Flow relations enable a flow representation of the respective views, which can be modelled as a directed graph (not acyclic in this case) [14]. The class *FlowElement* is used as a base class for architectural elements that exchange flows as described. In turn, this class is derived from *HierarchicalElement*, as in this research, all flow elements were also found to participate in hierarchies. An example of a logical flow is shown in Figure 4.4.

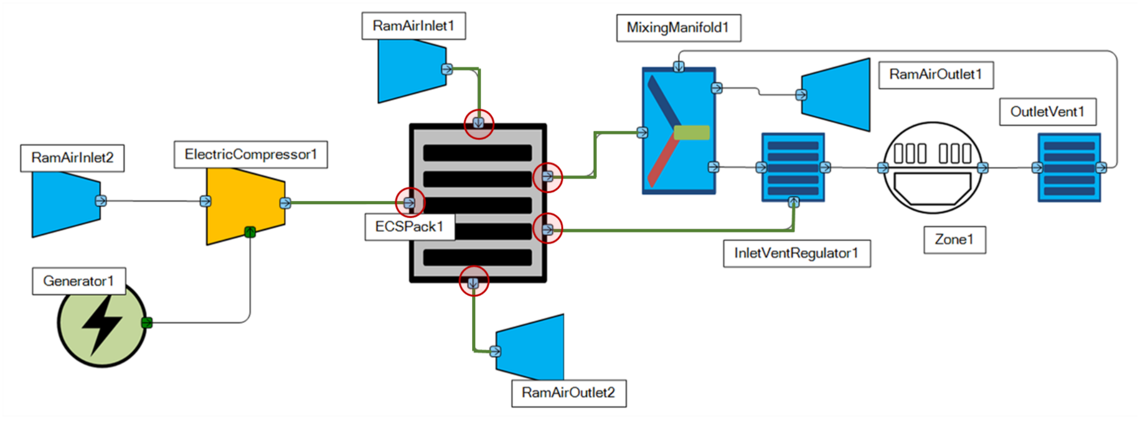


Figure 4.4: Example of a logical flow view

Table 4.3 introduces the main attributes of *FlowElement*. More details about ports, links and rules regarding flow exchange are provided in the sections below. The ports of the solution *ECSPack1*, which is a flow element, are circled in red in Figure 4.4. The links with one of their ends in these ports are highlighted in green.

Table 4.3: Description of FlowElement attributes.

Attribute Name	**Description**
InputPorts	Ports through which an energy, material or signal flow enters the element.
OutputPorts	Ports through which an energy, material or signal flow exit the element.
Ports	Combines input and output ports

Ports

Ports describe the interfaces of an element with other elements through which flow (energy, material, signal) is exchanged. Ports can be connected to other ports forming connections and links. The class *Port* is used to model ports and its attributes are summarised in Table 4.4. Three basic rules are proposed so that all port connections must adhere to:

1. An output port must be connected to an input port and vice versa.
2. A port cannot be connected to itself or any other port belonging to its containing element.
3. A port can be connected only to ports with a compatible *Flow* attribute.

Table 4.4: Description of Port attributes.

Attribute Name	**Description**
Name	Word or small set of words used to refer to a particular port within the element containing such port, e.g. ‘Air Input’ or ‘Rotational Output’
Parent	Element that contains the port
Flow	A flow is the object part of a ‘verb-object’ description of function [20]. It represents the energy, material or signal receiving the action of the function.
ExternalConnection	Object representing the connection that provides flow to or receives flow from the port.

Links

Links represent any pair of ports that are connected through a series of zero or more ports. Links represent the exchange of flow between two elements — as explained in Section 4.3.3. The class *Link* models links through the attributes presented in Table 4.5.

Table 4.5: Description of Link attributes.

Attribute Name	**Description**
FromPort	Port from which the flow originates. First port in *PathPorts*
ToPort	Port which receives the flow. Last port in *PathPorts*
PathPorts	Series of ports containing *FromPort*, *ToPort* and any intermediate ports in the case of links spanning multiple connections

Connections

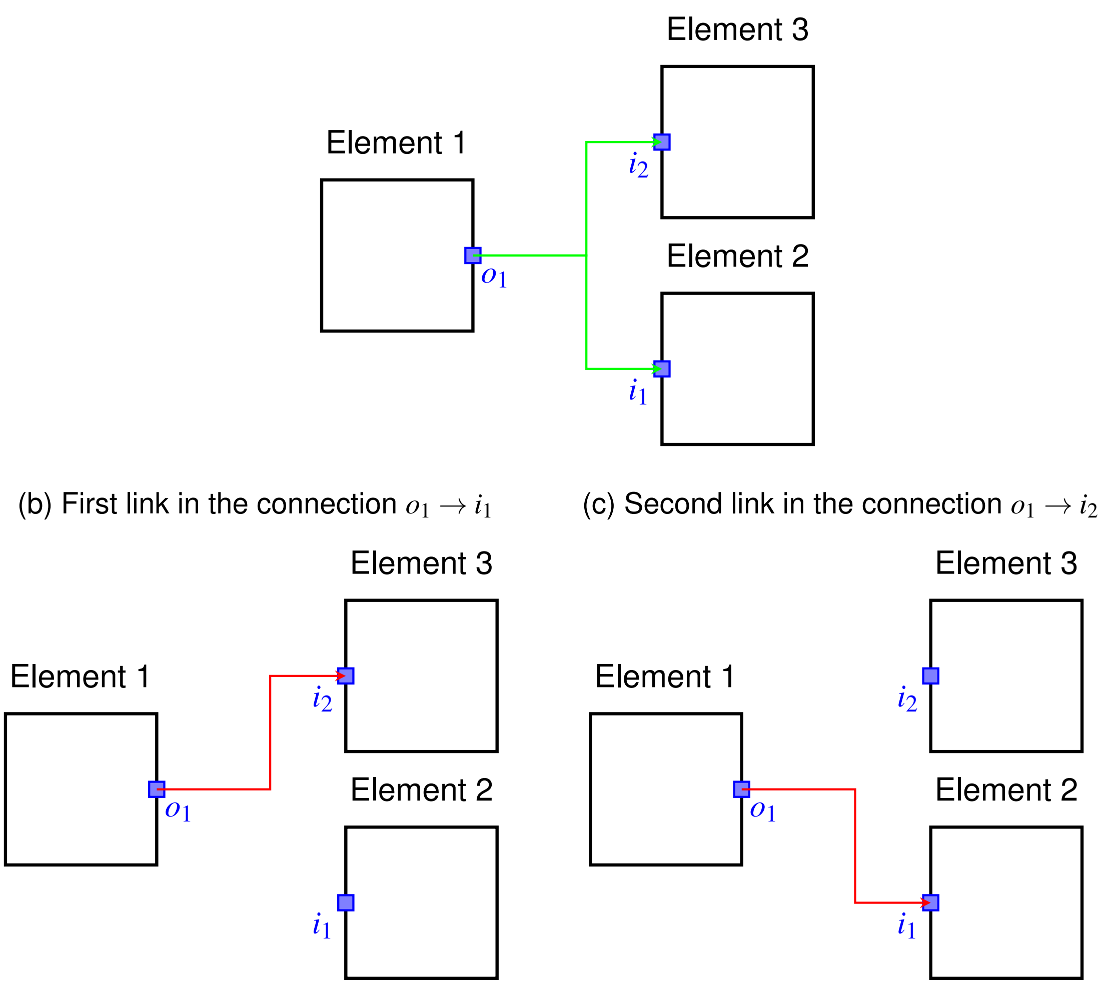
Connections are groups of links that represent the exchange of flow between two or more elements. A connection with N_i inputs and N_o outputs can be represented as N_iN_o links with only two ports in the path. As a consequence, the input flow is provided to the connection by one or more input ports, then the connection distributes the flow to one or more output ports. Figure 4.5 illustrates the difference between links and connections. Subfigure 4.5a shows a connection composed of two links, which are shown in Subfigure 4.5b and Subfigure 4.5b respectively.

As a consequence of the connection rules proposed in 4.3.2, all inputs ports of a connection are outputs ports of an element and vice versa (rule 1), no port appears more than once in a connection and the set of input elements is disjoint with respect to the set of output elements (rule 2), and all ports possess compatible flow attributes (rule 3). Connections are represented by the class *Connection*, whose attributes are summarised in Table 4.6.

Flows

As explained in the sections above, the concept of flow is fundamental for ports, connections and links. This concept originates from the functional basis for engineering design proposed by Hirtz et al. [20], which is introduced in Section 2.3.1.

Flows can be represented as a hierarchy with energy, material or signal flows



(a) Connection with one input o_1 and two outputs i_1 and i_2

(b) First link in the connection $o_1 \rightarrow i_1$

(c) Second link in the connection $o_1 \rightarrow i_2$

Figure 4.5: Equivalent links and connection

Table 4.6: Description of Connection attributes.

Attribute Name	**Description**
Flow	Energy, material or signal flowing through the connection.
InputPorts	Ports through which an energy, material or signal flow enters the element.
OutputPorts	Ports through which an energy, material or signal flow exit the element.
Ports	Combines input and output ports

at the top. These flows are decomposed at lower levels as more specific details are provided. The class modelling flows is the *Flow* class.

Table 4.7: Description of Flow attributes.

Attribute Name	**Description**
Type	Name of the flow as it appears in the functional basis, e.g. ‘gas’, ‘hydraulic’
Description	Description of the flow as provided in the functional basis, e.g. ‘Gas: Any collection of molecules characterized by random motion and the absence of bonds between the molecules’ [20]
Parameters	Series of variables used to describe the state of a particular flow. E.g. ‘pressure’, ‘density’ and ‘temperature’.
Parent	Flow immediately higher in the hierarchy.
Children	Flows immediately lower in the hierarchy.

Operations

Operations correspond to the term ‘function’ from the functional basis. In this thesis, the term operation is preferred as it avoids the confusion originated by the apparently recursive definition of functions consisting of a combination of flow and function. Operations can also be represented as a hierarchy with terms that can be decomposed at lower levels as their specificity increases. The class modelling flows is the *Operation* class, its most relevant attributes are described in Table 4.7.

4.3.3 Combined Hierarchical-Flow Relations

More complex flow relations appear when several hierarchical levels are considered compared to flow relations in a single hierarchical level. In order to handle flow connections between parents and children, additional rules for establishing

Table 4.8: Description of Operation attributes.

Attribute Name	**Description**
Type	Name of the operation as it appears in the functional basis, e.g. ‘connect’, ‘increment’
Description	Description of the operation as provided in the functional basis, e.g. ‘Increment: To enlarge a flow in a predetermined and fixed manner’ [20]
Syntax	Brief, structured description of the operation. For example ‘connect *flowA* to *flowB*’ or ‘increment *flowA*’
Inputs	Descriptions of the valid inputs for the operation: minimum and maximum number, and type of flows.
Outputs	Descriptions of the valid outputs for the operation: minimum and maximum number, and type of flows.
Constraints	Additional constraints imposed on input and output flows such as being of the same type or possessing a common parent.
Parent	Operation immediately higher in the hierarchy.
Children	Operations immediately lower in the hierarchy.

their validity connections are required as well as a more specialized type of ports referred to as internal ports.

Internal Ports

Internal ports are those ports that can be connected to child elements of the element containing the port. As a consequence, internal ports only exist in non-leaf elements — those who have children. Internal ports have two connections, an external connection like the one in common ports, and an internal connection to handle links with children of the element. An example of components with internal ports is presented in Figure 4.6, where both *EPS* and *FCS* possess this kind of ports.

The class *InternalPort* is used to model internal ports, it extends the class *Port* with the attribute presented in Table 4.9. There are three additional rules that all port connections must adhere to when considering combined hierarchical and flow relations:

1. An internal port can be connected internally only to ports belonging to a child of the element that contains the port.
2. A port (including internal ones) can only be connected externally to another port that belongs to an element at the same hierarchical level or to the parent of the element that contains the port (following the first set of rules in any case).
3. Internal connections are always either from an input port to another input port and from parent to children, or from an output port to another output port and from children to parent (following basic rules 1 and 2).

Table 4.9: Description of InternalPort attributes.

Attribute Name	**Description**
InternalConnection	Object representing the connection that provides flow to or receives flow from the port. The port at the other end of the connection belongs to any of the element's children.

Links Spanning Multiple Connections

In the case of internal ports, any link with any of its ends in an internal port can be extended by including links from the other connection (internal or external) of the internal port. This is the fact motivating the existence of the *PathPorts* attribute of class *Link*, which can be used to trace the flow path from one port to another through various internal ports. Figure 4.6 shows a link from an output port of *Generator* to an input port of *Actuator*, passing through two additional internal ports belonging to their respective parents, *EPS* and *FCS*.

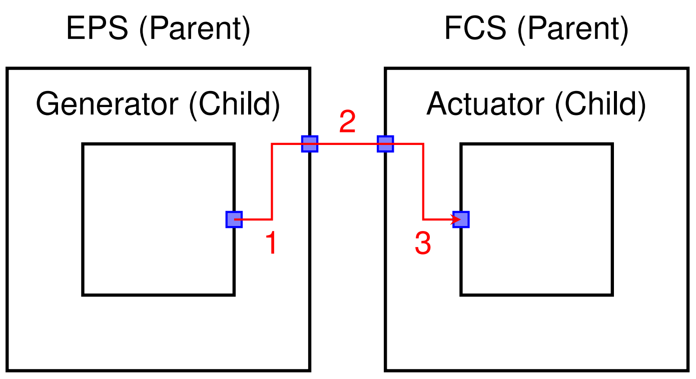


Figure 4.6: Example of a link spanning multiple connections

4.4 Views of the architecture

4.4.1 Requirements View

The first view that is analysed in detail is the Requirements View, whose principal element is the requirement. An example of a requirements view is shown in Figure 4.7. Three kinds of requirements are considered: functional, performance and safety requirements. Functional and performance requirements are those directly transformed from the stakeholders' needs [14]. Their main difference is that performance requirements need to be satisfied by obtaining a particular parameter value from the computational analysis within an acceptable range, whereas functional requirements are satisfied by the creation of a function and subsequent fulfilment of such function by one or more solutions. Safety requirements generally originate from the fact that the main stakeholders' needs (e.g. transport passengers and cargo in the case of aircraft) must be fulfilled safely (as required by law or regulations).

The base class for all requirements is the *Requirement* class. Requirements are represented respectively by classes *FunctionalRequirement*, *Performance-Requirement* and *SafetyRequirement*. The class diagram for requirements is presented in Figure 4.8 — it only includes inheritance relations as hierarchical relations have already been explained in Section 4.3.2. Descriptions of the most

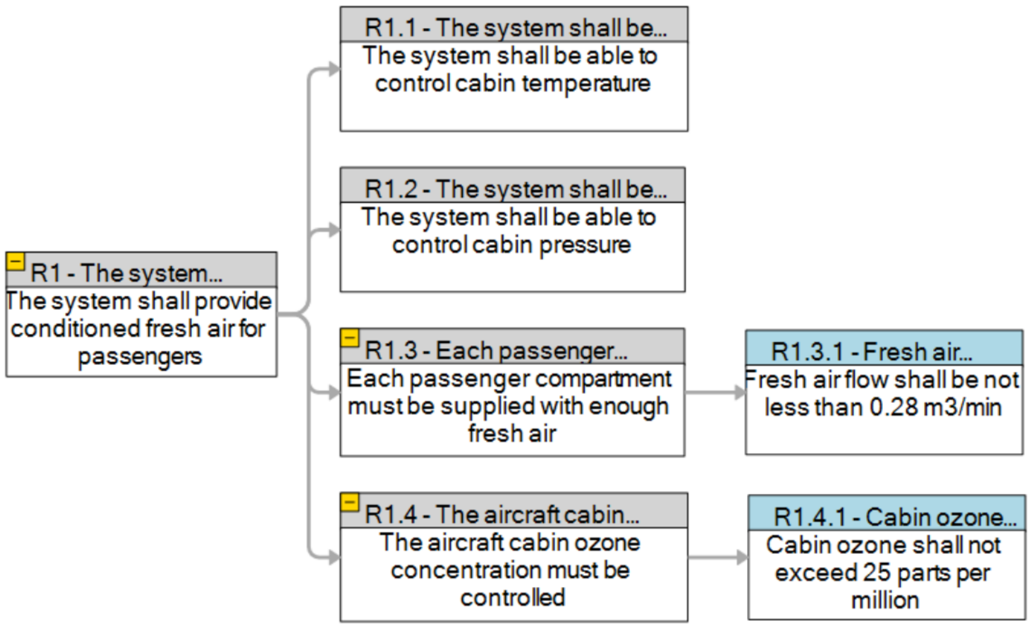

R1 - The system...
The system shall provide conditioned fresh air for passengers
R1.1 - The system shall be...
The system shall be able to control cabin temperature
R1.2 - The system shall be...
The system shall be able to control cabin pressure
R1.3 - Each passenger...
Each passenger compartment must be supplied with enough fresh air
R1.3.1 - Fresh air...
Fresh air flow shall be not less than 0.28 m3/min
R1.4 - The aircraft cabin...
The aircraft cabin ozone concentration must be controlled
R1.4.1 - Cabin ozone...
Cabin ozone shall not exceed 25 parts per million

Figure 4.7: Example of a requirements view (From [138])

relevant attributes are provided in Table 4.10.

Table 4.10: Description of requirement attributes.

Attribute Name	**Description**
Description	Description of the requirement, all details of the requirement can be expressed using natural language. E.g. The system shall provide conditioned fresh air for passengers.
HazardSeverity	Classification of hazard severity according to values provided in CS-25 [65, p. 2-F-49]
IsQualitative	Indicates if the requirement needs to be demonstrated in a qualitative or quantitative (probability of failure) manner

Functional Requirements

Functional requirements are those directly transformed from the stakeholders' needs and that can be satisfied by the creation of a function and subsequent fulfilment of such function by one or more solutions. This leads to the existence of a one-to-many relationship between a functional requirement and the functions

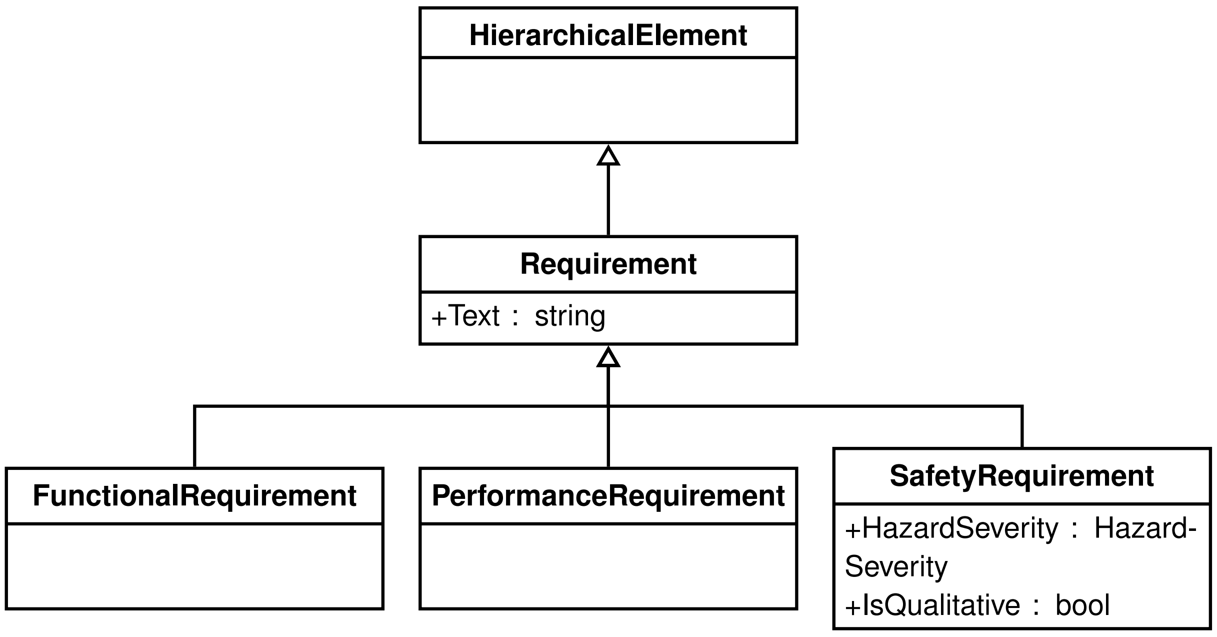


Figure 4.8: Class diagram for requirement classes inheritance hierarchy

derived from it. This relation is presented in more detail in Section 4.5.1.

Performance Requirements

Performance requirements are those that determine that to fulfil stakeholders' needs, the value of a particular parameter from the computational analysis must lay within an acceptable range. This translates into the one-to-many relationship between a performance requirement and its respective parameter. This kind of relation is stated in more detail in Section 4.5.2.

Safety Requirements

Safety requirements originate from the necessity of satisfying stakeholders' needs safely (as required by as required by law or regulations). Apart from the inherited *text* member, safety attributes also contain information about whether they are qualitative and their severity, which in the case of quantitative requirements will determine the target for the probability of failure as specified in CS-25 [65, p.

2-F-50].

4.4.2 Functional View

The second view to be analysed is the Functional View and its main element is the function. Figure 4.9 displays an example of a functional view. Functions are the actions that the system has to perform to meet the stakeholders' needs [14]. Functions are expressed as a 'verb-noun' or similar combination following the functional basis propose by Hirtz et al. [20]. Leaf functions — those that are not further decomposed and situated at the bottom of the hierarchy — are created by instantiation from function templates. The template provides information regarding the operation performed by the function on the flow, and possible constraints on the input and output flows. Functions complement the template information with a brief description of the function within its context. Functions that are higher in the hierarchy are composite functions, which are the result of functional aggregation. Determining the children of a function is generally done via direct functional decomposition or iterative function-solution decomposition.

Figure 4.9: Example of a functional hierarchical view (From [138])

The base class for all functions is the *Function* class. The class diagram for functions is presented in Figure 4.10 — it includes inheritance relations and aggregation relations other than hierarchical and flow relations that have already been explained in Sections 4.3.2 and 4.3.1. Descriptions of the most relevant attributes are provided in Table 4.11. There are two kinds of functions classes:

Functions

Functions are instantiated from templates created according to functional information described in the functional basis terminology [20]. These functions repres-

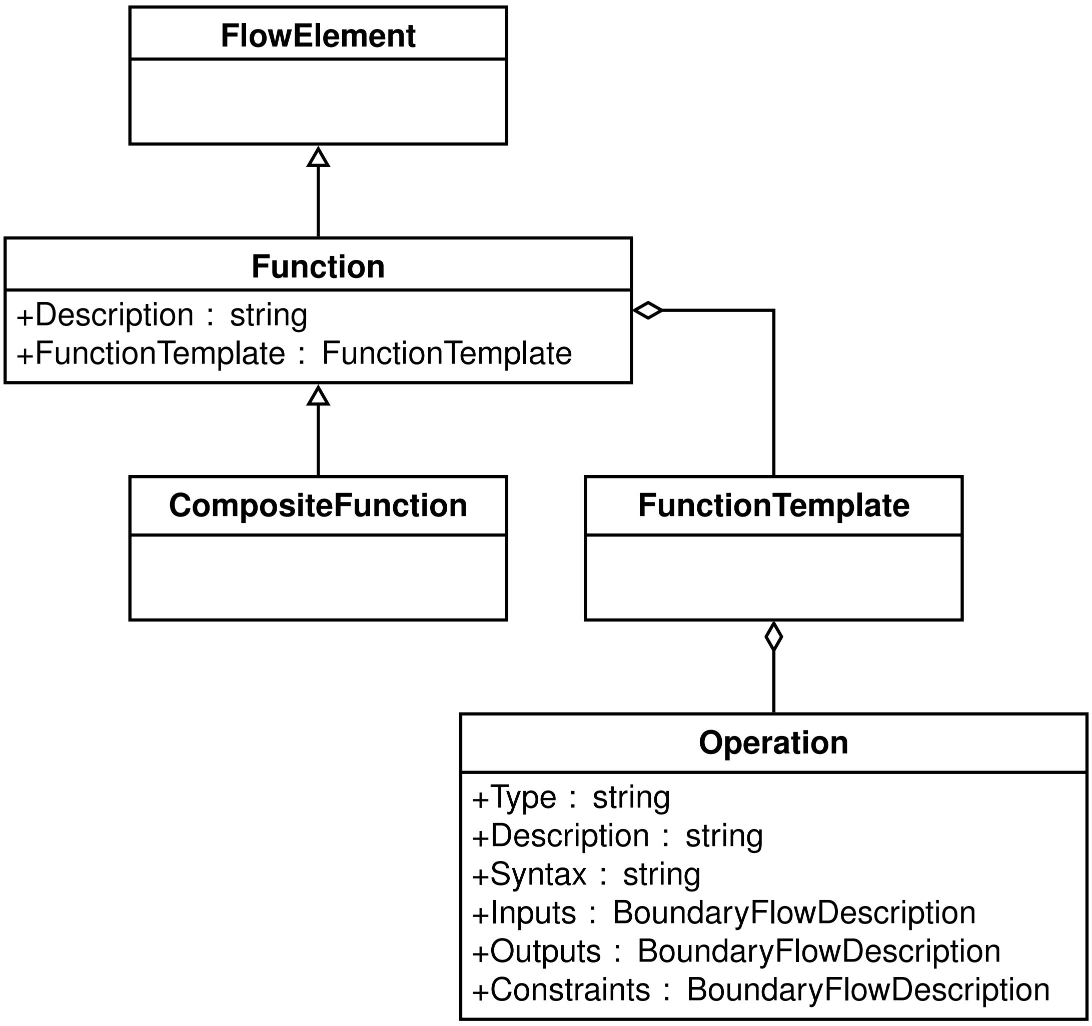

FlowElement
Function
+Description : string
+FunctionTemplate : FunctionTemplate
CompositeFunction
FunctionTemplate
Operation
+Type : string
+Description : string
+Syntax : string
+Inputs : BoundaryFlowDescription
+Outputs : BoundaryFlowDescription
+Constraints : BoundaryFlowDescription

Figure 4.10: Class diagram for Function class

ent the leaf elements of the functional hierarchy. The ports of objects of the class *Function* only possess external connections.

Composite functions

Composite functions are created by composition of other functions (leaf or composite). This kind of functions are represented by objects of the class *CompositeFunction* and their ports have both internal and external connections.

Table 4.11: Description of function attributes.

Attribute Name	Description
Description	Description of the function within its context using natural language. E.g. 'Compress ram air for air conditioning pack'
Template	Object used as a prototype to instantiate *Function* objects. More detailed information is available in Section 4.6.2

4.4.3 Logical View

Since the physical view is out of the scope of this research, the third and last view presented here is the Logical View. The central element of this view is the solution. An example of a logical view is shown in Figure 4.11. Solutions represent the physical elements (such as parts, components or subsystems) that perform (fulfill) the required functions. [14].

Similar to functions, leaf solutions are created by instantiation from solution templates. The template provides the information regarding the input and output flows of the component, computational behavioural models and their parameters or safety characteristics such as the probability of failure. Solutions complement the template information with a brief description of the solution within the architecture and are allowed to override the default values for their parameters to adapt to their context. The rest of the solutions — those that are situated higher in the hier-

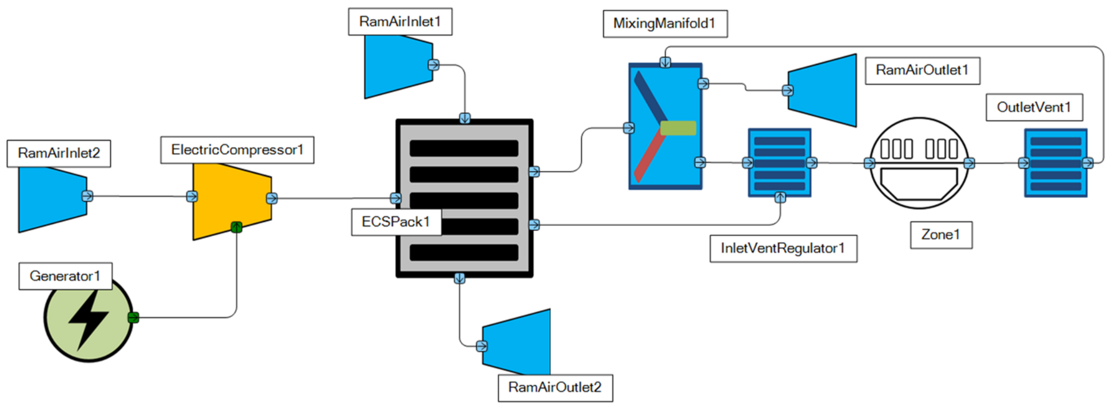


Figure 4.11: Example of a logical flow view (From [138])

archy and generally represent subsystems — are the result of the aggregation of several solutions. A third, novel kind of solution is proposed: the controller solution. Controller solutions represent controllers in the sense of STPA and enable automation of significant parts of the hazard assessment process as explained in Section 5.2.

The base class for all solutions is the *Solution* class. The class diagram for solutions is presented in Figure 4.12 — it only includes inheritance relations and aggregation relations other than hierarchical and flow relations that have already been explained in Sections 4.3.2 and 4.3.1. Descriptions of the most relevant attributes are provided in Table 4.12. There are three kinds of solution classes: solutions, composite solutions and controller solutions.

Solutions

Solutions are instantiated from templates containing behavioural models and their respective parameters. These functions represent the leaf elements of the logical hierarchy. The ports of objects of the class *Solution* only possess external connections.

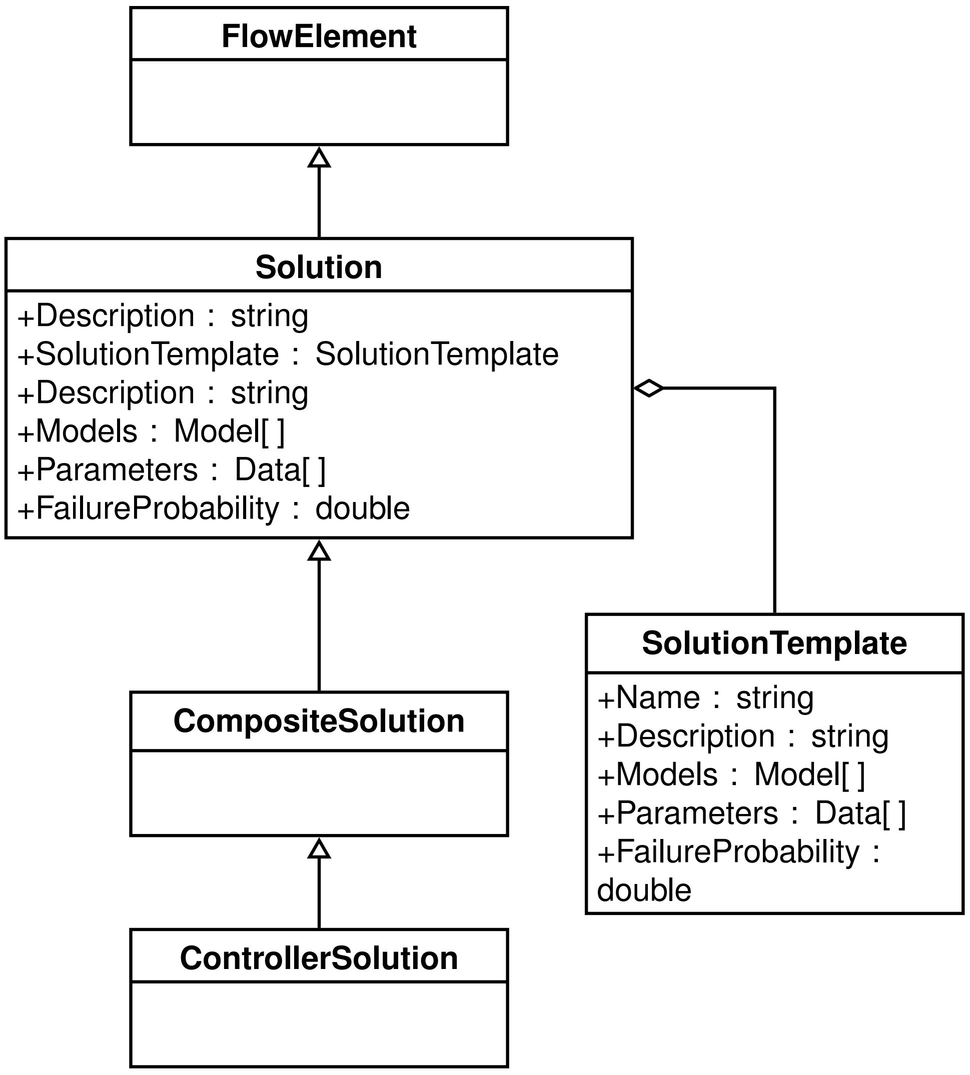


Figure 4.12: Class diagram for Solution class

Composite solutions

Composite functions are created by composition of other solution (leaf or composite). This kind of functions are represented by objects of the class *CompositeSolution* and their ports have both internal and external connections. Composite solutions have models and parameters representing aggregated magnitudes from their children solutions.

Controller solutions

Controller solutions are a special kind of solutions that have the role of a controller according to STPA hazard analysis. The class *ControllerSolution* is responsible to represent this kind of solutions.

Table 4.12: Description of solution attributes.

Attribute Name	**Description**
Description	Description of the solution within its context using natural language. E.g. ‘Primary Heat Exchanger in the Air Condition Pack’
Template	Object used as a prototype to instantiate *Solution* objects. More detailed information is available in Section 4.6.3
Models	Models represent the computational code (mathematical equations) for predicting the solutions’ behaviour or performance characteristics [14]. Each model calculates one or more outputs using the given quantities of one or more inputs — which are given by the parameters of the solution or quantities describing its flows. For example, a model for a resistor might employ Ohm’s law $V = IR$, which uses parameters V, I and R
Parameters	Parameters represent engineering quantities that describe some characteristics of solutions’ computational models. In the previous example, R is the component parameter describing the resistor.
Probability of failure	Probability of the component failing during the relevant timespan used in quantitative fault tree analysis.

4.5 Relations among Views

This chapter so far has focused on the relations amongst elements of the same kind and within a particular view. This section describes the relations between element that belong to different views and the objects required to model such relations.

4.5.1 Requirement-Function Relations

Function Satisfies Requirement

A relation of the kind 'Function satisfies requirement' specifies the mapping from a functional requirement to be satisfied. For example, the function 'Provide fresh air' could be mapped to the requirement 'Each passenger compartment must be supplied with enough fresh air' as it satisfies such requirement. Unlike other inter-view relations, no details beyond the requirement and functional involved are required to describe this relation. As a result, no additional class is proposed to model this kind of relation.

4.5.2 Requirement-Solution Relations

Solution Satisfies Requirement

A relation of the kind 'Solution satisfies requirement' represents the mapping from a performance requirement to a parameter belonging to a solution or any of its ports. To satisfy the requirement, the parameter value must lay within pre-specified bounds. For instance, the parameter 'Air flow' of the ECS could be mapped to the performance requirement 'Fresh air flow shall be not less than $0.28\,\mathrm{m^3/min}$', as that is the parameter that will determine if such requirement is met. This kind of relation is modelled through the *RequirementSolutionRelation* class, whose salient attributes are described in Table 4.13.

Table 4.13: Description of RequirementSolutionRelation attributes.

Attribute Name	**Description**
Requirement	Requirement to be satisfied
Solution	Solution providing the performance value to satisfy the requirement
Parameter	Parameter belonging to the solution or any of its ports, and minimum and maximum acceptable values for the requirement to be satisfied

4.5.3 Function-Solution Relations

The same classes used for function-solution relations in this thesis are used to model both the ‘Function-to-solution’ and ‘Solution-to-function’ relations described by Guenov et al. [14], as these relations are bidirectional.

Solution fulfils function

This relation and its respective class *SolutionFulfillsFunctionRelation* cover the cases when a component or a group of components satisfy a function. For example, the solution ‘Ram air inlet’ can be used to satisfy the function ‘Import air’, which can be reflected by a relation of the kind solution fulfils function. The main class attributes are presented in Table 4.14

Table 4.14: Description of SolutionFulfillsFunctionRelation attributes.

Attribute Name	**Description**
Function	Function to be fulfilled
Solution	Solution that provides the function
PortMappings	Pairs of solution ports and functional ports *(SolutionPort, FunctionalPort)* indicating which port that belongs to the solution corresponds to a particular port of the function, which is described as an operation on a flow that enters and exit through functional ports.

Solution derives function

Solution derives function relations, which are modelled by the class *SolutionDerivesFunctionRelation* represent the cases when using a new component results in a new function becoming necessary to ensure the proper working of the solution. For example, the solution ‘Electric actuator’ might derive the function ‘Provide electric power’, which can be modelled by a relation of the kind solution derives function. The attributes of *SolutionDerivesFunctionRelation* are presented in Table 4.15

Table 4.15: Description of SolutionDerivesFunctionRelation attributes.

Attribute Name	**Description**
Solution	Solution that induces the function
Function	Function induced by the solution
PortMappings	Pairs of solution ports and functional ports *(SolutionPort, FunctionalPort)* indicating which logical flow, needed by the component through one of its ports, correspond to which functional flow in the functional description.

Function to solution relations and the classes used to support them provide the foundation to model the architecting process known as ‘Functional-logical zigzagging’ [14].

4.6 Templates Libraries

As seen in Section 4.4, leaf functions and solutions are created with the help of function and solutions templates. The use of templates helps to maintain consistency in the architecture, as every element of a particular kind is created with the same attributes. The individual functions and solutions can be adapted to the particular context in which they are instantiated by providing values for their additional attributes or overriding template values when appropriate.

Another benefit of templates is that they encapsulate the functional and logical information that is common to every element of the same type, and that is independent of the context of a particular element. This enables the reuse of this encapsulated knowledge, potentially saving time when creating new architectures. Templates are stored in libraries where they can be accessed when needed.

4.6.1 Libraries

Libraries are collections of template or functional basis elements that can be accessed by using a key. Keys consists of one or more segments joined by a dot (**.**). The last segment represents the type of the template element. The rest of the segments are used for classifying the different items (similar to a folder structure in a computer's file system) or differentiating elements of the same type but implemented differently.

For instance, the key *'Aircraft.Pneumatic.Compressor'*, might be used to store a compressor template that has been created to be used in aircraft system architectures and works with pneumatic flows. It is important to understand that the keys used to describe an object do not limit the capabilities of the components, but their characteristics do. The ability of *'Aircraft.Pneumatic.Compressor'* to work with pneumatic flows depends on the type of ports used not on its name. The compressor might also be valid in architectures other than aircraft depending on the computational models and the assumptions these models employ.

There are four libraries employed in this research:

- **Flow Library:** which store instances of every flow defined in the functional basis [20].
- **Operation Library:** which store instances of every function defined in the functional basis [20].

- **Function library:** which contains every function template available to a particular architecting project. The templates can be reused in other projects.
- **Solution library:** which contains every solution template available to a particular architecting project. As with functions, the templates can be reused in other projects.

4.6.2 Function Templates

Apart from the key attribute, common to all library elements, function templates contain information regarding which flow ports required by the function and which kind of changes, transformations or constraints are imposed on the flows passing through those ports. The class representing function templates is the class *FunctionTemplate*. Table 4.16 describes its most relevant attributes.

Table 4.16: Description of FunctionTemplate attributes.

Attribute Name	**Description**
Key	String formed by one or more segments joined by a dot (**.**) that uniquely identifies a template. The last segment represents the type of the element
Description	Description of the function template using natural language. E.g. 'Compress ram air'
InputPorts	Ports through which an energy, material or signal flow enters the template.
OutputPorts	Ports through which an energy, material or signal flow exits the template.
Operation	Object representing changes, transformations or constraints imposed on the template's functional flow. See Section 4.3.2 for more information about operations.

4.6.3 Solution Templates

Solution templates contain the information required to model the behaviour of the physical component that the solution represents. This information comprises input and output solution ports, behavioural computational models and their parameters. Additionally, safety-relevant information such as the probability of failure is provided. The class representing solution templates is the class *SolutionTemplate*, whose salient attributes are presented in Table 4.17.

Table 4.17: Description of SolutionTemplate attributes.

Attribute Name	**Description**
Key	String formed by one or more segments joined by a dot (**.**) that uniquely identifies a template. The last segment represents the type of the element
Description	Description of the solution template using natural language. E.g. ‘Primary Heat Exchanger’
InputPorts	Ports through which an energy, material or signal flow enters the template.
OutputPorts	Ports through which an energy, material or signal flow exits the template.
Models	Models represent the computational code (mathematical equations) for predicting the solutions’ behaviour or performance characteristics [14]. Each model calculates one or more outputs using the given quantities of one or more inputs — which are given by the parameters of the solution or quantities describing its flows. For example, a model for a resistor might employ Ohm’s law $V = IR$, which uses parameters V, I and R
Parameters	Parameters represent engineering quantities that describe some characteristics of solutions’ computational models. In the previous example, R is the component parameter describing the resistor.
Probability of failure	Probability of the component failing during the relevant timespan used in quantitative fault tree analysis.

4.7 Summary and Conclusions

This chapter introduced the changes made to the RFLP framework proposed by Guenov et al. [14]. The changes are necessary to support the novel methods developed in this research. First, the chapter presented the hierarchical and flow relations, which are the most common relations within the RFLP views. These relations are used for guiding the various traversals that are employed by the methods presented in Chapter 5.

Then, the requirements, functional and logical views were presented along with their main elements. The types of requirements, functions and solutions utilised in this research were introduced. The objects for modelling inter-view relations were also discussed. Similarly, this information is also employed by the novel methods presented in this thesis.

The concept of templates applied to functions and solutions along with template libraries were discussed last. Templates are a key mechanism for reusing knowledge and ensure consistency between elements of the same kind. Templates are essential for the safety architecting enablers as these enablers create new solutions.

Chapter 5

Methodology II — Methods and Algorithms

5.1 Introduction

In this chapter, presented is a set of enablers developed to overcome the limitations of current methods stated in Chapter 3 and fulfil the objectives of this thesis (see Section 1.3). The proposed methods use as a starting point the definition of the architecture described according to the RFLP framework that is described in Chapter 4. The presented methods make extensive use of graphs and graph traversals, which are described in more detail in Section 2.4.

The chapter is divided into sections according to the methods. The objectives and corresponding methods are as follows:

- **Objective 1:** accelerate hazard analysis and integrate it with system architecting processes. Methods to automate the creation process of the control structures required for applying the STPA hazard analysis technique are developed. The methods are presented in Section 5.2.

- **Objective 2:** facilitate the interactive introduction of safety principles. Interactive enablers to support the implementation of physical redundancy, functional redundancy and containment are introduced in Section 5.3.

- **Objective 3:** enable a faster assessment of the safety and performance of architectures. Methods are created to automate the creation process of fault trees and are presented in Section 5.4. Also, methods are developed to automate substantial parts of the creation process of computational workflows used for sizing, which are introduced in Section 5.5.

Finally, the conclusions of this chapter are drawn in Section 5.6.

5.2 Methods to Support Hazard Assessment

This section describes two novel methods developed to support architects in the architecture hazard assessment process. The aim of the methods is two provide support for steps 2, 3 and 4 of the STPA hazard assessment process. The reader is referred to Section 3.5.7 for more details about STPA analysis. The first method focuses on using the information in the logical view to automatically elaborate consistent hierarchical control structures, which are used to identify unsafe control actions. The second method concentrates on the creation of more detailed control loops to support the identification of loss scenarios.

5.2.1 Hierarchical Control Structure

The hierarchical control structure is a graph formed by controller nodes and a process node at the bottom of the hierarchy. Each controller solution in the logical view is mapped to a controller node, the rest of the solutions are lumped into the process node. Starting from a controller κ the connections between its associated node and other nodes are determined by applying the following rules:

1. Traverse the logical view of the architecture starting from a port p in controller κ and including every component that can be reached by following flow

links in the direction of the flow. The traversal through a particular link is finished when a controller solution is found immediately after including the controller. This forms the set of components Σ_{I_κ} influenced by κ.

2. The controller solutions in Σ_{I_κ} represent the set of controllers K_{I_κ} influenced by κ.

3. A link is formed in the hierarchical control structure between the nodes representing κ and each controller κ_i in K_{I_κ}, excluding the initial controller κ. The type of the link is determined by comparing their position in the logical hierarchy:

   - If κ is higher than κ_i a link of type ‘ControlAction’ is added.
   - If κ is lower than κ_i a link of type ‘Feedback’ is added.
   - Otherwise, the link is of type ‘SameLevelInputOutput’.

   The initial port in the traversal p is associated with the link.

4. If $\Sigma_{I_\kappa} \setminus K_{I_\kappa} \neq \emptyset$, which represent the case where non-controllers solutions are directly controlled by κ, a link of type ‘ControlAction’ is added between the node representing κ and the process node.

5. Traverse the logical view starting from p including every component that can be reached by following flow links in the opposite direction to the flow. The traversal through a particular link is finished when a controller solution is found immediately after including the controller. This forms the set of components $\Sigma_{I'_\kappa}$ that influence κ.

6. The controller solutions in $\Sigma_{I'_\kappa}$ represent the set controllers $K_{I'_\kappa}$ that influence κ.

7. A link is formed in the hierarchical control structure between the nodes representing κ and each controller κ_i in $K_{I'_\kappa}$. The type of the link is determined

by comparing their position in the logical hierarchy:

- If κ_i is higher than κ a link of type ‘ControlAction’ is added.
- If κ_i is lower than κ a link of type ‘Feedback’ is added.
- Otherwise, the link is of type ‘SameLevelInputOutput’.

The initial port in the traversal p is associated with the link.

8. If $\Sigma_{I'_{\kappa}} \setminus K'_{I_{\kappa}} \neq \emptyset$, then a link of type ‘Feedback’ is added between the node representing κ and the process node.

9. Repeat steps 1–8 for the rest of the ports in κ.

(a) Logical view (b) Control Hierarchy

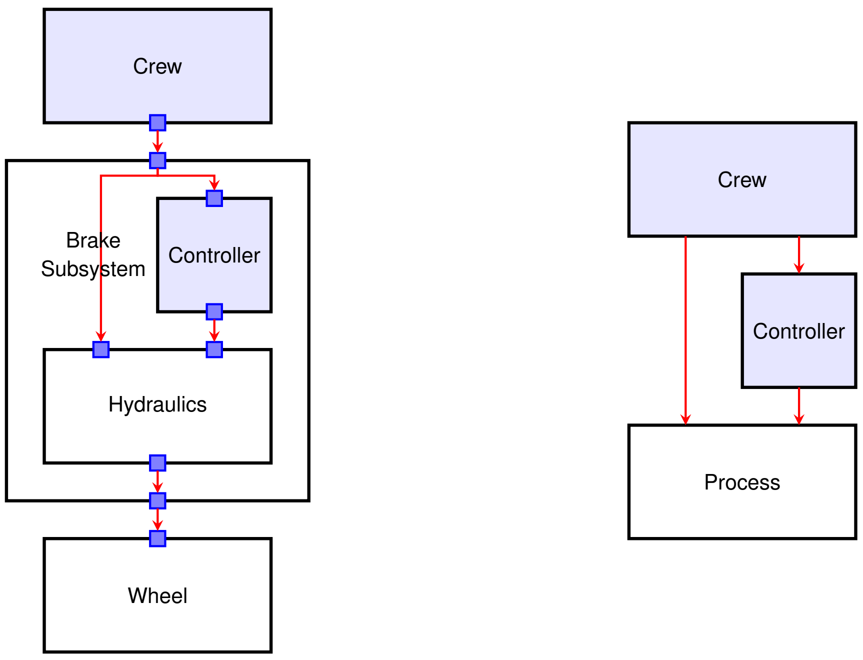


Figure 5.1: Example of the creation of a hierarchical control model

Figure 5.1 shows a simple example of a hierarchical control structure that is automatically created from the logical view of the system. Subfigure 5.1a displays the logical view of the system, which consist of a wheel brake subsystem. The hydraulic system that provides the power to the brake can be controlled either directly by the crew or via a controller. Both the controller and the hydraulics are defined as children of the brake subsystem composite solution. Subfigure 5.1b shows the resulting control hierarchy. The hierarchy is composed of the controllers (solutions highlighted in blue) and a process node where the rest of the solutions are lumped. Since the controller is lower than the crew in the hierarchy of solution, this node is also situated lower in the control hierarchy.

5.2.2 Detailed Control Loops

Creating a detailed control loop requires identifying four sets of components: a controller κ, actuators Σ_{A_κ}, sensors Σ_{S_κ} and process solutions Σ_{P_κ}. The developed method can automatically identify actuators, sensors and process solutions given a controller. The process, which is somewhat similar to the one in Section 5.2.1, is as follows:

1. Traverse the logical view starting from controller κ and including every solution that can be reached by following flow links in the direction of the flow. This forms the set of components Σ_{I_κ} influenced by κ.

2. The set of solutions in Σ_{I_κ} with a direct flow link from κ corresponds to Σ_{A_κ}.

3. Traverse the logical view starting from controller κ and including every solution that can be reached by following flow links in the opposite direction to the flow. This forms the set of components $\Sigma_{I'_\kappa}$ that influence κ.

4. The set of solutions in $\Sigma_{I'_\kappa}$ with a direct flow link to κ corresponds to Σ_{S_κ}.

5. Finally, use the following equation to obtain the set of process solutions

$$\Sigma_{P_\kappa} = (\Sigma_{I_\kappa} \cap \Sigma_{I'_\kappa}) \setminus (\Sigma_{A_\kappa} \cup \Sigma_{S_\kappa} \cup \kappa)$$

The union of the four sets form the set of components in the loop

$$\Sigma_{L_\kappa} = \Sigma_{I_\kappa} \cap \Sigma_{I'_\kappa} = \Sigma_{P_\kappa} \cup \Sigma_{A_\kappa} \cup \Sigma_{S_\kappa} \cup \kappa$$

The user can also provide different values for Σ_{A_κ} and Σ_{S_κ}, which will override the values obtained automatically taking components from Σ_{P_κ} as required. After the sets are identified, the method will classify the links among the various items in the control loop according to the generic control loop provided in the STPA handbook [92]. The criteria employed for this classification is presented in Table 5.1. The method helps to produce the information required by STPA's Step 4 *Identify Loss Scenarios* with little input from the architect (only the controller κ) and ensuring consistency with the architecture definition (solutions and their hierarchical and flow relations are described in the logical view).

Figure 5.2 shows a simple example of how a detailed control loop is created from the logical view of a wheel braking systems. Subfigure 5.2a displays the logical view of the system. The brake hydraulics are controlled by the controller, which is commanded by the crew and receives feedback from a sensor that measures the status of the wheel. Subfigure 5.2b presents the resulting STPA control loop corresponding to the solution labelled as 'Controller'. All solutions are classified according to their role in the control loop. Σ_{I_κ} is composed of the hydraulics, the wheel, the sensor and the controller. Σ_{A_κ} contains the hydraulics as they are the only component in Σ_{I_κ} with a direct link from κ. $\Sigma_{I'_\kappa}$ is composed of the crew, the sensor, the wheel, the hydraulics and the controller. Σ_{S_κ} contains the sensor as it is the only component in $\Sigma_{I'_\kappa}$ with a direct link to κ. Σ_{P_κ} corresponds to the wheel. The crew, which does not belong to the loop, is included in the figure as a direct influence on the controller.

Table 5.1: Comparison of safety principles

Link Type	From	To
Controller Inputs from Sensors	A sensor in the loop Σ_{S_κ}	Controller κ
Controller Inputs from Higher Level Controllers	A controller in $\Sigma_{I'_\kappa} \setminus \Sigma_{L_\kappa}$ with a direct link to κ and higher than κ in the controller hierarchy	Controller κ
Controller Inputs from Same Level Controllers	A controller in $\Sigma_{I'_\kappa} \setminus \Sigma_{L_\kappa}$ with a direct link to κ and at the same levels as κ in the controller hierarchy	Controller κ
Controller Inputs from Other Components	Solutions in $\Sigma_{I'_\kappa} \setminus \Sigma_{L_\kappa}$ with a direct link to κ and that are not controllers	Controller κ
Controller Outputs to Actuators	Controller κ	An actuator in Σ_{A_κ}
Controller Outputs to Other Components	Controller κ	A solution in $\Sigma_{I_\kappa} \setminus \Sigma_{A_\kappa}$ with a direct link to κ
Actuators Inputs from Sensors	A sensor in the loop Σ_{S_κ}	An actuator in Σ_{A_κ}
Actuators Inputs from Process Solutions	Process solutions in the loop Σ_{P_κ}	An actuator in Σ_{A_κ}
Actuators Inputs from Other Process Solutions	A solution in $\Sigma_{I'_\kappa} \setminus \Sigma_{L_\kappa}$ with a direct link to an actuator in Σ_{A_κ}	An actuator in Σ_{A_κ}
Actuators Inputs from Other Controllers	A controller in $\Sigma_{I'_\kappa} \setminus \Sigma_{L_\kappa}$ with a direct link to an actuator in Σ_{A_κ} and that is not a controller	An actuator in Σ_{A_κ}

Link Type	**From**	**To**
Actuators Outputs to Other Process Solutions	An actuator in Σ_{A_κ}	A solution in $\Sigma_{I_\kappa} \setminus \Sigma_{L_\kappa}$ with a direct link to an actuator in Σ_{A_κ} and that is not a controller
Actuators Outputs to Other Controllers	An actuator in Σ_{A_κ}	A controller in $\Sigma_{I_\kappa} \setminus \Sigma_{L_\kappa}$ with a direct link to an actuator in Σ_{A_κ}
Sensors Inputs from Actuators	An actuator in Σ_{A_κ}	A sensor in Σ_{S_κ}
Sensors Inputs from Other Process Solutions	A solution in $\Sigma_{I'_\kappa} \setminus \Sigma_{L_\kappa}$ with a direct link to a sensor in Σ_{S_κ} and that is not a controller	A sensor in Σ_{S_κ}
Sensors Inputs from Other Controllers	A controller in $\Sigma_{I'_\kappa} \setminus \Sigma_{L_\kappa}$ with a direct link to a sensor in Σ_{S_κ}	A sensor in Σ_{S_κ}
Sensors Outputs to Other Process Solutions	A sensor in Σ_{S_κ}	A solution in $\Sigma_{I_\kappa} \setminus \Sigma_{L_\kappa}$ with a direct link to a sensor in Σ_{S_κ} and that is not a controller
Sensors Outputs to Other Controllers	A sensor in Σ_{S_κ}	A controller in $\Sigma_{I_\kappa} \setminus \Sigma_{L_\kappa}$ with a direct link to a sensor in Σ_{S_κ}

Link Type	**From**	**To**
Process Inputs from Actuators	An actuator in Σ_{A_κ}	A solution in Σ_{P_κ}
Process Inputs from Other Process Solutions	A solution in $\Sigma_{I'_\kappa} \setminus \Sigma_{L_\kappa}$ with a direct link to a solution in Σ_{S_κ} and that is not a controller	A solution in Σ_{P_κ}
Process Inputs from Other Controllers	A controller in $\Sigma_{I'_\kappa} \setminus \Sigma_{L_\kappa}$ with a direct link to a solution in Σ_{S_κ}	A solution in Σ_{P_κ}
Process Outputs to Sensors	A solution in Σ_{P_κ}	A sensor in Σ_{S_κ}
Process Outputs to Other Process Solutions	A solution in Σ_{P_κ}	A solution in $\Sigma_{I_\kappa} \setminus \Sigma_{L_\kappa}$ with a direct link to a solution in Σ_{S_κ} and that is not a controller
Process Outputs to Other Controllers	A solution in Σ_{P_κ}	A controller in $\Sigma_{I_\kappa} \setminus \Sigma_{L_\kappa}$ with a direct link to a solution in Σ_{S_κ}

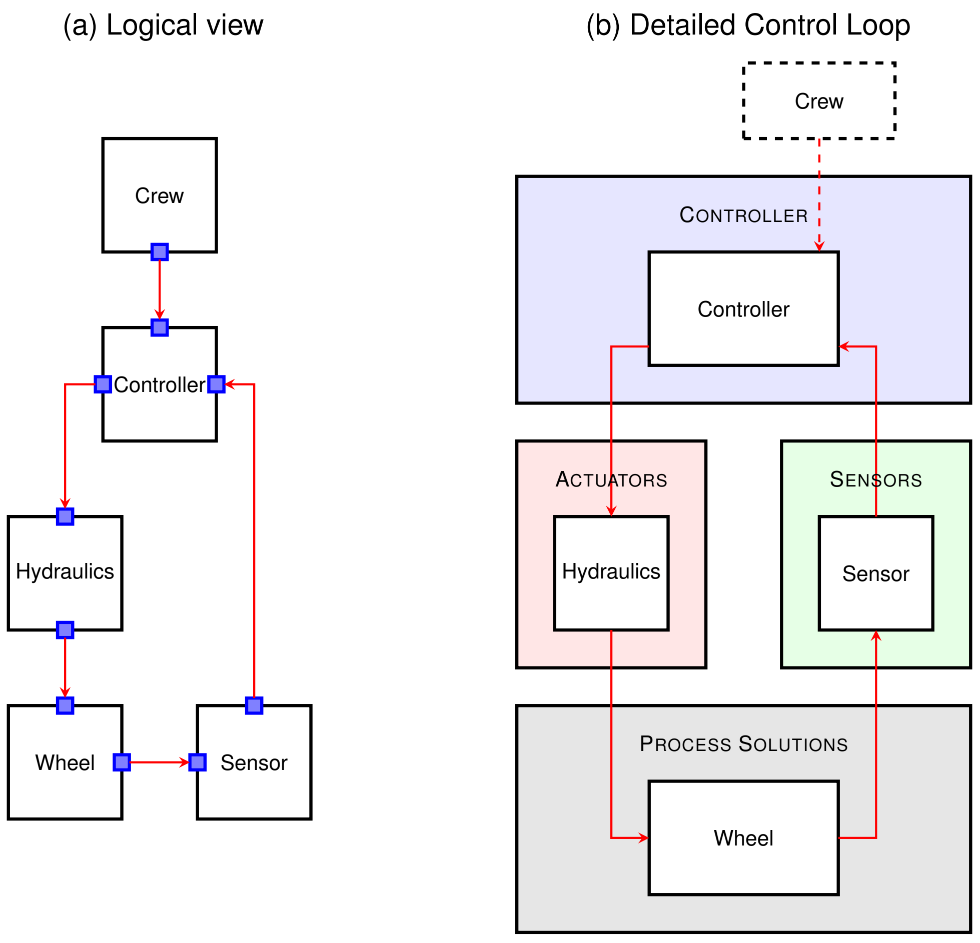


Figure 5.2: Example of the creation of a detailed control loop

5.3 Architecting Enablers for Safety Principles

This section describes three novel enablers developed to help architects to implement safety principles interactively into architectures. The enablers provide support to make architectures safer by introducing physical redundancy, functional redundancy and containment. The support is focused on the creation of new solutions and their connection with other solutions and elements from other views. Other considerations that might be relevant to the implementation of safety principles, such as the impact on mass, cost, volume or the probability of failure, are not considered directly by the enablers. However, these impacts can be determined by using the rest of the enablers presented in this chapter.

5.3.1 Physical Redundancy Enabler

The architecting enabler for physical redundancy helps the architect to increase the redundancy of the architecture by duplicating a part of the architecture one or more times. Figure 5.3 shows a conceptual example of an application of physical redundancy. Subfigure 5.3a show the original logical view and Subfigure 5.3b shows the result of duplicating the blue and orange components once.

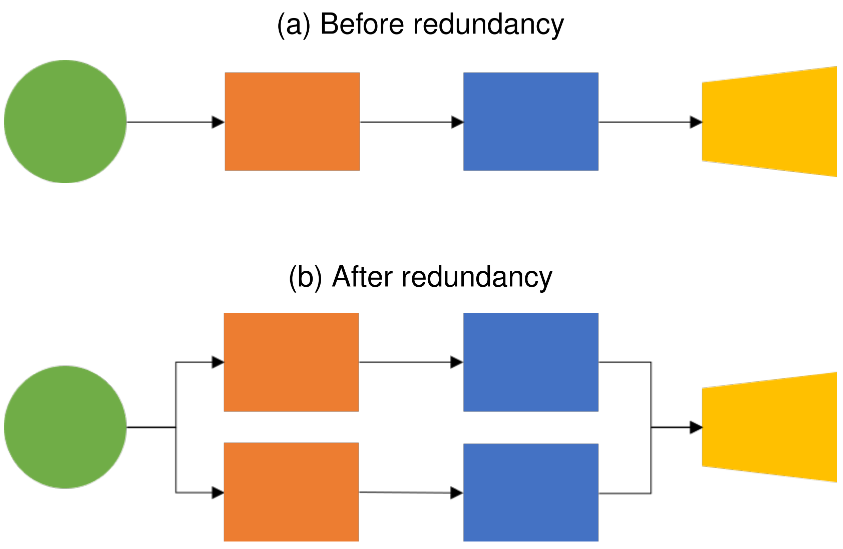


Figure 5.3: Example of physical redundancy

The enabler consists of several parts that are executed according to the logic presented in the flowchart in Figure 5.4. First, the extension of the redundant leg — which components will be replicated — is calculated by traversing the logical view of the architecture according to a set of default traversal rules and the user is facilitated with a preview of the components. Then, the architect can define new rules or override the default ones to increase or decrease the extension of the redundancy. After every update to the rules, the extension is recalculated and the preview is updated. When the architect is finished adding rules, the correctness of the rules is checked. If results are satisfactory, the desired number of redundant legs is created. Otherwise, the user is asked to further update the rules until no problems are detected.

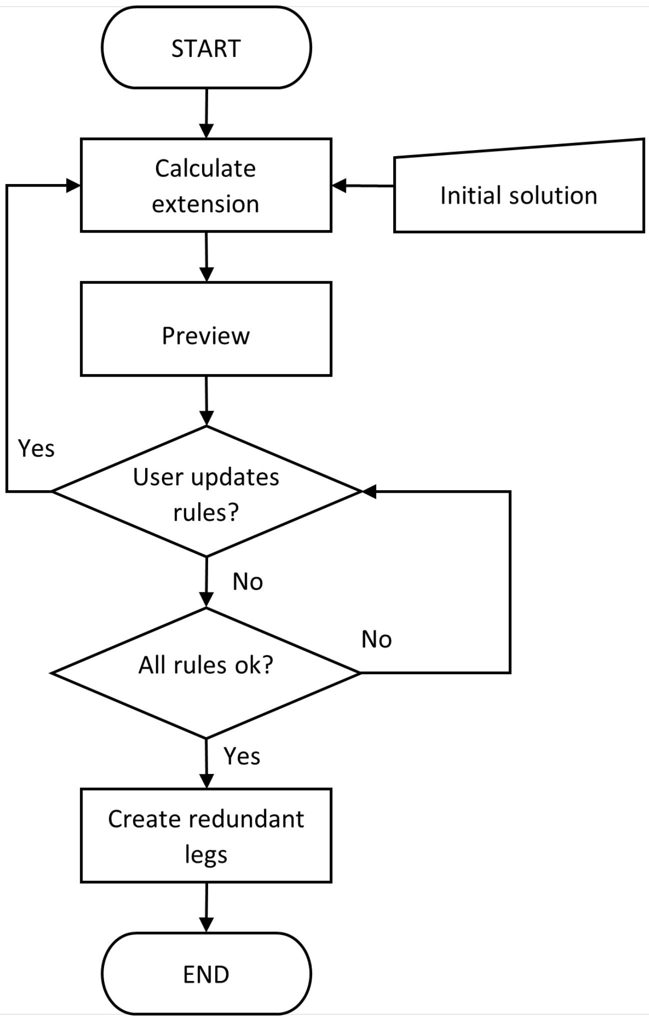


Figure 5.4: Flowchart for physical redundancy architecting enabler

Traversal Rules

Traversal rules help to determine the extension of the redundancy and achieve consistency with existing parts of the architecture. Rules are created both automatically by the enabler and manually by the architect. Rules have options that modify how the algorithms interpret them. Table 5.2 summarise all available traversal rules and their options. The default behaviour is to include components that provide flow to those already in the redundant leg and exclude the rest. If existing redundancy is found, it is up to the architect to decide whether to include it.

Table 5.2: Traversal rules used by the physical redundancy enabler.

Rule Type	**Options**
ExistingRedundancyRule	Include: includes existing redundancy (which becomes even more redundant).
	Merge: merges with existing redundancy (which remains as it is).
	Undefined: no option has been selected, it requires the architect to decide whether to include or merge.
BoundaryOutputRule	Stop: stop traversal through the port.
	Continue: do not stop traversal through the port.
	AutoContinue: same as Continue but automatically determined by the enabler.
UserDefinedStopRule	Stop: stop traversal through the port.
	AutoStop: same as Stop but automatically determined by the enabler.

Compute the Extension of the Redundant Leg

The extension of the redundancy is computed by traversing the logical view of the architecture starting from the initial solution, which is the input to the enabler. At each step, the traversal focuses on a component σ and determines the set of logical components to be analysed in subsequent steps. The set includes every component connected to σ through a valid link. Links from other components to σ are valid only if the ports at both ends of the link meet the following conditions. The *FromPort* in the link is valid if either:

1. Do not have a rule associated with it.
2. The rule is of type 'ExistingRedundancyRule' and the architect has selected the option to include the existing redundancy.
3. Or the rule is of type 'BoundaryOutputRule'.

The *ToPort* in the link is valid if either:

1. Do not have a rule associated with it.
2. The rule is of type 'ExistingRedundancyRule' and the architect has selected the option to include the existing redundancy or the option is undefined.
3. Or the rule is of type 'BoundaryOutputRule'.

Links to other components from σ are valid only if ports at both ends of the link:

1. Do not have a rule associated with them.
2. The rule is of type 'ExistingRedundancyRule' and the architect has selected the option to include the existing redundancy.
3. The rule is of type 'BoundaryOutputRule' and the architect has selected the option to continue through such port.

Default rules for including or excluding components are created by visiting in preorder — before the next elements of the traversal are determined — the logical components. Rules are created by use of the logic below:

1. For all ports, a rule of type ‘ExistingRedundancyRule’ with option ‘Undefined’ is created if the port’s connection represents existing redundancy.
2. For output ports, where the rule above is not applicable, a ‘BoundaryOutpu-tRule’ with option ‘Stop’ is created.

A second traversal, following the same rules as the first one is done to identify the links whose both ends belong to components included in the redundant leg during the first traversal. If the starting end of the link has a ‘UserDefinedSto-pRule’ with option ‘Stop’ — created during the first traversal — the option is set to ‘AutoContinue’ as such port does not belong to the boundary between included components and the rest of the architecture.

Update Rules for Computing the Extension

The user is allowed to modify existing rules: decide whether to include existing redundancy and to continue through outputs on the boundary. Additionally, architects can create (and remove) rules of type ‘UserDefinedStopRule’ to exclude (or stop excluding) components added to the redundant leg by default. ‘UserDefined-StopRule’ can be applied to solutions, ports and connections. When applied to a solution or connection, an additional ‘UserDefinedStopRule’ with ‘Stop’ option is added to each of their ports.

Create Redundant Legs

To create redundant legs, the architect is asked to provide the desired number of new legs to be created and the creation procedure is repeated once for every leg. First, for all solution σ in the leg:

1. A new solution σ' of the same kind of σ is created.
2. The hierarchical relations of σ are applied to σ'
3. The functional-logical relations of σ are applied to σ'

Then, for every solution σ in the leg:

1. The flow relations between σ and other solutions included in the redundant leg are applied to σ' and the respective solutions in the new leg.
2. The flow relations between σ and other solutions not included in the redundant leg (crossing the boundary) are applied to σ' connecting it to the pre-existing parts of the architecture.

5.3.2 Functional Redundancy Enabler

The architecting enabler for functional redundancy helps the architect to increase the redundancy of the architecture by adding one or more components that can perform one of the already fulfilled functions in the architecture in a dissimilar manner. Figure 5.5 shows a conceptual example of an application of functional redundancy. Subfigure 5.5a show the original logical view and Subfigure 5.5b shows the result of fulfilling the function φ_P, provided by the original blue component, by adding a different kind of component. The turquoise component although different from the original one is also able to provide φ_P. In turn, this component induces function φ_I, which is fulfilled by the orange component. Similarly, this component derives function φ_I', which is provided by the green component, which does not require any additional function to be fulfilled.

It consists of a series of processes that are executed repeatedly until all functions — the original function plus any induced by the new components — are fulfilled as the flowchart in Figure 5.6 illustrates.

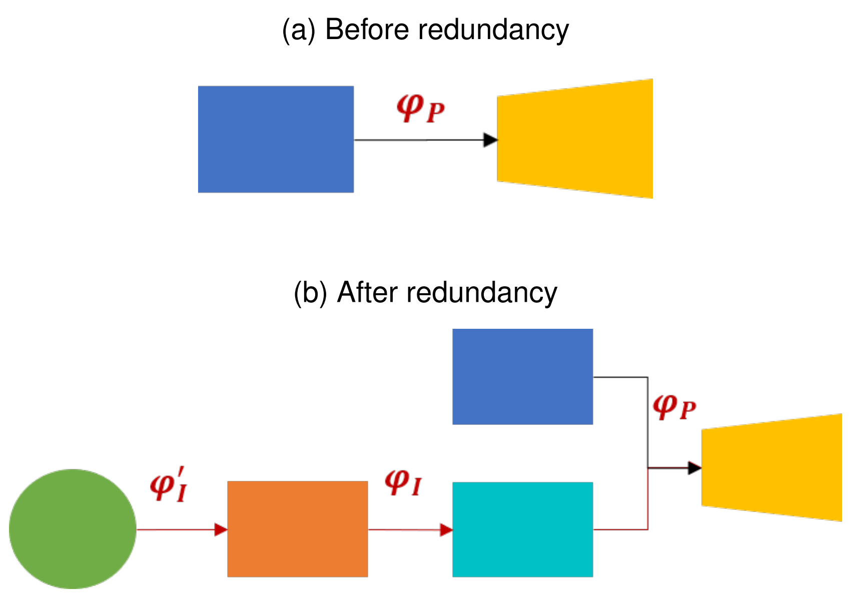


Figure 5.5: Example of physical redundancy

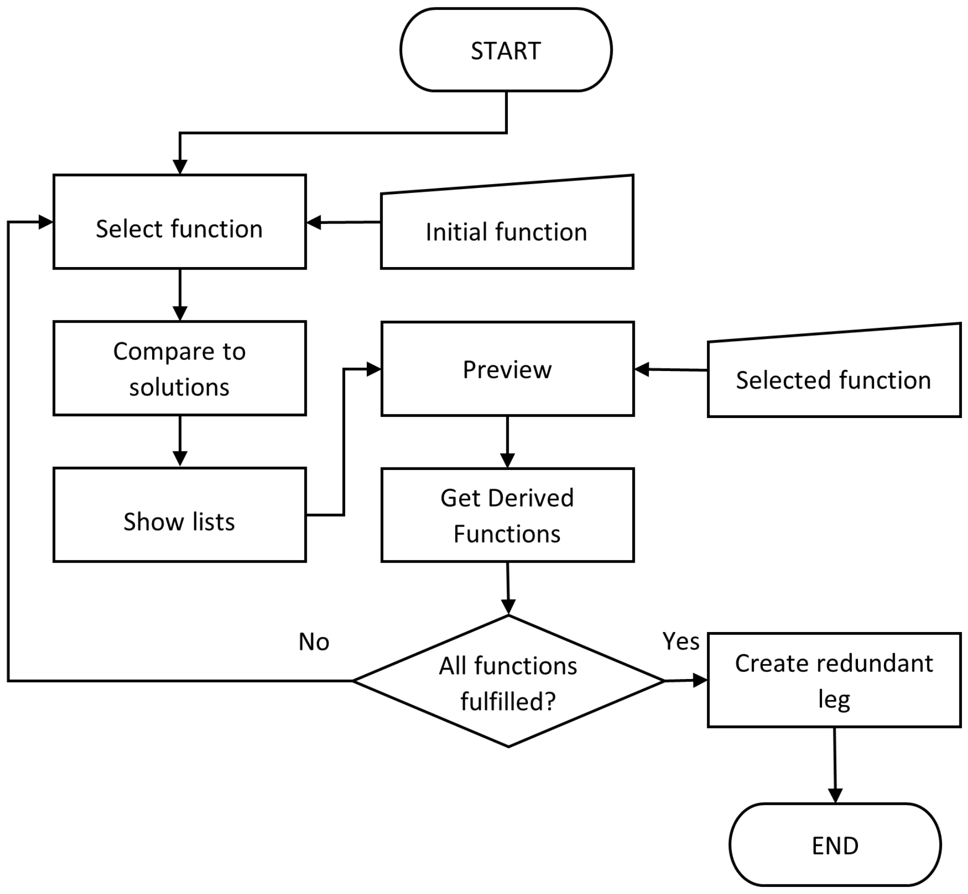


Figure 5.6: Flowchart for functional redundancy architecting enabler

First, a function is selected and then, solutions from the architecture and redundant leg, and solution templates from the library are compared and sorted according to their ability to fulfil the function. The results are displayed as lists to the users so they can decide which solution or template to use. Upon selection of the desired element, if not already present, the solution is added to the preview of the redundant leg. When a solution from the library is added to the leg, any derived function is added to the set of functions to be fulfilled. When all functions in the set are fulfilled, the new redundant leg is added to the architecture.

Determine Function-Solution Similarity

The similarity between a function ϕ to be fulfilled and a solution σ or solution template σ_τ is determined by comparing the functional definition of ϕ with those of the functions that are fulfilled by σ or that can be potentially fulfilled by instantiating σ_τ — according to the knowledge from the solution library. The comparison is done in terms of operation $\mathscr{O}$ and flow $\mathscr{F}$ (see Section 2.3.1 and Section 4.3.2 for more details about operations and flows).

The operation $\mathscr{O}_\phi$ obtained from function ϕ is compared to another operation $\mathscr{O}_\sigma$ obtained from a solution σ or solution template σ_τ. The comparison yields one out of three possible results: both operations are the same, they are related in the operation hierarchy (ancestor-descendant relation), or they are unrelated. The flow $\mathscr{F}_\phi$ is compared to another flow $\mathscr{F}_\sigma$, and one out of three results is obtained: both flows are the same, they are related in the flow hierarchy (ancestor-descendant relation), or they are unrelated. The results are combined to provide the overall metric of Function-SolutionsSimilarity as described in Table 5.3. Based on these results, three ordered lists are created: one with solutions already added to the leg, one with solutions already existing in the architecture and the other with templates from the solution library. The list groups the solutions by their comparison result and the groups are ordered as follows:

1. SameOpSameFlow: solutions with the higher degree of similarity are displayed first.

2. SameOpRelatedFlow: presents a high degree of similarity and, despite flows not being exactly the same, the solution is likely to be able to fulfil the function.

3. RelatedOpSameFlow: presents a lower degree of similarity and, whilst operations are related, the solution might still be able to fulfil the function.

4. RelatedOpRelatedFlow: the solution presents an even lower degree of similarity and therefore a lower probability of being able to fulfil the function.

5. NoOpSameFlow: although the same flow is used, the degree of similarity is low and the solution is unlikely to be able to fulfil the function.

6. NoOpRelatedFlow: similar to the case before but with a lower similarity between flows and a lower probability of function fulfilment.

7. Unrelated: the solution cannot provide the function and its excluded from the list.

The lower levels of the list are unlikely to be able to produce results, but they might be able to inspire the architect to find new solutions to include in the library or develop new solutions from the existing solutions that already present some similarity.

Operation comparison is done in terms of the operation types and their relative position in the operations hierarchy. Operation is an element-level attribute (there is only only one per element). Flow comparison is done in term of ports, ports' flow types and their relative position in the operations hierarchy. Flow is a port-level attribute (there is only one per port but many per element), which makes the comparison of flows a more complex process. To evaluate whether a solution can

Table 5.3: Table Function-Solution Similarity

$\mathscr{F}_\phi$ **and** $\mathscr{F}_\sigma \rightarrow$ $\mathscr{O}_\phi$ **and** $\mathscr{O}_\sigma \downarrow$	**Unrelated**	**Related**	**Equal**
Unrelated	*Unrelated*	*NoOpSameFlow*	*NoOpRelatedFlow*
Related	*Unrelated*	*RelatedOpRelatedFlow*	*RelatedOpSameFlow*
Equal	*Unrelated*	*SameOpRelatedFlow*	*SameOpSameFlow*

fulfil a function in terms of flow compatibility, all ports in the function definition must be matched to a port in the solution. Ports can only be matched to ports with equal (or related) flow and no port can be matched twice. This represents a maximum matching problem that can be solved by using the Hopcroft-Karp algorithm [149]. Bipartite graphs are created by adding to one set all function ports and to the other set all logical ports. Edges from ports from one set to ports from the other set exist only if their flows are equal (or related). If a maximum matching is found then $\mathscr{F}_\phi$ and $\mathscr{F}_\sigma$ are said to be *Equal* in the case where edges represent equal flows, *Related* when edges represent related flows and *Unrelated* otherwise. The maximum matching is stored to be reused when the redundant branch is created during the last process of the enabler.

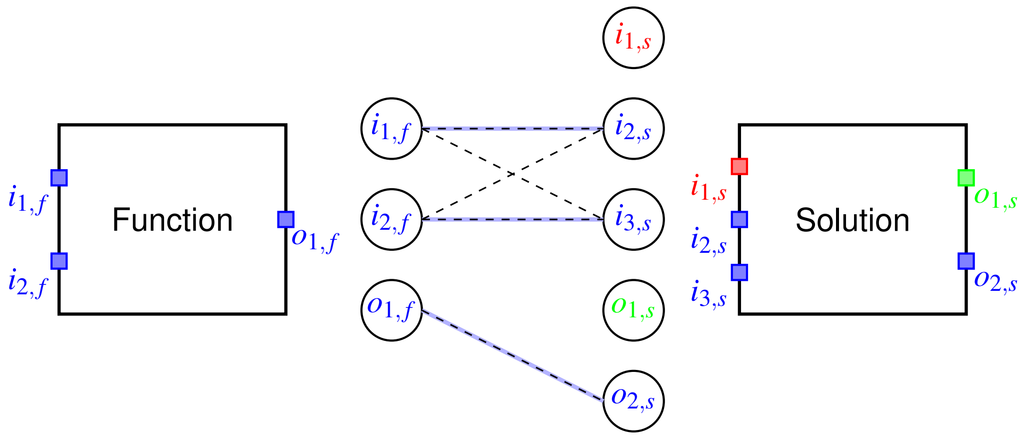


Figure 5.7: Example of a graph for flow matching

Figure 5.7 illustrate how the flow comparison is converted to a maximum matching problem. The function has two inputs $i_{1,f}$ and $i_{2,f}$ and one output $o_{1,f}$ of the same type (blue). The solution has an input $i_{1,s}$ of a second type (red) and an output $o_{1,s}$ of a third type (green). The rest of the ports in the solution $i_{2,s}$, $i_{3,s}$ and $o_{2,s}$ are of the same type as the functional ports. The bipartite graph contains the ports from the function in one set and the ports from the solution in the other set. Links are created only between ports of the same type, or related when an exact match cannot be found. A possible match is represented in the figure by highlighting in blue the links that connect the matched ports.

Connect Components using Functional-Logical Relations

When a solution σ_F is added to the leg — with the purpose of fulfilling function ϕ — it is necessary to connect σ_F to those solutions already present in the redundant leg. This is done by combining the maximum matching that relates ϕ and σ_F with the *SolutionDerivesFunctionRelation* object that relates the ϕ to the solution in the leg σ_D that derived function ϕ. This three-element functional-logical zigzagging ($\sigma_D \Rightarrow \phi \mapsto \sigma_F$) is used to map (and therefore connect) ports in σ_D to ports in σ_F.

Obtention of Derived Functions

When a solution is selected from the library, the functions it derives need to be taken into account to complete the redundant leg preview first, and then to create it in the architecture. The enabler then collects all functions from the derived function mappings related to the template in the solution library. If any of the input ports of the new solution is not covered by any of the mappings, a new mapping is added to the collection. The new mapping relates the input port of the solution to the output port of a new function of type ‘Import ⟨flow-type⟩’. For example, if an input port with flow of type ‘air’ is not covered, then a mapping between that port and the output port of function ‘Import air’ would be created.

Create Redundant Legs

After every function is fulfilled, the enabler then creates the new redundant leg. First, all solutions in the preview leg that are not already present in the architecture are created and added to the logical view. A hierarchical link is created between these solutions and the parent component σ_I of the component that performs the initial function provided to the enabler. Then, functions that were derived while using the enabler are created and added to the functional view. Similarly, hierarchical links are created between these functions and the parent of the first function. Later, functions and solution are linked according to the zigzagging relations ($\sigma_D \Rightarrow \phi \mapsto \sigma_F$) as explained earlier in this section. Finally, logical flow links are created between solutions in the leg according to the zigzag, and between the leg and σ_I.

5.3.3 Containment Enabler

The architecting enabler for containment helps the architect to study the propagation of disturbances, which originate from one or more solutions (disturbance sources), and potentially affect one or more susceptible solutions (disturbance sinks). Figure 5.8 shows a conceptual example of an application of containment. Subfigure 5.8a show the original logical view and Subfigure 5.8b shows the result of adding two barriers (two red lines crossing a link) to protect the component that needs protection (highlighted in blue) from the disturbances originated from the component highlighted in red.

The enabler can create and analyze the effects of including barriers — any solution that can stop the propagation of a disturbance. Examples of barriers are valves for hydraulic and pneumatic flows and circuit breakers for electrical flows. The best location for barriers can be determined manually or automatically by using a min-cut algorithm. After the location of barriers is decided and the resulting

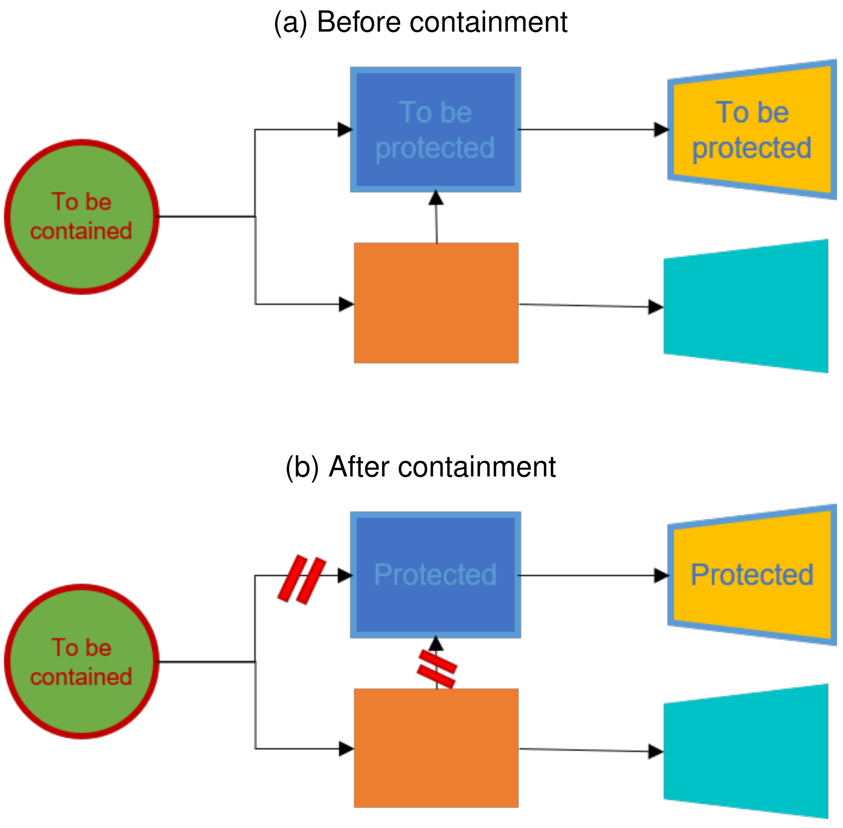


Figure 5.8: Example of containment

containment is deemed satisfactory, the enabler instantiates the barriers in the architecture and connects them to other components. This process is presented in the flowchart in Figure 5.9.

Propagation Algorithm

The propagation algorithm traverses the graph via the flow links defined in the logical views. The propagation can be done flow-wise to obtain affected components from a particular source, or counter flow-wise to obtain components with the potential to affect a particular flow. Modelling of propagation within components (from input to output ports) can be done with varying level of detail:

- Every input port propagates to every output port: this is the simplest way and does not require additional information, although the results might be conservative as some input-output combination might not propagate dis-

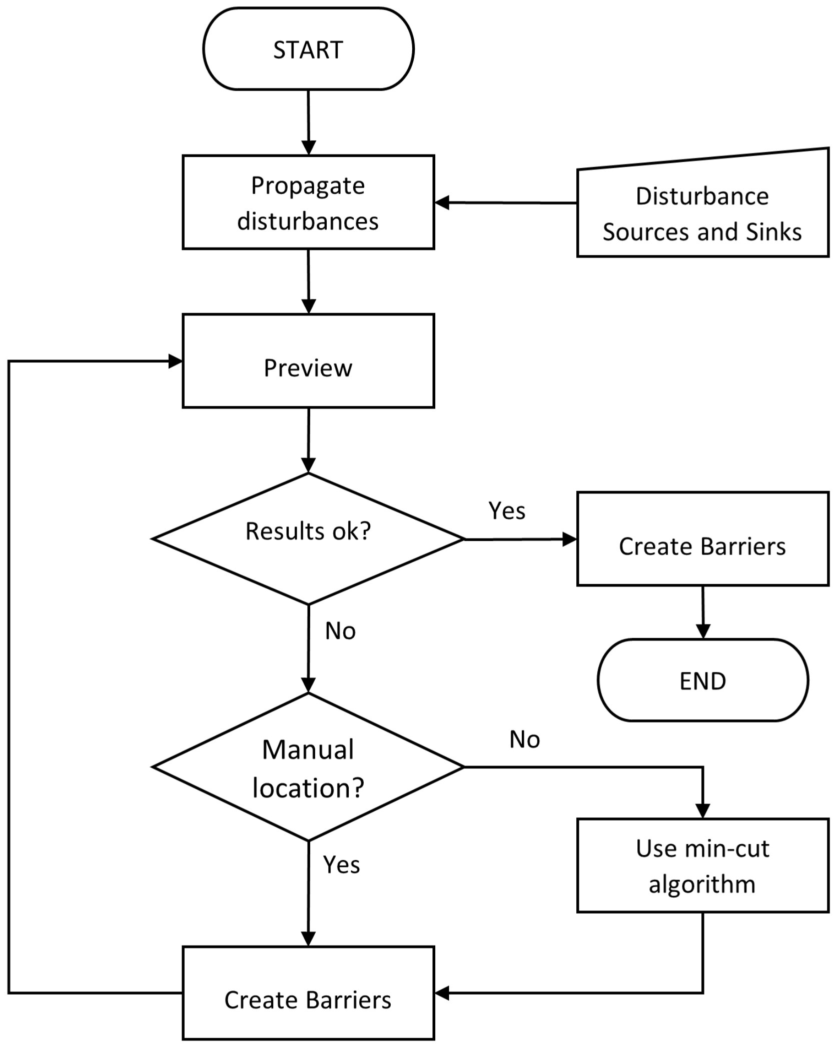


Figure 5.9: Flowchart for physical redundancy architecting enabler

turbances.

- Every input propagates only to a subset of or to all output ports: this method requires such information to be defined (e.g. in the solution template). It can provide more accurate result but requires more effort. Nonetheless, propagation information is reusable ant the extra effort is likely to be amortized when the same template is used in several architectures.

Both levels of detail can coexist in a single traversal. Barriers are components that stop propagation, which is modelled by disabling propagation from some input to some output ports of the component. Therefore any link with an end in any of the ports with disabled propagation keeps the traversal from continuing through that link.

Automatic placement of Barriers. Minimum Cut Algorithm

The automatic placement of barriers is enabled by the automated application of a minimum cut set algorithm (see Section 2.4.3) to a graph that models the disturbance of propagation according to the rules provided in the section above. The first step consists in building such a graph. Then, the algorithm is run in the second step. During a third and final step, the results from the algorithm are analysed to determine the best location of the barriers.

The propagation flow graph is formed by vertices, which represent logical ports, and connections, and edges, which represent the links that can propagate disturbances — both between ports and internal to the components. The set of solutions whose ports will be added to the graph is formed by the disturbance sources, the susceptible sinks, and any other solution that can be affected by a disturbance that originates in any of the sources and can also affect at least one sink.

Edges between those vertices that represent the ports that belong to a partic-

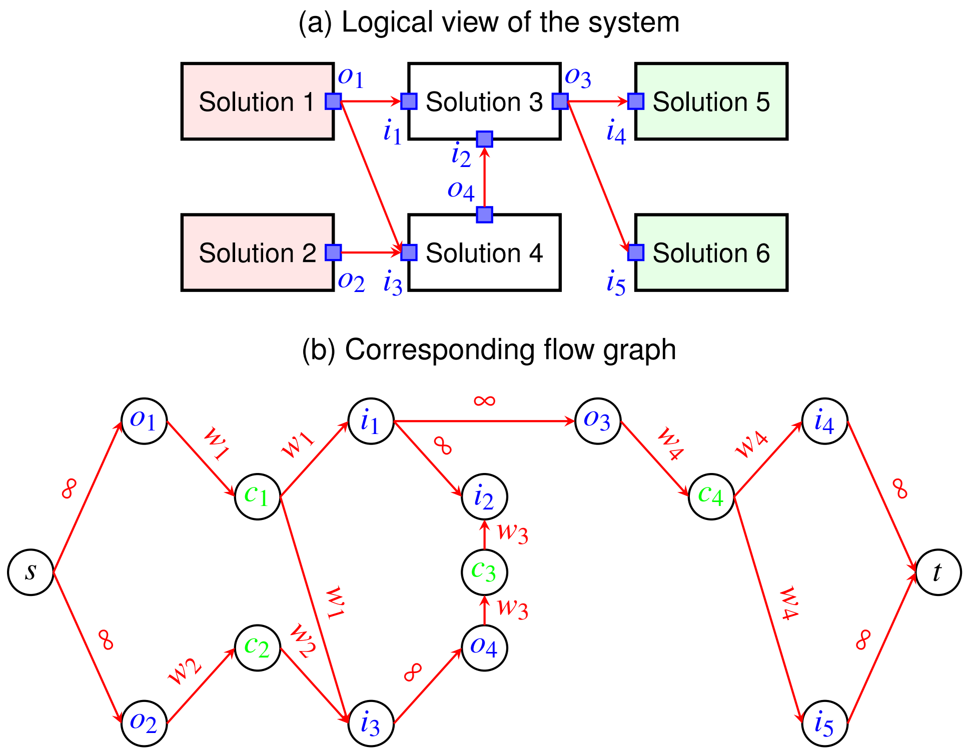


Figure 5.10: Example of the creation of the propagation flow graph

ular component are created according to the rules provided regarding the level of detail in the previous section. When a link exists in the logical view between ports that belong to two different components — excluding barrier ports, an edge is created in the graph between the vertices that correspond to the logical ports. Since the min-cut algorithm requires the graph to have a single source and a single sink, a dummy source vertex is created and connected to all vertices that represent input ports in source components. Similarly, a dummy sink vertex is created and connected to vertices that represent output ports in sink components.

Figure 5.10 illustrates the creation of the flow graph required for solving the minimum cut problem. The graph is automatically created from the logical view of the architecture. Disturbance sources are highlighted in red, and components that are susceptible to disturbances are highlighted in green. The vertices whose text is blue correspond to the ports in the logical view. The vertices whose text is green represent the connections between components. The vertices s and t are the dummy source and dummy sink vertices respectively. The edges between blue and green vertices are created according to the connection in the logical view. The edges between blue vertices represent the links internal to the components that can propagate disturbances. The rest of the edges are located between the dummy source and the output ports of disturbance sources, or the input ports of susceptible components and the dummy sink. Edges between blue vertices or involving any of the dummy vertices are given an infinite weight to avoid that the algorithm includes them in the minimal cut. The rest of the edges are given a weight w_i whose default value is one and that can be customised to express preference on the placement of the barriers.

The algorithm will find the set of edges whose removal causes the sink to become disconnected from the source and whose value is the minimum possible. The value is calculated by summing the values given to any edge in the set. The solution is not necessarily unique. To avoid spurious results, an infinite value is automatically given to edges that do not represent a valid flow link where a barrier can be placed. In particular, this includes links that are internal to the components and links that involve dummy vertices. The rest of the edges are given by default a value of one. This value can be overridden to simulate more expensive barrier additions.

After running the algorithm the results are mapped back from the propagation graph to the flow view and barriers are added to the preview in the locations computed by the algorithm.

5.4 Fault Tree Analysis Support

This section describes a novel method developed to support the architect while performing fault tree analyses (see Section 2.5 for more details). The method enables the automatic creation of fault trees. The support is composed of two parts: a novel algorithm to create fault trees from the architecture definition, and a way to analyse fault trees to obtain quantitative and qualitative results.

5.4.1 Fault Tree Creation Algorithm

The algorithm for creating the fault tree from the logical view presented here is based on work already published by the author [138]. The version presented here includes more detail about how subtrees are reused to avoid visiting components more than once. Additionally, the notation has been updated to be consistent with the object model presented in Chapter 4

The algorithm is based on two mutually recursive functions — the first function calls the second function, which in turn calls the first one until either of them reaches the base case when no call is made to the other function. The starting point of the algorithm is the function CREATE-FAULT-TREE (Algorithm 1), which is used to initialize the tree and the auxiliary data structures, namely the set of parent components in the recursion tree and the set of visited components. After initialisation, the function CREATE-TREE-RECURSIVE-COMPONENT is called with the top event provided to the algorithm (an output of the component whose fault is being analysed).

Function CREATE-TREE-RECURSIVE-COMPONENT (Algorithm 2) the first of the two mutually recursive functions, creates the logical gate that models the failure of a component (parent of the function's input argument 'output-port'). The gate created is an OR gate whose inputs are the failure of the component itself and the failures of each of the components inputs. Input failures are modelled by calling CREATE-TREE-RECURSIVE-COMPONENT with each port as an input parameter. Lines 2 to 4 of the function check whether the port has already been visited, and returns the previously created subtree if that is the case. Line 5 obtains the parent component of the port, and line 6 adds the component to the set of parents. Lines 7 to 11 get the gate inputs, which correspond to the solution's input ports. Line

Algorithm 1: Fault Tree creation

```
1 Function CREATE-FAULT-TREE(top-event)
      Inputs : An output logical port representing the top event of the tree
               (top-event).
      Output: The fault tree corresponding to top-event
2     tree ← $\langle\emptyset,\emptyset\rangle$ // A fault tree is described by the tuple
       $T = \langle E, G\rangle$, where $E$ is the set of events (basic and
       intermediate) and $G$ is the set of gates
3     parents ← $\emptyset$ // Set of parent nodes in the tree, used to avoid
       loops in the tree
4     visited ← $\emptyset$ // Set of nodes visited so far by the algorithm,
       used to avoid visiting a node more than once
5     return CREATE-TREE-RECURSIVE-COMPONENT(top-event, tree, parents,
       visited)
6 end
```

12 removes the component from the set of parents, and line 13 adds the port to the set of visited elements so lines 2 to 5 will work as intended in future calls to the functions. Finally, line 15 calls ADD-GATE and returns the fault (sub) tree.

Function CREATE-TREE-RECURSIVE-CONNECTION (Algorithm 3) the second of the two mutually recursive functions, creates the logical gate that models the failure of a connection (the one connected to the function's input argument 'input-port'). The type of gate created depends on the kind of redundancy that the connection represents. If the connection represents redundant components, the gate will be an *AND* gate, otherwise an *OR* gate. More complex gates (such as *k/N* for voting systems where k correct inputs out of N are needed or *SPARE* gates for modelling dynamic redundancy) could be considered if the necessary information was included in the connections. Connection's input failures are modelled by calling CREATE-TREE-RECURSIVE-SOLUTION with each port as an input parameter. As in the first function, lines 2 to 4 of the function check whether the port has already been visited, and returns the previously created subtree if that is the case. Line 5 gets the external connection of the port, and line 6 gets the type of gate as described above. Lines 7 to 11 get the gate inputs that correspond

Algorithm 2: Fault Tree creation recursive, component part

```
1  Function CREATE-TREE-RECURSIVE-COMPONENT(output-port, tree, parents, visited)
       Inputs : An output logical port representing the top event of the
                sub-tree being created (output-port).
                The fault tree being created (tree).
                The set of parents in the tree (parents).
                The set of nodes visited by the algorithm so far (visited).
       Output: The fault tree corresponding to output-port
2      if output-port $\in$ visited then
3          return GET-GATE(output-port);
4      end
5      component $\leftarrow$ output-port.parent
6      parents $\leftarrow$ parents $\cup$ component // The component will be a
        parent in the subsequent recursive calls
7      gate-inputs $\leftarrow \emptyset$
8      for each input $\in$ component.input-ports
9          gate-input $\leftarrow$ CREATE-TREE-RECURSIVE-CONNECTION(input, tree,
            parents, visited)
10         gate-inputs $\leftarrow$ gate-inputs $\cup$ gate-input
11     end
12     parents $\leftarrow$ parents $\setminus$ component // Recursive calls are finished,
        so the component is not a parent anymore
13     visited $\leftarrow$ visited $\cup$ output-port // Mark as visited so it is not
        visited again
14     return ADD-GATE(tree, output-port, component $\cup$ gate-inputs, OR))
        // OR gate representing the failure of the component or any
        of its inputs
15 end
```

to the connection's input ports. Only input ports whose parent does not belong to the set of visited parents are considered in the loop. Also similar to the first function line 13 adds the port to the set of visited elements so lines 2 to 5 can work as intended. Finally, line 15 calls ADD-GATE and returns the fault (sub) tree.

Algorithm 3: Fault Tree creation recursive, connection part

```
1  Function CREATE-TREE-RECURSIVE-CONNECTION(input-port, tree, parents,
     visited)
       Inputs : An input logical port representing the top event of the
                sub-tree being created (input-port).
                The fault tree being created (tree).
                The set of parents in the tree (parents).
                The set of nodes visited by the algorithm so far (visited).
       Output: The fault tree corresponding to input-port
2      if input-port $\in$ visited then
3          return GET-GATE(input-port);
4      end
5      connection $\leftarrow$ input-port.connection
6      gate-type $\leftarrow$ GET-GATE-TYPE(connection)
7      gate-inputs $\leftarrow \emptyset$
8      for each input $\in$ GET-INPUTS(connection)
9          gate-input $\leftarrow$ CREATE-TREE-RECURSIVE-COMPONENT(input, tree,
             parents, visited)
10         gate-inputs $\leftarrow$ gate-inputs $\cup$ gate-input
11     end
12     visited $\leftarrow$ visited $\cup$ input-port // Mark as visited so it is not
         visited again
13     return ADD-GATE(tree, input-port, gate-inputs, gate-type)) // OR gate
         representing the failure of the component or any of its
         inputs
14 end
```

The actual creation of gates, which puts together the desired inputs, happens during calls to function ADD-GATE (4). If there is only one input (lines 2 and 3), the gate does not provide any additional information and therefore is skipped to provide a more compact, easier to visualize view of the tree. Otherwise, the algorithm will create the desired gate (lines 5 to 8) and append the gate to the tree (lines 9 to 11). Additionally, the gate is stored associated with the port (function's argument with name *output*) that is provided as input to either of the recursive functions. This allows the reuse of the subtree that originates from the newly created gate when the algorithm tries to revisit the port, which results in fewer computations required and more compact trees.

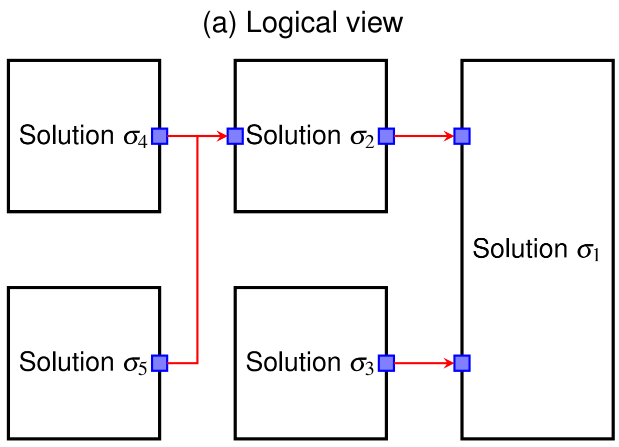
Algorithm 4: Add gate to the Fault Tree

```
Function ADD-GATE(tree, output, inputs, type)
    Inputs : The fault tree being created (tree).
             A logical port representing the output of the gate (output).
             The set of gate inputs (inputs).
             The type of gate to be created (type).
    Output: A gate with the specified type and inputs
    if $|\text{inputs}| = 1$ then // Skip the current gate by returning its only input
        return inputs;
    else
        gate ← $\{\}$
        gate.type ← type
        gate.inputs ← inputs
        gate.output ← output
        $\langle E, G \rangle$ ← tree
        $E \leftarrow E \cup$ output $\cup$ inputs
        $G \leftarrow G \cup$ gate; tree ← $\langle E, G \rangle$
        SET-GATE(output, gate) // Map output to the newly created gate so it can be recovered later by calling GET-GATE using output as key
        return output;
    end
end
```

5.4.2 Fault Tree Qualitative and Quantitative Analysis

The algorithm for the qualitative evaluation of fault trees is based on the MOCUS algorithm [150], which has been adapted to the data structures used in this research. Qualitative evaluation provides the Minimal Cut Sets and Minimal Path Sets of the tree. Quantitative evaluation of the fault tree is based on the inclusionexclusion principle. The equations labelled as 11.1 in the Reliability Engineering Handbook by Kececioglu [151, p. 236] are implemented to obtained quantitative results. The combined analyses allow to obtain the probability of failure of the top event, the relative importance of the various cut sets, and to rank the failures of solutions according to their contribution to the overall fault condition according to several importance measures such as Fussell-Vesley or Birnbaum importance measures.

Figure 5.11 shows a simple example of the creation of a fault tree. The logical view of the system is displayed in Subfigure 5.11a. Solution σ_1 requires the inputs from solutions σ_2 and σ_3. In turn, solution σ_2 requires the input from either σ_4 or σ_5, as these solutions are redundant. The resulting fault tree is presented in Subfigure 5.11b. Failure of σ_1 to receive adequate input is represented by the node labelled 'Failure σ_1'. This failure is either provoked by the failure of σ_3 or the failure of σ_2 to provide a valid output, which is modelled by the second 'OR' gate. This output failure is the result of σ_2 or both of the redundant solutions σ_4 or σ_5, which is modelled by the 'AND' gate.



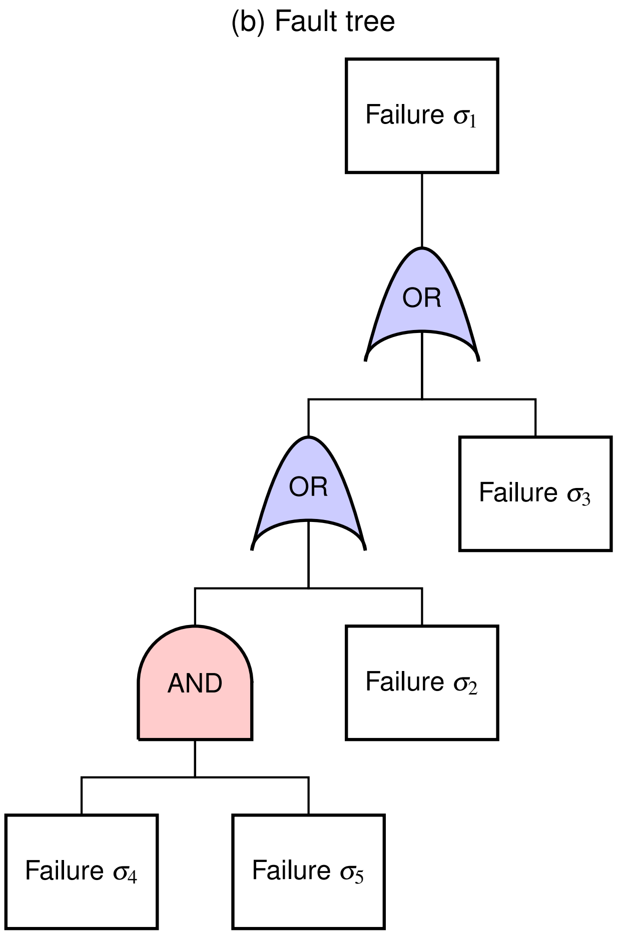


Figure 5.11: Example of the creation of a fault tree

5.5 Architecture Sizing And Performance Support

This section describes a novel method developed to support architects while sizing architectures and determining their performance. The method is built upon the state-of-the-art RFLP sizing enablers developed by Bile [16] and Bile et al. [15]. The enabler creates a computational workflow (see Section 2.6) for each subsystem. The workflow is then executed to determine the size of the system. However, the existing enabler only considers one environmental condition to size the system. In this thesis, this limitation is overcome by enabling the consideration of more detailed information about the system context as well as various system configurations.

The enhanced sizing process is described by the flowchart in Figure 5.12. The grey part of the flowchart compiles all architecture configurations to be used to created computational sizing workflows. Configurations can be motivated by the use of safety principles such as redundancy, which leads to *Failure Configurations* or by other design considerations, such as trying to optimise the system to varying environmental conditions. *Failure Configurations* are those that consider that a part of the system has failed, but thanks to existing redundancy the rest of the system can provide its intended functions. This kind of configuration will ensure that components are sized to account for the additional demand imposed on them in such failure situations. An example of failure configuration is the one engine failure scenario. Results from STPA hazard assessment or FTA (red part of the flowchart) can help to establish relevant configurations. An example of design configuration is the use of different air sources for avionics ventilation depending on the flight phase and environmental conditions.

The next part of the process is to call the workflow creation method by Bile [16] for each system and configuration. For a total number of subsystems N_{SS} and configurations N_C, a naive approach to the problem yields a total number of calls

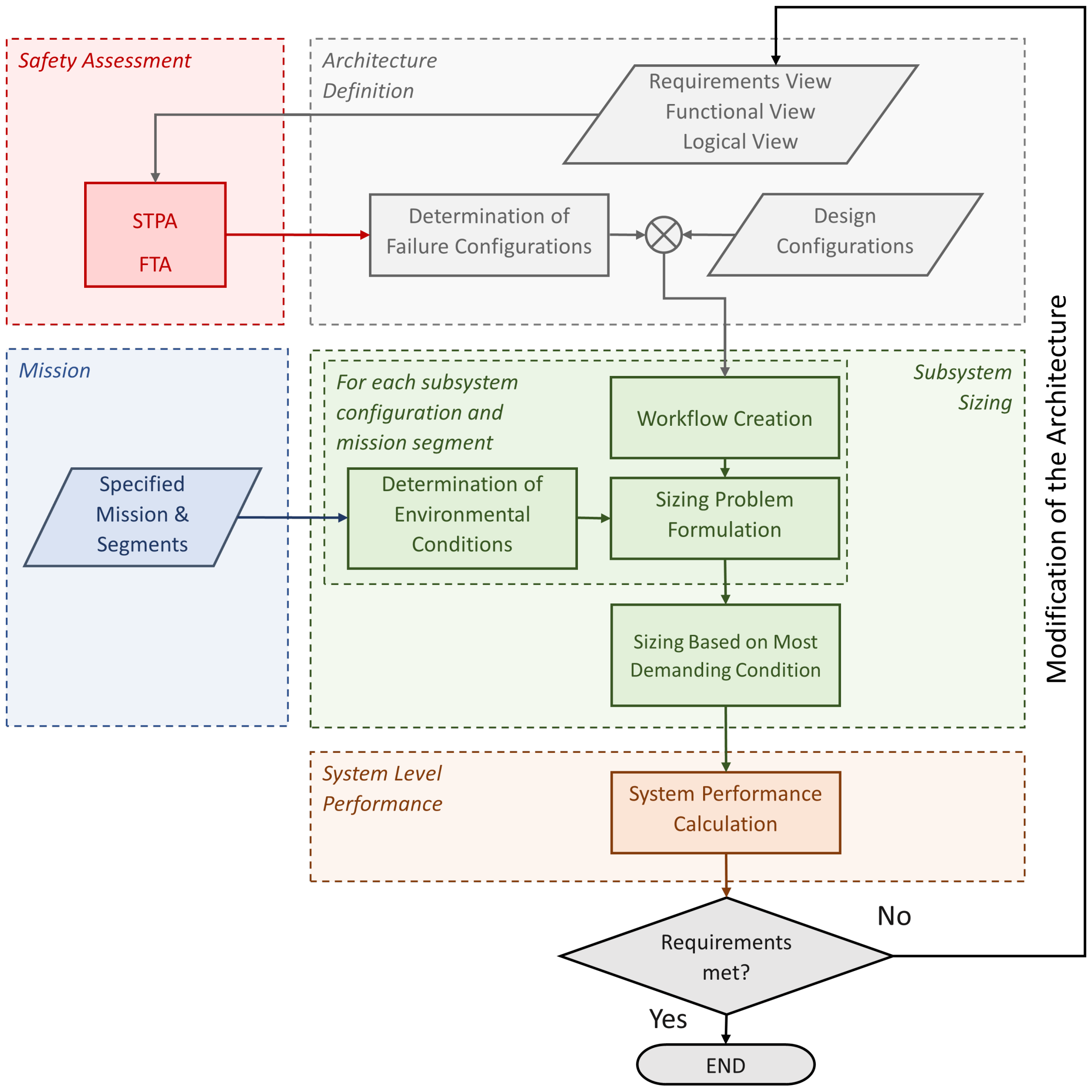

Safety Assessment
STPA
FTA
Architecture Definition
Requirements View
Functional View
Logical View
Determination of Failure Configurations
Design Configurations
Mission
Specified Mission & Segments
For each subsystem configuration and mission segment
Subsystem Sizing
Workflow Creation
Determination of Environmental Conditions
Sizing Problem Formulation
Sizing Based on Most Demanding Condition
System Level Performance
System Performance Calculation
Requirements met?
No
Yes
END
Modification of the Architecture

Figure 5.12: Flowchart for the process of sizing architectures and obtaining their performance

$$N_{WC} = N_{SS} \cdot N_C$$

to the workflow scheduling algorithm. This algorithm is one of the most expensive parts of the process, both in computational time and manual setup, so minimising the number of calls is of great importance. A more careful observation of the problem shows that, for a particular subsystem, only scenarios that affect the subsystem in a different way lead to different workflows. This way, the number of calls can be reduced to

$$N'_{WC} = \sum_{s=1}^{N_{SS}} N_{C_s}$$

where N_{C_s} is the number of distinct configurations from the perspective of subsystem s. The number N'_{WC} can be considerably smaller than N_{WC} when each configuration s affect only a small number of subsystems and therefore N_{C_s} is small compared to N_C.

Figure 5.13 shows an example of a system with two subsystems and two different configurations. All components are active in the first configuration, which is shown in Subfigure 5.13a. By contrast, the second configuration contains a failed component in ‘Subsystem 1’, as displayed in Subfigure 5.13b. The naive approach would require computing $N_C = 4$ workflows, two workflows per scenarios. But using the proposed approach, it is identified that ‘Subsystem 2’ has the same state in both configurations, which saves one workflow computation. Therefore, the number of required workflows becomes $N_C = 3$.

The next step in the workflow is to use mission information (blue part of the flowchart) to determine the possible environmental conditions that the system might encounter. After the sizing problem is formulated — inputs and outputs of the workflow are determined as well as any attribute of the subsystem to be

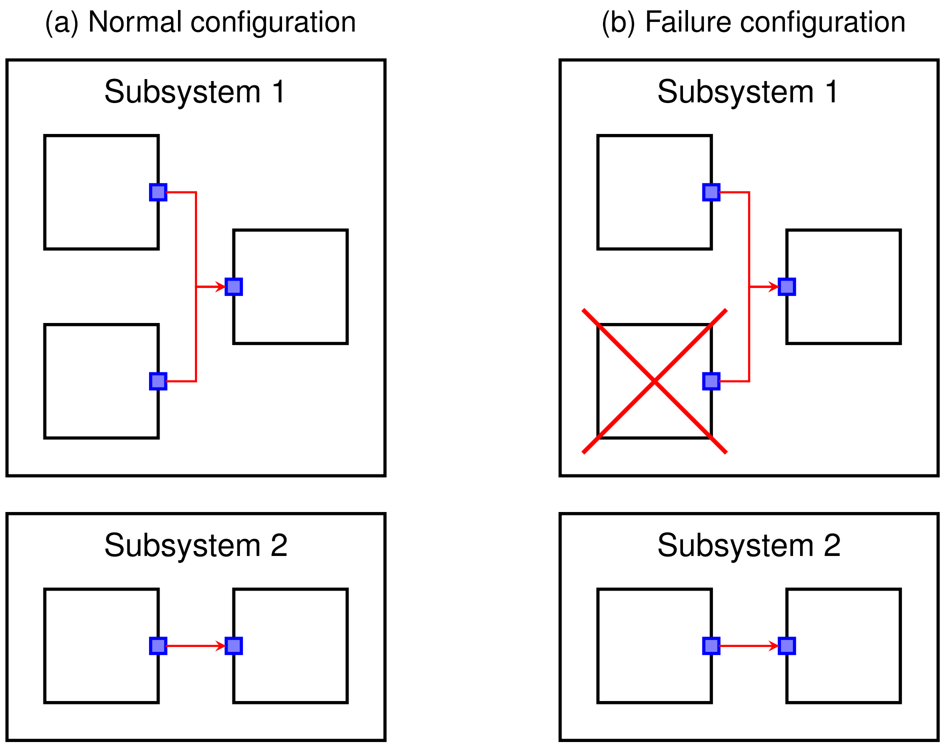


Figure 5.13: Scenarios and workflow creation

optimized — the workflow is executed with different values for the environmental variables. The results from this process are then used to establish the most demanding condition and size the system accordingly. Finally, the results from each sized subsystem can be aggregated up to the system level to obtain system-level performance. The procedure can be repeated until the results are satisfactory.

5.5.1 Modelling of Contextual Information

The concept of *Environment* is proposed to model contextual information regarding the different values that environmental variables adopt throughout the system's mission. The environment is composed of a computational workflow and a default *Environment Region* that can be further divided into smaller regions. The workflow is used to compute all dependent environment variables from a set of independent variables provided by the regions. The regions can be linked to one or more scenarios.

And environment region is composed of a collection of environment variables (such as altitude, temperature, velocity or pressure) and their upper and lower bounds. Only values inside the bounds belong to the region. The shape of the region can be refined further by using inequality constraints, which are computational models created from equations relating two or more environment variables. The default region is the one with the largest extension and any other region must be contained within it. Regions can override the default bounds to make them smaller and provide alternative constraint models. Dividing the default environment into smaller regions not only makes it easier to define each region but also allows the modelling of configurations that only apply to specific parts of the mission (and therefore only to certain regions of the environment).

When sizing a subsystem, as explained in the section above, a workflow will be created for each of the unique scenarios mapped to such subsystem. When finding the maximum demanding condition, the value of the environment variables required by the scenario's workflow will be determined by running the environment workflow. The input values to the environment's workflow are drawn from the environment regions that are linked to that scenarios.

5.5.2 Omnidirectional Connection Models

One of the most common problems that were encountered when applying the workflow scheduling methods [16] was the existence of overdetermined connection models. This situation arises when the value of the various flow attributes at both ends of the connection are provided by the component models corresponding to models at both ends of the connection. To mitigate this problem, omnidirectional models are proposed, which introduce extra variables so the model can be used in any direction. The benefits of this kind of models are twofold. Omnidirectional models allow architects to create workflows first and diagnose conflicts later when more information and better visualization techniques are available. Second, they defer the creation of the actual connection model until all inputs and outputs are determined, which makes it possible to discard any unused additional variable and formulate the model's equations in the right direction (avoiding the computational cost of solving the reversed model),

5.6 Summary and Conclusions

This chapter introduced a set of methods that were developed to overcome the limitations of current methods found during the *Descriptive Study I* stage of the research, which corresponds to the literature review presented in Chapter 3. The limitations of the methods were determined with respect to their ability to fulfil the objectives of this thesis. Consequently, the methods that support hazard assessment (Objective 1) were discussed first. A method that automates the creation of STPA hierarchical controls structures was presented. This was followed by the introduction of a method for automatically creating detailed STPA control loop models. Both methods utilise the hierarchical and flow relations within the logical view to create the desired models.

Architecting enablers for the introduction of safety principles, as required by Objective 2, were presented next. In particular, these enablers use the information contained in the functional and logical views of the architecture together with template libraries to support the architecting of physical and functional redundancy and containment. Graph theory is used to automate several parts of the process. Graph traversals determine the extension of redundancy and the propagation of disturbances. Maximum matching problems are solved to determine the similarity between functions to be fulfilled and candidate solutions to fulfil them. The minimum cut problem is solved to determine the optimal location of barriers.

The last two methods discussed in this chapter correspond to Objective 3 and aim to enable a faster assessment of the safety and performance of architectures. A method to automate the creation of fault trees was presented. The method uses the flow relations in the logical view and the functional-logical relations, which indicate the presence or lack of redundancy, to automatically create fault trees. Finally, a state-of-the-art sizing method was extended to consider various scenarios and environmental conditions. The creation of connection models was improved to reduce the amount of manual work required and increase the flexibility of the automated workflow creation process.

Chapter 6

Evaluation

6.1 Introduction

Several studies were performed to test and evaluate the novel methods and enablers developed during the course of this research. In this chapter, the methods employed, the results, and the conclusions from these studies are provided.

As stated in Section 1.5, the work presented in this thesis corresponds to prescriptive engineering design research and could be classified as a 'Type 3' study, according to the classification by Blessing and Chakrabarti [1]. A Type 3 study is where the majority of the effort is spent in producing a 'design support tool'. In this research, the design support tool consists of the different methods developed and presented in Chapter 5. Additionally, the methods were implemented within a prototype object-oriented software, 'Aircadia Architect', to facilitate their evaluation.

Three types of evaluations are usually performed for this type of research:

- **Support** evaluation checks the consistency and correct functioning of the developed tool. This kind of assessment is also referred to as 'verification'. In this research, the support evaluation was conducted by applying the proposed techniques to two case studies (see Section 6.3) and comparing the results with those provided by manual application of the methods or by other authors and tools.

- **Application** evaluation investigates whether the application of the methods impacts the design process as expected. In this case, this implies assessing that the objectives are achieved and as a result so is the aim of the research. As stated in Section 1.3, the aim is to improve the efficiency and effectiveness of design for safety as an integral part of the systems architecting process.

- **Success** evaluation intends to determine whether the developed methods can provide value and be useful in practice. As it was not possible to test the performance of the methods on real industrial projects, the evaluation consisted in the obtention of feedback from a panel of specialists from Airbus and Cranfield University, to whom the work was presented.

The rest of this chapter is organised as follows. Section 6.2 introduces the prototype software tool that implements the proposed techniques and was used to evaluate the research. Section 6.3 describes the two use cases that were developed to evaluate the tools and present the results to experts from industry. The results from the support evaluation are presented and discussed in Section 6.4. The industrial success evaluation and its results are presented in 6.5. Both sections contain information relevant to the application evaluation. Finally, conclusions are drawn in Section 6.6.

6.2 Software Prototype

AirCADia Architect is a prototype software tool that was developed for demonstrating the RFLP representation proposed by Guenov et al. [14]. The software used the object model formalised by Guenov et al. [14] as a foundation. In this thesis, the RFLP framework is extended as explained in Chapter 4. Consequently, the extended object model proposed in this research has been used to extend the prototype tool adding new capabilities for architecting of safety principles, STPA hazard assessment, FTA and sizing and performance analysis. AirCADia Architect is not publicly available outside Cranfield University, but it can be obtained by contacting Advanced Engineering Design Group.

AirCADia Architect has been used to implement and help to evaluate the various use cases proposed in this chapter. Beyond the capabilities described by the methods in Chapter 4, AirCADia Architect provides a high degree of interactivity to the users to facilitate the understanding of the architecture definition and the meaning of the analysis results. A more detailed description of the interactive support provided by the various enablers is presented below.

6.2.1 Interactive Support

AirCADia Architect supports several kinds of user interactions that intend to make the use of the tool easier and contribute to a higher understanding of the architecture and the application of the methods. AirCADia Architect is capable of providing an interactive visualization of results so architects can understand the rationale behind them. The tool also traces information within and across views so the designers can understand the impacts of changes in the whole system.

STPA Hierarchical Control Structure

Once the hierarchical control structure is created, it can be interactively explored. When a node is clicked, all links related to that node are highlighted and the flow variables transmitted through the links are presenter to the user. By clicking on links, it is possible to create STPA's unsafe control actions associated with the control action that is related to the link. Interaction with links also allows overriding the default hierarchical order (see Section 5.2.1), changing links of type ‘SameLevelInputOutput’ to ‘ControlAction’ or ‘Feedback’ and vice versa.

STPA Detailed Control Loops

After a detailed control loop is created, it can be interactively explored in AirCADia Architect. When a component node is clicked, all links related to that node and the nodes at the opposite end of the links are highlighted. This way, it becomes evident which components are related to which, and what position in the control loop occupy (see Section 5.2.2). Additionally, the corresponding component in the logical view of the architecture are highlighted so they can be seen in their original context. The elements from other views related to that component are highlighted as well.

Physical Redundancy Enabler

This enabler interacts with the architect by presenting a preview of the redundant leg, computed as explained in Section 5.3.1, showing included components and those with links crossing the boundary. It also displays a list of rules (see Section 5.3.1), which the user can add, select, delete or modify. When a rule is selected the corresponding element is selected and scrolled to (displayed on the centre) in the preview. The architect can add ‘UserDefinedStopRule’ rules by clicking the components with which the rule will be associated. The component also highlights the extension of the redundant leg in the architecture’s logical view, using blue for the included components and connections, and red for those components that require a further decision from the architect.

Functional Redundancy Enabler

The functional redundancy enabler interacts with the architect by presenting a preview of the redundant leg, and a list of functions yet to be fulfilled. When the user clicks on a function three lists are populated with the results obtained from applying the comparison algorithm (see 5.3.2). These three lists correspond to solutions from the leg, solutions from the architecture and solution templates from the library. When the architect selects an item from any of the lists, the preview and list of functions are updated accordingly.

Containment Enabler

The enabler interacts with the architect by presenting a preview of all solutions located between the disturbance sources and sinks. It also displays any barrier that has been added to the preview — manually or by the algorithm for automatic placement detailed in Section 5.3.3. Clicking on a component computes the set A of solutions affected by a perturbation originating from that component and the set B of solutions from which a disturbance could reach the clicked component. Solutions are coloured differently depending on the set to which they belong, namely $A \setminus B$ (can be affected but cannot affect), $B \setminus A$ (can affect but cannot be affected) or $A \cap B$ (both).

Fault Tree Analysis Support

To help architects to understand the results provided by the algorithm and the meaning of the different gates in the fault tree, the support for fault tree analysis displays the tree itself and a list with the results, which can be interactively explored. Clicking on the fault tree highlights the parts of the logical view that were considered when creating a particular gate or basic event. Clicking on a minimal cut or path set highlights both their constitutive elements in the logical view and also their respective basic events in the fault tree.

Architecture Sizing and Performance Support

Regarding sizing and performance support, AirCADia Architect allows user to interactively explore the various workflows suggested by the algorithm. It shows which models are over-constrained and which variables can be traced to the model to help architects to remove the excessive constraints. Interactive traceability is also provided, allowing users to gain insight into the relations among variables. Once a workflow is executed the results can be explored by hovering the mouse on top of the components and ports in the logical view, which display the values of their respective parameters.

6.3 Case Studies

To be able to evaluate the methods and demonstrated how they can be applied in a realistic scenario, two case studies were developed. The first use case is inspired by the systems architecture of an Airbus A320 and it was used to perform the success evaluation of all methods by presenting it to industry experts and obtaining feedback from them. It was also employed for the application evaluation of several methods as indicated in Table 6.1. Another case study was developed for the application evaluation of the methods related to STPA hazard assessment. This use case is inspired by the Wheel Brake System Example from the ARP 4761 [58], also used by Leveson et al. [61] to evaluate STPA against existing techniques.

6.3.1 A320 Inspired System Architecture Use Case

This use case starts with the creation of a baseline systems architecture of an aircraft similar to an Airbus A320. However, in this architecture, all the parts whose purpose is improving the safety of the architecture, such as redundant compon-

Table 6.1: Evaluation types and use cases employed

Method	**Support Evaluation**	**Application Evaluation**	**Success Evaluation**
STPA Hierarchical	WBS	WBS	Industrial Evaluation
STPA Detailed	WBS	WBS	Industrial Evaluation
Physical Redundancy	A320	A320	Industrial Evaluation
Functional Redundancy	A320	A320	Industrial Evaluation
Containment	A320	A320	Industrial Evaluation
Fault Tree Analysis	A320	A320	Industrial Evaluation
Sizing And Performance	A320	A320	Industrial Evaluation

ents or containment mechanisms, were disregarded. The proposed architecture could be interpreted as the outcome of a design process where only functional and performance requirements were considered. The architecture provides a realistic starting point for the processing of architecting safety into the architecture.

The architecture includes only a subset of the systems that can be found in a real aircraft. In particular, it includes the Cabin (CAB), the Environmental Control System (ECS), the Flight Control System (FCS), the Hydraulic System (HYD), the Pneumatic System (PNE) and the Engine (ENG). It also contains various relevant controllers and means of controlling the subsystems above, which are included to model the interaction between the crew and the aircraft, which is relevant for hazard assessment. Figure 6.1 shows the logical view of the architecture as displayed in AirCADia Architect, except for the labels, which have been magnified.

More details about the ENG, PNE and HYD subsystems are provided in Figure 6.2. In this figure, the constituents components of those three subsystems and their interconnections are shown, superimposed to the top-level view of the architecture. Figures 6.4 and 6.3 show the ECS and FCS in detail respectively.

The use case continues by improving the safety of the architecture, which

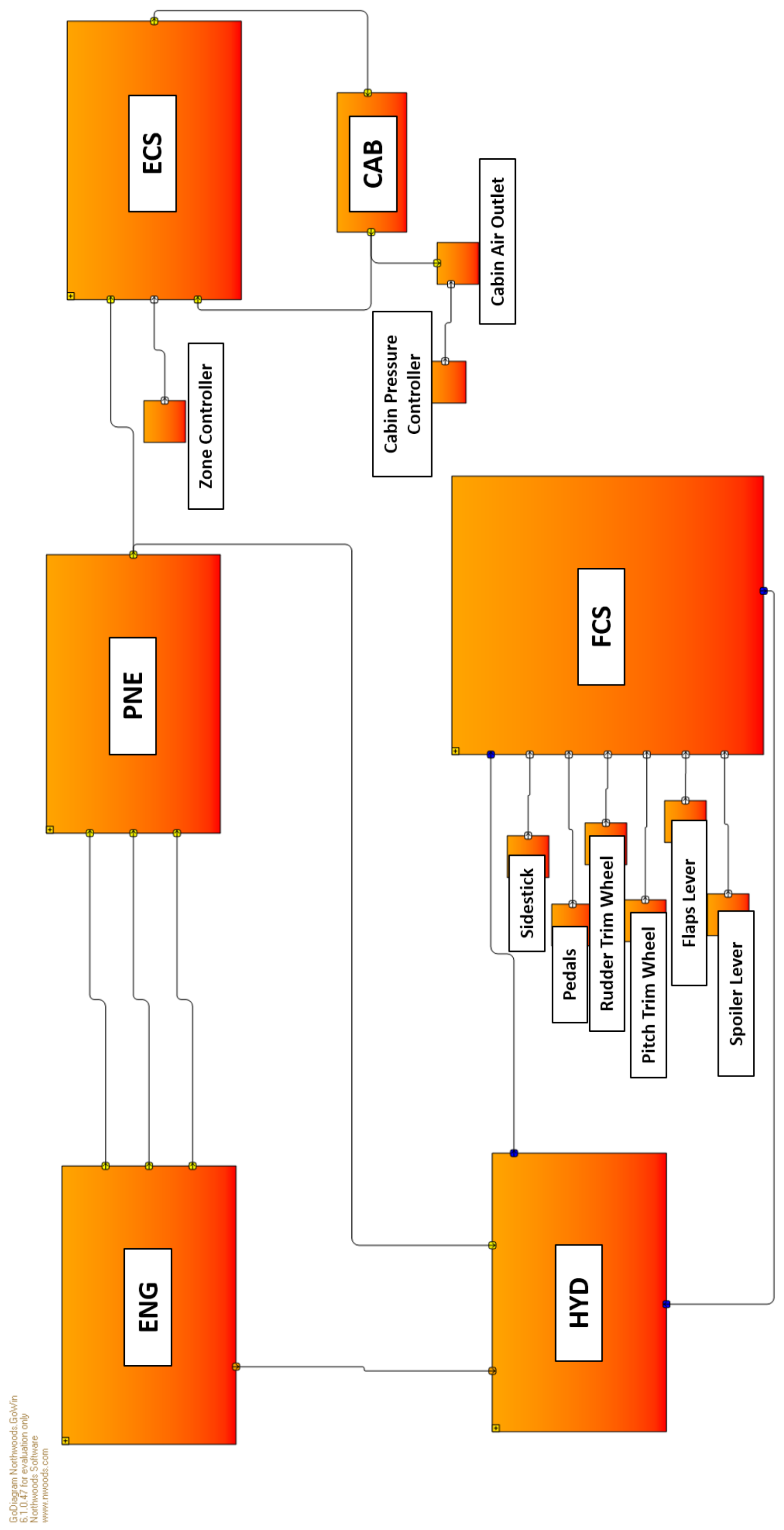


Figure 6.1: Initial systems architecture

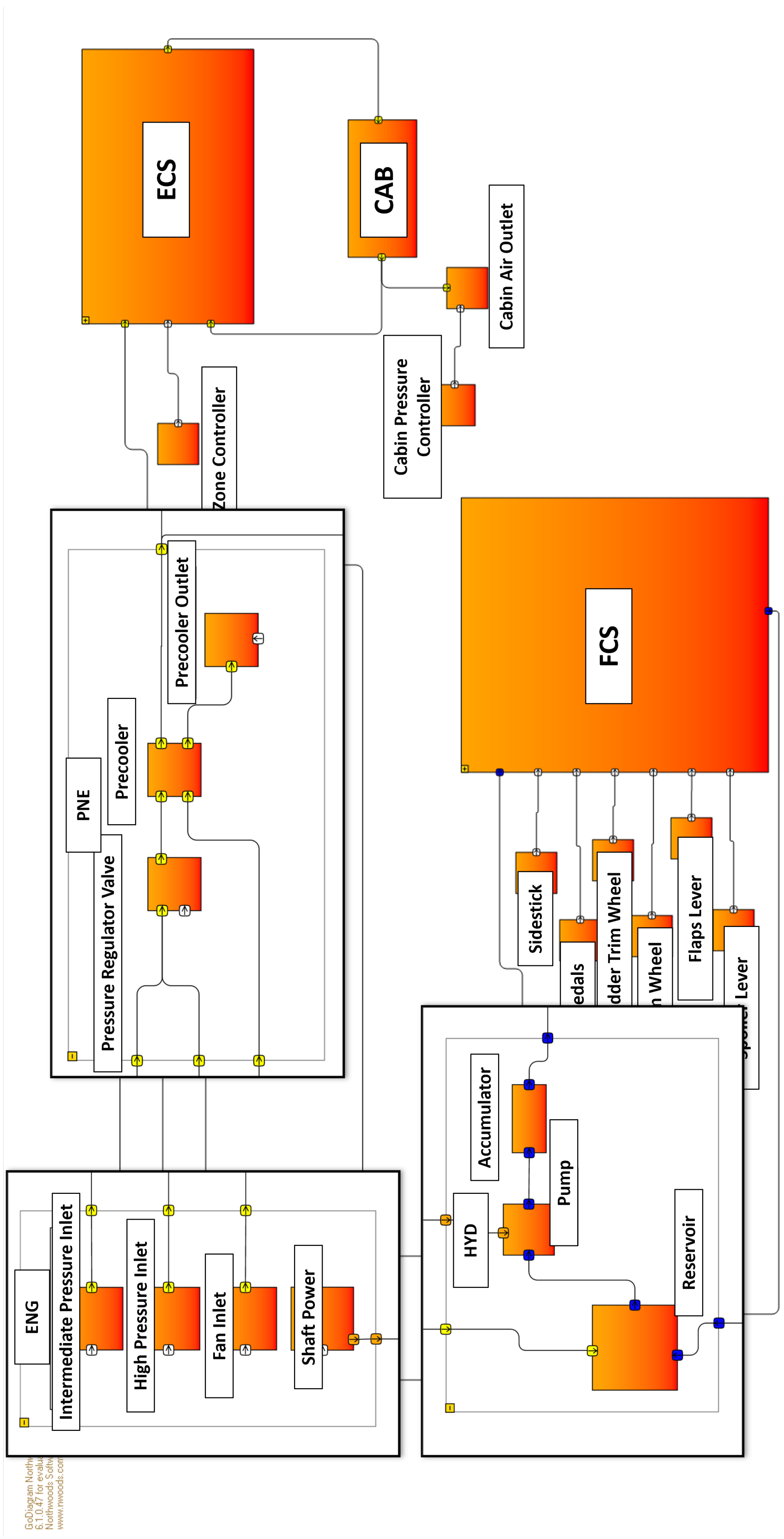


Figure 6.2: Detail view of ENG, PNE and HYD subsystems

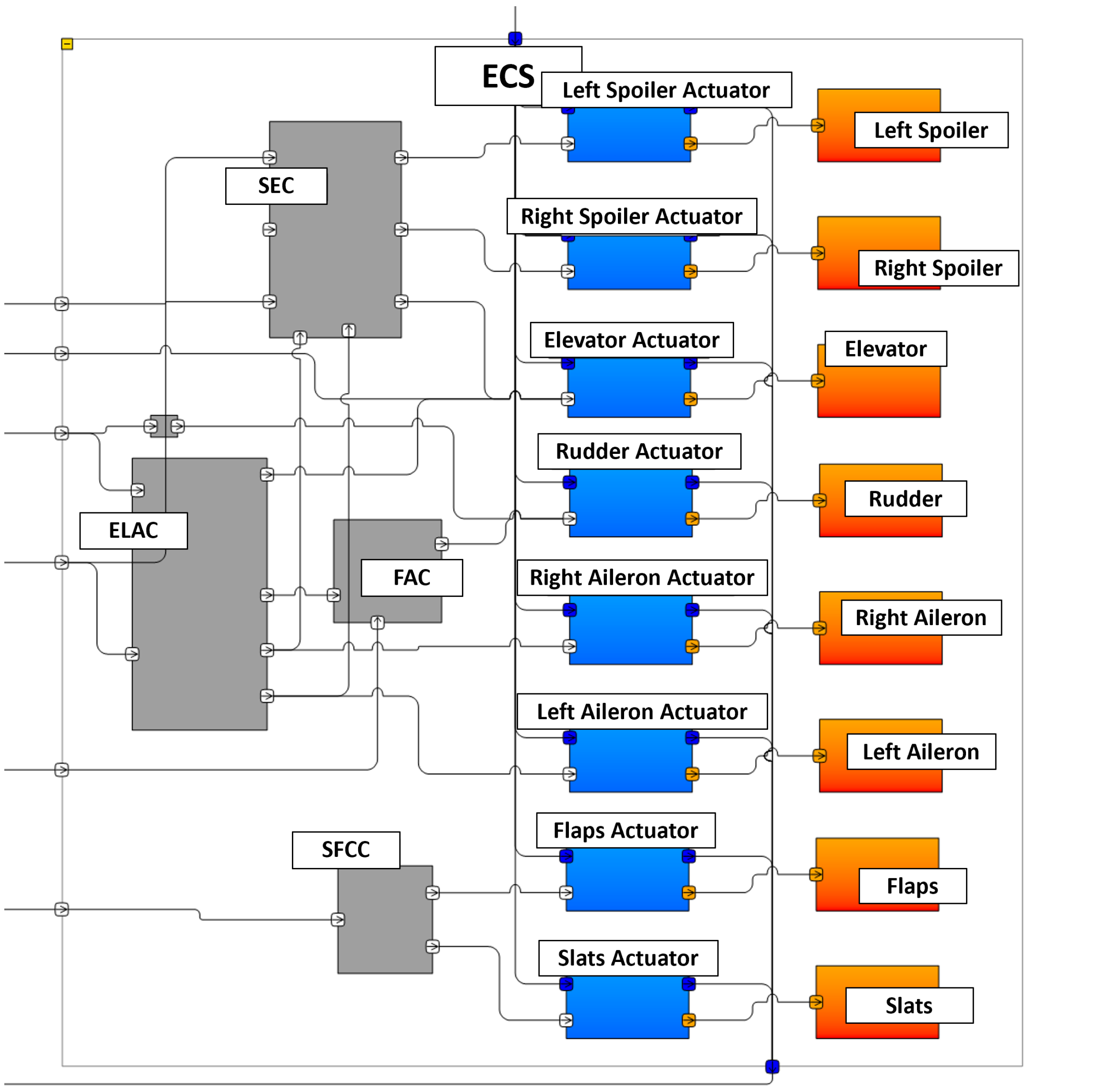


Figure 6.3: Detail view of ENG, PNE and HYD subsystems

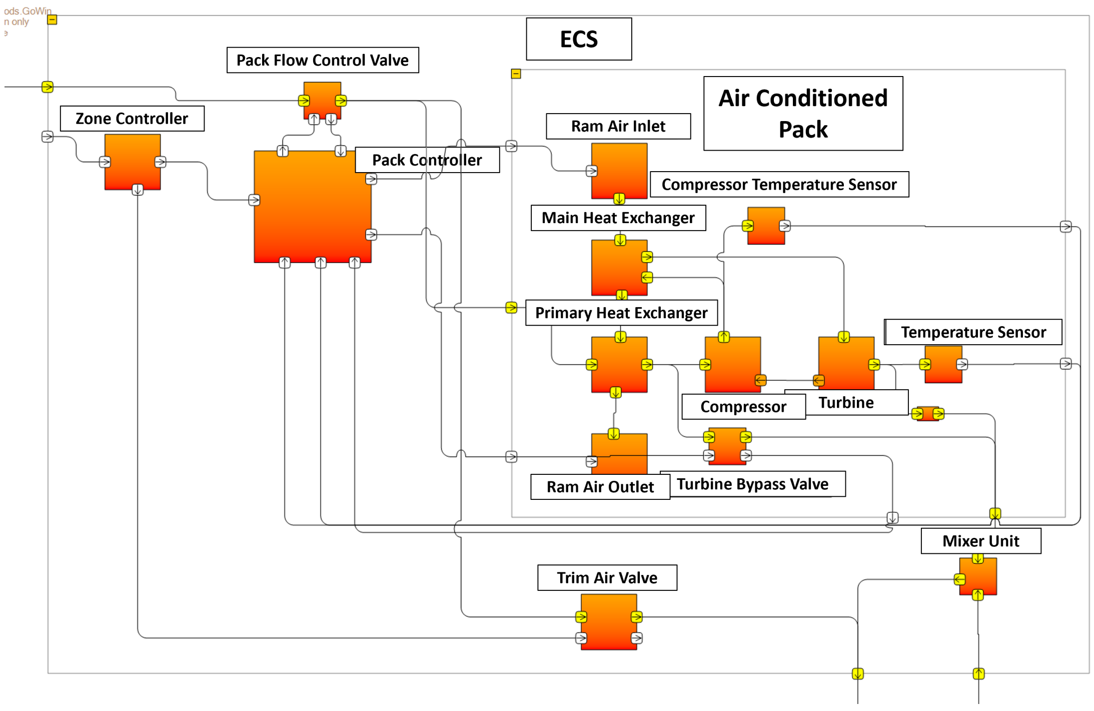
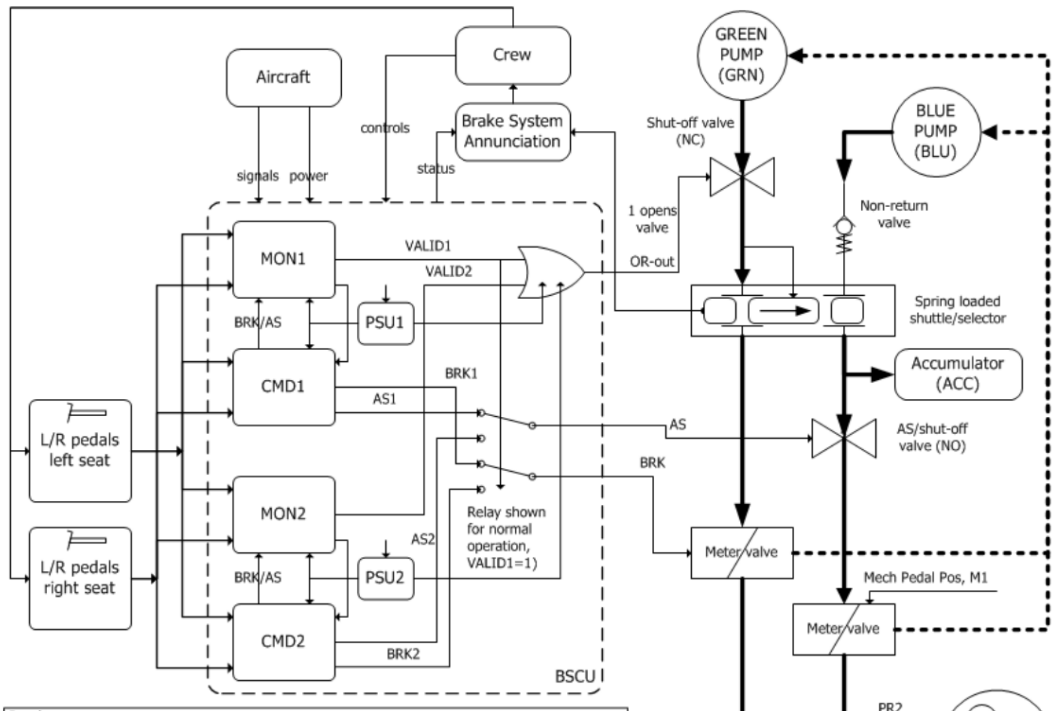


Figure 6.4: Detail view of the ECS subsystems

allows demonstration of the enablers developed in this research. First, the safety of architecture is assessed through STPA analysis, using the proposed methods when appropriate. The results from this process allow the definition of safety requirements. Then, the enablers for architecting safety principles are used to introduce redundant component and containment mechanisms where they are needed to comply with the new requirements. Finally, the architecture is analysed by using the FTA and sizing supports so the effects of the architecture changes can be assessed not only in terms of safety but also on performance.

6.3.2 ARP4761 Wheel Brake System Use Case

The second use case consists in the application of the STPA enablers to the Wheel Brake System (WBS) example proposed in the ARP 4761 [58]. The results from this use case are to be compared to those obtained by Leveson et al. [61]. To facilitate comparison, the architecture includes the assumptions made by Leveson et al., which can be found in the Appendix of the original document [61, pp. 68–72]. In particular, the diagram and textual description of the WBS found in the appendix of [61] was used to populate the logical view of the architecture in AirCADia Architect. The original diagram is reproduced in Figure 6.5 for convenience.



Logic
PR1=GRN.(VALID1+VALID2).(BRK1.VALID1+BRK2./VALID1)
PR2=M1.BLU.(/GRN+/VALID1./VALID2).(AS1.VALID1+AS2./VALID1)
PR1 reduces to GRN.(BRK1.VALID1+BRK2.VALID2./VALID1)
VALIDx = 1, channel is valid

Figure 6.5: WBS systems architecture (From [61])

As shown in the diagram the system has two hydraulic channels (green and blue) capable of applying pressure on the brakes of the aircraft wheels. The hydraulic part of the system is controlled by the Brake System Control Unit (BSCU) and by the pedal's mechanical input. The controller uses the manual brake commands from the pedals as well as inputs from the rest of the aircraft such as wheel speed or auto-brake mode. Apart from outputting control commands for various hydraulic valves, it also communicates its status to the Brake System Annunciation. The annunciation informs the crew about the state of the system so they can make the necessary decision to control the WBS through manual brake (using pedals) or automatic brake commands. More details about the system can be found in the original report [61];

Figure 6.6 portrays the system as implemented in AirCADia Architect. Some connections and components, which were implicit in the original diagram, but provided in the system's description, have been made explicit in this implementation. This is because to work properly, the tool and the methods require links to be included explicitly.

6.4 Support and Application Evaluation

The main focus of the section is the support evaluation of the research, which verifies the consistency and correct functioning of the support tool. The results presented in this section are also relevant to prove that the aim and objectives of the research are achieved, thus contributing to the application evaluation.

6.4.1 STPA Support

The support for performing hazard assessment using STPA methodology is evaluated by applying it to the WBS use case and then comparing the results obtained by Leveson et al. [61], who applied STPA manually to a very similar use case. The WBS architecture in this thesis was obtained by interpreting the text and diagrams in the appendix of the original report comparing STPA and ARP 4761 [61]. However, the appendix was found to be ambiguous to some extent and to omit information that is used later by the authors to perform STPA analysis. This resulted in the author of this research implementing a system in AirCADia Architect that is different to some extent from the system analysed by the authors of the original reference. More detail regarding what are the differences and their impact on the results are presented in the sections below.

Most of the discrepancies between the system implemented in this research

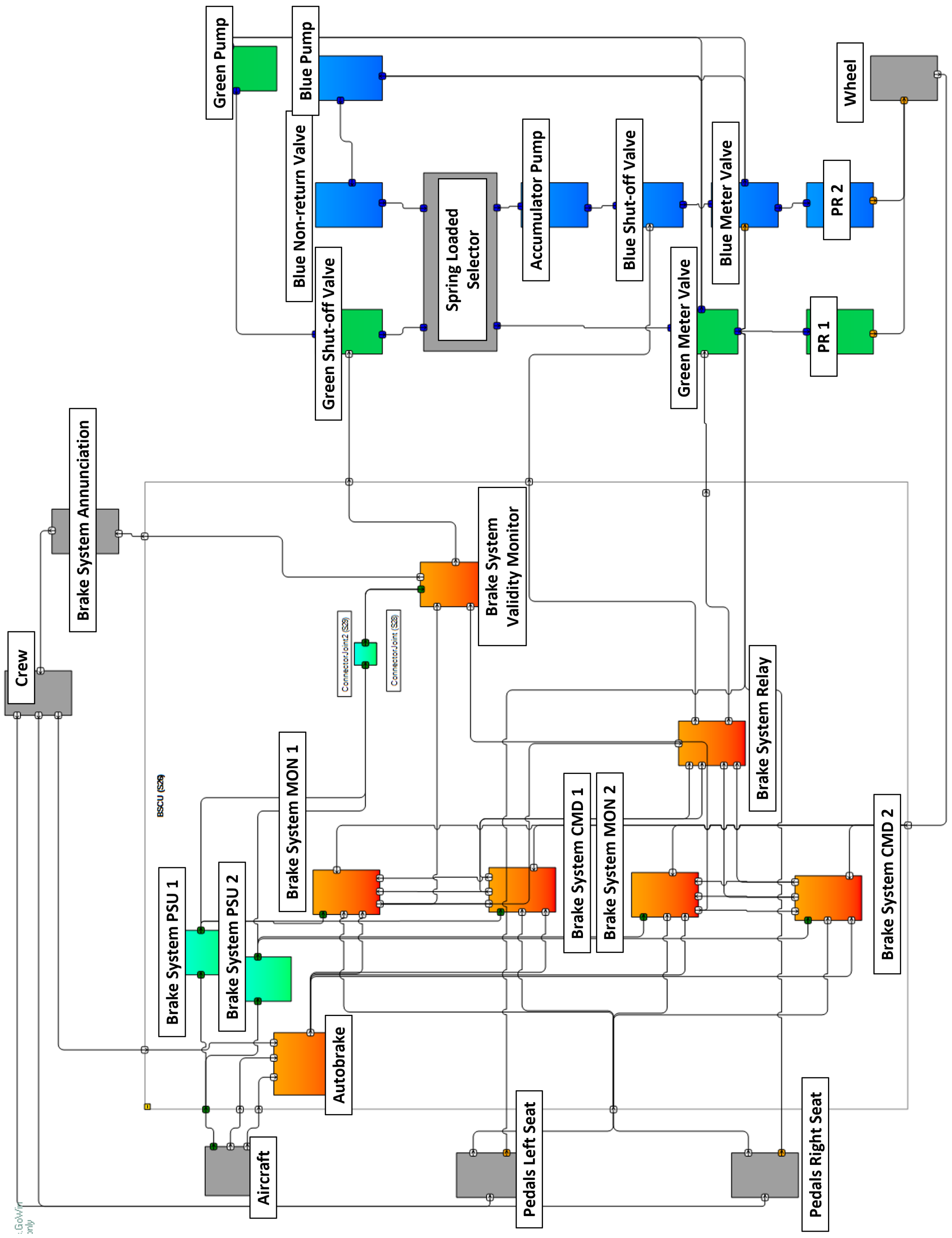


Figure 6.6: WBS systems architecture in AirCADia Architect

and the original one could be corrected after having compared the results in detail and having gained more insight into the system that Leveson et al. [61] had in mind. This information would make possible to model in AirCADia Architect a system that produces result even closer to the original ones. However, the author believes that a comparison using only the appendix is fairer as this is likely what the original analyst used for their STPA analysis. As a result, the system as interpreted from the appendix is used for evaluation.

STPA Hierarchical Control Structure

The first part of the STPA support evaluation focuses on the ability of the STPA support to automatically model the control hierarchies used for STPA analysis. The control hierarchy obtained by the authors of the original report [61] is shown in Figure 6.7. Figure 6.8 shows the control hierarchy as automatically modelled by AirCADia Architect. Regarding the included components and their position in the hierarchy, the similarity is high. The crew is situated at the top of the hierarchy. At an intermediate level, we can find the brake system control unit's (BSCU) controllers. Finally, the WBS hydraulics and wheel are located at the bottom of the hierarchy.

There are some differences though. The algorithm that creates the hierarchy (See 5.2.2), lumps by default all process components in one single node called 'Process Node'. But in the original report, the WBS hydraulics and the wheel appear as two separate entities. It would not be difficult to override the default grouping to obtain even more similar results. Another difference is the fact the original hierarchy has merged the two redundant control channels of the BSCU, whereas the automated method considers them individually. There is also a small difference in the order of controllers within the BSCU. The original report situates the autobrake at a higher level than other BSCU controllers, but it appears at the same level in Figure 6.8. This is due to the fact, that the autobrake was not even

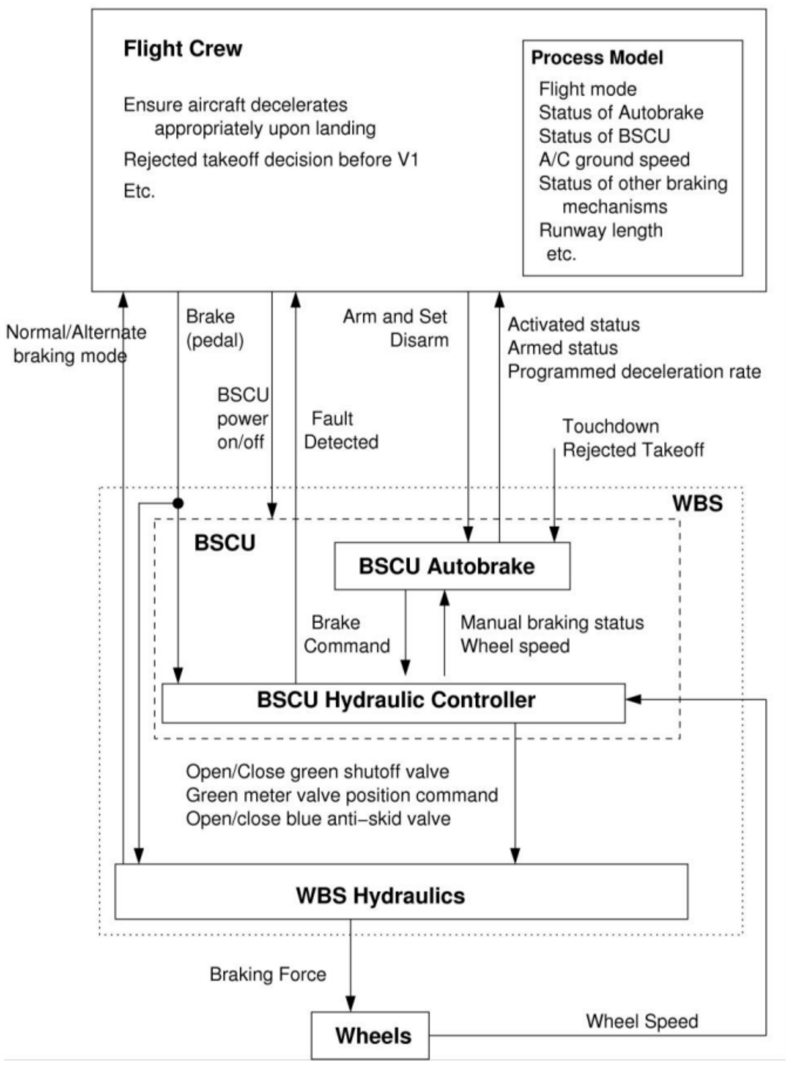


Figure 6.7: Control hierarchy obtained by Leveson et al. [61]. (From [61])

included in the original diagram (Figure 6.5), so it was situated at the same level as the other controllers when implementing the system in AirCADia Architect. The default order can be easily overridden by changing the type of the two links between the autobrake and each of the CMDs, obtaining identical ordering.

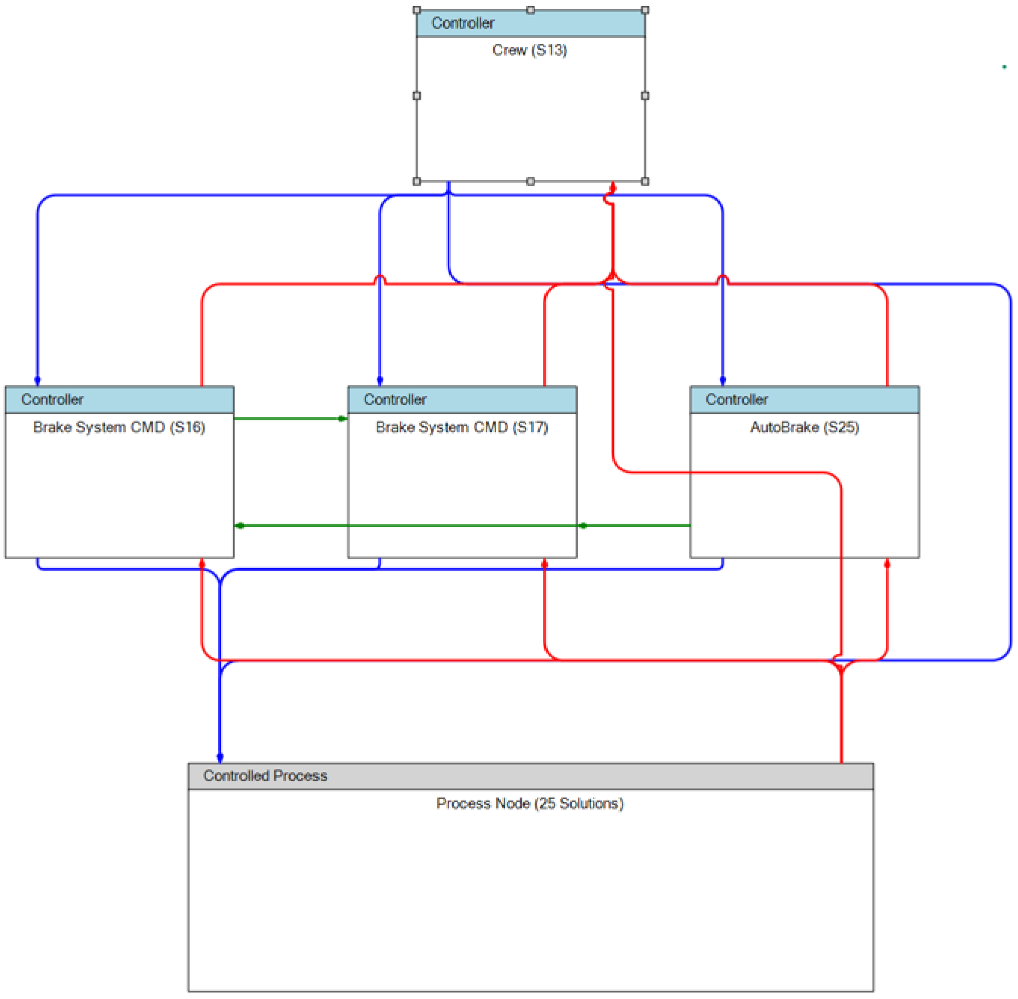


Figure 6.8: Control hierarchy obtained by AirCADia Architect

Regarding the description of the signals carried by each of the links, AirCADia Architect does not show all of them simultaneously as it results in clutter but displays the relevant information when a controller is clicked. Figure 6.9 shows the descriptions of the links associated with the CMD controller after such component is clicked. Tables 6.2 and 6.3 display the comparison between the original results and those in this research.

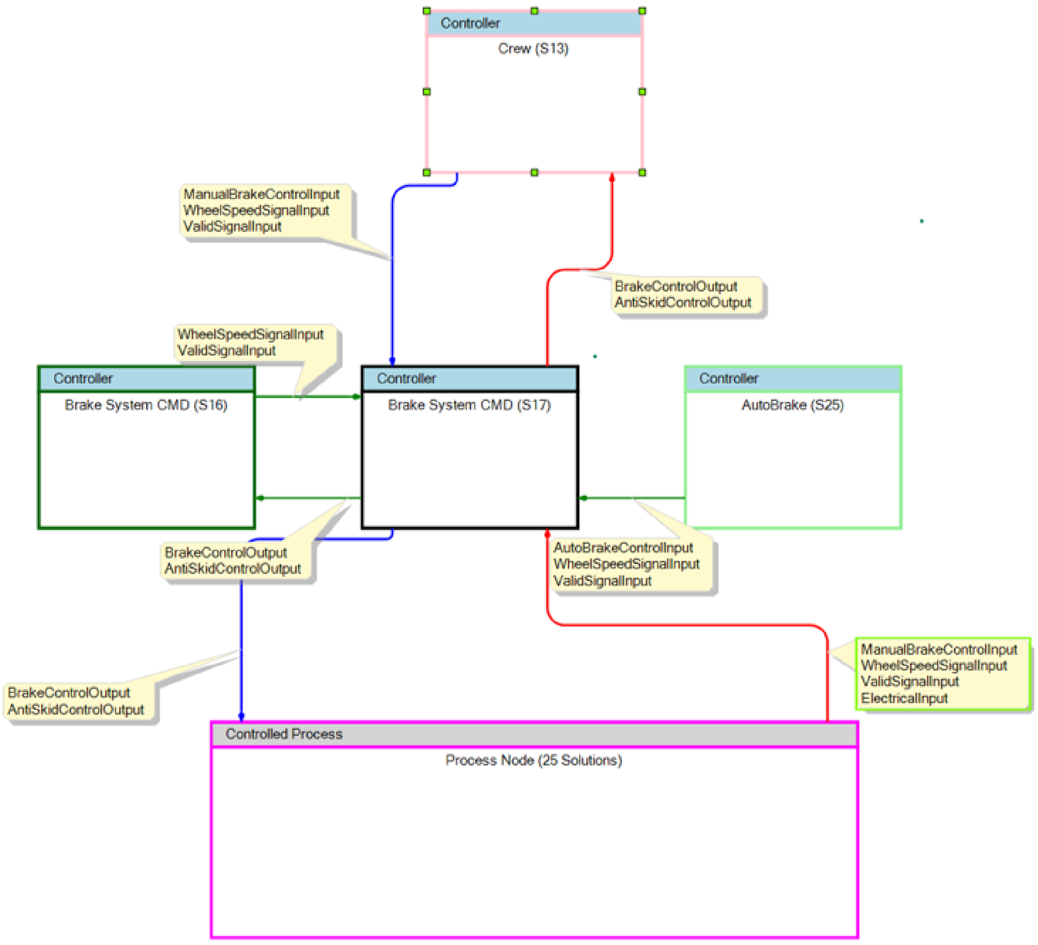


Figure 6.9: Control and feedback signals from CMD

The comparison results regarding control commands in the control hierarchy are shown in Table 6.2. Control command are those links from controllers situated at one level in the hierarchy to other controllers or process solutions situated at lower levels. Links at the same level originating from the BSCU CMD unit in this thesis have also been included in the table. Although the wording used to designate the commands in the original document and this research differ, which may seem to indicate that the results are different, their meaning is equivalent. Therefore, once the different terminology is taken into account, the results show a considerable degree of similarity. For simplicity, all signal descriptions shown next to each other in the tables throughout this section are equivalent unless explicitly stated. The differences are stated explicitly and explained in detail.

Table 6.2: Control commands in control hierarchy

From	**To**	**Original Signals**	**Signals**
Crew	BSCU	Power on/off	–
Crew	Autobrake	Arm and Set	Autobrake
		Disarm	Autobrake
Crew	Hydraulic Controller	Brake (pedal)	Manual brake 1 & 2 (x2)
Crew	WBS Hydraulics	Brake (pedal)	Manual brake 1 & 2
Autobrake	Hydraulic Controller	Brake command	Control command (x2)
	Process	–	Control command
Hydraulic Controller	WBS Hydraulics	Open/Close green valve	Brake command
		Green position command	–
		Open/Close blue valve	Anti-skid command
Hydraulic Controller	Hydraulic Controller	–	Wheel speed (x2)
		–	Fault signal (x2)
WBS Hydraulics	Wheels	Braking Force	–
Aircraft	Autobrake	Touchdown	–
		Rejected take-off	–

The power on/off command to the BSCU discovered in the original system was not discovered by the STPA support as it was not included in the architecture. This is because that command is not explicit in the diagram from Figure 6.7

and was not considered relevant when interpreting the textual description of the system. The same reasoning applies to the command from the crew to the autobrake, which explains why the original analysis considers two different commands (‘Arm and Set’ and ‘Disarm’) and the analyses in this document only discovered one command (‘Autobrake’). There are also some differences regarding what is considered as the process in the original analysis (hydraulics and wheels separately) and what was identified as a process by the algorithm (everything solution that is not a controller). This fact explains why the analysis in this thesis identified an additional command from the autobrake to the process (corresponding to the flow link between the autobrake and the 2 MONs), and why the commands from the hydraulic controller to the hydraulics are different — it seems like the original report included more component inside ‘Hydraulic controller’ as opposed to only including the CMD. It also explains why the original analysis included ‘Braking force’ as a command.

The fact that the STPA support considers the two CMD individually lead to the identification of two additional commands between CMDs (‘Wheel speed’ and ‘Fault signal’) not considered in the original analysis. It is important to note at this point, that the algorithm finds links between components through any existing flow path as long as any of the ports in such path belongs to a controller. Although the link between one CMD and the other is not apparent, that link exists by traversing the hydraulics up to the wheel and coming back via the wheel speed feedback signal, and it does not cross any controller. The independently considered CMD also lead to the discovery of many of the command and feedback links twice, which has been indicated as (x2) in both Table 6.2 and Table 6.3. The last discrepancy regarding control command is the fact that in Figure 6.7, ‘Touchdown’ and ‘Rejected take-off’ have been drawn as downward links, whereas in this research these links come from the process node and are therefore upward feedback links, shown in their corresponding table.

Table 6.3: Feedback signals in control hierarchy

From	To	Original Signals	Signals
Autobrake	Crew	Activated status	Fault signal
		Armed status	–
		Programmed deceleration	–
Hydraulic Controller	Crew	Fault detected	Fault signal
WBS Hydraulics	Crew	Braking mode	–
Process	Crew	–	Fault signal
Hydraulic Controller	Autobrake	Manual braking status	–
		Wheel speed	–
Wheels	Hydraulic Controller	Wheel speed	–
Process	Hydraulic Controller	–	Manual Brake (x2)
		–	Wheel speed (x2)
		–	Fault signal (x2)
		–	Electrical power (x2)

Table 6.3 presents the feedback signals in the control hierarchy. Feedback signals are those links from process solutions or controller situated at one level in the hierarchy to other controllers or process solutions situated at higher levels. The explanations for discrepancies in the case of control commands are also applicable to feedback signals. The difficulty of interpretation of the autobrake's textual description led to the consideration of a different set of signals: 'Activated status', 'Armed status' and 'Programmed deceleration' in the original report, and 'Fault signal' in this thesis. It has also caused the signals between the hydraulic controller and the autobrake 'Manual braking status' and 'Wheel speed' not to be included in the architecture.

The fact the original diagram (see Figure 6.7) does not include a 'Fault detected' link between the hydraulics and the crew, explains why it was not included in the architecture implemented in AirCADia Architect and therefore not discovered by the method. However, an additional feedback signal was discovered between the process and the crew; this is because the validity monitor is included in the process in the architecture used in this research. The partition of the set of process solutions in the original document explains why the reported link between the wheels and the hydraulic controller was not discovered. However, this link is discovered between the process and the CMD controller, as well as the electrical power, fault signal and manual brake links from other process solutions.

STPA Detailed Control Loops

The second part of the STPA support evaluation focuses on the ability of the STPA support to automatically model detailed control loops for STPA analysis. Figure 6.10 shows a control loop from the original report [61]. Figure 6.11 shows the control loop as automatically created by AirCADia Architect. There is a high similarity between both loops regarding the components that belong to the various parts of the loop, namely controller, actuators, sensors and controlled process. The crew was the chosen controller for this loop, the brake pedals were identified as the actuators and the annunciation system is the ‘sensor’ providing feedback to the crew. The controller process is the wheel brake system, including the hydraulic controller and the hydraulics.

Regarding the description of the links with origin or destination in any of the parts of the control loop, AirCADia Architect shows them only after links are clicked by the user, as otherwise the view is easily cluttered. The implemented STPA support also can export a summary of the results with the constituent components and links in a textual format. Table 6.4 presents the comparison of links obtained by both the original manual analysis and the automated analysis by the methods proposed in this research.

The original loop is less detailed than the one provided in this research because of two reasons. The first one is because it did not provide any description for the links starting from or ending in the WBS. The second reason is that it does not distinguish between any of the two pedals or redundancies in the WBS.

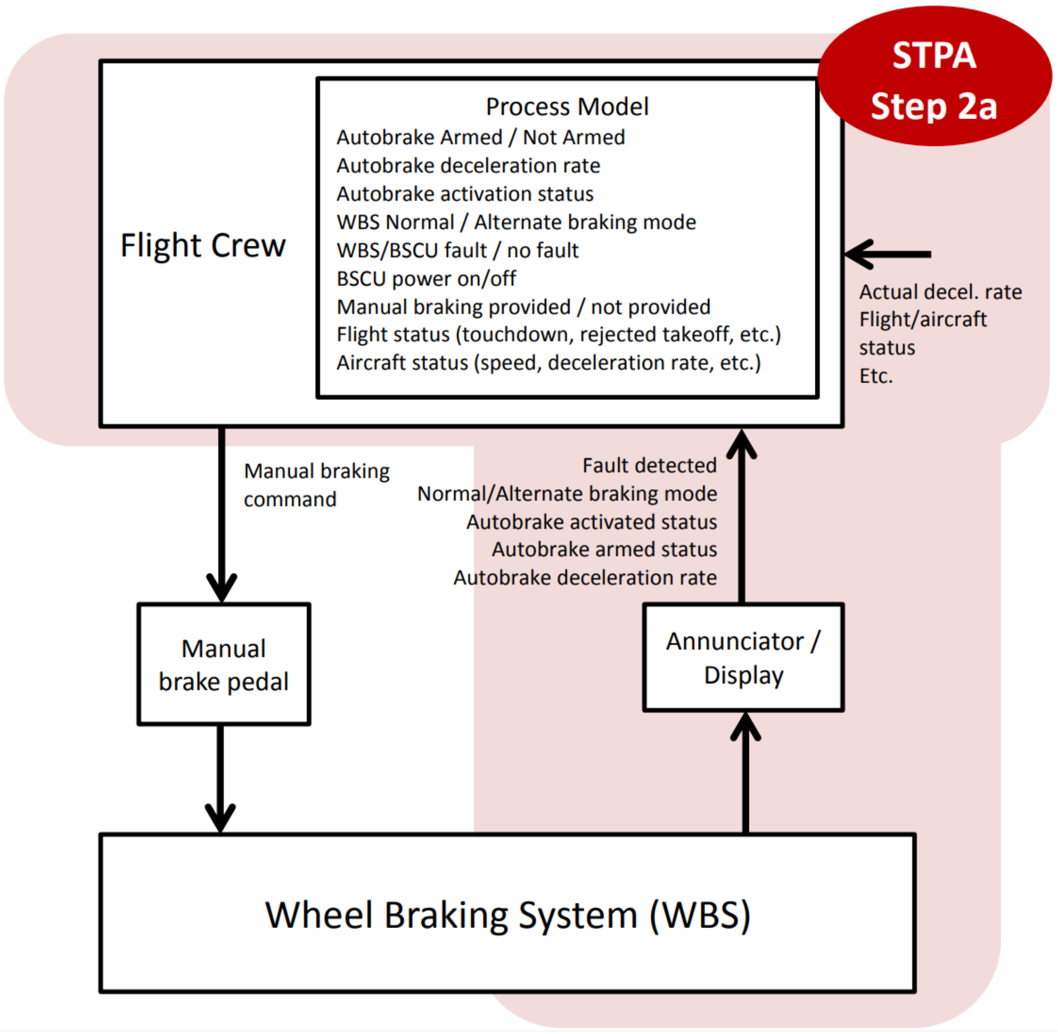

STPA
Step 2a
Flight Crew
Process Model
Autobrake Armed / Not Armed
Autobrake deceleration rate
Autobrake activation status
WBS Normal / Alternate braking mode
WBS/BSCU fault / no fault
BSCU power on/off
Manual braking provided / not provided
Flight status (touchdown, rejected takeoff, etc.)
Aircraft status (speed, deceleration rate, etc.)
Actual decel. rate
Flight/aircraft status
Etc.
Manual braking command
Fault detected
Normal/Alternate braking mode
Autobrake activated status
Autobrake armed status
Autobrake deceleration rate
Manual brake pedal
Annunciator / Display
Wheel Braking System (WBS)

Figure 6.10: Crew control loop obtained by Leveson et al. [61]. (From [61])

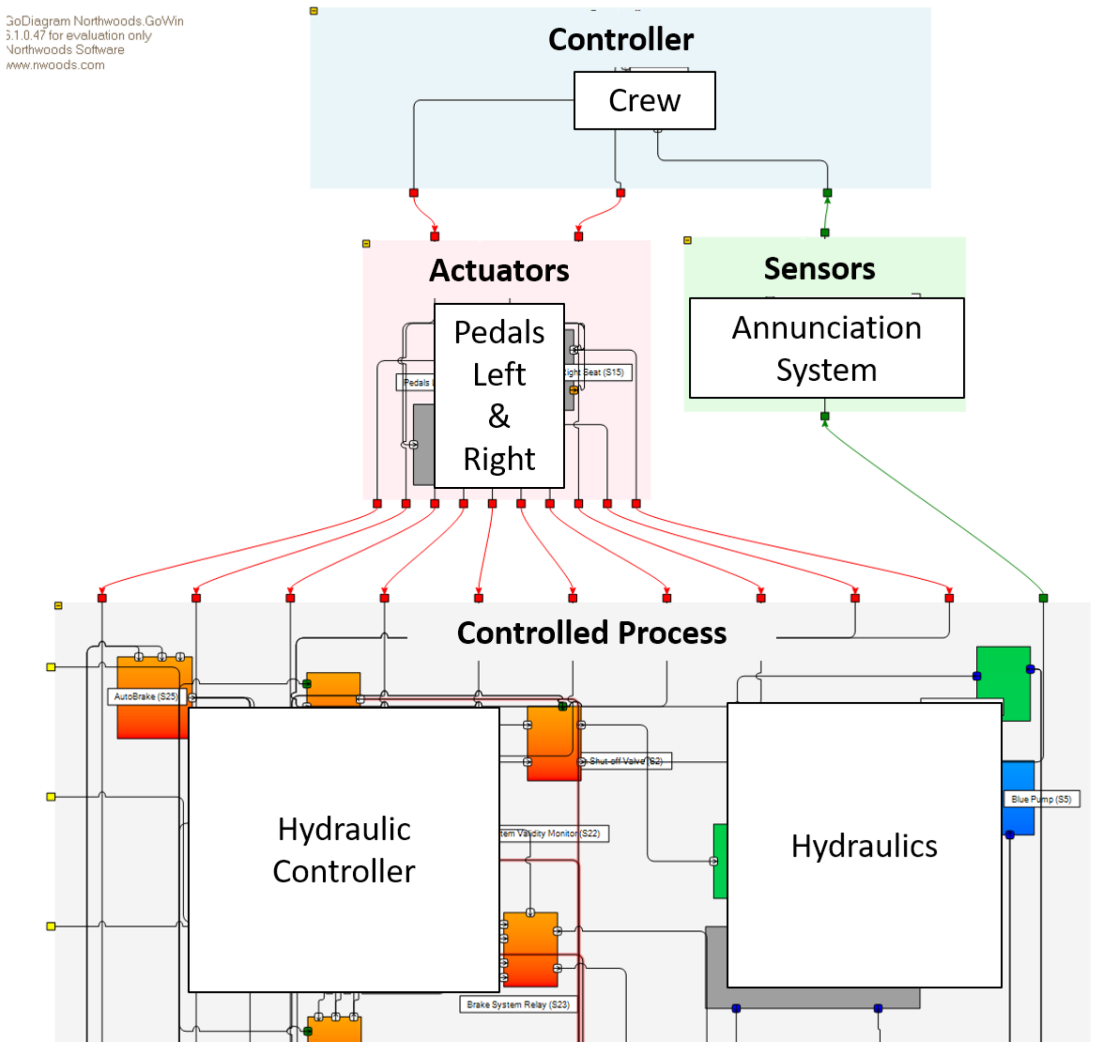


Figure 6.11: Crew control loop obtained by AirCADia Architect

Because of this lack of detail, Table 6.4 shows a great number of links that are not included in the original analysis. These links include all links categorised as 'Process Inputs From Actuators', 'Process Inputs From Other Process Solutions', 'Process Outputs To Sensors', and the second link in 'Controller Outputs To Actuators'. There are however a few links that appear in the original loop but have not been identified by the algorithm used in this research (see Section 5.2.2). These links are in categories 'Controller Inputs From Sensors' and 'Controller Inputs From Other Process Solutions' and were not included in the architecture implemented in AirCADia Architect because they were not explicit in the drawing and either missing or not inferred from the interpretation of the textual description of the system.

The other two detailed control loops obtained in the original have also been created in AirCADia Architect. The obtained results compared to the manual analysis in a similar way to the first loop, presented in detail above.

Discussion of Results

Although it was not possible to recreate the exact system the original analysts had in mind from the drawings and textual description provided in their report [61], the results obtained by the methods developed in this thesis presented a considerable degree of similarity. The main sources of discrepancy were the different terminology and the different interpretation of the system. Whereas part of the difference could be due to an erroneous interpretation of the original document, it is also true that Leveson et al. use links for STPA analysis that are not present in their drawings and that the description of the system is ambiguous to some extent. Furthermore, the comparison presented in Reference [61] also had to interpret and make assumptions the original ARP 4761 WBS example which they use to compare STPA with other methods. The need for interpretation and its possible impact on results highlight the importance of having a common unam-

Table 6.4: Signals in the crew control loop

From	**To**	**Original Signals**	**Signals**
		Controller Inputs From Sensors	
Annunciation	Crew	Fault Detected	FaultSignalOutput
		Braking mode	–
		Autobrake activated	–
		Autobrake armed	–
		Autobrake deccel. rate	–
		Controller Inputs From Other Process Solutions	
Unspecified	Crew	Various signals	–
		Controller Outputs To Actuators	
Crew	Pedals Left Seat	Manual Braking	Manual Brake
Crew	Pedals Right Seat	Manual Braking	Manual Brake
		Process Inputs From Actuators	
Pedals Left Seat	Blue Meter Valve	–	Mechanical Position
Pedals Right Seat	Blue Meter Valve	–	Mechanical Position
Pedals Left Seat	CMD1	–	Brake Command
Pedals Right Seat	CMD1	–	Brake Command
Pedals Left Seat	MON1	–	Brake Command
Pedals Right Seat	MON1	–	Brake Command
Pedals Left Seat	CMD2	–	Brake Command
Pedals Right Seat	CMD2	–	Brake Command
Pedals Left Seat	MON2	–	Brake Command
Pedals Right Seat	MON2	–	Brake Command
		Process Inputs From Other Process Solutions	
PSU	CMD1	–	ElectricalOutput
PSU	MON1	–	ElectricalOutput
PSU	CMD2	–	ElectricalOutput
PSU	MON2	–	ElectricalOutput
PSU	Validity Monitor	–	ElectricalOutput
PSU	Validity Monitor	–	ElectricalOutput
Aircraft	AutoBrake	–	TouchDownSignalOutput
Aircraft	AutoBrake	–	RejectedTakeOffSignalOutput
		Process Outputs To Sensors	
Validity Monitor	Annunciation	–	Fault Signal

biguous description of the system, to which various analysts or experts can refer. The developed methods work by using an RFLP model that is common to all analyses and whose elements are clearly defined, which mitigates interpretation and communication issues and ensures everybody works with the same system.

The ability to partition the set of process solutions exhibited by the manual approach, in contrast with the single-node approach employed by the algorithm used for creating the hierarchical structure (see Section 5.2.1) also lead to some differences. It could be interesting to remove this limitation, allowing the algorithm to consider different groups of process solutions, to be able to match the original results even closer and offer more flexibility in future analyses.

6.4.2 Safety Principles Enablers

Physical Redundancy Enabler

The enabler for architecting physical redundancy is applied to the A320 inspired use case. It is utilised to transform the initial architecture into one more similar to the one in the actual aircraft in terms of safety features. Physical redundancy is a commonly employed principle in aircraft systems architectures, so there are plenty of situations where the enabler can be tested. Table 6.5 displays all the test cases grouped by the subsystem to which the original components belong. Column 'Component' indicates the first element added to the new redundant legs, used as a starting point for the algorithm that calculates the extension of the leg. Column 'User Defined Rules' displays how many rules were input to the algorithm to obtain the desired result. Finally, column 'N_{legs}' indicates how many redundant legs were created. Therefore, $N_{legs}+1$ is the total number of components of each type after redundancy is applied.

Redundancy was applied to the engine to model the fact that the aircraft posses two engines. Since detailed modelling of the engine is out of scope,

Table 6.5: Physical redundancy cases

Subsystem	**Component**	**User Defined Rules**	N_{legs}
ENG	Shaft Power	No rules	1
	Fan Inlet	No rules	1
	High Pressure Inlet	No rules	1
	Intermediate Pressure Inlet	No rules	1
HYD	Accumulator	1 Stop rule, and 2 Merge with existing rules	1
PNE	Precooler	2 Merge with existing rules, and 1 Continue through output rule	1
ECS	Compressor	1 Merge with existing rule, and 2 Continue through output rules	1
FCS	Elevator Actuator	2 Merge with existing rules	1
	Elevator	1 Include existing redundancy rule, 2 Merge with existing rule, 2 Stop rules	1
	Rudder Actuator	1 Stop rule, and 1 Merge with existing rule	2
	Flaps Actuator	2 Stop rules	1
	Slats Actuator	2 Stop rules	1
	Right Aileron	2 Stop rules	1
	Left Aileron	2 Stop rules	1
	Right Spoiler	2 Stop rules	4
	Left Spoiler	2 Stop rules	4

the engine consists of three air inlets (fan, intermediate and high pressure) and a component representing the extracted shaft power. Due to this modelling, the redundancy required four simple operations where no additional rules were required. Figure 6.12 reflects the four stages of the process.

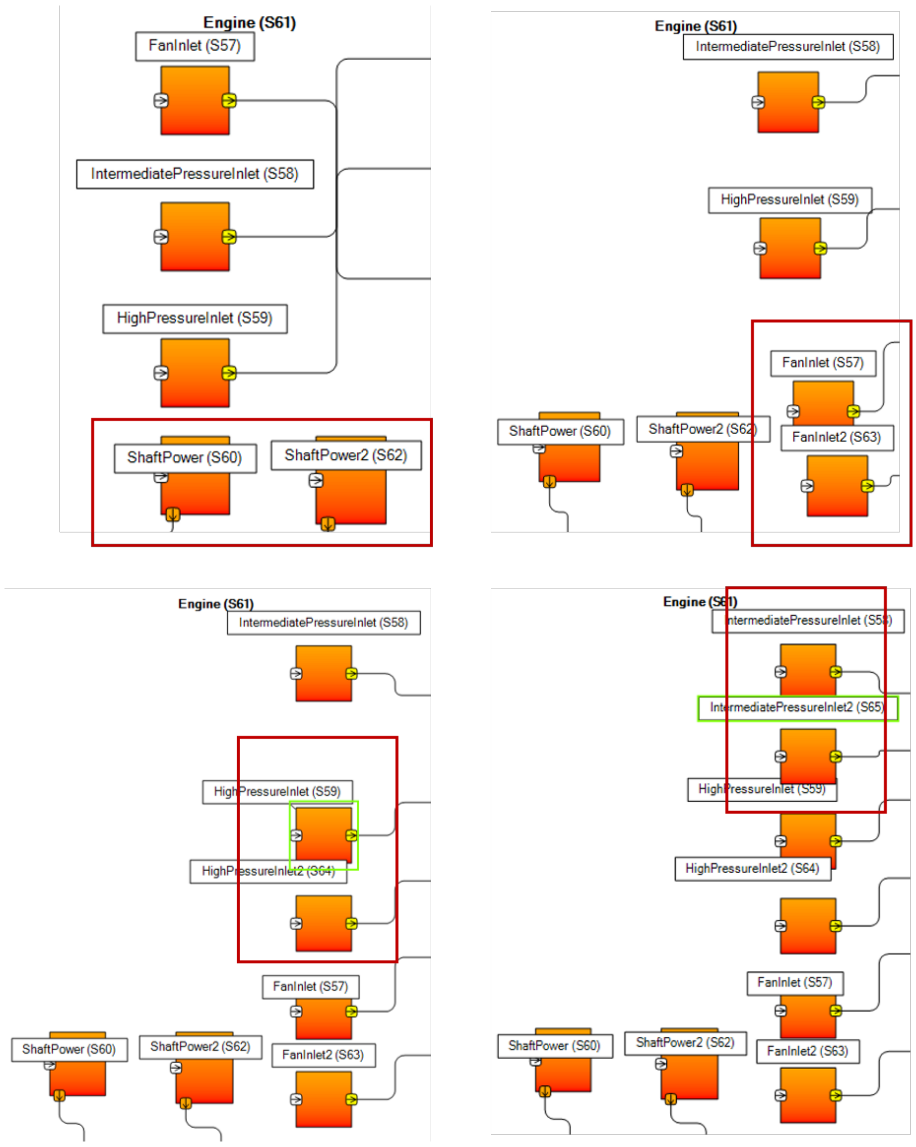


Figure 6.12: Application of physical redundancy to the engine

Regarding the hydraulic subsystem, the physical redundancy enabler was applied to create an additional redundant circuit. The application was more complex this time. It required the definition of a Stop rule at the reservoir pneumatic input and merging the new leg with existing redundancy at two points. The first point is the reservoir's hydraulic input, which receives hydraulic flow from many actuators. The second one is the pump's shaft power input receiving at this point power from both engines. The redundant leg included eventually all components in the hydraulic subsystem.

At the pneumatic system, the enabler was used to created one redundant leg. Starting from the precooler, the process required to define two rules to merge with existing redundancy at the pressure regulator valve's and precooler's inputs. This redundancy is due to the existence of the two engines. A third and final rule was required to include the air outlet connected to the precooler, as by default, the algorithm does not include components by following output connections. The redundant leg consisted of all components in the pneumatic subsystem. Figure 6.13 show the resulting architecture.

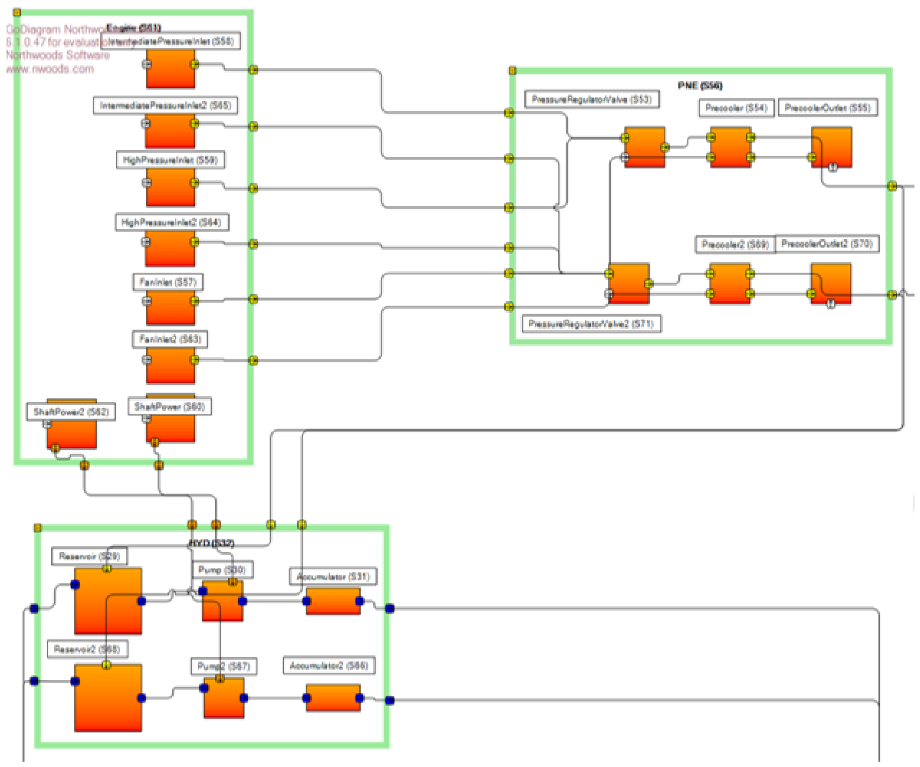


Figure 6.13: Application of physical redundancy PNE and HYD systems

The application of physical redundancy to the environmental control system started from the air pack's compressor. It required to merge with the existing PNE redundancy at the flow control valve's pneumatic input. It was also necessary to include the air outlet connected to the primary heat exchanger for reasons identical to those given for the PNE subsystem. In this case, the redundant leg included every component in the air conditioning pack, which was duplicated as a whole, plus the flow control valve and zone and pack controllers.

Finally, in the case of the flight control system, physical redundancy was applied to most control surfaces and actuators. The elevator was converted from a single-surface single-actuator system to a double-surface system with two actuators per surface. This was done in two steps. The first step added the redundant actuator and required two rules to mere with the existing hydraulic and control redundancies respectively. Then, in a second step, the enabler was applied to the elevator surface, in this case including the existing redundant actuators to obtain the desired configuration. It also required two additional merge rules, one for each actuator's control input; and two stop rules, one for each actuator's hydraulic input. Unlike in the first step, there was not hydraulic redundancy at the actuator, as only one of the hydraulic circuits was connected to each one of the actuators.

The rudder was left unchanged, but a triple-redundant actuator system was implemented. The algorithm required the creation of a 'Merge with existing redundancy' rule at the actuator's control input, and a stop at the actuator's hydraulic input, similar to the elevator case. For the rest of the control surfaces — flaps, slats, right and left ailerons, and right and left spoilers — the process of obtaining the desired redundancy required the addition of two stop rules, one for each input port of the corresponding actuators. The number of new legs created for each control surfaces is indicated in Table 6.5. Figure 6.14 shows the resulting FCS architecture. The blue actuators are the original ones, whereas the green ones represent the ones added for redundancy purposes; a lighter shade of green represents the third rudder actuator.

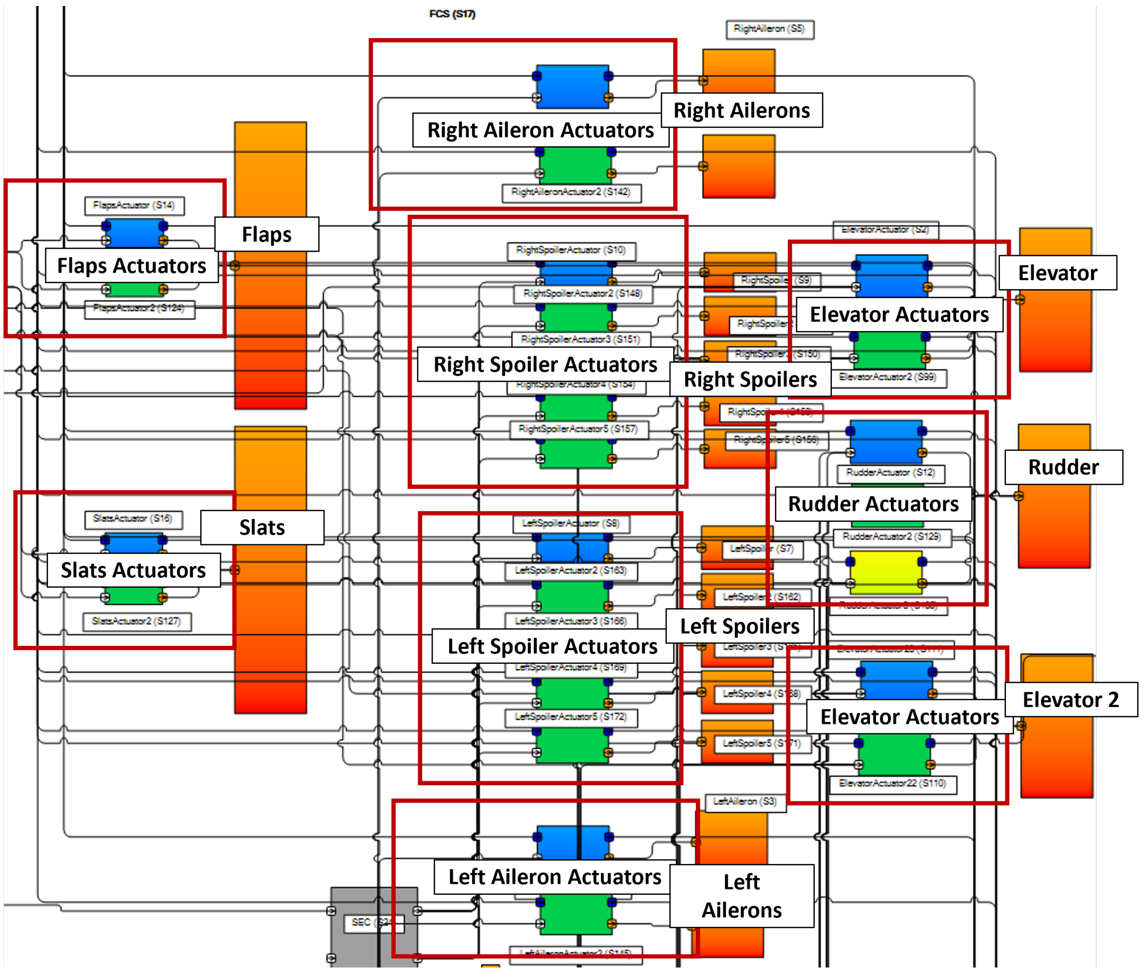


Figure 6.14: Application of physical redundancy to the FCS

Functional Redundancy Enabler

Using the architecture resulting from the inclusion of physical redundancy as a starting point, the enabler for architecting functional redundancy is applied to the use case. This enabler is also used to bring the architecture closer to a real one in terms of safety features. Although less abundant than physical redundancy — at least in the parts of the architecture that are considered, there are still many examples of functional redundancy in aircraft architectures. The test cases for the functional redundancy enabler are summarised in Table 6.6. Column 'Function (Component)' indicates the function given as input to the enabler and the component that originally provides such function enclosed in parenthesis. Column 'Added Components' reflect the components that form the new redundant leg, in order of addition.

Table 6.6: Functional redundancy cases

Subsystem	Function (Component)	Added Components
HYD	Stabilise Pressure (Accumulator)	Accumulator, Ram Air Turbine, Reservoir, Slats Actuator (from architecture)
	Supply Hydraulic (Electric Pump)	Electric Pump, Electric Generator
PNE	Supply Air (Precooler)	APU, Ram Air Inlet
ECS	Supply Fresh Air (Mixer Unit)	Ram Air inlet (from architecture)
FCS	Control Pitch (ELAC)	Elevator Trim Wheel (already implemented)
	Control Pitch (ELAC)	SEC (already implemented)
	Control Rudder (FAC)	Pedals (already implemented)

Functional redundancy was applied to the hydraulic system to create a new circuit. Furthermore, this third circuit is dissimilar to the already existing ones. For this purpose, the function 'Stabilise Pressure' from the hydraulic accumulator is selected as the starting point of the enabler for architecting functional redundancy. At this point, the algorithm for determining function-solution similarity (See

Section 5.3.2) is applied by the enabler. The library component that scores the highest is unsurprisingly the accumulator template (matches operation and flow), followed by other hydraulic components (match flow). Elements from the architecture are ranked analogously.

A new accumulator is instantiated from the library, the information contained in the template results in the derivation of function ‘Supply Hydraulic’, awaiting to be fulfilled. In this case, the highest template matches are the hydraulic, electric and ram air turbine pumps, followed by other hydraulic components. Since the goal is to make a dissimilar leg, the ram air turbine is selected, which derives the function ‘Supply Hydraulic’ once more. This time, the reservoir template is selected instead, which results in two additional derived functions, one per input port.

The pneumatic input induced the ‘Import Pneumatic’, which is fulfilled by one of the precoolers already present in the architecture’s pneumatic system. This component was amongst the highest-ranked in the list of solutions in architecture, and since it is already in the architecture does not induce additional functions. Finally, ‘Import Hydraulic’, which is the second function induced by the reservoir — as it has a hydraulic input port that requires hydraulic fluid — is fulfilled by selecting the slats actuator from the architecture to close the hydraulic circuit. This component also scored high in the ranking of solution similar to the function. Since the component comes from the architecture, no more functions are induced and consequently, all functions are fulfilled. The new leg is added to the architecture resulting in a hydraulic system as shown in Figure 6.15 — minus the electric pump and electric generator to be added next.

Functional redundancy was applied to the hydraulic system a second time. In this case, to provide an alternative way of fulfilling the function provided by the ram air turbine pump. The first solution added to the new leg is an electric pump instantiated using the electric pump template from the library. Analogously to the

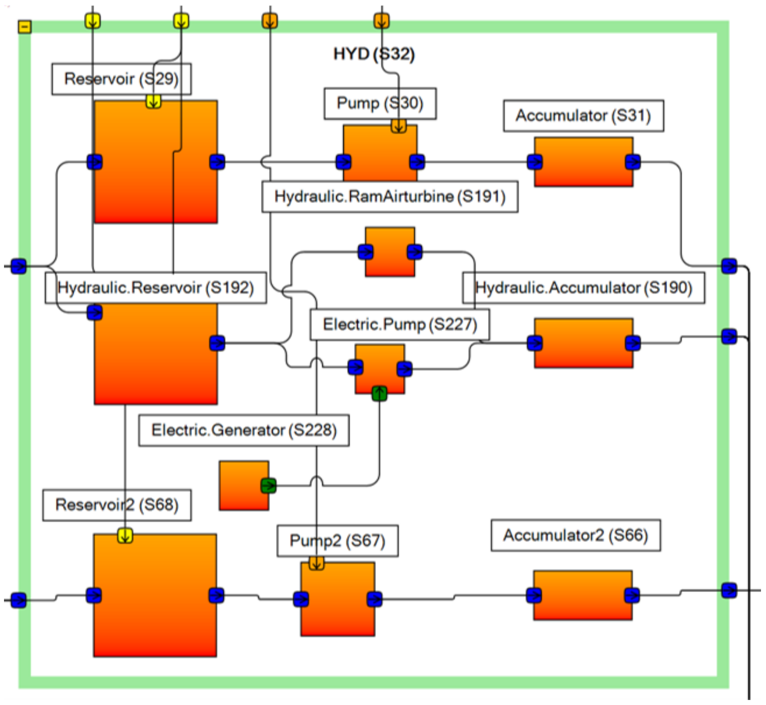


Figure 6.15: Application of functional redundancy to the HYD

first application of the functional redundancy enabler, the pump introduced two derived functions, one per input port. The electric input induced the ‘Import Electric’, which is fulfilled by adding the highest ranked solution from the library, an electric generator. This generator represents the electric system of the aircraft, which is out of scope in this use case and not modelled in detail. In a more comprehensive study, the corresponding component from the electrical system would be chosen instead. The electric generator did not induce additional functions, consequently, the only function left to be fulfilled is ‘Import Hydraulic’, induced by the pump’s hydraulic port. In this case, the function is fulfilled by the hydraulic reservoir belonging to the third circuit. Since no more functions require fulfilment, the leg is added to the architecture. The final hydraulic system is presented in Figure 6.15.

The function ‘Supply Air’, provided by the pneumatic system’s precoolers was used to start the functional redundancy workflow. In this case, amongst the highest ranked templated from the library, the auxiliary power unit APU was selected. This induced the function 'Import Air', whose fulfilment lead to selecting an additional ram air inlet from the library. The process resulted in the system shown in Figure 6.16. This ram air inlet was reused to provide the function Supply Fresh Air (fulfilled by the air ECS' mixer unit) in a redundant manner.

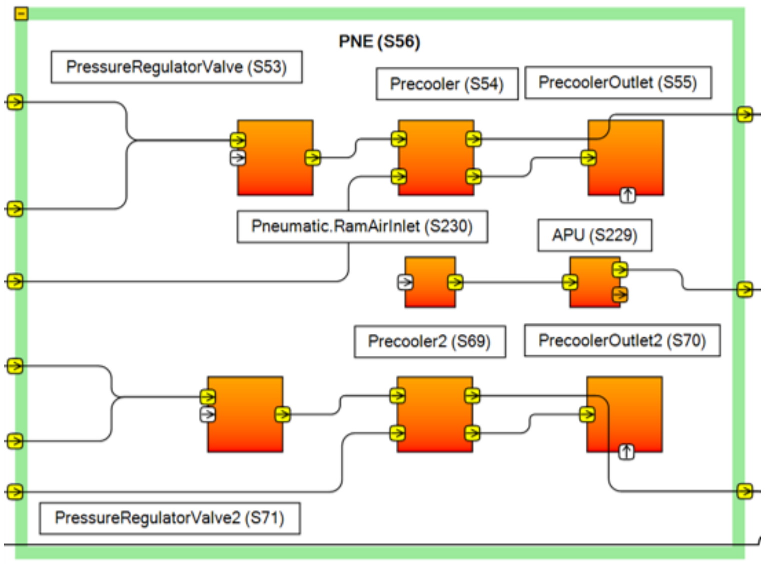


Figure 6.16: Application of functional redundancy to the PNE

The remaining identified cases of functional redundancy, as stated in Table 6.6, belong to the flight control system. And were already implemented in the initial architecture, as it was considered relevant for demonstrating the STPA support to the industrial evaluators. The algorithm for function-solution similarity was used in these test cases to check its consistency. As expected, the controllers in the architecture and library obtained the highest similarity values.

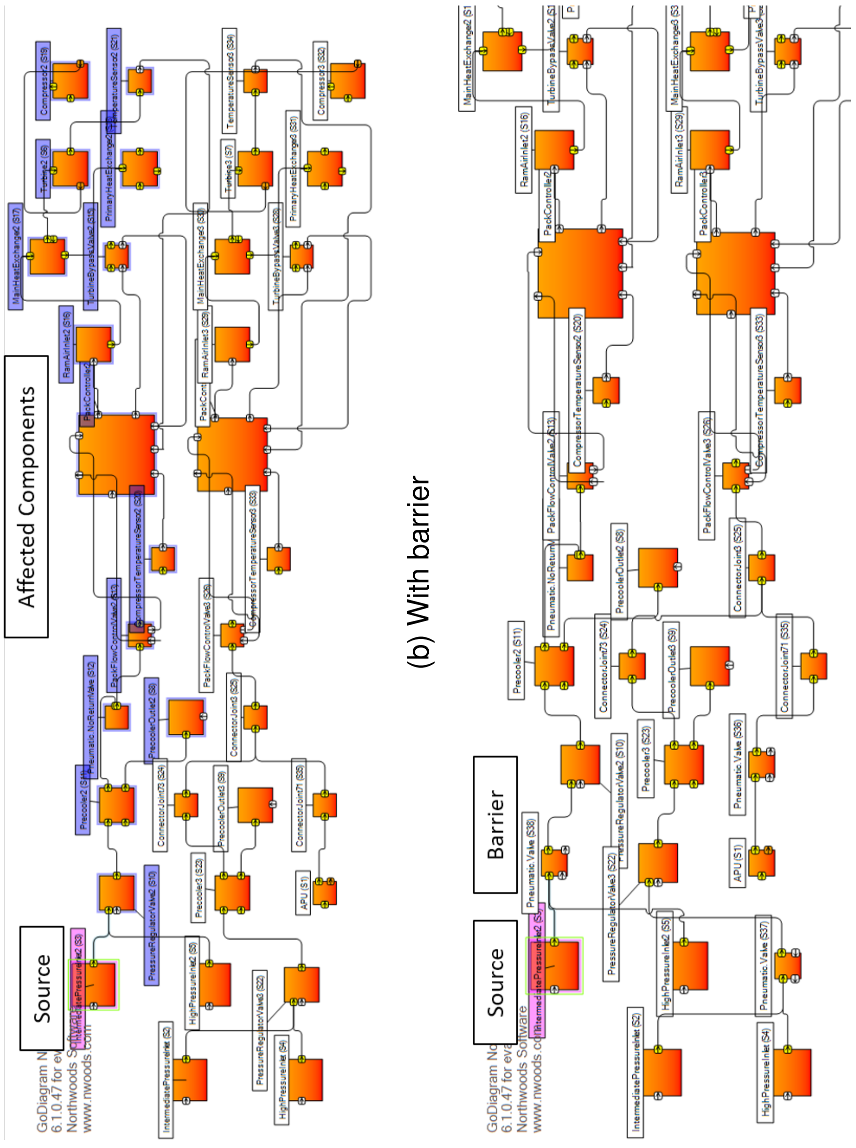
Containment Enabler

The final step regarding the architecting of safety principles consisted in using the containment enabler to support the addition of barrier components to protect those components that are susceptible to disturbances. The opportunities to apply this safety principle were more limited than those of the other two enablers. The main reason behind this limitation is not the low number of barriers in actual architecture, which is high, but the fact that many of them had already been added for demonstration or functional purposes other than safety. Table 6.7 summarises the application of the containment enabler to the architecture's subsystems. Column 'Sources' indicates the components that can generate the disturbances to be contained. Column 'Susceptible' display the components to be protected. The elements in both columns need to be selected manually. Column '$N_{barriers}$' indicates the number of barriers that were required to provide the desired containment characteristics.

Table 6.7: Containment cases

Case	**Sources**	**Susceptible**	$N_{barriers}$
Case 1	Engine components and APU	FCS actuators	6
Case 2	Engine components and APU	Precooler and ECS components	2

The containment enabler facilitates the exploration of the propagation of disturbances between their sources and the components to be protected. Figure 6.17a displays the propagation of a disturbance originating from the engine's intermediate pressure inlet when there is no barrier. It shows how the disturbance can propagate through the PNE, reaching the ECS components. Figure 6.17b present the same situation but with a barrier next to the source this time, which contains the disturbance protecting ECS components from it.



(a) Without barrier

(b) With barrier

Figure 6.17: Disturbance propagation exploration in AirCADia Architect

Apart from manual exploration, the enabler can also use a minimum cut algorithm (see Section 5.3.3) to determine automatically the minimum number of barriers required to isolate all sources and suggest locations for the barriers. The results provided by the algorithm for both cases are presented in Tables 6.8 and 6.9. The number of barriers suggested by the algorithm matches that of the actual system used for reference, except in the case of the two barriers between the engine shaft power and the HYD pumps, which are not specified in the original reference. But on the contrary, the suggested locations for the barriers differ. This does not indicate that the results from the algorithm are incorrect, as there might be several minimum cuts of identic value and the algorithm only selects one. However, it indicates that the usefulness of the algorithm could be improved either by providing more detailed information expressing the preference of some locations over the others (assigning different costs to the edges) or by modifying it to compute more than one result when possible.

Table 6.8: Min-cut algorithm results for Case 1

Connection	**Kind**
Original shaft power → Original HYD pump	Rotational
Redundant shaft power → Redundant HYD pump	Rotational
Original hydraulic system leg → Original FCS actuators	Hydraulic
Redundant hydraulic system → Redundant FCS actuators	Hydraulic
Original HYD pump → Original Accumulator	Hydraulic
Redundant HYD pump → Redundant Accumulator2	Hydraulic

Discussion

The enablers were able to reproduce most of the safety features from the architecture of a real aircraft at the level of detail employed in this research. They made it possible to transform the initial architecture, lacking most safety measures

Table 6.9: Min-cut algorithm results for Case 2

Connection	**Kind**
Original intermediate and high pressure inlets → Original Precooler Pneumatic	
Redundant intermediate and high pressure inlets → Redundant Precooler	Pneumatic
APU → Pack flow control valve	Pneumatic

to a much safer architecture approaching realistic standards. More importantly, they were able to do so by automating many repetitive aspects and ensuring consistency at every step of the process. Additionally, the enablers gather from the RFLP definition the relevant information when implementing each of the safety features in the evaluation architecture, helping architects to understand the effect of their decision in the architecture, or what additional considerations need to be accounted for when making the architecture safer.

The methods also shown some limitations as some safety-related components, such as the cross-feed valve in the pneumatic system or the power transfer unit located between two hydraulic circuits, could not be modelled appropriately by any of the enablers and had to be added manually*.

6.4.3 Fault Tree Analysis Support

The support for automating the creation of fault trees from the architecture definition is evaluated by applying it to the A320 inspired system architecture. Fault trees are created using the developed support at different stages of implementation of physical redundancy and then results are checked to ensure that they are in line with the hypothesis used by the fault tree creation algorithm (see Section 5.4.1) and the presence of more or less redundancy in the architecture.

*It might be possible to reuse most parts of the functional redundancy enabler to support a more generic functional-logical zigzagging and therefore support this scenario as well. However, this is out of the scope of this thesis and it was not tested

Based on the application of the physical redundancy enabler in Section 6.4.2, the four stages of redundancy included in the FTA methods evaluation are:

1. **Initial architecture:** the architecture used at this stage has not been modified to make it more redundant.

2. **Second engine:** the architecture is modified to take into account the existence of two engines.

3. **Redundant engine + pneumatics:** a second redundant channel is added to the hydraulic system.

4. **Redundant engine + pneumatics + air pack:** a new redundant leg containing the air conditioning pack and adjacent ECS components is created,

The FTA support automatically creates a fault tree starting from a selected component. The tree intends to model the inability of the selected component to fulfil its function. The chosen component in this evaluation is the ECS's mixer unit that delivers air to the cabin. The tree is presented to the user as depicted in Figure 6.18 and analysed both qualitatively and quantitatively. The quantitative results consist of the minimal cut sets and minimal path sets. Each series of sets is also presented in its filtered version, where only one representative set is kept for each group of symmetric sets. Symmetric sets have the same number and type of components (common solution template) element-wise, and these components belong to redundant legs providing the same function. Quantitative results include the probability of failure of the top-event (the selected solution fails to provide its function), the relative probability of failure of each set with respect to the probability of failure of the top-event, and two rankings of solutions according to the Fussell-Vesely and Birnbaum importance measures respectively. The probabilities of failure assigned to the individual components have not been determined rigorously but they allow to demonstrate the methods.

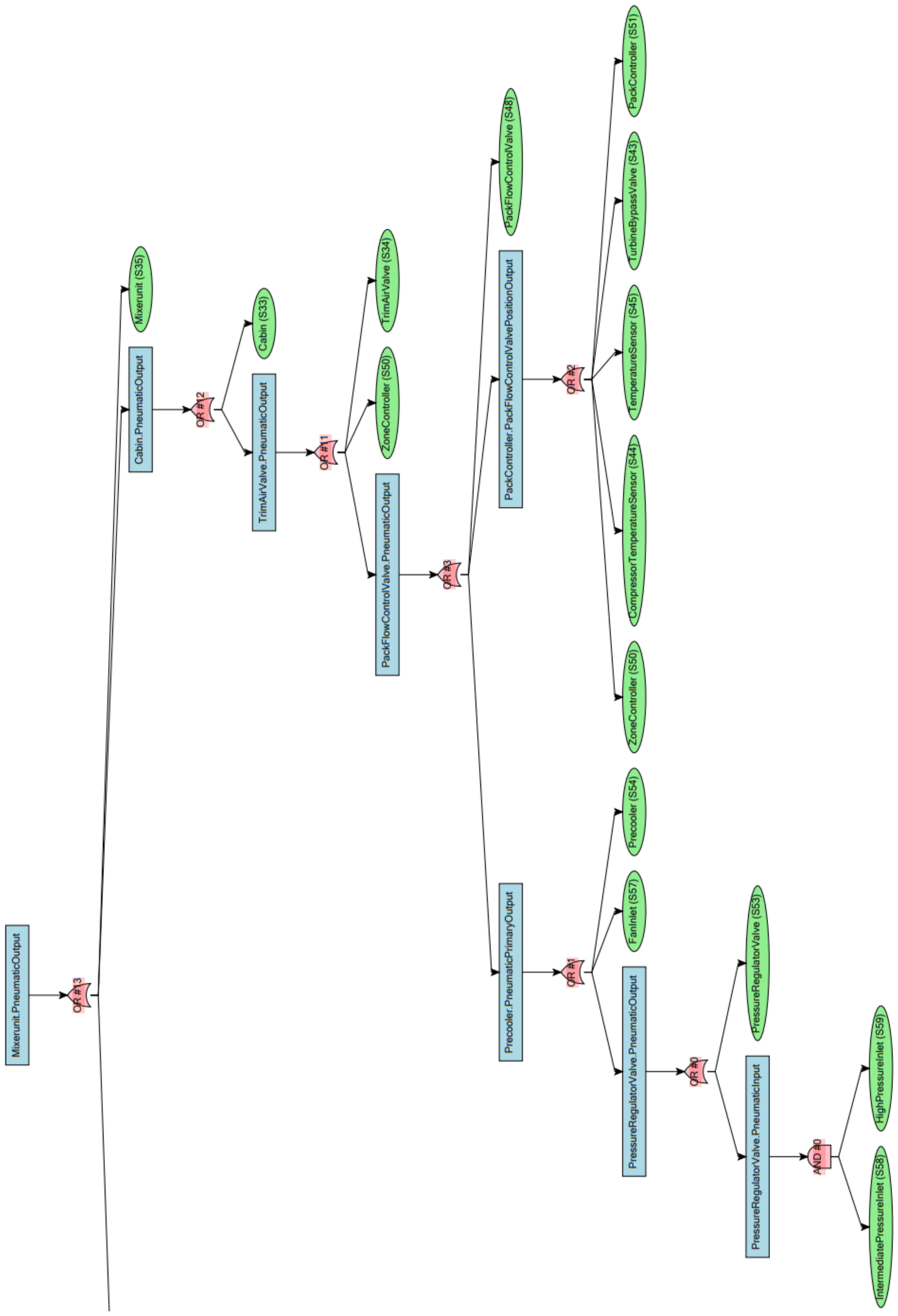

Mixerunit.PneumaticOutput
OR #13
Mixerunit (S35)
Cabin.PneumaticOutput
OR #12
Cabin (S33)
TrimAirValve.PneumaticOutput
OR #11
TrimAirValve (S34)
ZoneController (S50)
PackFlowControlValve.PneumaticOutput
OR #3
PackFlowControlValve (S48)
PackController.PackFlowControlValvePositionOutput
OR #2
PackController (S51)
TurbineBypassValve (S43)
TemperatureSensor (S45)
CompressorTemperatureSensor (S44)
ZoneController (S50)
Precooler.PneumaticPrimaryOutput
OR #1
Precooler (S54)
FanInlet (S57)
PressureRegulatorValve.PneumaticOutput
OR #0
PressureRegulatorValve (S53)
PressureRegulatorValve.PneumaticInput
AND #0
HighPressureInlet (S59)
IntermediatePressureInlet (S58)

Figure 6.18: Detail of the fault tree created for the initial architecture

The reports generated by AirCADia Architect with the FTA results can be found in Appendix B; in this section, only a summary with the most relevant results will be presented. Table 6.10 shows the probability of failure for the trees corresponding to the four stages of redundancy. As expected, the probability decreases monotonically as redundancy increases. However, the contribution of each stage is of different orders of magnitude. The largest impact is that of the third stage, where redundancy is added to the PNE.

Table 6.10: Probability of failure for the four stages of redundancy

Stage	**Probability of failure** P_{top}
Initial	$1.011 \cdot 10^{-006}$
Engine	$1.010 \cdot 10^{-006}$
Pneumatics	$9.001 \cdot 10^{-009}$
ECS	$9.000 \cdot 10^{-009}$

A closer look at the cut sets brings more insight into the variation of the results with the various degrees of physical redundancy. Table 6.11 shows the minimal cut sets for the tree created for the initial architecture. The lack of redundancy causes mosts cut sets to have only one element, except for the last cut. This set, composed of the two engine core inlets, reflect the only redundancy that affects the mixer unit. The information about the relative probability of the cut set P_{set} with respect to the total probability of failure P_{top} indicates that the PNE's precooler is the main responsible component.

Table 6.12 shows the minimal cut sets for the tree created after modelling the redundancy representing the second engine. The effect of redundancy can be seen in the two last cut sets, which have grown from one and two components to two and four components respectively. However, since the probability of failure of the individual components is low, their contribution to P_{top} small and its value is only slightly improved.

Table 6.11: Minimal cut sets for Stage 1

Components	P_{set}/P_{top}
Mixer Unit	$9.891 \cdot 10^{-004}$
Cabin	$9.891 \cdot 10^{-004}$
ZoneController	$9.891 \cdot 10^{-004}$
TrimAirValve	$9.891 \cdot 10^{-004}$
PackFlowControlValve	$9.891 \cdot 10^{-004}$
CompressorTemperatureSensor	$9.891 \cdot 10^{-004}$
TemperatureSensor	$9.891 \cdot 10^{-004}$
TurbineBypassValve	$9.891 \cdot 10^{-004}$
PackController	$9.891 \cdot 10^{-004}$
FanInlet	$9.891 \cdot 10^{-004}$
Precooler	$9.891 \cdot 10^{-001}$
PressureRegulatorValve	$9.891 \cdot 10^{-004}$
IntermediatePressureInlet, HighPressureInlet	$9.891 \cdot 10^{-013}$

The next stage of redundancy consists in the addition of a new leg in the pneumatic system. The FTA algorithm is then applied to the architecture obtaining the minimal cut sets shown in Table 6.13. The single-component cut sets containing the precooler and the pressure regulator valve are replaced by four sets of two components each. The sets represent the four possible combinations of choosing one component from the first leg (precooler or valve) and one from the second leg (precooler or valve). Since this time two precoolers need to fail at the simultaneously to provoke the failure, P_{top} is vastly reduced. The more influential components are now those with a low probability of failure.

The last stage of redundancy is reached after duplicating the air pack, pack flow control valve and related controllers in the environmental control systems. This time the number of minimal cut sets has more than doubled. Every single

¶The zero-valued probability is due to the precision of the C# decimal numerical type used in AirCADia Architect

Table 6.12: Minimal cut sets for Stage 2

Components	P_{set}/P_{top}
Mixer Unit	$9.901 \cdot 10^{-004}$
Cabin	$9.901 \cdot 10^{-004}$
ZoneController	$9.901 \cdot 10^{-004}$
TrimAirValve	$9.901 \cdot 10^{-004}$
PackFlowControlValve	$9.901 \cdot 10^{-004}$
CompressorTemperatureSensor	$9.901 \cdot 10^{-004}$
TemperatureSensor	$9.901 \cdot 10^{-004}$
TurbineBypassValve	$9.901 \cdot 10^{-004}$
PackController	$9.901 \cdot 10^{-004}$
Precooler	$9.901 \cdot 10^{-001}$
PressureRegulatorValve	$9.901 \cdot 10^{-004}$
FanInlet, FanInlet2	$9.901 \cdot 10^{-013}$
IntermediatePressureInlet, IntermediatePressureInlet2, HighPressureInlet, HighPressureInlet2	0.000¶

Table 6.13: Minimal cut sets for Stage 3

Components	P_{set}/P_{top}
Mixer Unit	$1.111 \cdot 10^{-001}$
Cabin	$1.111 \cdot 10^{-001}$
ZoneController	$1.111 \cdot 10^{-001}$
TrimAirValve	$1.111 \cdot 10^{-001}$
PackFlowControlValve	$1.111 \cdot 10^{-001}$
CompressorTemperatureSensor	$1.111 \cdot 10^{-001}$
TemperatureSensor	$1.111 \cdot 10^{-001}$
TurbineBypassValve	$1.111 \cdot 10^{-001}$
PackController	$1.111 \cdot 10^{-001}$
FanInlet, FanInlet2	$1.111 \cdot 10^{-010}$
Precooler2, Precooler	$1.111 \cdot 10^{-004}$
PressureRegulatorValve2, Precooler	$1.111 \cdot 10^{-007}$
Precooler2, PressureRegulatorValve	$1.111 \cdot 10^{-007}$
PressureRegulatorValve2, PressureRegulatorValve	$1.111 \cdot 10^{-010}$
IntermediatePressureInlet, IntermediatePressureInlet2, HighPressureInlet, HighPressureInlet2	$0.000 \cdot 10^{+000}$

one-component set has been replaced by one or more larger sets except for the mixer unit and the cabin. These two components are now the greatest contributors to $\mathrm{P}_{\mathrm{top}}$, which has been reduced considerably but not as much as in the previous stage.

Discussion

The FTA support was able to generate fault trees from the architecture definition with very little additional user's input. Providing the component that performs the function whose failure represents the tree's top event is enough for the algorithm to construct the tree. Qualitative and quantitative evaluation of the fault trees confirmed the ability of the algorithm to generate fault trees that reflect trends in the probability of failure caused by an increasing degree of redundancy. It was also possible to differentiate between the effect of making redundant major or minor contributors to the total probability. Safety experts will likely have to review and possibly extend or correct parts of the trees as the design evolves but the method presented in this thesis enable architects to obtain quick feedback regarding their architectural decision without requiring inputs, which has the potential of speeding up the architecting process.

Table 6.14: Minimal cut sets for Stage 4

Components	P_{set}/P_{top}
Cabin	$4.990 \cdot 10^{-001}$
Mixer Unit	$4.990 \cdot 10^{-001}$
RamAirInlet, RamAirInlet2	$4.990 \cdot 10^{-010}$
RamAirInlet2, MainHeatExchanger	$4.990 \cdot 10^{-007}$
RamAirInlet, MainHeatExchanger2	$4.990 \cdot 10^{-007}$
MainHeatExchanger2, MainHeatExchanger	$4.990 \cdot 10^{-004}$
MainHeatExchanger2, PrimaryHeatExchanger, Turbine	$4.990 \cdot 10^{-010}$
MainHeatExchanger2, TurbineBypassValve, Turbine	$4.990 \cdot 10^{-013}$
MainHeatExchanger2, PrimaryHeatExchanger, CJ2	$4.990 \cdot 10^{-004}$
PrimaryHeatExchanger2, Turbine2, MainHeatExchanger	$4.990 \cdot 10^{-010}$
PrimaryHeatExchanger2, CJ3, MainHeatExchanger	$4.990 \cdot 10^{-004}$
RamAirInlet, PrimaryHeatExchanger2, Turbine2	$4.990 \cdot 10^{-013}$
RamAirInlet, PrimaryHeatExchanger2, CJ3	$4.990 \cdot 10^{-007}$
MainHeatExchanger2, TurbineBypassValve, CJ2	$4.990 \cdot 10^{-007}$
TurbineBypassValve2, CJ3, MainHeatExchanger	$4.990 \cdot 10^{-007}$
RamAirInlet, TurbineBypassValve2, Turbine2	$4.990 \cdot 10^{-016}$
RamAirInlet, TurbineBypassValve2, CJ3	$4.990 \cdot 10^{-010}$
RamAirInlet2, PrimaryHeatExchanger, Turbine	$4.990 \cdot 10^{-013}$
RamAirInlet2, TurbineBypassValve, Turbine	$4.990 \cdot 10^{-016}$
RamAirInlet2, PrimaryHeatExchanger, CJ2	$4.990 \cdot 10^{-007}$
RamAirInlet2, TurbineBypassValve, CJ2	$4.990 \cdot 10^{-010}$
TurbineBypassValve2, Turbine2, MainHeatExchanger	$4.990 \cdot 10^{-013}$
And 16 more sets of four components each	$\leq 4.990 \cdot 10^{-004}$

6.4.4 Architecture Sizing And Performance Support

The support for sizing and determining the performance of the architecture is evaluated by applying it to the A320 inspired system architecture. Sizing workflows are created for the flight control system and hydraulic system. The proposed methods allow for the consideration of different configurations such as those created by failed components or determined by other design considerations. The configurations used for this evaluation, which were selected to enable demonstration of the novel aspects of the sizing support, are:

- **All Active:** in this configuration all components are active and none of them has failed.
- **FAC Fail:** similar to 'All Active' but the flight augmentation computer in the flight control system has failed.
- **Left Aileron Actuator Fail:** the actuator of the left aileron has failed but the rest of the components are available.

Similarly, it is also possible to consider several environmental regions when sizing the system. The regions represent the allowed values of variables that model the context of the aircraft. In particular, three regions are considered for this evaluation, namely 'Whole Envelope', 'Landing' and 'Cruise'. The 'Whole Envelope' is chosen as the default region, which is used whenever no other region is explicitly linked to a scenario. The environmental variables considered and their lower and upper bounds are shown in Table 6.15. 'Landing' and 'Cruise' regions override the upper and lower bounds respectively of some of the environmental variables as indicated in Table 6.15

The last step before workflows can be created is to map one or more regions to each configuration. By default, configurations are mapped to the default region, which in this case is the 'Whole Envelope' region. However, for evaluation pur-

Table 6.15: Bounds of environmental variables

Variable	**Lower Bound**	**Upper Bound**
	Whole Envelope	
Altitude (m)	0	12000
Velocity (m/s)	0	250
Dynamic Pressure (Pa)	0	1000000
Mach	0	0.9
Angle of Attack (rad)	-1	1
Elevator Deflection (rad)	-1	1
	Landing	
Altitude	0	3000
Velocity	0	125
Dynamic Pressure	0	350000
Mach	0	0.4
	Cruise	
Altitude	3000	12000
Velocity	125	250
Dynamic Pressure	350000	1000000
Mach	0.4	0.9

poses, the ‘FAC Fail’ configuration is mapped exclusively to the ‘Cruise’ region, and the ‘Left Aileron Actuator Fail’ mapped exclusively to the ‘Landing’ region.

A workflow needs to be created for each combination of subsystem and configuration. To reduce the number of required workflows, the method ‘projects’ each configuration into each of the subsystems. Since all configurations are the same from the HYD’s perspective, there is no need to create three different workflows and only one is enough. The situation is different for the FCS as both scenarios affect it. Thus, the number of required workflows is three.

The creation of the workflow for the hydraulic system is simple because it does not depend on any environmental variables and the algorithm is capable of determining a schedule for the workflow without user input. However, after examining the result it was discovered that a better workflow with fewer reversed models could be obtained by indicating to the algorithm that the volumetric flow at the HYD’s hydraulic output is a known variable. An excerpt of the workflow as presented by AirCADia Architect can be seen in Figure 6.19. The total number of reversed models is one, and the model corresponds to a connection model.

The creation of workflow for the flight control system is somewhat more involved as the system depends on environment variables (the actuators interact with the environment) requiring that the variables belonging to the components are mapped to the corresponding environmental variables. The mapping process was required only once as the mapping can be reused for all the workflows. Unlike the first time, the workflows provided by the algorithm using default inputs were satisfactory, not requiring additional action for any of the three workflows. Ten connection models required to be reversed to schedule the workflow.

Discussion

The sizing support was able to handle the fast creation of computational workflows for the HYD and FCS subsystems requiring only a small amount of user

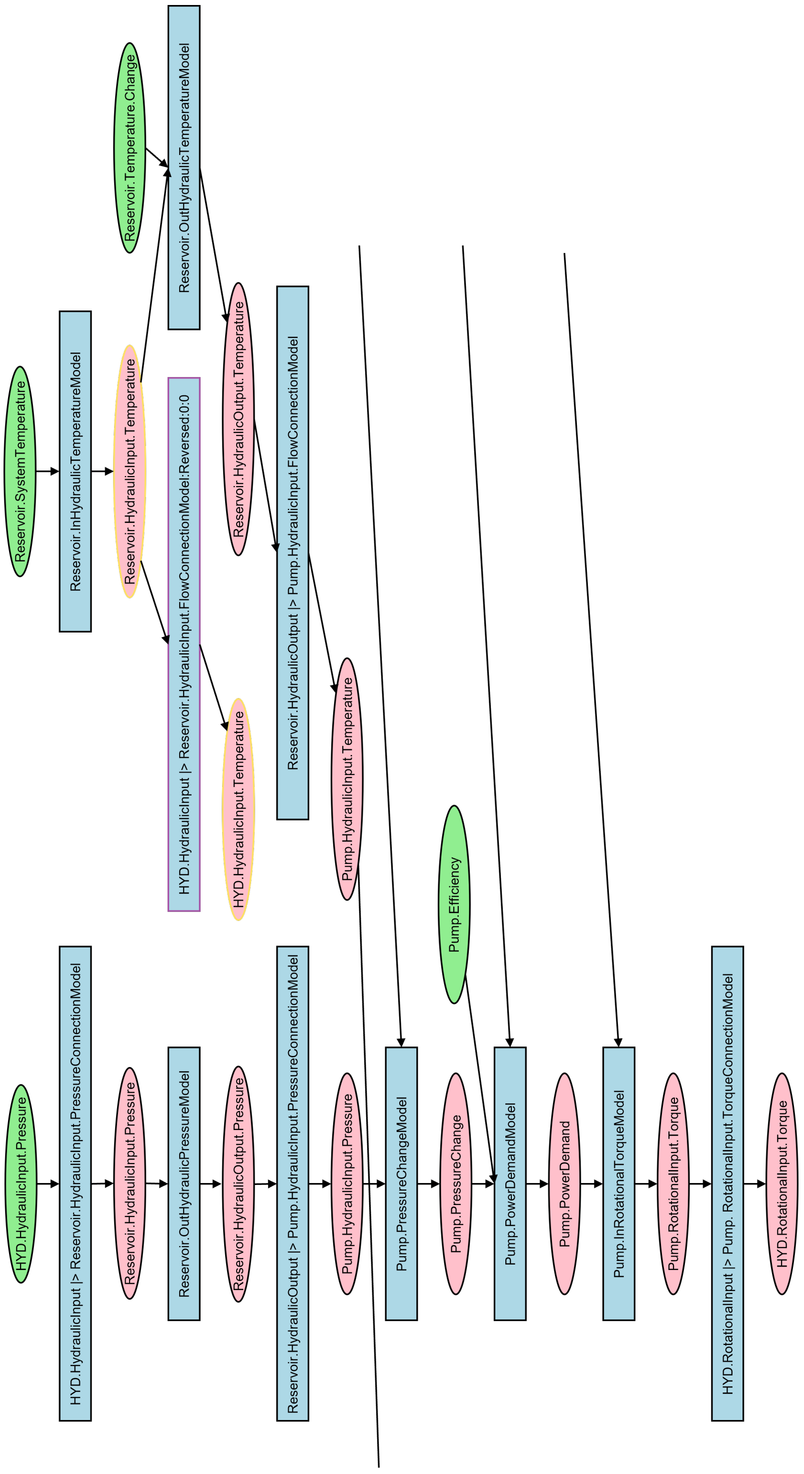

Reservoir.Temperature.Change
Reservoir.OutHydraulicTemperatureModel
Reservoir.SystemTemperature
Reservoir.InHydraulicTemperatureModel
Reservoir.HydraulicInput.Temperature
HYD.HydraulicInput |> Reservoir.HydraulicInput.FlowConnectionModel:Reversed:0:0
Reservoir.HydraulicOutput.Temperature
HYD.HydraulicInput.Temperature
Reservoir.HydraulicOutput |> Pump.HydraulicInput.FlowConnectionModel
Pump.HydraulicInput.Temperature
Pump.Efficiency
HYD.HydraulicInput.Pressure
HYD.HydraulicInput |> Reservoir.HydraulicInput.PressureConnectionModel
Reservoir.HydraulicInput.Pressure
Reservoir.OutHydraulicPressureModel
Reservoir.HydraulicOutput.Pressure
Reservoir.HydraulicOutput |> Pump.HydraulicInput.PressureConnectionModel
Pump.HydraulicInput.Pressure
Pump.PressureChangeModel
Pump.PressureChange
Pump.PowerDemandModel
Pump.PowerDemand
Pump.InRotationalTorqueModel
Pump.RotationalInput.Torque
HYD.RotationalInput |> Pump. RotationalInput.TorqueConnectionModel
HYD.RotationalInput.Torque

Figure 6.19: excerpt of HYD workflow

input, and taking into account the interaction of the system with the environment and several configurations that model component failures. The quality of the provided workflows is high as the number of reversed models is low compared with the total number of models in each of the workflows. Furthermore, the reversed models are connection models, which are generally easier to reverse and their numerical treatment, when required, is more robust. The presented support enables architects to accelerate the process of obtaining feedback regarding the impact on performance of different architectures, without the need of involving simulation experts at every step.

6.5 Industrial evaluation

6.5.1 Purpose

The final part of the evaluation consisted in an industrial evaluation. The industrial evaluation is only a first attempt to the success evaluation of the support (see Table 6.1) as it was not possible to test the research in a real industrial project. A group composed of four specialists from Airbus and one from Cranfield University participated in the evaluation session. The main purpose of the session was to obtain feedback regarding:

- The usefulness and industrial relevance of the proposed methods.
- The availability of the information required by the methods during conceptual design.
- Their ability to reduce the number or duration of time-consuming activities and increase interactivity.
- How to improve the techniques in the future to make them more useful and relevant.

6.5.2 Approach

A feedback questionnaire combining closed-form Likert-type questions with open-ended questions was created. The Likert questions were divided into groups that correspond to each of the methods plus one additional group of questions about the overall framework. The open questions referred to the overall framework as well. An online tool called Microsoft Forms [152] was used to elaborate the questionnaire, distribute it during the evaluation session and collect the responses from the participants. A printed version of this questionnaire can be viewed in Appendix C.

The evaluation session took place on the 9^{th} of December 2020 and it was done via an online videoconference. The session was structured as follows:

1. Introductory presentation (30 min). The presentation served to introduce the participants to the research, the use case (A320 inspired systems architecture) and AirCADia Architect; combined with a live demonstration of the application of the methods to the use case using AirCADia Architect. Due to temporal limitations, the demonstration only included one example for each of the methods developed in the research.

2. Questions and discussion (30 min). At this point, the participants were given some time to ask questions about the research and discuss the presented methods.

3. Questionnaire (15 min). The questionnaire was distributed and explained to the participants, including their right to withdraw and how to do so. The respondents were given up to seven days to complete the questionnaire, modify their answers or withdraw.

Several days prior to the session the participants were given a brief report introducing the research and its background as well as a copy of the consents form describing their consent to participate, ability to withdraw, and handling of the collected data (see Appendix D).

The information of the five participants is summarised in Table 6.16. Five participants may seem a small number, but it should be noted that there appears to be no specific rules on how to determine appropriate sample size in qualitative research. As Patton [153, p. 184] suggest, the sample size depends on the available resources, time allotted and purpose of the research.

Table 6.16: Participant information

	Workplace	**Years of Experience**
Participant 1	Cranfield University	5
Participant 2	Airbus	25
Participant 3	Airbus	35
Participant 4	Airbus	30
Participant 5	Airbus	15

6.5.3 Results and Discussion

Below are presented the aggregated responses for each of the groups of Likert questions. The valid responses for any of the questions are ‘Strongly Disagree’, ‘Disagree’, ‘Neutral’, ‘Agree’ and ‘Strongly Agree’. In the figures presenting the results, ‘Strongly Disagree’ and ‘Strongly Agree’ have been abbreviated to ‘S. Disagree’ and ‘S. Agree’ respectively; the vertical axes represent the number of participants that selected a particular category. A different colour has been used for each participant. The assignment of colour is consistent throughout all the figures, e.g. the red answer correspond always to the same participant. No information regarding which colour represent which participant is given because the questionnaire is anonymous. The question numbers are in accordance with those in the original form. This is the reason the first group of questions starts at number six, as the first five questions refer to personal data.

STPA Support

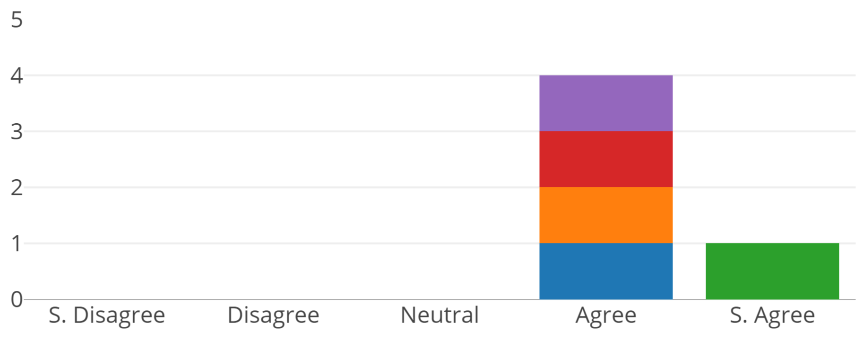
This group of questions is focused on the STPA enabler, consisting of the methods for automatically creating hierarchical control structures and detailed control loops. It is composed of three questions reproduced in Table 6.17. The answers to these questions are summarised in Figure 6.20.

Table 6.17: Likert questions related to STPA

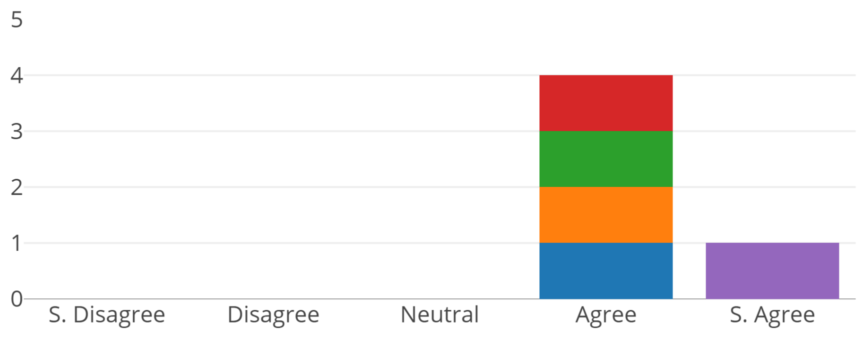
Question Number	**Please indicate to what extent you agree or disagree with the following statements**
Question 6.1	The proposed enablers enhance the interactivity of the safety assessment process.
Question 6.2	The proposed enablers result in a tighter integration between the architecture definition (in particular, the logical view) and the safety assessment models.
Question 6.3	The approach of STPA combined with RFLP satisfy the industrial need for safety assessment capabilities.

Results for questions 6.1 and 6.2 show a high degree of agreement (four ‘Agree’ and one ‘Strongly Agree’). This supports the claim that the proposed STPA enabler can increase interactivity and integration with the architecture definition during the safety assessment process. Responses for question 6.3 range between ‘Disagree’ and ‘Agree’, which indicates that STPA support can satisfy a relevant number of capabilities needed by industry but not all of them, which signals the existence of room for improvement regarding this area.

(a) Question 6.1



(b) Question 6.2



(c) Question 6.3

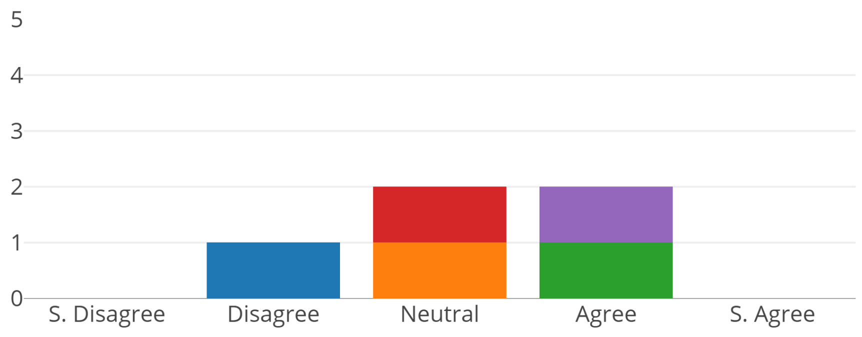


Figure 6.20: Answers to Likert questions related to STPA

Safety Architecting Enablers

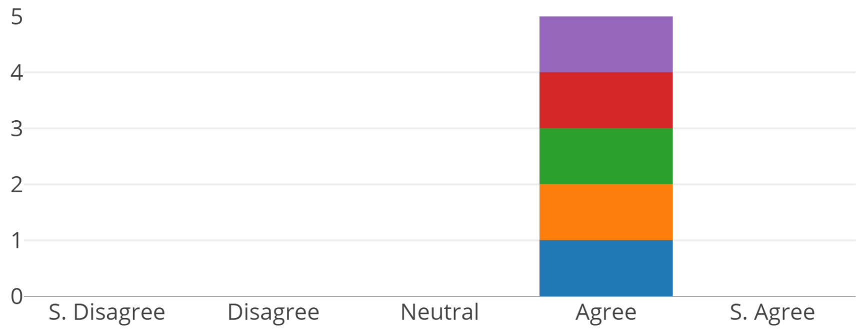
This group of questions focuses on the three enablers for architecting safety principles, namely physical redundancy, functional redundancy and containment. The three questions displayed in Table 6.18 represent this group. The responses to the questions are presented in Figure 6.21.

Results for questions 7.1 shows unanimous agreement (five ‘Agree’), supporting the claim that the proposed enabler can increase the interactivity of architecting the safety principles stated above. Responses to question 7.2 show more variety but also show a high degree of agreement, indicating the potential of the methods to reduce the number of time-consuming manual activities required. The answers to question 7.3 show a much lower level of agreement; the majority are ‘Neutral’ and even one is ‘Strongly Disagree’. This indicates that the supported safety principles can satisfy only a part of industrial requirements but there are still many more principles to consider.

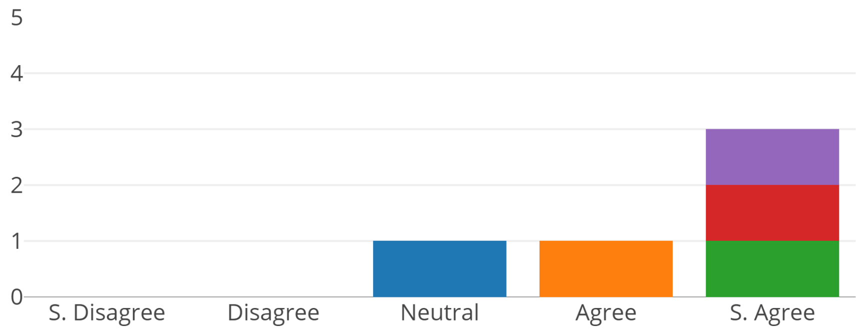
Table 6.18: Likert questions related to Safety architecting enablers

Question Number	**Please indicate to what extent you agree or disagree with the following statements**
Question 7.1	The proposed enablers enhance interactivity while architecting physical and functional redundancy, and containment.
Question 7.2	The proposed enablers reduce the number of time-consuming manual activities required for architecting physical and functional redundancy, and containment.
Question 7.3	Physical and functional redundancy, and containment cover to a sufficient extent the industrial requirement for safety principles.

(a) Question 7.1



(b) Question 7.2



(c) Question 7.3

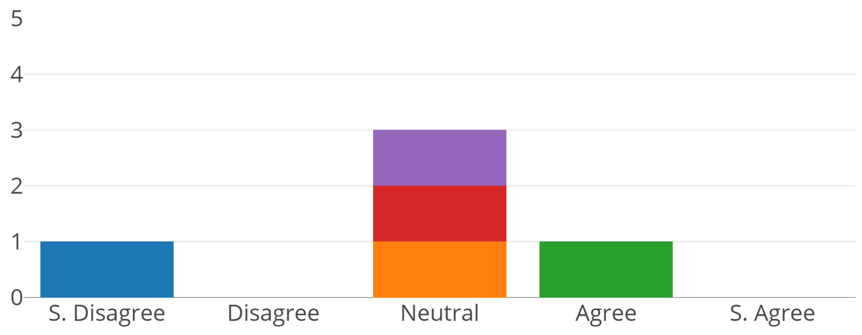


Figure 6.21: Answers to Likert questions related to Safety architecting enablers

Fault Tree Analysis Support

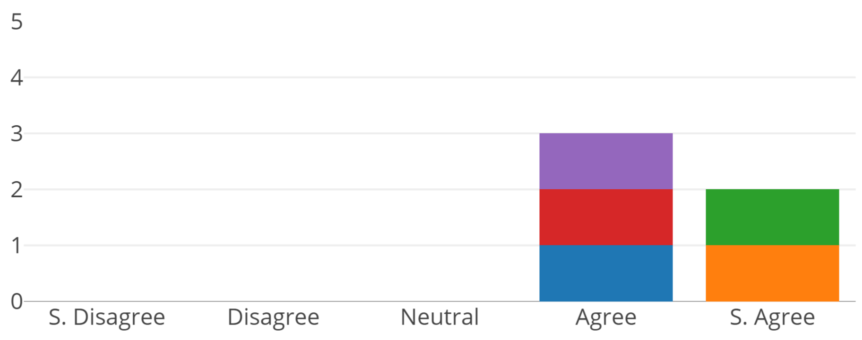
This group of questions is centred around the methods for Fault Tree Analysis. It consists of the three questions presented in Table 6.19. The results gathered from the participants are shown in Figure 6.22.

Table 6.19: Likert questions related to Fault Tree Analysis

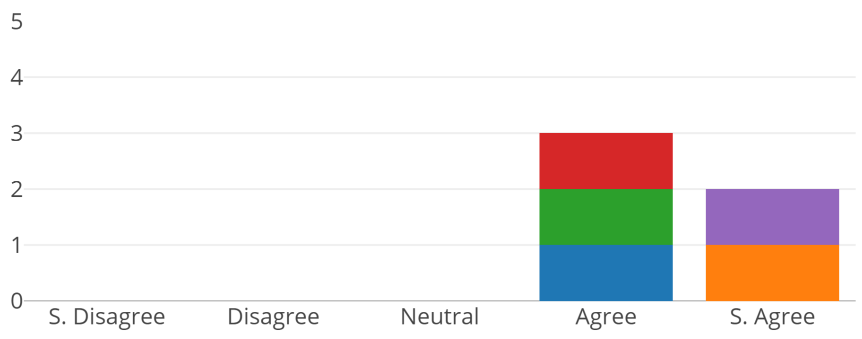
Question Number	**Please indicate to what extent you agree or disagree with the following statements**
Question 8.1	The proposed enablers reduce the number of time-consuming manual activities for FTA creation.
Question 8.2	The proposed enablers enhance interactivity of the exploration of FTA results.
Question 8.3	The level of detail of the FTA created by this technique is commensurate with conceptual design.

Answers for questions 7.1, 7.2 and 7.3 are similar (majority of ‘Agree’ and some ‘Strongly Agree’), shows a high degree of agreement. These results supporting the claims that the proposed FTA enabler reduces the number of time-consuming manual activities required for FTA creation, enhances interactivity while exploring FTA results, and operates at a level of detail appropriate for conceptual design.

(a) Question 8.1



(b) Question 8.2



(c) Question 8.3

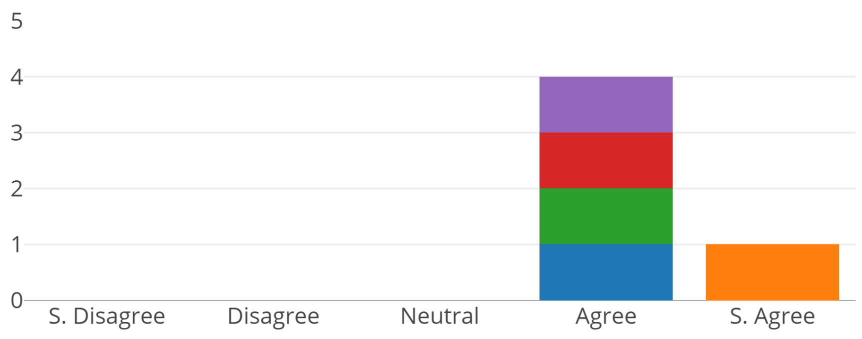


Figure 6.22: Answers to Likert questions related to Fault Tree Analysis

Sizing Support

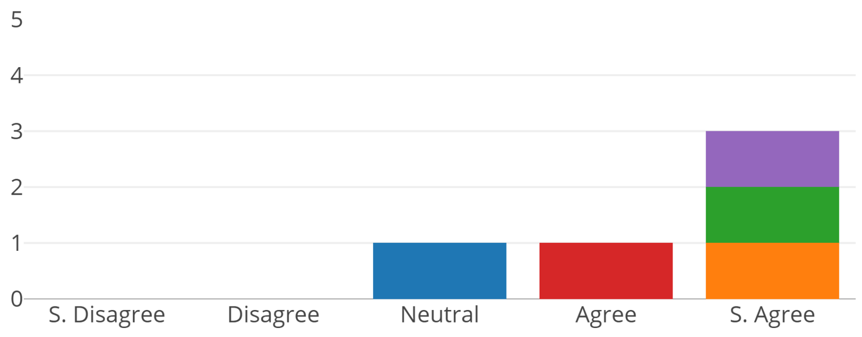
This group of questions refers to the proposed Sizing support. Table 6.20 contains the two questions that comprise this group. The answers to these questions are summarised in Figure 6.23.

Table 6.20: Likert questions related to Sizing

Question Number	**Please indicate to what extent you agree or disagree with the following statements**
Question 9.1	The proposed enablers reduce the number of time-consuming manual activities for sizing workflow creation.
Question 9.2	The level of detail regarding context definition (configuration/scenarios and environmental conditions) is appropriate for conceptual design.

Responses to question 9.1 range from ‘Neutral’ to ‘Strongly Agree’, which is the most chosen value. This indicates the capability of the proposed enabler to reduce the number of time-consuming activities when creating computational workflows used for sizing. Results to question 9.2 are distributed around ‘Agree’, supporting the claim the level of detail about configurations, scenarios and environmental conditions is appropriate for conceptual design.

(a) Question 9.1



(b) Question 9.2

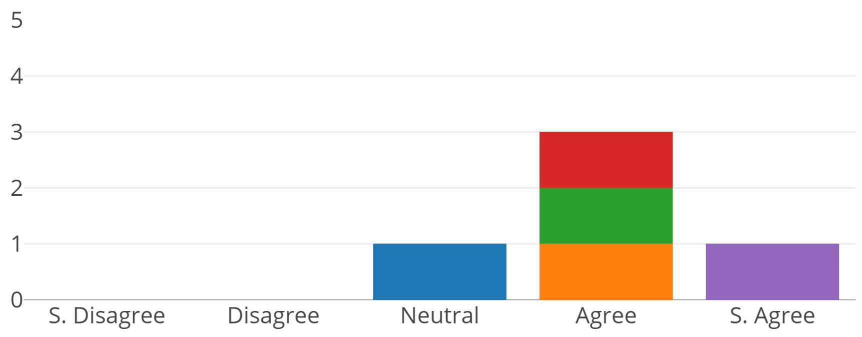


Figure 6.23: Answers to Likert questions related to Sizing

Overall Framework

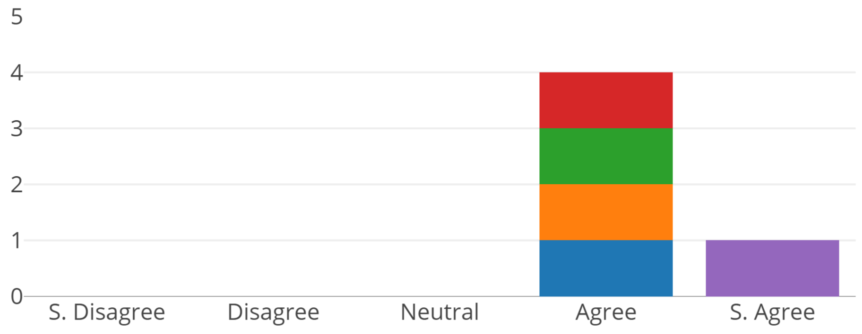
The last group of Likert questions refers to the overall framework consisting of all the proposed methods. The three questions belonging to this group are shown in Table 6.21. The responses to these questions displayed in Figure 6.24.

Table 6.21: Likert questions related to Overall framework

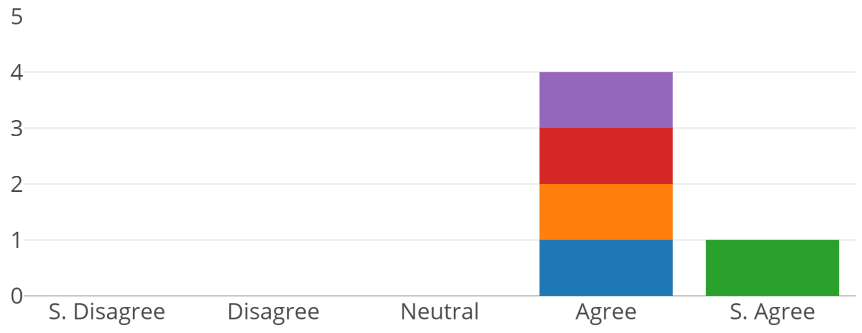
Question Number	**Please indicate to what extent you agree or disagree with the following statements**
Question 10.1	The techniques presented allow the exploration of more design alternatives during conceptual design, specifically regarding different ways of complying with safety requirements.
Question 10.2	The additional exploration capabilities will contribute to a higher level of understanding regarding how to achieve safety and the effects on system performance.
Question 10.3	An interactive software tool similar to the one presented, integrated with existing tools used in the company, would be useful and add value to the company.

The answers to questions 10.1, 10.2 and 10.3 show a majority of 'Agree' values, some 'Strongly Agree' and one 'Neutral' in question 10.3. This shows a high level of agreement, which support the claim that the proposed techniques allow the exploration of more design alternatives during conceptual design, contributing to a higher level of understanding on how to achieve safety and which effects it has on system performance. The results also show that a tool like AirCADia Architect, integrated with existing tools, can add value to the industry.

(a) Question 10.1



(b) Question 10.2



(c) Question 10.3

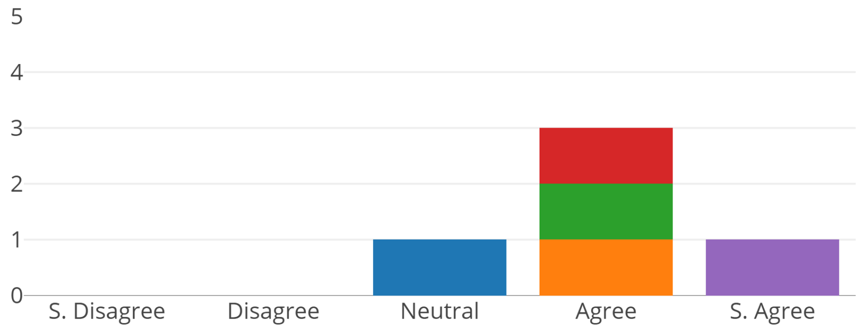


Figure 6.24: Answers to Likert questions related to Overall framework

Open Questions

The three open-ended questions that close the questionnaire are reproduced in Table 6.22. Regarding question 11, the participants highlighted the following positive aspects of the methods: automation and ability to speed up processes (mentioned by participants 1, 2 and 3), linking between different domains to ensure consistency (answered by participants 1 and 4), and ability of the methods to "incorporate safety considerations... without the need for expert input at every stage" (participant 5). Furthermore, these positive characteristic are in line with the aim and objectives of the research (see Chapter 1.3), which is especially important as their achievement is what the success evaluation intends to prove.

Table 6.22: open questions

Question Number	Question Text
Question 11	Is there anything that you particularly liked about any of the presented methods? Which one of them do you believe would be the most useful? How do you think the methods could add value in the conceptual design stage?
Question 12	Is there anything that you did not particularly like about any of the presented methods? Any particular capability that the methods fail to support?
Question 13	Regarding the certification process, how relevant is the early consideration of safety? How could these methods contribute towards the certification of aircraft in the future?

Question 12 helped to understand the limitations of the proposed methods and obtain ideas for further improvement. Respondents 3 and 5 highlighted the lack of verification and validation in the framework and suggested their addition. Participant 5 also worried about the methods might not be able to handle novel configurations. Participant 1 warned about the possibility that "the initial work to setup such a tool on an industrial scale may be extensive". Finally, respondent 2 highlighted the importance of including hazards both from past experience and more formal methods.

Question 13 explored the relationship between early consideration of safety, the certification process and the presented methods. The answers from participants 4 and 5 highlighted the importance of considering safety at early design stages and its potential to accelerate the certification process. Regarding the ability of the methods to contribute to certification of aircraft, respondent 2 indicated that " more model-based demonstration (including safety one) shall be introduced in the future but should be carefully looked at before fully trusting them". This is in line with participants 3 and 5 reiterating the importance of validation and verification for the methods to be used in industry. Participants 1 and 5 were positive with the ability of the methods to explore multiple solutions or allow for novel trades.

Based on the presented results, it is possible to conclude that the industrial evaluation was successful, with the presented methods having the potential to make safety architecting at early design stages more efficient and contribute to the exploration of more design alternatives. However, it is also important to consider that the methods still present limitations that hinder their ability to be used in an industrial setup, and that should be considered in future work. Further evaluation is also desirable, preferably in a more realistic setup and with the architects using the software.

6.6 Summary and Conclusions

The results from the support and application evaluation studies indicated that the proposed methods work correctly and provide expected results. The STPA support was evaluated by applying it to the WBS case study. The results were compared to those obtained via manual application of the methods by Leveson et al. The degree of similarity was considerable even though there were some discrepancies, which were due to missing information in the original description of the system and its possible misinterpretation. This problem highlights the importance of having a common unambiguous description of the system that methods can use, such as the one used in this research.

The enablers for the architecting of safety principles were able to reproduce most of the safety features from the architecture of a real aircraft at the level of detail employed in this research. They enabled the implementation of safety measure by automating many repetitive aspects and ensuring consistency at every step of the process. The enablers work interactively, providing relevant information at each step, which can help architects to understand the effect of implementing safety principles in the architecture.

The FTA support was able to generate fault trees automatically from the architecture definition with very little additional user's input. Qualitative and quantitative evaluation of the fault showed that the generated fault trees that reflect the expected trends in the probability of failure and cut sets caused by an increasing degree of redundancy. The FTA support can help architects to obtain quick feedback after changes are made to the architecture, which has the potential of accelerating the architecting process.

The sizing support was able to handle the fast creation of computational workflows taking into account the interaction of the system with the environment and several configurations that model component failures. The workflows presented a low number of reversed models, which is generally an indication of good quality. The support is expected to speed up the determination of the impact of architecture changes on performance.

The industrial feedback provided valuable information regarding application and success evaluations. Although this only constituted an initial application and success evaluation, the responses from the expert were positive, indicating the ability of the methods to contribute towards the achievement of the objectives and aim of the research. The evaluation presented in this chapter has proven the ability of the methods to make safety an integral part of the systems architecting process, increasingly automating assessment and system development activities, improving the efficiency and effectiveness of design for safety.

However, the industrial evaluation identified some areas where further work is needed. The participants highlighted the lack of verification and validation in the framework. They also pointed out the importance of being able to include hazards from past experience and formal methods. The participants also raised concerns regarding the ability of the methods to handle novel configurations and the required amount of initial work to set up the tool.

It is also important to discuss the challenges the use of the proposed methods presents in an actual industrial setup. To get the maximum benefit from the methods, the architecture must be modelled by following the proposed RFLP framework, a framework that can be easily converted to and from the proposed framework or adapt the methods to their own framework. The more difficult the conversion between frameworks (e.g. time or resource consuming), the smaller the expected benefit from using the proposed methods. Regarding the type of data required, at a minimum, the methods require that the logical view (components and connections) is defined and computational models for each component are available for the sizing enabler. The result obtained from the methods can be enhanced by providing additional information such as the requirement and functional views, inter-view relations, probability of failure, disturbance propagation inside components, and functional-logical knowledge (e.g. which solution satisfies which function and what functions are derived as a result). An architecting framework that contains these types of data is likely to be easily convertible to the presented RFLP framework or the methods could be easily adaptable to such a framework.

Chapter 7

Summary and Conclusions

7.1 Introduction

This chapter concludes the main body of this thesis. First, a summary of the research is provided in Section 7.2. The summary focuses on how the proposed methods achieve the aims and objectives of the research. This is followed by a discussion of the research findings and contribution to knowledge in Section 7.3. Finally, the limitation of the research and suggestions for future work are presented in Section 7.4.

7.2 Research Summary

7.2.1 Research Clarification

The research presented in this thesis is structured according to the four stages of engineering design research by Blessing and Chakrabarti [1]. The first stage, *Research Clarification* consisted of a literature review that identified areas of improvement regarding the design of safe systems and provided enough information to formulate the aim and objectives. First, the literature review clarified the concepts of safety, reliability, and resilience; definitions for these terms were obtained after compiling and identifying the most frequent traits of existing definitions in the literature. It was also found that both reliability and resilience are closely related to safety. Methods for improving reliability or resilience can also have a positive impact on safety, however safety cannot be restricted to reliability as some accidents — those caused by interactions of working, rather than component failure — cannot be prevented by increasing component reliability.

The literature review also clarified how the safety assessment process, used to show compliance with certification requirements, and the rest of the aircraft development processes relate to each other. On the safety assessment side, two main kinds of activities were identified, namely hazard assessment (creation and verification of safety requirements) and safety analysis (validation of the system against safety requirements). On the system development side, it was found that it is fundamental to support the architecting of safety principles. It is also important to and be able to assess the impact on performance as soon as possible after safety-motivated changes are made to the architecture.

As a result of the research clarification, the following aim (restated here for convenience) was proposed:

To improve the efficiency and effectiveness of design for safety as an integral part of the systems architecting process.

The following objectives were formulated in support of the aim:

1. Automate and integrate the hazard assessment process with the systems architecting process to allow a seamless definition of safety requirements.

2. Develop interactive methods to support the introduction of safety principles during architecture definition.

3. Automate the creation of system safety and performance models, enabling swift assessment of the candidate architectures.

7.2.2 Descriptive Study I

The *Research Clarification* was followed by the *Descriptive Study I*, which consisted in a detailed literature review regarding the most relevant topics concerning the objectives. First, accident models were reviewed to better understand the hazard assessment and safety analysis methods that the models underpin. Then, the methods themselves were studied and STPA hazard assessment and FTA were chosen as the most appropriate methods for which to develop computational support, which is intended to automate them and integrate them with the rest of system development processes and tools as required by the research objectives.

Several computational tools to support STPA were reviewed. It was concluded that none of them supports the automated creation of detailed STPA control loops from the architecture definition and only one of them supports the automated creation of hierarchical control structures. Furthermore, none of them was integrated nor easily integrable with tools that support other parts of the architecture definition such as requirements or functions.

Regarding FTA, the existing methods for automating the creation of fault trees were studied. It was found that the reviewed approaches require the architectural information to be translated to different specific languages, which limits their ability to be integrated easily with other tools that provide sizing and performance information.

To support the architecting of safety into the design, several safety principles were reviewed. Many of them were taken from the field of resilience, as it was shown earlier in the review how resilience can indeed contribute to safety. The principles of physical redundancy, functional redundancy, and containment were identified as the most applicable within the scope of this research, considering that the objective was to develop a computation support tool to support the inclusion of safety principles within the architecture.

Two limitations were found with current methods to support the sizing of complex systems present. Some of the methods require extensive manual setup or expert knowledge, hindering the ability to provide results swiftly after the architecture definition is modified. Others, despite being more automated, can only consider certain combinations of systems and components. A method free from both limitations was found, but its applicability is limited as it considers only a single flight condition to determine the sizes of the system. More flight conditions and configurations need to be considered to determine the most demanding conditions for a system, especially when designing novel architecture where past experience is insufficient.

7.2.3 Prescriptive Study I

In the *Prescriptive Study I* stage, several methods were developed to overcome the limitations of current approaches identified in the literature review. All methods used the RFLP definition of the architecture as an input, which is expected to facilitate the integration of the safety and development activities and help to improve the consistency of the results. The RFLP framework by Guenov et al. was identified as a promising framework for modelling the architecture. However, during the development of the methods, it was found that this RFLP approach was insufficient and some parts needed to be extended. Therefore, the first part of the methodology chapter revolves around the improved RFLP framework. The following parts of the methodology focused on methods to support hazard assessment, enablers for architecting safety principles, and methods for safety analysis and sizing.

RFLP Framework

The most common relations in the framework, namely hierarchical and flow relations, were presented first, including the object model that supports such relations. Regarding flow relations, the exchange of matter, energy or information between components is modelled using concepts from the Functional Basis (see Section 2.3.1). Flow is exchanged through ports and links between ports. Links can be seen in groups called connections, which represent mixing and distributions of flow as well as the existence or lack of redundancy.

The main elements in the RFLP views are requirements, functions and solutions. Requirements are presented in a hierarchy. Three kinds of requirements are considered, namely functional, performance and safety requirements. Function and solutions are related both hierarchically and via flow relations. Solutions are categorised as leaf solutions, composite solutions (subsystems), and controllers.

Objects that model relations between elements that belong to different views were also presented. The relations that were considered are 'Function satisfies requirement', 'Solution satisfies requirement', 'Solution fulfils function' and 'Solution derives function'. Finally, the concepts of template and template library were introduced. Templates facilitate the creation of functions and solutions ensuring consistency between similar elements and avoiding having to define the same element more than once.

Methods to Support Hazard Assessment

Two novel methods were developed to support architects in the hazard assessment process, in particular the methods, provide support for creating the models required in steps 2, 3 and 4 of the STPA hazard assessment process. The first method uses the information in the logical view of the architecture to automatically elaborate consistent hierarchical control structures, which can be used to identify unsafe control actions. The flow and hierarchical relation in the logical view determine the position in the hierarchy of the controllers and what type of information (feedback, control or same level signals) is exchanged among them and the controlled processes.

The second method enables the automated creation of more detailed control loops to support the identification of loss scenarios. Given a controller, the sets of actuators, sensors and process solutions are determined either automatically by the method or manually by the user. The links between sets are determined similarly to the ones in the method for creating hierarchical control structures.

Enablers for Architecting Safety Principles

Three novel enablers were developed to help architects interactively implement safety principles into architectures. Specifically, the enablers support the introduction of physical redundancy, functional redundancy and containment.

The architecting enabler for physical redundancy helps architects to increase the redundancy of the architecture by duplicating one or more times a part of the architecture. The extension of the redundant leg — which components will be replicated — is calculated by traversing the logical view according to a set of rules that the architect can modify. Once the extension is satisfactory, the desired number of redundant legs is created.

The enabler for functional redundancy supports increasing the redundancy of the architecture by adding one or more components that can perform one of the already fulfilled functions in the architecture in a dissimilar manner. The solutions in the architecture and redundant leg and solution templates from the library are compared and sorted according to their ability to fulfil the function. When a solution from the library is added to the leg, new functions might be derived as a result. These functions need to be fulfilled, which is achieved by repeating the process.

The architecting enabler for containment helps architects to study the propagation of disturbances that originate from one or more solutions (disturbance sources) and could affect one or more susceptible solutions (disturbance sinks). The enabler can create and analyse the effects of including barriers — any solution that is capable of stopping the propagation of a disturbance. The best location for barriers can be determined manually or automatically by using a min-cut algorithm.

Method for Automated Creation of Fault Trees

A safety method that enables the automatic creation and interactive analyses of fault tree analyses was developed. A novel algorithm is used to create fault trees from the architecture definition. The flow relations in the logical view are used together with the information about existing redundancy to determine the type and inputs of the fault tree gates. Trees created this way can be analysed by applying standard qualitative and quantitative techniques. The results are linked to the different parts of the tree and logical view and can be visualized interactively.

Method for Automated Creation of Sizing Workflows

A method for automating the creation of sizing workflows was built on top of an existing state-of-the-art RFLP sizing enabler. The developed methods make it possible to consider more detailed information about the system context and system configurations, such as those failure configurations that the system can handle thanks to the architected safety principles such as redundancy.

7.2.4 Descriptive Study II

The *Descriptive Study II* stage consists of the evaluation of the research. Several studies were performed to test and evaluate the novel methods and enablers developed during the course of this research. In order to be evaluated, the methods were implemented within a prototype object-oriented software called *AirCADia Architect*. Three types of evaluation were carried out in this research, namely support, application and success evaluation. Two use cases were developed for evaluation purposes. One of the use cases is inspired in the wheel brake system example from the ARP 4761 [58], the other one in the systems architecture of an Airbus A320 including the cabin, ECS, FCS, PNE, HYD and engine systems.

Support evaluation, which is also referred to as verification, checked the consistency and correct functioning of the developed tool. The support evaluation was conducted by applying the proposed techniques to the two case studies and comparing the results with those provided by manual application of the methods, existing automated tools, or existing designs. The results indicated that the proposed methods work correctly and provide expected results.

Application evaluation investigated whether the application of the methods contributes towards achieving the aim of the research. It was also based on the application of the methods to the use cases. The result showed the ability of the methods to reduce the time required for architecting and assessing the impact of safety as they automate significant parts of the process reducing the number of required manual activities. Greater consistency in the architecture definition and between the definition and STPA, FTA and sizing analyses was also observed. The potential of the methods to work with a common architecture definition, mitigating the problems of interpreting ambiguous architectural diagrams or descriptions from various sources, was demonstrated as well.

Success evaluation intended to determine whether the developed methods can provide value and be useful in practice. As it was not possible to test the performance of the methods on a real industrial project, the evaluation consisted in the obtention of feedback from a panel of specialists from Airbus and Cranfield University. The research and a live demonstration based on the A320 inspired use case was presented to the panel. After the presentation, an online questionnaire was given to the participants. The responses from the experts were positive, indicating the ability of the methods to contribute towards the achievement of the objectives and aim of the research. The feedback also provided suggestions to improve the presented work and make it more industrially relevant.

7.3 Novelty and Contribution to Knowledge

This research has resulted in the following novelty and contribution to knowledge and engineering practice:

- Even though resilience is a popular topic spanning many research fields, the literature review showed there is no clear definition of resilience and that it is unclear how resilience relates to safety. The analysis of several resilience-related references was employed to propose a definition for resilience that is relevant within the scope of this research. After careful consideration, it was concluded that resilience does not depart significantly from the aims of safety and reliability when applied to this research, and therefore there is no need for such term. Nevertheless, resilience principles were found to be relevant for this research and relabelled as safety principles.

- Two novel methods for automating the creation process of the necessary models for STPA hazard analysis of systems architectures. In particular:

  - The first method creates hierarchical control structures, which comprise the whole system.

  - The second method allows architects to focus on a particular part of the system as it creates detailed controlled loops of such part.

  The automation of model creation reduces the time required for STPA hazard analysis. It also increases the consistency of the results with respect to the architecture definition and other analysis as it uses the common RFLP model as input to the methods.

- Three novel interactive enablers developed to facilitate the introduction of safety principles, namely physical redundancy, functional redundancy and containment. The application of these principles is expected to modify the architecture in such a way that safety is increased. The enablers provide the relevant information for implementing each one of the principles, which is expected to lead to better decisions from architects. The automation of instantiation and linking of functions and solution reduces the required time for architecting safety. It also reduces the likelihood of making errors while performing these repetitive tasks.

- A novel method to automate the creation of fault trees. The method traverses the architecture and adds gates to the fault tree. The type of each gate depends on the existing redundancy in the architecture. Gates' inputs correspond to other gates or failures of individual components. The elements of the tree are linked to the architecture to facilitate the interactive exploration of results. The method reduces the time required for FTA and the consistency of the results as the source of information is the common RFLP model.

- A method developed to automate substantial parts of the creation process of computational workflows used for sizing. The method extends a state-of-the-art method that was designed for only one sizing point. The novel method is capable of accounting for different scenarios (including failure scenarios) and flying conditions. This way, the time spent on sizing is reduced while being able to account for complex sizing problems.

- An improved RFLP framework. It was developed to overcome the limitations of existing RFLP frameworks that made them unsuitable to be used with the developed methods. The framework allows the definition of architectures in such a way that all proposed methods refer to the same model and thus become tightly integrated. This reduced the likelihood of different analyses evolving at a different pace and referring to different versions of the system.

7.4 Limitations and Future Work

There are several limitations regarding the proposed techniques, which could be addressed in future work. These limitations and avenues for future research are as follows:

- The methods for creating hierarchical control structures currently lump together all process solutions in one node. However, several examples of hierarchical control structures that do not adhere to this rule can be found in Reference [61], which was used to prepare the case study employed for the evaluation of STPA support (see Section 6.3.2). It is important to study how to provide a way to group process solutions in more than one node without losing the benefits of automation.
- Similar to the point above, there are examples of detailed control loops that do not match exactly the generic structure proposed in the STPA handbook [92] and used by the method developed in this research. It needs to be investigated how to provide more flexibility to create control loops without the additional manual input offsetting the benefits of automation.
- The visualization of STPA models, in particular the automatic layout of the items, was not found completely satisfactory in examples that include many components. Visualization of the models seems to be an important part of the STPA process, so improving this aspect is crucial for extending the applicability of the proposed methods to larger systems.

- Merging the components created by the redundancy enablers with the existing architecture is not trivial when the interfaces of subsystems are involved. A set of default rules, which produced good results in the evaluation case studies, was proposed. However, the applicability of these rules to other cases needs to be studied and new rules and options might need to be included.

- A central part of the functional redundancy enabler is the functional-logical information (e.g. the ability of a solution to fulfil a function and its need for derived functions) contained in solutions and templates. At a basic level, this information is inferred from component and function interfaces. However, more detailed functional-logical information is expected to lead to a more precise ranking of solutions. Since it is difficult to define the relations a priori, it would be beneficial to develop ways to capture this information as the architecture is defined so it can be reused later.

- The modelling of disturbance propagation used in this thesis does not differentiate between types of disturbance. Removing this limitation would introduce more complexity but it would also enable more realistic studies.

- The minimum cut algorithm used to determine the optimal placement of barriers takes into account the weights associated with graph edges. These weights might be used to express the user's preference for the location of barriers but no mechanism to do so is currently available. Also, the algorithm outputs only one minimum cut, but more cuts with the same value might exist. Removing these limitations is a suggested avenue for future research.

- The method for automated fault tree creation considers only the failure of components without specifying which kind of failure. Furthermore, only *AND* and *OR* gates are applicable with the current level of detail regarding redundancy. If the architecting framework is extended to model information about failure modes and redundancy, the fault tree creation algorithm can be extended to generate trees that include this information. The traversal logic of the algorithm is expected to be reusable for this purpose.

- Depending on the kind of sizing workflow obtained by applying the proposed method, a unique solution for the sizing problem might not exist. Therefore, additional criteria need to be employed to obtain a single solution (e.g. optimise weight or power consumption). The set of variables that describe system attributes is determined by optimising the system according to such criteria. By contrast, finding the most demanding condition implies finding the set of variables that models the environment so that the performance of the system is minimised. Methods for solving this double optimization problem in an efficient matter need to be researched further.

- Scheduling workflows for systems involving a great number of models can be challenging when the models generate conflicts in their default configuration, e.g. the default workflow is overdetermined. Support methods are required to help architects resolve these issues, including better visualization of workflows.

- This research focused on particular aspects of the safety and system development processes. Other relevant activities and analyses, such as determining the layout of components or thermal analysis among others, need to be performed during the early design stages. Adapting the methods to the RFLP framework would be beneficial to obtain a better integration and increase consistency among the analyses. This would also test the ability of the proposed framework to model the required aspects of the system during conceptual design and inform further development of the framework. One possibility for extending the framework is including the physical view of the system, which was out of scope in this research.

- When a part of the architecture is modified, all the analyses related to that part are likely to become outdated and their results might become invalid. The author believes that, in response to certain minor changes, it would be possible to automatically repeat some analyses so they are kept up to date without requiring new user input. Which analyses and which kind of changes can be handled in this way need to be determined in future research.

- The industrial evaluation highlighted the lack of verification and validation in the framework. Verification and validation are suggested to be added in future work. One of the participants also warned about the amount of initial work required for using the proposed methods. Another respondent highlighted the importance of including hazards from past experience and formal methods. Quantifying the effort required for the initial setup and how to re-use this information and past experience for future projects is an interesting avenue for future research, especially if it involves a real industrial project.

References

[1] L. T. M. Blessing and A. Chakrabarti. *DRM, a Design Research Methodology*. London: Springer London, 2009.
DOI: 10.1007/978-1-84882-587-1.

[2] European Aviation Safety Agency. *Annual Safety Review 2020*. European Aviation Safety Agency, 2020.
DOI: 10.2822/147804.

[3] International Air Transport Association. *IATA Annual Review 2020*. Amsterdam: International Air Transport Association, 2020.
URL: https://www.iata.org/en/publications/annual-review/ (accessed on 16/04/2021).

[4] D. G. Ullman. *The Mechanical Design Process*. 4th ed. McGraw-Hill Series in Mechanical Engineering. McGraw-Hill Higher Education, 2009.

[5] *Boeing 737 MAX Groundings*. In: *Wikipedia*. 2021.
URL: https://en.wikipedia.org/w/index.php?title=Boeing_737_MAX_groundings&oldid=1017342329 (accessed on 17/04/2021).

[6] Komite Nasional Keselamatan Transportasi. *Aircraft Accident Investigation Report PT. Lion Mentari Airlines Boeing 737-8 (MAX); PK-LQP*. Aircraft Accident Investigation Report. Tanjung Karawang, West Java, Republic of Indonesia: Komite Nasional Keselamatan Transportasi, 2019.
URL: http://knkt.dephub.go.id/knkt/ntsc_aviation/baru/2018%20-%20035%20-%20PK-LQP%20Final%20Report.pdf (accessed on 17/04/2021).

[7] Aircraft Accident Investigation Bureau. *Interim Investigation Report on Accident to the B737-8 (MAX) Registered ET-AVJ Operated by Ethiopian Airlines On 10 March 2019*. Addis Ababa, Ethiopia: Aircraft Accident Investigation Bureau, Ministry of Transport, 2020.
URL: https://web.archive.org/web/20200310004955/http://www.aib.gov.et/wp-content/uploads/2020/documents/accident/ET-302%20%20Interim%20Investigation%20%20Report%20March%209%202020.pdf (accessed on 17/04/2021).

[8] The International Council on Systems Engineering. *About Systems Engineering*. 2021.
URL: https://www.incose.org/about-systems-engineering/system-and-se-definition/systems-engineering-definition (accessed on 19/04/2021).

[9] Object Management Group. *MBSE Wiki*. 2020.
URL: https://www.omgwiki.org/MBSE/doku.php (accessed on 20/04/2021).

[10] SysML.org. *SysML Open Source Project*. 2021.
URL: https://sysml.org/ (accessed on 23/05/2021).

[11] The Modelica Association. *Modelica Language*. 2021.
URL: https://modelica.org/modelicalanguage.html (accessed on 23/05/2021).

[12] M. Guenov, A. Molina-Cristobal, V. Voloshin, A. Riaz, A. S. van Heerden, S. Sharma, C. Cuiller and T. Giese. 'Aircraft Systems Architecting - a Functional-Logical Domain Perspective'. In: *16th AIAA Aviation Technology, Integration, and Operations Conference*. AIAA AVIATION Forum. American Institute of Aeronautics and Astronautics, 2016.
DOI: 10.2514/6.2016-3143.

[13] VDI Department of Product Development and Mechatronics. *Design Methodology for Mechatronic Systems (VDI 2206)*. 2004.
URL: https://www.vdi.de/richtlinien/details/vdi-2206-entwicklungsmethodik-fuer-mechatronische-systeme (accessed on 24/05/2021).

[14] M. D. Guenov, A. Riaz, Y. H. Bile, A. Molina-Cristobal and A. S. Heerden. ‘Computational Framework for Interactive Architecting of Complex Systems’. In: *Systems Engineering* 23.3 (2020), pp. 350–365.
DOI: 10.1002/sys.21531.

[15] Y. H. Bile, A. Riaz, M. D. Guenov and A. Molina-Cristobal. ‘Towards Automating the Sizing Process in Conceptual (Airframe) Systems Architecting’. In: *2018 AIAA/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference*. Kissimmee, Florida: American Institute of Aeronautics and Astronautics, 2018.
DOI: 10.2514/6.2018-1067.

[16] Y. H. Bile. ‘Component-Driven Computational Design of Complex Engineering Systems’. PhD thesis. Cranfield University, 2019.

[17] The Object Management Group. *What Is UML — Unified Modeling Language*. 2005.
URL: https://www.uml.org/what-is-uml.htm (accessed on 25/08/2021).

[18] B. Rumpe. *Modeling with UML*. Cham: Springer International Publishing, 2016.
DOI: 10.1007/978-3-319-33933-7.

[19] R. B. Stone and K. L. Wood. ‘Development of a Functional Basis for Design’. In: *Journal of Mechanical Design* 122.4 (2000), pp. 359–370.
DOI: 10.1115/1.1289637.

[20] J. Hirtz, R. B. Stone, D. A. McAdams, S. Szykman and K. L. Wood. ‘A Functional Basis for Engineering Design: Reconciling and Evolving Previous Efforts’. In: *Research in Engineering Design* 13.2 (2002), pp. 65–82.
DOI: 10.1007/s00163-001-0008-3.

[21] A. D. Little, K. L. Wood and D. A. McAdams. ‘Functional Analysis: A Fundamental Empirical Study for Reverse Engineering, Benchmarking and Redesign’. In: *Volume 3: 9th International Design Theory and Methodology Conference*. Sacramento, California, USA: American Society of Mechanical Engineers, 1997.
DOI: 10.1115/DETC97/DTM-3879.

[22] S. Szykman, J. W. Racz and R. D. Sriram. 'The Representation of Function in Computer-Based Design'. In: *Volume 3: 11th International Conference on Design Theory and Methodology*. Las Vegas, Nevada, USA: American Society of Mechanical Engineers, 1999, pp. 233–246.
DOI: 10.1115/DETC99/DTM-8742.

[23] G. Pahl, W. Beitz, J. Feldhusen and K.-H. Grote. *Engineering Design*. 3rd ed. London: Springer London, 2007.
DOI: 10.1007/978-1-84628-319-2.

[24] D. C. Karnopp, D. L. Margolis and R. C. Rosenberg. *System Dynamics: Modeling, Simulation, and Control of Mechatronic Systems*. Hoboken, NJ, USA: John Wiley & Sons, Inc., 2012.
DOI: 10.1002/9781118152812.

[25] T. H. Cormen, C. E. Leiserson, R. L. Rivest and C. Stein. *Introduction to Algorithms*. 3rd ed. Cambridge, Mass: MIT Press, 2009.

[26] L. R. Ford and D. R. Fulkerson. *Flows in Networks*. 1st ed. A Rand Corporation Research Study. Princeton University Press, 1962. JSTOR: `j.ctt183q0b4`.

[27] J. Edmonds and R. M. Karp. 'Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems'. In: *Journal of the ACM* 19.2 (1972), pp. 248–264.
DOI: 10.1145/321694.321699.

[28] W. E. Vesely, F. F. Goldberg, N. H. Roberts and D. F. Haasl. *Fault Tree Handbook*. 1981.
URL: `https://www.nrc.gov/reading-rm/doc-collections/nuregs/staff/sr0492/index.html` (accessed on 24/05/2021).

[29] E. Ruijters and M. Stoelinga. 'Fault Tree Analysis: A Survey of the State-of-the-Art in Modeling, Analysis and Tools'. In: *Computer Science Review* 15-16 (2015), pp. 29–62.
DOI: 10.1016/j.cosrev.2015.03.001.

[30] Y. Dutuit and A. Rauzy. ‘Efficient Algorithms to Assess Component and Gate Importance in Fault Tree Analysis’. In: *Reliability Engineering & System Safety* 72.2 (2001), pp. 213–222.
DOI: 10.1016/S0951-8320(01)00004-7.

[31] S. Jimeno, A. Riaz, M. Guenov and A. Molina-Cristobal. ‘Enabling Interactive Safety and Performance Trade-Offs in Early Airframe Systems Design’. In: *AIAA Scitech 2020 Forum*. Orlando, FL: American Institute of Aeronautics and Astronautics, 2020.
DOI: 10.2514/6.2020-0550.

[32] S. Hosseini, K. Barker and J. E. Ramirez-Marquez. ‘A Review of Definitions and Measures of System Resilience’. In: *Reliability Engineering & System Safety* 145 (2016), pp. 47–61.
DOI: 10.1016/j.ress.2015.08.006.

[33] R. Patriarca, J. Bergström, G. Di Gravio and F. Costantino. ‘Resilience Engineering: Current Status of the Research and Future Challenges’. In: *Safety Science* 102 (2018), pp. 79–100.
DOI: 10.1016/j.ssci.2017.10.005.

[34] D. Henry and J. E. Ramirez-Marquez. ‘Generic Metrics and Quantitative Approaches for System Resilience as a Function of Time’. In: *Reliability Engineering & System Safety* 99 (2012), pp. 114–122.
DOI: 10.1016/j.ress.2011.09.002.

[35] V. B. Sitterle, D. F. Freeman, S. R. Goerger and T. R. Ender. ‘Systems Engineering Resiliency: Guiding Tradespace Exploration within an Engineered Resilient Systems Context’. In: *Procedia Computer Science* 44 (2015), pp. 649–658.
DOI: 10.1016/j.procs.2015.03.013.

[36] K. Tierney and M. Bruneau. ‘Conceptualizing and Measuring Resilience: A Key to Disaster Loss Reduction’. In: *TR News* 250 (2007).
URL: `https://trid.trb.org/view/813539` (accessed on 24/05/2020).

[37] E. Hollnagel. ‘The Four Cornerstones of Resilience Engineering’. In: *Resilience Engineering Perspectives, Volume 2*. 2009.

[38] D. D. Woods. ‘Four Concepts for Resilience and the Implications for the Future of Resilience Engineering’. In: *Reliability Engineering & System Safety* 141 (2015), pp. 5–9.
DOI: 10.1016/j.ress.2015.03.018.

[39] R. Westrum. ‘A Typology of Resilience Situations’. In: *Resilience Engineering*. Ed. by E. Hollnagel, D. D. Woods and N. Leveson. 1st ed. CRC Press, 2017, pp. 55–65.
DOI: 10.1201/9781315605685-8.

[40] A. M. Madni and S. Jackson. ‘Towards a Conceptual Framework for Resilience Engineering’. In: *IEEE Systems Journal* 3.2 (2009), pp. 181–191.
DOI: 10.1109/JSYST.2009.2017397.

[41] R. Francis and B. Bekera. ‘A Metric and Frameworks for Resilience Analysis of Engineered and Infrastructure Systems’. In: *Reliability Engineering & System Safety* 121 (2014), pp. 90–103.
DOI: 10.1016/j.ress.2013.07.004.

[42] J. R. Enos. ‘Achieving Resiliency in Major Defense Programs through Nonfunctional Attributes’. In: *Systems Engineering* 22.5 (2019), pp. 389–400.
DOI: 10.1002/sys.21488.

[43] R. Neches and A. M. Madni. ‘Towards Affordably Adaptable and Effective Systems’. In: *Systems Engineering* 16.2 (2013), pp. 224–234.
DOI: 10.1002/sys.21234.

[44] M. J. Chalupnik, D. C. Wynn and P. J. Clarkson. ‘Comparison of Ilities for Protection against Uncertainty in System Design’. In: *Journal of Engineering Design* 24.12 (2013), pp. 814–829.
DOI: 10.1080/09544828.2013.851783.

[45] H. T. Tran, M. Balchanos, J. C. Domerçant and D. N. Mavris. ‘A Framework for the Quantitative Assessment of Performance-Based System Resilience’. In: *Reliability Engineering & System Safety* 158 (2017), pp. 73–84.
DOI: 10.1016/j.ress.2016.10.014.

[46] P. Uday, R. Chandrahasa and K. Marais. ‘System Importance Measures: Definitions and Application to System-of-Systems Analysis’. In: *Reliability Engineering & System Safety* 191 (2019).
DOI: 10.1016/j.ress.2019.106582.

[47] A. M. Farid. ‘Static Resilience of Large Flexible Engineering Systems: Axiomatic Design Model and Measures’. In: *IEEE Systems Journal* 11.4 (2017), pp. 2006–2017.
DOI: 10.1109/JSYST.2015.2428284.

[48] B. M. Ayyub. ‘Systems Resilience for Multihazard Environments: Definition, Metrics, and Valuation for Decision Making’. In: *Risk Analysis* 34.2 (2014), pp. 340–355.
DOI: 10.1111/risa.12093.

[49] S. Jackson and T. L. J. Ferris. ‘Resilience Principles for Engineered Systems’. In: *Systems Engineering* 16.2 (2013), pp. 152–164.
DOI: 10.1002/sys.21228.

[50] P. Uday and K. Marais. ‘Designing Resilient Systems-of-Systems: A Survey of Metrics, Methods, and Challenges’. In: *Systems Engineering* 18.5 (2015), pp. 491–510.
DOI: 10.1002/sys.21325.

[51] S. Jackson. ‘Resilience of the Aircraft System’. In: *Systems Engineering for Commercial Aircraft: A Domain-Specific Adaptation*. London, England: Routledge, 2015, pp. 229–243.
DOI: 10.1201/9781003075042.

[52] P. Martin-Breen and J. M. Anderies. 'Resilience: A Literature Review'. In: (2011). URL: https://opendocs.ids.ac.uk/opendocs/handle/20.500.12413/3692 (accessed on 24/05/2021).

[53] J. C. Whitson and J. E. Ramirez-Marquez. 'Resiliency as a Component Importance Measure in Network Reliability'. In: *Reliability Engineering & System Safety* 94.10 (2009), pp. 1685–1693. DOI: 10.1016/j.ress.2009.05.001.

[54] Y. Y. Haimes. 'On the Definition of Resilience in Systems'. In: *Risk Analysis* 29.4 (2009), pp. 498–501. DOI: 10.1111/j.1539-6924.2009.01216.x.

[55] S. Jackson. *Architecting Resilient Systems: Accident Avoidance and Survival and Recovery from Disruptions*. Hoboken, NJ, USA: John Wiley & Sons, Inc., 2009. DOI: 10.1002/9780470544013.

[56] R. W. Burch. *Resilient Space Systems Design: An Introduction*. Boca Raton: CRC Press, 2019. DOI: 10.1201/9780429053603.

[57] B. D. Youn, C. Hu and P. Wang. 'Resilience-Driven System Design of Complex Engineered Systems'. In: *Journal of Mechanical Design* 133.10 (2011). DOI: 10.1115/1.4004981.

[58] S-18 Aircraft and Sys Dev and Safety Assessment Committee. *Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment*. United States: SAE International, 1996. DOI: 10.4271/ARP4761.

[59] N. Leveson. *Engineering a Safer World: Systems Thinking Applied to Safety*. Engineering Systems. Cambridge, Mass: MIT Press, 2012.

[60] JPL Special Review Board. *Report on the Loss of the Mars Polar Lander and Deep Space 2 Missions*. 2000. URL: https://ntrs.nasa.gov/api/citations/20000061966/downloads/20000061966.pdf (accessed on 29/03/2021).

[61] N. Leveson, C. Wilkinson, C. Fleming, J. Thomas and I. Tracy. *A Comparison of STPA and the ARP 4761 Safety Assessment Process*. 2014.
URL: http://sunnyday.mit.edu/STAMP/ARP4761-Comparison-Report-final-2.pdf (accessed on 25/05/2021).

[62] E. Hollnagel and G. Sundström. 'States of Resilience'. In: *Resilience Engineering*. Ed. by E. Hollnagel, D. D. Woods and N. Leveson. 1st ed. CRC Press, 2017, pp. 339–346.
DOI: 10.1201/9781315605685-29.

[63] M. G. Richards, D. E. Hastings, D. H. Rhodes and A. L. Weigel. 'Defining Survivability for Engineering Systems'. In: *Conference on Systems Engineering Research*. 2007.
URL: https://www.researchgate.net/publication/228371484_Defining_Survivability_for_Engineering_Systems (accessed on 25/05/2021).

[64] S-18 Aircraft and Sys Dev and Safety Assessment Committee. *Guidelines for Development of Civil Aircraft and Systems*. United States: SAE International, 2010.
DOI: 10.4271/ARP4754A.

[65] European Aviation Safety Agency. *Certification Specifications for Large Aeroplanes CS-25*. 2020.
URL: https://www.easa.europa.eu/certification-specifications/cs-25-large-aeroplanes (accessed on 25/05/2021).

[66] Federal Aviation Administration. *Title 14 of the Code of Federal Regulations. Part 25. Airworthiness Standards: Transport Category Airplanes.* 2020.
URL: https://ecfr.federalregister.gov/current/title-14/chapter-I/subchapter-C/part-25 (accessed on 11/03/2021).

[67] J. Delange and P. Feiler. 'Architecture Fault Modeling with the AADL Error-Model Annex'. In: *2014 40th EUROMICRO Conference on Software Engineering and Advanced Applications*. Verona, Italy: IEEE, 2014, pp. 361–368.
DOI: 10.1109/SEAA.2014.20.

[68] F. Mhenni, N. Nguyen and J.-Y. Choley. ‘Automatic Fault Tree Generation from SysML System Models’. In: *2014 IEEE/ASME International Conference on Advanced Intelligent Mechatronics*. Besacon: IEEE, 2014, pp. 715–720.
DOI: 10.1109/AIM.2014.6878163.

[69] A. Joshi, S. Vestal and P. Binns. *Automatic Generation of Static Fault Trees from AADL Models*. 2007.
URL: `http://hdl.handle.net/11299/217313`.

[70] A. Majdara and T. Wakabayashi. ‘Component-Based Modeling of Systems for Automated Fault Tree Generation’. In: *Reliability Engineering & System Safety* 94.6 (2009), pp. 1076–1086.
DOI: 10.1016/j.ress.2008.12.003.

[71] Y. Papadopoulos, J. McDermid, R. Sasse and G. Heiner. ‘Analysis and Synthesis of the Behaviour of Complex Programmable Electronic Systems in Conditions of Failure’. In: *Reliability Engineering & System Safety* 71.3 (2001), pp. 229–247.
DOI: 10.1016/S0951-8320(00)00076-4.

[72] Federal Aviation Administration. *System Safety Handbook*. 2000.
URL: `https://www.faa.gov/regulations_policies/handbooks_manuals/aviation/risk_management/ss_handbook/` (accessed on 15/03/2021).

[73] C. A. Ericson. *Hazard Analysis Techniques for System Safety: Ericson/Hazard Analysis Techniques for System Safety*. Hoboken, NJ, USA: John Wiley & Sons, Inc., 2005.
DOI: 10.1002/0471739421.

[74] E. Hollnagel. *FRAM: The Functional Resonance Analysis Method: Modelling Complex Socio-Technical Systems*. 1st ed. CRC Press, 2017.
DOI: 10.1201/9781315255071.

[75] H. W. Heinrich. *Industrial Accident Prevention: A Scientific Approach*. United Kingdom: McGraw-Hill book Company, Incorporated, 1931.

[76] F. E. Bird and R. G. Loftus. *Loss Control Management*. 1st ed. Loganville, Ga: Intl Loss Control Inst, 1976.

[77] J. Reason. *Human Error*. United Kingdom: Cambridge University Press, 1990.

[78] J. Larouzee and J.-C. Le Coze. 'Good and Bad Reasons: The Swiss Cheese Model and Its Critics'. In: *Safety Science* 126 (2020). DOI: 10.1016/j.ssci.2020.104660.

[79] J. Reason. 'Human Error: Models and Management'. In: *BMJ* 320.7237 (2000), pp. 768–770. DOI: 10.1136/bmj.320.7237.768.

[80] N. Leveson. 'A New Accident Model for Engineering Safer Systems'. In: *Safety Science* 42.4 (2004), pp. 237–270. DOI: 10.1016/S0925-7535(03)00047-X.

[81] N. G. Leveson. *A New Approach To System Safety Engineering*. 2002. URL: `https://dspace.mit.edu/bitstream/handle/1721.1/71860/16-358j-spring-2005/contents/readings/book2.pdf` (accessed on 19/03/2021).

[82] J. Rasmussen. 'Risk Management in a Dynamic Society: A Modelling Problem'. In: *Safety Science* 27.2 (1997), pp. 183–213. DOI: 10.1016/S0925-7535(97)00052-0.

[83] J. Rasmussen and I. Svedung. *Proactive Risk Management in a Dynamic Society*. Swedish Rescue Services A, 2000.

[84] I. Svedung and J. Rasmussen. 'Graphic Representation of Accident Scenarios: Mapping System Structure and the Causation of Accidents'. In: *Safety Science* 40.5 (2002), pp. 397–417. DOI: 10.1016/S0925-7535(00)00036-9.

[85] M. Modarres, M. P. Kaminskiy and V. Krivtsov. *Reliability Engineering and Risk Analysis : A Practical Guide*. CRC Press, 2016. DOI: 10.1201/9781315382425.

[86] M. Rausand and A. Høyland. *System Reliability Theory: Models, Statistical Methods, and Applications*. 2nd ed. Wiley Series in Probability and Statistics. Hoboken, NJ: Wiley-Interscience, 2004.

[87] S. Distefano and L. Xing. ‘A New Approach to Modeling the System Reliability: Dynamic Reliability Block Diagrams’. In: *RAMS ’06. Annual Reliability and Maintainability Symposium, 2006.* Newport Beach, CA, USA: IEEE, 2006, pp. 189–195.
DOI: 10.1109/RAMS.2006.1677373.

[88] S. Distefano and A. Puliafito. ‘Dependability Evaluation with Dynamic Reliability Block Diagrams and Dynamic Fault Trees’. In: *IEEE Transactions on Dependable and Secure Computing* 6.1 (2009), pp. 4–17.
DOI: 10.1109/TDSC.2007.70242.

[89] United States Department of Defense. *Procedures for Performing a Failure Mode, Effects and Criticality Analysis*. 1980.

[90] J. Dunjó, V. Fthenakis, J. A. Vílchez and J. Arnaldos. ‘Hazard and Operability (HAZOP) Analysis. A Literature Review’. In: *Journal of Hazardous Materials* 173.1-3 (2010), pp. 19–32.
DOI: 10.1016/j.jhazmat.2009.08.076.

[91] Y. Papadopoulos and M. Maruhn. ‘Model-Based Synthesis of Fault Trees from Matlab-Simulink Models’. In: *Proceedings International Conference on Dependable Systems and Networks*. Goteborg, Sweden: IEEE Comput. Soc, 2001, pp. 77–82.
DOI: 10.1109/DSN.2001.941393.

[92] N. G. Leveson and J. P. Thomas. *STPA Handbook*. 2018.
URL: `http://psas.scripts.mit.edu/home/get_file.php?name=STPA_handbook.pdf` (accessed on 12/03/2021).

[93] A. Abdulkhaleq. ‘An Open Tool Support for System- Theoretic Process Analysis’ (MIT, Boston). 2014.
URL: `http://psas.scripts.mit.edu/home/wp-content/uploads/2014/03/Asim_A-STPA.pdf` (accessed on 25/05/2021).

[94] SE-Stuttgart. *XSTAMPP*. Version 3.1.1. 2019.
URL: `https://github.com/SE-Stuttgart/XSTAMPP` (accessed on 31/03/2021).

[95] A. Abdulkhaleq and S. Wagner. ‘XSTAMPP: An eXtensible STAMP Platform As Tool Support for Safety Engineering’. In: *2015 STAMP Conference*. MIT, Boston, 2015.
DOI: 10.13140/2.1.3862.0486.

[96] The Eclipse Foundation. *Eclipse IDE*. Version 2021-03. 2021.
URL: `http://www.eclipse.org/eclipseide/` (accessed on 31/03/2021).

[97] S. S. Krauss, M. Rejzek, C. W. Senn and C. Hilbes. ‘SAHRA - An Integrated Software Tool for STPA’. In: *4th European STAMP Workshop*. Zurich, Switzerland, 2016.
DOI: 10.21256/zhaw-4926.

[98] S. S. Krauss, M. Rejzek, M. U. Reif and C. Hilbes. ‘Towards a Modeling Language for Systems-Theoretic Process Analysis (STPA) : Proposal for a Domain Specific Language (DSL) for Model Driven Systems-Theoretic Process Analysis (STPA) Based on UML’. In: (2016).
DOI: 10.21256/ZHAW-1175.

[99] Sparx Systems. *Sparx Enterprise Architect*. Version 15.2. 2021.
URL: `https://www.sparxsystems.com/products/ea/index.html` (accessed on 31/03/2021).

[100] Risk Management Studio. *STPA Software Solution — Risk Management Studio*. 2019.
URL: `https://www.riskmanagementstudio.com/stpa-software-solution/#` (accessed on 31/03/2021).

[101] S. H. Björnsdóttir and M. Rejzek. ‘Embedding STPA into a Highly Successful Risk Management Software Application’. In: *6th MIT STAMP Workshop*. Boston, USA: ZHAW Zürcher Hochschule für Angewandte Wissenschaften, 2017.
DOI: 10.21256/ZHAW-3306.

[102] Information-technology Promotion Agency, Japan. *STAMP Workbench 1.0.1 Documentation*. 2018.

URL: https://www.ipa.go.jp/sec/stamp_wb/manual_en/tutorial/basic/basic.html (accessed on 31/03/2021).

[103] Astah. *Astah System Safety*. Version 5.0. 2021.
URL: https://astah.net/products/astah-system-safety/ (accessed on 31/03/2021).

[104] Shamal Faily. *CAIRIS 2.3.8 Documentation*. 2021.
URL: https://cairis.readthedocs.io/en/latest/stpa.html (accessed on 01/04/2021).

[105] J. Thomas and D. Suo. 'A Tool-Based STPA Process' (Cambridge, MA). 2015.
URL: http://psas.scripts.mit.edu/home/wp-content/uploads/2015/03/Thomas-Suo-Tool-based-STPA-process.pdf (accessed on 01/04/2021).

[106] Volpe National Transportation Systems Center. *SafetyHAT: A Transportation System Safety Hazard Analysis Tool*. 2014.
URL: https://www.volpe.dot.gov/infrastructure-systems-and-technology/advanced-vehicle-technology/safetyhat-transportation-system (accessed on 01/04/2021).

[107] A. A. Abdellatif and F. Holzapfel. 'The Utilization of STPA Techniques for System Design Safety Enhancement'. In: *AIAA Scitech 2021 Forum*. AIAA SciTech Forum. American Institute of Aeronautics and Astronautics, 2021.
DOI: 10.2514/6.2021-0565.

[108] J. P. Thomas. 'Extending and Automating a Systems-Theoretic Hazard Analysis for Requirements Generation and Analysis'. PhD thesis. Massachusetts Institute of Technology, 2013.
URL: https://dspace.mit.edu/handle/1721.1/81055 (accessed on 01/04/2021).

[109] J. P. Thomas and N. G. Leveson. 'Generating Formal Model-Based Safety Requirements for Complex, Software- and Human-Intensive Systems'. In: *Safety-Critical Systems Club* (2013).
URL: http://sunnyday.mit.edu/SSS-conference-stpa.pdf (accessed on 28/03/2021).

[110] N. G. Leveson, M. P. E. Heimdahl and J. D. Reese. 'Designing Specification Languages for Process Control Systems: Lessons Learned and Steps to the Future'. In: *ACM SIGSOFT Software Engineering Notes* 24.6 (1999), pp. 127–145. DOI: 10.1145/318774.318937.

[111] J. Holt and S. Perry. *SysML for Systems Engineering: A Model-Based Approach*. 3d edition. IET Professional Applications of Computing Series 20. Stevenage: The Institution of Engineering and Technology, 2018.

[112] U. Lindemann, M. Maurer and T. Braun. *Structural Complexity Management: An Approach for the Field of Product Design*. Berlin: Springer, 2009. DOI: 10.1007/978-3-540-87889-6.

[113] M. Roth, M. Wolf and U. Lindemann. 'Integrated Matrix-Based Fault Tree Generation and Evaluation'. In: *Procedia Computer Science* 44 (2015), pp. 599–608. DOI: 10.1016/j.procs.2015.03.027.

[114] J. Xiang, K. Yanoo, Y. Maeno and K. Tadano. 'Automatic Synthesis of Static Fault Trees from System Models'. In: *2011 Fifth International Conference on Secure Software Integration and Reliability Improvement*. Jeju Island, Korea (South): IEEE, 2011, pp. 127–136. DOI: 10.1109/SSIRI.2011.32.

[115] M. Clavel, F. Durán, S. Eker, S. Escobar, P. Lincoln, N. Martí-Oliet, J. Meseguer, R. Rubio and C. Talcott. *Maude Manual (Version 3.0)*. 2020. URL: `http://maude.cs.illinois.edu/w/images/e/ee/Maude-3.0-manual.pdf` (accessed on 05/04/2021).

[116] P. Feiler, D. Gluch and J. Hudak. *The Architecture Analysis & Design Language (AADL): An Introduction*. 2006.

[117] J. Delange, P. Feiler, D. Gluch and J. J. Hudak. *AADL Fault Modeling and Analysis Within an ARP4761 Safety Assessment*. 2014, 1466338 Bytes. DOI: 10.21236/ADA610294.

[118] P. Feiler and J. Delange. 'Automated Fault Tree Analysis from AADL Models'. In: *ACM SIGAda Ada Letters* 36.2 (2017), pp. 39–46.
DOI: 10.1145/3092893.3092900.

[119] G. Point and A. Rauzy. 'AltaRica : Constraint Automata as a Description Language'. In: *Journal Européen des Systèmes Automatisés* 33.8-9 (1999), pp. 1033–1052.
URL: http://www.altarica-association.org/members/arauzy/Publications/pdf/PointRauzy1999-AltaRicaConstraintLanguage.pdf (accessed on 16/05/2021).

[120] S. Li and X. Li. 'Study on Generation of Fault Trees from Altarica Models'. In: *Procedia Engineering* 80 (2014), pp. 140–152.
DOI: 10.1016/j.proeng.2014.09.070.

[121] The MathWorks, Inc. *Simulink*. 2021.
URL: https://www.mathworks.com/products/simulink.html (accessed on 05/04/2021).

[122] G. Latif-Shabgahi and F. Tajarrod. 'A New Approach for the Construction of Fault Trees from System Simulink'. In: *2009 International Conference on Availability, Reliability and Security*. Fukuoka, Japan: IEEE, 2009, pp. 712–717.
DOI: 10.1109/ARES.2009.172.

[123] C. Schallert. 'Inclusion of Reliability and Safety Analysis Methods in Modelica'. In: *International Modelica Conference*. 2011, pp. 616–627.
DOI: 10.3384/ecp11063616.

[124] A. Tundis, L. Rogovchenko-Buffoni, A. Garro, M. Nyberg and P. Fritzson. 'Performing Fault Tree Analysis of a Modelica-Based System Design Through a Probability Model'. In: *Proceedings of Workshop on Applied Modeling and Simulation*. Buenos Aires, Argentina, 2013.
URL: https://www.researchgate.net/publication/272055346_Performing_Fault_Tree_Analysis_of_a_Modelica-Based_System_Design_Through_a_Probability_Model (accessed on 25/05/2021).

[125] S. Jackson and T. L. J. Ferris. 'Designing Resilient Systems'. In: *Resilience and Risk*. Ed. by I. Linkov and J. M. Palma-Oliveira. NATO Science for Peace and Security Series C: Environmental Security. Dordrecht: Springer Netherlands, 2017, pp. 121–144.
DOI: 10.1007/978-94-024-1123-2˙4.

[126] E. Hollnagel, D. D. Woods and N. Leveson, eds. *Resilience Engineering: Concepts and Precepts*. 1st ed. CRC Press, 2006.
DOI: 10.1201/9781315605685.

[127] M. G. Richards. 'Multi-Attribute Tradespace Exploration for Survivability'. PhD thesis. Massachusetts Institute of Technology, 2009.
URL: `https://dspace.mit.edu/handle/1721.1/53217` (accessed on 23/05/2020).

[128] J. Maxwell and R. F. Emerson. 'Observations of the Resilience Architecture of the Firefighting and Emergency Response Infrastructure'. In: *INSIGHT* 12.2 (2009), pp. 45–47.
DOI: 10.1002/inst.200912245a.

[129] C. Haddon-Cave, Great Britain, Parliament and House of Commons. *The Nimrod Review: An Independent Review into the Broader Issues Surrounding the Loss of the RAF Nimrod MR2 Aircraft XV230 in Afghanistan in 2006 : Report*. London: TSO, 2009.

[130] C. E. Billings. *Aviation Automation: The Search for a Human-Centered Approach*. Mahwah, N.J: Lawrence Erlbaum Associates Publishers, 1997.

[131] N. Leveson. *SafeWare: System Safety and Computers*. Reading, Mass: Addison-Wesley, 1995.

[132] J. T. Reason. *Managing the Risks of Organizational Accidents*. Aldershot, Hants, England ; Brookfield, Vt., USA: Ashgate, 1997.

[133] C. Perrow. *Normal Accidents: Living with High-Risk Technologies*. Princeton Paperbacks. Princeton, N.J: Princeton University Press, 1999.

[134] M. G. Richards, D. E. Hastings, A. M. Ross and D. H. Rhodes. '7.1.1 Survivability Design Principles for Enhanced Concept Generation and Evaluation'. In: *INCOSE International Symposium* 19.1 (2009), pp. 1055–1070.
DOI: 10.1002/j.2334-5837.2009.tb01001.x.

[135] G. Pumpuni-Lenss, T. Blackburn and A. Garstenauer. 'Resilience in Complex Systems: An Agent-Based Approach: RESILIENCE IN COMPLEX SYSTEMS'. In: *Systems Engineering* 20.2 (2017), pp. 158–172.
DOI: 10.1002/sys.21387.

[136] M. Ouyang, L. Dueñas-Osorio and X. Min. 'A Three-Stage Resilience Analysis Framework for Urban Infrastructure Systems'. In: *Structural Safety* 36-37 (2012), pp. 23–31.
DOI: 10.1016/j.strusafe.2011.12.004.

[137] N. Yodo and P. Wang. 'Engineering Resilience Quantification and System Design Implications: A Literature Survey'. In: *Journal of Mechanical Design* 138.11 (2016).
DOI: 10.1115/1.4034223.

[138] S. Jimeno, A. Molina-Cristobal, A. Riaz and M. Guenov. 'Incorporating Safety in Early (Airframe) Systems Design and Assessment'. In: *AIAA Scitech 2019 Forum*. San Diego, California: American Institute of Aeronautics and Astronautics, 2019.
DOI: 10.2514/6.2019-0553.

[139] S. Liscouët-Hanke. 'A Model-Based Methodology for Integrated Preliminary Sizing and Analysis of Aircraft Power System Architecture'. PhD thesis. Université de Toulouse, 2008.

[140] S. Liscouët-Hanke, J.-C. Maré and S. Pufe. 'Simulation Framework for Aircraft Power System Architecting'. In: *Journal of Aircraft* 46.4 (2009), pp. 1375–1380.
DOI: 10.2514/1.41304.

[141] C. de Tenorio. 'Methods for Collaborative Conceptual Design of Aircraft Power Architectures'. PhD thesis. Georgia Institute of Technology, 2010.
URL: `https://smartech.gatech.edu/handle/1853/34818` (accessed on 25/01/2021).

[142] C. de Tenorio, D. Mavris, E. Garcia and M. Armstrong. ‘Methodology for Aircraft System Architecture Sizing’. In: *26th Congress of International Council of the Aeronautical Sciences*. Anchorage, Alaska, USA, 2008, p. 11.
URL: http://www.icas.org/ICAS_ARCHIVE/ICAS2008/PAPERS/371.PDF (accessed on 24/05/2021).

[143] C. de Tenorio, M. Armstrong and D. Mavris. ‘Architecture Subsystem Sizing and Coordinated Optimization Methods’. In: *47th AIAA Aerospace Sciences Meeting Including The New Horizons Forum and Aerospace Exposition*. Aerospace Sciences Meetings. Orlando, Florida: American Institute of Aeronautics and Astronautics, 2009.
DOI: 10.2514/6.2009-37.

[144] I. Chakraborty. ‘Subsystem Architecture Sizing and Analysis for Aircraft Conceptual Design’. PhD thesis. Georgia Institute of Technology, 2015.
URL: https://smartech.gatech.edu/handle/1853/54427 (accessed on 06/04/2021).

[145] I. Chakraborty, D. N. Mavris, M. Emeneth and A. Schneegans. ‘An Integrated Approach to Vehicle and Subsystem Sizing and Analysis for Novel Subsystem Architectures’. In: *Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering* 230.3 (2015), pp. 496–514.
DOI: 10.1177/0954410015594399.

[146] I. Chakraborty and D. N. Mavris. ‘Integrated Assessment of Aircraft and Novel Subsystem Architectures in Early Design’. In: *Journal of Aircraft* 54.4 (2017), pp. 1268–1282.
DOI: 10.2514/1.C033976.

[147] D. Judt and C. Lawson. ‘Methodology for Automated Aircraft Systems Architecture Enumeration and Analysis’. In: *12th AIAA Aviation Technology, Integration, and Operations (ATIO) Conference and 14th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*. Aviation Technology, Integration, and Operations (ATIO) Conferences. Indianapolis, Indiana: American Institute of Aeronautics and

Astronautics, 2012.
DOI: 10.2514/6.2012-5648.

[148] D. M. Judt and C. Lawson. ‘Development of an Automated Aircraft Subsystem Architecture Generation and Analysis Tool’. In: *Engineering Computations* 33.5 (2016), pp. 1327–1352.
DOI: 10.1108/EC-02-2014-0033.

[149] J. E. Hopcroft and R. M. Karp. ‘An \$n{\^5/2}\$ Algorithm for Maximum Matchings in Bipartite Graphs’. In: *SIAM Journal on Computing* 2.4 (1973), pp. 225–231.
DOI: 10.1137/0202019.

[150] J. Fussell, E. Henry and N. Marshall. *MOCUS: A Computer Program to Obtain Minimal Sets from Fault Trees*. ANCR–1156, 4267950. 1974, ANCR–1156, 4267950.
DOI: 10.2172/4267950.

[151] D. Kececioglu. *Reliability Engineering Handbook*. Vol. 2. Englewood Cliffs, NJ: Prentice-Hall, 1991.

[152] *Microsoft Forms*.
URL: `https://forms.office.com` (accessed on 15/02/2021).

[153] M. Q. Patton. *Qualitative Research and Evaluation Methods*. 3 ed. Thousand Oaks, Calif: Sage Publications, 2002.

[154] United States Department of Defence. *Standard Practice for System Safety*. 2000.
URL: `http://sesam.smart-lab.se/IG_Prgsak/Publikat/MIL-STD-882D.pdf` (accessed on 28/03/2021).

[155] P. Uday and K. B. Marais. ‘Resilience-Based System Importance Measures for System-of-Systems’. In: *Procedia Computer Science* 28 (2014), pp. 257–264.
DOI: 10.1016/j.procs.2014.03.033.

[156] K. Barker, J. E. Ramirez-Marquez and C. M. Rocco. ‘Resilience-Based Network Component Importance Measures’. In: *Reliability Engineering & System Safety*

117 (2013), pp. 89–97.
DOI: 10.1016/j.ress.2013.03.012.

[157] R. Filippini and A. Silva. ‘A Modeling Framework for the Resilience Analysis of Networked Systems-of-Systems Based on Functional Dependencies’. In: *Reliability Engineering & System Safety* 125 (2014), pp. 82–91.
DOI: 10.1016/j.ress.2013.09.010.

[158] H. Mehrpouyan, B. Haley, A. Dong, I. Y. Tumer and C. Hoyle. ‘Resiliency Analysis for Complex Engineered System Design’. In: *Artificial Intelligence for Engineering Design, Analysis and Manufacturing* 29.1 (2015), pp. 93–108.
DOI: 10.1017/S0890060414000663.

[159] T. L. J. Ferris. ‘A Resilience Measure to Guide System Design and Management’. In: *IEEE Systems Journal* 13.4 (2019), pp. 3708–3715.
DOI: 10.1109/JSYST.2019.2901174.

[160] F. Ren, T. Zhao, J. Jiao and Y. Hu. ‘Resilience Optimization for Complex Engineered Systems Based on the Multi-Dimensional Resilience Concept’. In: *IEEE Access* 5 (2017), pp. 19352–19362.
DOI: 10.1109/ACCESS.2017.2755043.

[161] N. Yodo and P. Wang. ‘Resilience Allocation for Early Stage Design of Complex Engineered Systems’. In: *Journal of Mechanical Design* 138.9 (2016).
DOI: 10.1115/1.4033990.

[162] A. K. Raz and R. C. Kenley. ‘Multi-Disciplinary Perspectives for Engineering Resilience in Systems’. In: *2019 IEEE International Conference on Systems, Man and Cybernetics (SMC)*. Bari, Italy: IEEE, 2019, pp. 761–766.
DOI: 10.1109/SMC.2019.8914180.

Appendix A

Definitions from literature review

A.1 Safety

1. **Patriarca et al. [33]:** *no definition*.

    - Resilience Engineering (RE) is a paradigm for safety management that focuses on systems coping with complexity and balancing productivity with safety.

2. **Madni and Jackson [40]:** system property that encompasses the behaviour of and interactions among subsystems, software, organizations, and humans.

    - Conventional risk management approaches employ hindsight and calculations of failure probabilities, resilience engineering is a proactive approach that looks for ways to enhance the ability of organizations to explicitly monitor risks, and to make appropriate trade-offs between required safety levels and production and economic pressures.
    - Predictable Versus Unpredictable Disruptions: Traditional system safety analysis, as documented in MIL-STD-882 depends, for the most part, on designing systems to survive previously encountered disruptions.

Even so, the expectation exists that a system designed for resilience will avoid, survive, and recover from unpredicted disruptions also.

3. **Francis and Bekera [41]:** freedom from those conditions that can cause death, injury, occupational illness, damage to or loss of equipment or property, or damage to the environment. (From MIL-STD-882 [154])

    - Resilience engineering is distinguished from traditional safety management in that, instead of identifying and alleviating risk factors, it aims to build system that can withstand unanticipated disruptions. Anticipate, circumvent, and recover rapidly from events that threaten safety.
    - While a resilient system is likely to be safe, the converse is not necessarily true. Safety may be obtained at the cost of other objectives. Systems may not be designed with enough adaptive capacity and thus, not resilience.

4. **Uday and Marais [50]:** the objective of ensuring accident prevention through actions on multiple safety levers, such as technical, organizational, or regulatory. It values human life (or property loss) over other performance traits.

    - With respect to resilience, safety can be thought of as the aspect of survivability that is related to minimizing loss of life (or property).
    - In some cases, both these attributes go together. In other cases, the emphasis is on performance.

5. **Jackson [51]:** *no definition*.

    - Safety endeavours to prevent the failure of a system. Resilience goes beyond safety in that it calls for mechanisms to anticipate failure and to enable the system to recover from a major disruption.

6. **Haimes [54]:** *no definition*.

  - Resilience engineering is a paradigm for safety management that focuses on how to help people cope with complexity under pressure to achieve success.

7. **Jackson [55]:** *no definition*.
   - Thus, one could say that safety, which is achieving a system goal or objective without loss of life, is the whole point of system resilience.
   - In addition, system safety does not normally concern itself with the anticipation factor in system resilience.
   - Traditional safety analysis does not treat the interaction of components.

8. **Richards et al. [63]:** *no definition*.
   - An extension of the traditional fields of reliability engineering and safety management, resilience engineering

9. **ARP 4754A [64]:** the state in which risk is acceptable. Risk is the combination of the frequency (probability) of an occurrence and its associated level of severity. The levels of severity are defined in line with CS-25 [65, §AMC 25.1309], which focuses on the effects on the aircraft, crew and passengers.

10. **Leveson [59]:** freedom from accidents. An accident is defined as an undesired and unplanned event that results in a loss (including loss of human life or injury, property damage, environmental pollution, and so on).

A.2 Resilience

1. **Sitterle et al. [35]:** a resilient system is one which 1) is trusted and effective in a wide range of contexts (robustness), 2) easily adapted through reconfiguration or replacements.

2. **Patriarca et al. [33]:** feature if a system that allows it to respond to an unanticipated disturbance that can lead to failure and then resume operations quickly and with minimum decrement in performance. (related to robust, buoyant, elastic, flexible)

3. **Tierney and Bruneau [36]:** R4, Robustness, Redundancy, Resourcefulness, Rapidity.

4. **Hollnagel [37]:** responding, monitoring, learning, anticipating.

5. **Woods [38]:** (from different fields: safety, complexity, human organizations, ecology...) system property that arises from 1) how a system rebounds from disrupting events 2) its robustness (absorb perturbations) 3) its graceful extensibility (avoid sudden failure) 4) its sustained adaptability (governance policies to sustain function over changing conditions).

6. **Westrum [39]:** system property consisting of two of the following 1) Avoidance: anticipation of a mishap. 2) Survival: resist destruction. 3) Recovery: sometimes with reduced performance.

7. **Madni and Jackson [40]:** multi-faceted capability encompassing avoiding (anticipation), absorbing (robustness), adapting to (reconfiguration) and recovering (restoration) from disruptions.

8. **Francis and Bekera [41]:** capability of effectively combating disrupting events: 1) Anticipate and absorb: forecast triplets and prepare. 2) Adapt 3) Recover.

9. **Enos [42]:** ability of a system to react to and return to full function after an interruption to system operation.

10. **Neches and Madni [43]:** ability of a system to adapt affordably and perform effectively across a wide range of operational contexts, defined by mission, environment, threat, and force disposition.

11. **Neches and Madni [43]:** robustness that is achieved through thoughtful, informed design that makes systems both effective and reliable in a wide range of contexts.

12. **Chalupnik, Wynn and Clarkson [44]:** capacity of a system to tolerate disturbances while retaining its structure and function, focusing on a system's recovery from perturbation

13. **Hosseini, Barker and Ramirez-Marquez [32]:** ability of an entity or system to return to normal condition after the occurrence of an event that disrupts its state.

14. **Henry and Ramirez-Marquez [34]:** ability of an entity to recover from an external disruptive event.

15. **Tran et al. [45]:** ability to prepare and plan for, absorb, recover from, and more successfully adapt to adverse events.

16. **Uday, Chandrahasa and Marais [46]:** ability of a system, process or organization to react to, survive, and recover from undesired but not necessarily unplanned or unexpected changes (disruptions)

17. **Uday and Marais [155]:** ability of a system to survive and recover from changes, represented as a combination of survivability and recoverability.

18. **Farid [47]:** combination of a static 'survival' property, which measures the degree of performance after a disruption, and a dynamic 'recovery' property,

which measures how quickly the performance returns to normal operation.

19. **Ayyub [48]:** ability to prepare for and adapt to changing conditions and withstand and recover rapidly from disruptions. Include deliberate attack types, accidents, or naturally occurring threats or incidents. It can be measured based on the persistence of a corresponding functional performance under uncertainty in the face of disturbances.
20. **Jackson and Ferris [49]:** ability to adapt to changing conditions and prepare for, withstand, and rapidly recover from disruption.
21. **Uday and Marais [50]:** ability of a system or organization to react to and recover from disturbances at an early stage with minimal effect on its dynamic stability, usually represented as a combination of survivability and recoverability.
22. **Jackson [51]:** ability of the system to withstand a major disruption within acceptable degradation parameters and to recover within an acceptable time and composite costs and risks.
23. **Martin-Breen and Anderies [52]:** ability of a system to resist external forces, shocks, and disturbances and can quickly return to its normal state.
24. **Whitson and Ramirez-Marquez [53]:** ability of a system to maintain function when shocked and the speed at which it recovers from to achieve a desired state.
25. **Haimes [54]:** ability of a system to withstand a major disruption within acceptable degradation parameters and to recover within an acceptable time and composite costs and risks.
26. **Jackson [55]:** combination of at least two of the following: avoidance, survival and recovery.

27. **Burch [56]:** ability of an architecture to support the functions necessary for mission success in spite of hostile action or adverse conditions. Higher probability, shorter periods of reduced capability, and across a wider range of scenarios, conditions, and threats.

28. **Youn, Hu and Wang [57]:** resilience is the degree of a passive survival rate (or reliability) plus a proactive survival rate (or restoration).

29. **Barker, Ramirez-Marquez and Rocco [156]:** resilience can be defined as the time dependent ratio of recovery over loss.

30. **Filippini and Silva [157]:** ability to resist, react and recover, in a word the resilience.

31. **Yodo and Wang [137]:** ability to survive and recover from the likelihood of damage due to disruptive events or mishaps. In engineering, speed of returning to equilibrium... how fast an engineered system can adapt to deviation following a misfortune....

32. **Mehrpouyan et al. [158]:** 'maintaining system functions despite the existence of failures'.

33. **Ferris [159]:** 'ability to provide required capability in the face of adversity'. What degradation and restorative possibilities are appropriate in the face of threats need to be determined.

34. **Ren et al. [160]:** 'system-level capability to adapt to a disturbance and then recover from the disturbance'.

35. **Yodo and Wang [161]:** ability to anticipate, respond, and adapt to changes in order to reduce the magnitude and duration of disruptive events.

    - 'the ability of the system to continue with its intended functionalities without failure by itself does not define resiliency behaviour in a system. A resilient system is also a system that should be able to recover the system misfortunes to its prime operating state, given adequate resources and time'.

36. **Raz and Kenley [162]:** system's ability to recover from internal failures, faults, errors the effects of the evolving operational environment.

A.3 Reliability

1. **Patriarca et al. [33]:** property of a system consisting of having an acceptably low probability of failure.
    - Reliable if the failure probability is acceptable low while (in contrast) something is resilient if it has the ability to recover from irregular variations, disruptions, and degradation of expected working conditions
2. **Madni and Jackson [40]:** ability of the system to perform required functions under stated conditions for a specified period of time.
    - Resilience offers a vastly different approach; it creates safety by anticipating and planning for the unexpected. From this perspective, safety becomes a dynamic capability that requires reinforcement and investment on an ongoing basis.
3. **Francis and Bekera [41]:** *no definition*.
    - System robustness and reliability are prototypical pre-disruption characteristics of a resilient system
4. **Enos [42]:** probability that a system or component will satisfy its requirements over a given period of time and under given conditions.
    - Linked to repairability which in turn helps to achieve resilience.
5. **Chalupnik, Wynn and Clarkson [44]:** probability that a system will not fail under the full range of operating conditions, including both expected unexpected conditions.
    - This document view resilience as something similar to reliability including only unexpected conditions under varying environment and requirements.

6. **Hosseini, Barker and Ramirez-Marquez [32]:** *no definition*.
    - Engineering resilience as the sum of the passive survival rate (reliability) and proactive survival rate (restoration) of a system.
7. **Tran et al. [45]:** *no definition*.
    - Traditional systems engineering: '...resist disruptions through classical reliability methods, such as redundancy at the component level'. Resilience, sustaining and recovering critical system functions often through adaptation.
    - Focusing on known and expected threats (e.g. through robust design, vulnerability assessment, and reliability engineering) leaves a system dangerously susceptible to unanticipated ones.
8. **Uday and Marais [50]:** ability of a system and its components to perform required functions under stated conditions for a specified period of time.
    - Resilience is a function of system properties such as component reliability or reconfigurability.
    - Simple systems and components: resilience and reliability equivalent. Complex systems and SoSs: the difference grows with complexity.
    - Simplifying hypothesis of traditional reliability and risk approaches become less and less reasonable as complexity augments.
9. **Ayyub [48]:** *no definition*.
    - Some resilience metrics as a function of reliability.
10. **Jackson and Ferris [49]:** *no definition*.
    - Reliability is relevant for implementing physical redundancy to improve resilience. But resilience considers a broader spectrum of failure modes than traditional reliability analysis.

11. **Whitson and Ramirez-Marquez [53]:** probability that the system performs its intended function, for a specific mission time, under normal and known operating conditions.
    - Resiliency quantifies the probability that the system performs its intended function, for a specific mission time, when in the presence of external causes.
12. **Jackson [55]:** *no definition*.
    - An unreliable system will have a difficult time during the survival phase of resilience. First line against disruptions, but it is not the whole story.
13. **Burch [56]:** *no definition*.
    - The key difference is that reliability is defined by the system's response to failures internal to the system while resilience is defined by the system's response to external threats or conditions.
14. **Youn, Hu and Wang [57]:** ability of an engineered system to maintain its capacity and performance above a safety limit during a given period of time under stated conditions.
    - Engineering resilience as the sum of the passive survival rate (reliability) and proactive survival rate (restoration) of a system.
15. **Richards et al. [63]:** *no definition*.
    - To be resilient, systems must not only be reliable but also able to recover from disturbances.
16. **ARP 4761 [58, p. 10]:** 'the probability that an item will perform a required function under specified conditions, without failure, for a specified period of time'.

17. **Leveson [59]:** probability that the behaviour of a component will satisfy its specified behavioural requirements over time and under given conditions.

Appendix B

Fault Tree Analysis Reports

This chapter presents the fault tree analysis results used for evaluating the research. Section 2.5 provides more information regarding the meaning of the results.

FTA Analysis Results - FTAView

TOP Event

- `1.011E-006`

Minimal Cut Sets (relative probability)

- [Mixerunit (S35)] `P: 9.891E-004`
- [Cabin (S33)] `P: 9.891E-004`
- [ZoneController (S50)] `P: 9.891E-004`
- [TrimAirValve (S34)] `P: 9.891E-004`
- [PackFlowControlValve (S48)] `P: 9.891E-004`
- [CompressorTemperatureSensor (S44)] `P: 9.891E-004`
- [TemperatureSensor (S45)] `P: 9.891E-004`
- [TurbineBypassValve (S43)] `P: 9.891E-004`
- [PackController (S51)] `P: 9.891E-004`
- [FanInlet (S57)] `P: 9.891E-004`
- [Precooler (S54)] `P: 9.891E-001`
- [PressureRegulatorValve (S53)] `P: 9.891E-004`
- [IntermediatePressureInlet (S58),HighPressureInlet (S59)] `P: 9.891E-013`

Minimal Cut Sets Filtered

- [Mixerunit (S35)]
- [Cabin (S33)]
- [ZoneController (S50)]
- [TrimAirValve (S34)]
- [PackFlowControlValve (S48)]
- [CompressorTemperatureSensor (S44)]
- [TemperatureSensor (S45)]
- [TurbineBypassValve (S43)]
- [PackController (S51)]
- [FanInlet (S57)]
- [Precooler (S54)]
- [PressureRegulatorValve (S53)]
- [IntermediatePressureInlet (S58),HighPressureInlet (S59)]

Fussell-Vesely Ranking

1. Precooler (S54) : `9.891E-001`
2. PressureRegulatorValve (S53) : `9.891E-004`
3. FanInlet (S57) : `9.891E-004`
4. ZoneController (S50) : `9.891E-004`
5. CompressorTemperatureSensor (S44) : `9.891E-004`
6. TemperatureSensor (S45) : `9.891E-004`
7. TurbineBypassValve (S43) : `9.891E-004`
8. PackController (S51) : `9.891E-004`

9. PackFlowControlValve (S48) : `9.891E-004`
10. TrimAirValve (S34) : `9.891E-004`
11. Cabin (S33) : `9.891E-004`
12. Mixerunit (S35) : `9.891E-004`
13. IntermediatePressureInlet (S58) : `9.891E-013`
14. HighPressureInlet (S59) : `9.891E-013`
15. PrimaryHeatExchanger (S38) : `0.000E+000`
16. Compressor (S41) : `0.000E+000`
17. RamAirInlet (S37) : `0.000E+000`
18. MainHeatExchanger (S39) : `0.000E+000`
19. Turbine (S42) : `0.000E+000`
20. ConnectorJoint2 (S46) : `0.000E+000`

Birnbaum Ranking

1. PressureRegulatorValve (S53) : `1.000E+000`
2. FanInlet (S57) : `1.000E+000`
3. Precooler (S54) : `1.000E+000`
4. ZoneController (S50) : `1.000E+000`
5. CompressorTemperatureSensor (S44) : `1.000E+000`
6. TemperatureSensor (S45) : `1.000E+000`
7. TurbineBypassValve (S43) : `1.000E+000`
8. PackController (S51) : `1.000E+000`
9. PackFlowControlValve (S48) : `1.000E+000`
10. TrimAirValve (S34) : `1.000E+000`
11. Cabin (S33) : `1.000E+000`
12. Mixerunit (S35) : `1.000E+000`
13. IntermediatePressureInlet (S58) : `1.000E-009`
14. HighPressureInlet (S59) : `1.000E-009`
15. PrimaryHeatExchanger (S38) : `0.000E+000`
16. Compressor (S41) : `0.000E+000`
17. RamAirInlet (S37) : `0.000E+000`
18. MainHeatExchanger (S39) : `0.000E+000`
19. Turbine (S42) : `0.000E+000`
20. ConnectorJoint2 (S46) : `0.000E+000`

Minimal Path Sets

- [IntermediatePressureInlet (S58),Mixerunit (S35),Cabin (S33),ZoneController (S50),TrimAirValve (S34),CompressorTemperatureSensor (S44),PackFlowControlValve (S48),TemperatureSensor (S45),TurbineBypassValve (S43),PackController (S51),FanInlet (S57),Precooler (S54),PressureRegulatorValve (S53)]
- [Mixerunit (S35),Cabin (S33),ZoneController (S50),TrimAirValve (S34),CompressorTemperatureSensor (S44),PackFlowControlValve (S48),TemperatureSensor (S45),TurbineBypassValve (S43),PackController (S51),FanInlet (S57),Precooler (S54),PressureRegulatorValve (S53),HighPressureInlet (S59)]

Minimal Path Sets Filtered

- [IntermediatePressureInlet (S58),Mixerunit (S35),Cabin (S33),ZoneController (S50),TrimAirValve (S34),CompressorTemperatureSensor (S44),PackFlowControlValve (S48),TemperatureSensor (S45),TurbineBypassValve (S43),PackController (S51),FanInlet (S57),Precooler (S54),PressureRegulatorValve (S53)]
- [Mixerunit (S35),Cabin (S33),ZoneController (S50),TrimAirValve (S34),CompressorTemperatureSensor (S44),PackFlowControlValve (S48),TemperatureSensor (S45),TurbineBypassValve (S43),PackController (S51),FanInlet (S57),Precooler (S54),PressureRegulatorValve (S53),HighPressureInlet (S59)]

FTA Analysis Results - FTAView

TOP Event

- `1.010E-006`

Minimal Cut Sets (relative probability)

- [Mixerunit (S35)] `P: 9.901E-004`
- [Cabin (S33)] `P: 9.901E-004`
- [ZoneController (S50)] `P: 9.901E-004`
- [TrimAirValve (S34)] `P: 9.901E-004`
- [PackFlowControlValve (S48)] `P: 9.901E-004`
- [CompressorTemperatureSensor (S44)] `P: 9.901E-004`
- [TemperatureSensor (S45)] `P: 9.901E-004`
- [TurbineBypassValve (S43)] `P: 9.901E-004`
- [PackController (S51)] `P: 9.901E-004`
- [Precooler (S54)] `P: 9.901E-001`
- [PressureRegulatorValve (S53)] `P: 9.901E-004`
- [FanInlet (S57),FanInlet2 (S62)] `P: 9.901E-013`
- [IntermediatePressureInlet (S58),IntermediatePressureInlet2 (S64),HighPressureInlet (S59),HighPressureInlet2 (S63)] `P: 0.000E+000`

Minimal Cut Sets Filtered

- [Mixerunit (S35)]
- [Cabin (S33)]
- [ZoneController (S50)]
- [TrimAirValve (S34)]
- [PackFlowControlValve (S48)]
- [CompressorTemperatureSensor (S44)]
- [TemperatureSensor (S45)]
- [TurbineBypassValve (S43)]
- [PackController (S51)]
- [Precooler (S54)]
- [PressureRegulatorValve (S53)]
- [FanInlet (S57),FanInlet2 (S62)]
- [IntermediatePressureInlet (S58),IntermediatePressureInlet2 (S64),HighPressureInlet (S59),HighPressureInlet2 (S63)]

Fussell-Vesely Ranking

1. Precooler (S54) : `9.901E-001`
2. PressureRegulatorValve (S53) : `9.901E-004`
3. ZoneController (S50) : `9.901E-004`
4. CompressorTemperatureSensor (S44) : `9.901E-004`
5. TemperatureSensor (S45) : `9.901E-004`
6. TurbineBypassValve (S43) : `9.901E-004`

7. PackController (S51) : `9.901E-004`
8. PackFlowControlValve (S48) : `9.901E-004`
9. TrimAirValve (S34) : `9.901E-004`
10. Cabin (S33) : `9.901E-004`
11. Mixerunit (S35) : `9.901E-004`
12. FanInlet (S57) : `9.901E-013`
13. FanInlet2 (S62) : `9.901E-013`
14. IntermediatePressureInlet (S58) : `0.000E+000`
15. IntermediatePressureInlet2 (S64) : `0.000E+000`
16. HighPressureInlet (S59) : `0.000E+000`
17. HighPressureInlet2 (S63) : `0.000E+000`
18. PrimaryHeatExchanger (S38) : `0.000E+000`
19. Compressor (S41) : `0.000E+000`
20. RamAirInlet (S37) : `0.000E+000`
21. MainHeatExchanger (S39) : `0.000E+000`
22. Turbine (S42) : `0.000E+000`
23. ConnectorJoint2 (S46) : `0.000E+000`

Birnbaum Ranking

1. PressureRegulatorValve (S53) : `1.000E+000`
2. Precooler (S54) : `1.000E+000`
3. ZoneController (S50) : `1.000E+000`
4. CompressorTemperatureSensor (S44) : `1.000E+000`
5. TemperatureSensor (S45) : `1.000E+000`
6. TurbineBypassValve (S43) : `1.000E+000`
7. PackController (S51) : `1.000E+000`
8. PackFlowControlValve (S48) : `1.000E+000`
9. TrimAirValve (S34) : `1.000E+000`
10. Cabin (S33) : `1.000E+000`
11. Mixerunit (S35) : `1.000E+000`
12. FanInlet (S57) : `1.000E-009`
13. FanInlet2 (S62) : `1.000E-009`
14. PrimaryHeatExchanger (S38) : `0.000E+000`
15. Compressor (S41) : `0.000E+000`
16. RamAirInlet (S37) : `0.000E+000`
17. MainHeatExchanger (S39) : `0.000E+000`
18. Turbine (S42) : `0.000E+000`
19. ConnectorJoint2 (S46) : `0.000E+000`
20. IntermediatePressureInlet (S58) : `-1.005E-014`
21. IntermediatePressureInlet2 (S64) : `-1.005E-014`
22. HighPressureInlet (S59) : `-1.005E-014`
23. HighPressureInlet2 (S63) : `-1.005E-014`

Minimal Path Sets

- [IntermediatePressureInlet (S58),Mixerunit (S35),Cabin (S33),ZoneController (S50),TrimAirValve (S34),CompressorTemperatureSensor (S44),PackFlowControlValve

(S48),TemperatureSensor (S45),TurbineBypassValve (S43),PackController (S51),FanInlet (S57),Precooler (S54),PressureRegulatorValve (S53)]

- [IntermediatePressureInlet (S58),Mixerunit (S35),Cabin (S33),ZoneController (S50),TrimAirValve (S34),CompressorTemperatureSensor (S44),PackFlowControlValve (S48),TemperatureSensor (S45),TurbineBypassValve (S43),PackController (S51),Precooler (S54),FanInlet2 (S62),PressureRegulatorValve (S53)]
- [Mixerunit (S35),Cabin (S33),ZoneController (S50),TrimAirValve (S34),CompressorTemperatureSensor (S44),PackFlowControlValve (S48),TemperatureSensor (S45),TurbineBypassValve (S43),PackController (S51),FanInlet (S57),Precooler (S54),PressureRegulatorValve (S53),IntermediatePressureInlet2 (S64)]
- [Mixerunit (S35),Cabin (S33),ZoneController (S50),TrimAirValve (S34),CompressorTemperatureSensor (S44),PackFlowControlValve (S48),TemperatureSensor (S45),TurbineBypassValve (S43),PackController (S51),FanInlet (S57),Precooler (S54),PressureRegulatorValve (S53),HighPressureInlet (S59)]
- [Mixerunit (S35),Cabin (S33),ZoneController (S50),TrimAirValve (S34),CompressorTemperatureSensor (S44),PackFlowControlValve (S48),TemperatureSensor (S45),TurbineBypassValve (S43),PackController (S51),FanInlet (S57),Precooler (S54),PressureRegulatorValve (S53),HighPressureInlet2 (S63)]
- [Mixerunit (S35),Cabin (S33),ZoneController (S50),TrimAirValve (S34),CompressorTemperatureSensor (S44),PackFlowControlValve (S48),TemperatureSensor (S45),TurbineBypassValve (S43),PackController (S51),Precooler (S54),FanInlet2 (S62),PressureRegulatorValve (S53),IntermediatePressureInlet2 (S64)]
- [Mixerunit (S35),Cabin (S33),ZoneController (S50),TrimAirValve (S34),CompressorTemperatureSensor (S44),PackFlowControlValve (S48),TemperatureSensor (S45),TurbineBypassValve (S43),PackController (S51),Precooler (S54),FanInlet2 (S62),PressureRegulatorValve (S53),HighPressureInlet (S59)]
- [Mixerunit (S35),Cabin (S33),ZoneController (S50),TrimAirValve (S34),CompressorTemperatureSensor (S44),PackFlowControlValve (S48),TemperatureSensor (S45),TurbineBypassValve (S43),PackController (S51),Precooler (S54),FanInlet2 (S62),PressureRegulatorValve (S53),HighPressureInlet2 (S63)]

Minimal Path Sets Filtered

- [IntermediatePressureInlet (S58),Mixerunit (S35),Cabin (S33),ZoneController (S50),TrimAirValve (S34),CompressorTemperatureSensor (S44),PackFlowControlValve (S48),TemperatureSensor (S45),TurbineBypassValve (S43),PackController (S51),FanInlet (S57),Precooler (S54),PressureRegulatorValve (S53)]
- [IntermediatePressureInlet (S58),Mixerunit (S35),Cabin (S33),ZoneController (S50),TrimAirValve (S34),CompressorTemperatureSensor (S44),PackFlowControlValve (S48),TemperatureSensor (S45),TurbineBypassValve (S43),PackController (S51),Precooler (S54),FanInlet2 (S62),PressureRegulatorValve (S53)]
- [Mixerunit (S35),Cabin (S33),ZoneController (S50),TrimAirValve (S34),CompressorTemperatureSensor (S44),PackFlowControlValve (S48),TemperatureSensor (S45),TurbineBypassValve (S43),PackController (S51),FanInlet (S57),Precooler (S54),PressureRegulatorValve (S53),IntermediatePressureInlet2 (S64)]
- [Mixerunit (S35),Cabin (S33),ZoneController (S50),TrimAirValve (S34),CompressorTemperatureSensor (S44),PackFlowControlValve

(S48),TemperatureSensor (S45),TurbineBypassValve (S43),PackController (S51),FanInlet (S57),Precooler (S54),PressureRegulatorValve (S53),HighPressureInlet (S59)]

- [Mixerunit (S35),Cabin (S33),ZoneController (S50),TrimAirValve (S34),CompressorTemperatureSensor (S44),PackFlowControlValve (S48),TemperatureSensor (S45),TurbineBypassValve (S43),PackController (S51),FanInlet (S57),Precooler (S54),PressureRegulatorValve (S53),HighPressureInlet2 (S63)]
- [Mixerunit (S35),Cabin (S33),ZoneController (S50),TrimAirValve (S34),CompressorTemperatureSensor (S44),PackFlowControlValve (S48),TemperatureSensor (S45),TurbineBypassValve (S43),PackController (S51),Precooler (S54),FanInlet2 (S62),PressureRegulatorValve (S53),IntermediatePressureInlet2 (S64)]
- [Mixerunit (S35),Cabin (S33),ZoneController (S50),TrimAirValve (S34),CompressorTemperatureSensor (S44),PackFlowControlValve (S48),TemperatureSensor (S45),TurbineBypassValve (S43),PackController (S51),Precooler (S54),FanInlet2 (S62),PressureRegulatorValve (S53),HighPressureInlet (S59)]
- [Mixerunit (S35),Cabin (S33),ZoneController (S50),TrimAirValve (S34),CompressorTemperatureSensor (S44),PackFlowControlValve (S48),TemperatureSensor (S45),TurbineBypassValve (S43),PackController (S51),Precooler (S54),FanInlet2 (S62),PressureRegulatorValve (S53),HighPressureInlet2 (S63)]

FTA Analysis Results - FTAView

TOP Event

- `9.000E-009`

Minimal Cut Sets (relative probability)

- [Mixerunit (S35)] `P: 1.111E-001`
- [Cabin (S33)] `P: 1.111E-001`
- [TrimAirValve (S34)] `P: 1.111E-001`
- [ZoneController (S50)] `P: 1.111E-001`
- [PackFlowControlValve (S48)] `P: 1.111E-001`
- [CompressorTemperatureSensor (S44)] `P: 1.111E-001`
- [TemperatureSensor (S45)] `P: 1.111E-001`
- [TurbineBypassValve (S43)] `P: 1.111E-001`
- [PackController (S51)] `P: 1.111E-001`

Minimal Cut Sets Filtered

- [Mixerunit (S35)]
- [Cabin (S33)]
- [TrimAirValve (S34)]
- [ZoneController (S50)]
- [PackFlowControlValve (S48)]
- [CompressorTemperatureSensor (S44)]
- [TemperatureSensor (S45)]
- [TurbineBypassValve (S43)]
- [PackController (S51)]

Fussell-Vesely Ranking

1. ZoneController (S50) : `1.111E-001`
2. CompressorTemperatureSensor (S44) : `1.111E-001`
3. TemperatureSensor (S45) : `1.111E-001`
4. TurbineBypassValve (S43) : `1.111E-001`
5. PackController (S51) : `1.111E-001`
6. PackFlowControlValve (S48) : `1.111E-001`
7. TrimAirValve (S34) : `1.111E-001`
8. Cabin (S33) : `1.111E-001`
9. Mixerunit (S35) : `1.111E-001`
10. IntermediatePressureInlet (S58) : `0.000E+000`
11. IntermediatePressureInlet2 (S62) : `0.000E+000`
12. HighPressureInlet (S59) : `0.000E+000`
13. HighPressureInlet2 (S63) : `0.000E+000`
14. PressureRegulatorValve2 (S67) : `0.000E+000`
15. FanInlet (S57) : `0.000E+000`
16. FanInlet2 (S64) : `0.000E+000`

17. Precooler2 (S65) : `0.000E+000`
18. RamAirInlet2 (S69) : `0.000E+000`
19. MainHeatExchanger2 (S71) : `0.000E+000`
20. PrimaryHeatExchanger2 (S70) : `0.000E+000`
21. Turbine2 (S74) : `0.000E+000`
22. Compressor2 (S73) : `0.000E+000`
23. CompressorTemperatureSensor2 (S76) : `0.000E+000`
24. TemperatureSensor2 (S77) : `0.000E+000`
25. TurbineBypassValve2 (S75) : `0.000E+000`
26. ZoneController2 (S81) : `0.000E+000`
27. PackController2 (S79) : `0.000E+000`
28. PackFlowControlValve2 (S80) : `0.000E+000`
29. PrimaryHeatExchanger (S38) : `0.000E+000`
30. Compressor (S41) : `0.000E+000`
31. RamAirInlet (S37) : `0.000E+000`
32. MainHeatExchanger (S39) : `0.000E+000`
33. Turbine (S42) : `0.000E+000`
34. ConnectorJoint2 (S46) : `0.000E+000`
35. ConnectorJoint3 (S78) : `0.000E+000`

Birnbaum Ranking

1. IntermediatePressureInlet (S58) : `0.000E+000`
2. IntermediatePressureInlet2 (S62) : `0.000E+000`
3. HighPressureInlet (S59) : `0.000E+000`
4. HighPressureInlet2 (S63) : `0.000E+000`
5. PressureRegulatorValve2 (S67) : `0.000E+000`
6. FanInlet (S57) : `0.000E+000`
7. FanInlet2 (S64) : `0.000E+000`
8. Precooler2 (S65) : `0.000E+000`
9. RamAirInlet2 (S69) : `0.000E+000`
10. MainHeatExchanger2 (S71) : `0.000E+000`
11. PrimaryHeatExchanger2 (S70) : `0.000E+000`
12. Turbine2 (S74) : `0.000E+000`
13. Compressor2 (S73) : `0.000E+000`
14. CompressorTemperatureSensor2 (S76) : `0.000E+000`
15. TemperatureSensor2 (S77) : `0.000E+000`
16. TurbineBypassValve2 (S75) : `0.000E+000`
17. ZoneController2 (S81) : `0.000E+000`
18. PackController2 (S79) : `0.000E+000`
19. PackFlowControlValve2 (S80) : `0.000E+000`
20. PrimaryHeatExchanger (S38) : `0.000E+000`
21. Compressor (S41) : `0.000E+000`
22. RamAirInlet (S37) : `0.000E+000`
23. MainHeatExchanger (S39) : `0.000E+000`
24. Turbine (S42) : `0.000E+000`
25. ConnectorJoint2 (S46) : `0.000E+000`
26. ConnectorJoint3 (S78) : `0.000E+000`

27. ZoneController (S50) : `-8.000E-009`
28. CompressorTemperatureSensor (S44) : `-8.000E-009`
29. TemperatureSensor (S45) : `-8.000E-009`
30. TurbineBypassValve (S43) : `-8.000E-009`
31. PackController (S51) : `-8.000E-009`
32. PackFlowControlValve (S48) : `-8.000E-009`
33. TrimAirValve (S34) : `-8.000E-009`
34. Cabin (S33) : `-8.000E-009`
35. Mixerunit (S35) : `-8.000E-009`

Minimal Path Sets

- [CompressorTemperatureSensor (S44),Mixerunit (S35),Cabin (S33),ZoneController (S50),TrimAirValve (S34),PackFlowControlValve (S48),TemperatureSensor (S45),TurbineBypassValve (S43),PackController (S51)]

Minimal Path Sets Filtered

- [CompressorTemperatureSensor (S44),Mixerunit (S35),Cabin (S33),ZoneController (S50),TrimAirValve (S34),PackFlowControlValve (S48),TemperatureSensor (S45),TurbineBypassValve (S43),PackController (S51)]

FTA Analysis Results - FTAView

TOP Event

- `9.001E-009`

Minimal Cut Sets (relative probability)

- [Mixerunit (S35)] `P: 1.111E-001`
- [Cabin (S33)] `P: 1.111E-001`
- [ZoneController (S50)] `P: 1.111E-001`
- [TrimAirValve (S34)] `P: 1.111E-001`
- [PackFlowControlValve (S48)] `P: 1.111E-001`
- [CompressorTemperatureSensor (S44)] `P: 1.111E-001`
- [TemperatureSensor (S45)] `P: 1.111E-001`
- [TurbineBypassValve (S43)] `P: 1.111E-001`
- [PackController (S51)] `P: 1.111E-001`
- [FanInlet (S57),FanInlet2 (S64)] `P: 1.111E-010`
- [Precooler2 (S65),Precooler (S54)] `P: 1.111E-004`
- [PressureRegulatorValve2 (S67),Precooler (S54)] `P: 1.111E-007`
- [Precooler2 (S65),PressureRegulatorValve (S53)] `P: 1.111E-007`
- [PressureRegulatorValve2 (S67),PressureRegulatorValve (S53)] `P: 1.111E-010`
- [IntermediatePressureInlet (S58),IntermediatePressureInlet2 (S62),HighPressureInlet (S59),HighPressureInlet2 (S63)] `P: 0.000E+000`

Minimal Cut Sets Filtered

- [Mixerunit (S35)]
- [Cabin (S33)]
- [ZoneController (S50)]
- [TrimAirValve (S34)]
- [PackFlowControlValve (S48)]
- [CompressorTemperatureSensor (S44)]
- [TemperatureSensor (S45)]
- [TurbineBypassValve (S43)]
- [PackController (S51)]
- [FanInlet (S57),FanInlet2 (S64)]
- [Precooler2 (S65),Precooler (S54)]
- [PressureRegulatorValve2 (S67),Precooler (S54)]
- [Precooler2 (S65),PressureRegulatorValve (S53)]
- [PressureRegulatorValve2 (S67),PressureRegulatorValve (S53)]
- [IntermediatePressureInlet (S58),IntermediatePressureInlet2 (S62),HighPressureInlet (S59),HighPressureInlet2 (S63)]

Fussell-Vesely Ranking

1. ZoneController (S50) : `1.000E+000`
2. CompressorTemperatureSensor (S44) : `1.000E+000`

3. TemperatureSensor (S45) : `1.000E+000`
4. TurbineBypassValve (S43) : `1.000E+000`
5. PackController (S51) : `1.000E+000`
6. PackFlowControlValve (S48) : `1.000E+000`
7. TrimAirValve (S34) : `1.000E+000`
8. Cabin (S33) : `1.000E+000`
9. Mixerunit (S35) : `1.000E+000`
10. PrimaryHeatExchanger (S38) : `0.000E+000`
11. Compressor (S41) : `0.000E+000`
12. RamAirInlet (S37) : `0.000E+000`
13. MainHeatExchanger (S39) : `0.000E+000`
14. Turbine (S42) : `0.000E+000`
15. ConnectorJoint2 (S46) : `0.000E+000`
16. IntermediatePressureInlet (S58) : `-1.111E-019`
17. IntermediatePressureInlet2 (S62) : `-1.111E-019`
18. HighPressureInlet (S59) : `-1.111E-019`
19. HighPressureInlet2 (S63) : `-1.111E-019`
20. FanInlet (S57) : `-1.111E-001`
21. FanInlet2 (S64) : `-1.111E-001`
22. Precooler (S54) : `-1.112E+002`
23. Precooler2 (S65) : `-1.112E+002`
24. PressureRegulatorValve (S53) : `-1.112E+002`
25. PressureRegulatorValve2 (S67) : `-1.112E+002`

Birnbaum Ranking

1. FanInlet (S57) : `1.200E-025`
2. FanInlet2 (S64) : `1.200E-025`
3. IntermediatePressureInlet (S58) : `8.400E-026`
4. IntermediatePressureInlet2 (S62) : `8.400E-026`
5. HighPressureInlet (S59) : `8.400E-026`
6. HighPressureInlet2 (S63) : `8.400E-026`
7. PrimaryHeatExchanger (S38) : `0.000E+000`
8. Compressor (S41) : `0.000E+000`
9. RamAirInlet (S37) : `0.000E+000`
10. MainHeatExchanger (S39) : `0.000E+000`
11. Turbine (S42) : `0.000E+000`
12. ConnectorJoint2 (S46) : `0.000E+000`
13. ZoneController (S50) : `-5.600E-026`
14. CompressorTemperatureSensor (S44) : `-5.600E-026`
15. TemperatureSensor (S45) : `-5.600E-026`
16. TurbineBypassValve (S43) : `-5.600E-026`
17. PackController (S51) : `-5.600E-026`
18. PackFlowControlValve (S48) : `-5.600E-026`
19. TrimAirValve (S34) : `-5.600E-026`
20. Cabin (S33) : `-5.600E-026`
21. Mixerunit (S35) : `-5.600E-026`
22. Precooler (S54) : `-1.105E-014`

23. Precooler2 (S65) : `-1.105E-014`
24. PressureRegulatorValve (S53) : `-1.011E-012`
25. PressureRegulatorValve2 (S67) : `-1.011E-012`

Minimal Path Sets

- [IntermediatePressureInlet (S58),Mixerunit (S35),Cabin (S33),ZoneController (S50),TrimAirValve (S34),CompressorTemperatureSensor (S44),PackFlowControlValve (S48),TemperatureSensor (S45),TurbineBypassValve (S43),PackController (S51),FanInlet (S57),Precooler (S54),PressureRegulatorValve (S53)]
- [Mixerunit (S35),Cabin (S33),ZoneController (S50),TrimAirValve (S34),CompressorTemperatureSensor (S44),PackFlowControlValve (S48),TemperatureSensor (S45),TurbineBypassValve (S43),PackController (S51),IntermediatePressureInlet (S58),FanInlet (S57),Precooler2 (S65),PressureRegulatorValve2 (S67)]
- [Mixerunit (S35),Cabin (S33),ZoneController (S50),TrimAirValve (S34),CompressorTemperatureSensor (S44),PackFlowControlValve (S48),TemperatureSensor (S45),TurbineBypassValve (S43),PackController (S51),IntermediatePressureInlet (S58),Precooler2 (S65),FanInlet2 (S64),PressureRegulatorValve2 (S67)]
- [Mixerunit (S35),Cabin (S33),ZoneController (S50),TrimAirValve (S34),CompressorTemperatureSensor (S44),PackFlowControlValve (S48),TemperatureSensor (S45),TurbineBypassValve (S43),PackController (S51),FanInlet (S57),Precooler2 (S65),PressureRegulatorValve2 (S67),IntermediatePressureInlet2 (S62)]
- [Mixerunit (S35),Cabin (S33),ZoneController (S50),TrimAirValve (S34),CompressorTemperatureSensor (S44),PackFlowControlValve (S48),TemperatureSensor (S45),TurbineBypassValve (S43),PackController (S51),FanInlet (S57),Precooler2 (S65),PressureRegulatorValve2 (S67),HighPressureInlet (S59)]
- [Mixerunit (S35),Cabin (S33),ZoneController (S50),TrimAirValve (S34),CompressorTemperatureSensor (S44),PackFlowControlValve (S48),TemperatureSensor (S45),TurbineBypassValve (S43),PackController (S51),FanInlet (S57),Precooler2 (S65),PressureRegulatorValve2 (S67),HighPressureInlet2 (S63)]
- [Mixerunit (S35),Cabin (S33),ZoneController (S50),TrimAirValve (S34),CompressorTemperatureSensor (S44),PackFlowControlValve (S48),TemperatureSensor (S45),TurbineBypassValve (S43),PackController (S51),Precooler2 (S65),FanInlet2 (S64),PressureRegulatorValve2 (S67),IntermediatePressureInlet2 (S62)]
- [Mixerunit (S35),Cabin (S33),ZoneController (S50),TrimAirValve (S34),CompressorTemperatureSensor (S44),PackFlowControlValve (S48),TemperatureSensor (S45),TurbineBypassValve (S43),PackController (S51),Precooler2 (S65),FanInlet2 (S64),PressureRegulatorValve2 (S67),HighPressureInlet (S59)]
- [Mixerunit (S35),Cabin (S33),ZoneController (S50),TrimAirValve (S34),CompressorTemperatureSensor (S44),PackFlowControlValve (S48),TemperatureSensor (S45),TurbineBypassValve (S43),PackController (S51),Precooler2 (S65),FanInlet2 (S64),PressureRegulatorValve2 (S67),HighPressureInlet2 (S63)]
- [IntermediatePressureInlet (S58),Mixerunit (S35),Cabin (S33),ZoneController (S50),TrimAirValve (S34),CompressorTemperatureSensor (S44),PackFlowControlValve (S48),TemperatureSensor (S45),TurbineBypassValve (S43),PackController (S51),Precooler (S54),FanInlet2 (S64),PressureRegulatorValve (S53)]

- [Mixerunit (S35),Cabin (S33),ZoneController (S50),TrimAirValve (S34),CompressorTemperatureSensor (S44),PackFlowControlValve (S48),TemperatureSensor (S45),TurbineBypassValve (S43),PackController (S51),FanInlet (S57),Precooler (S54),PressureRegulatorValve (S53),IntermediatePressureInlet2 (S62)]
- [Mixerunit (S35),Cabin (S33),ZoneController (S50),TrimAirValve (S34),CompressorTemperatureSensor (S44),PackFlowControlValve (S48),TemperatureSensor (S45),TurbineBypassValve (S43),PackController (S51),FanInlet (S57),Precooler (S54),PressureRegulatorValve (S53),HighPressureInlet (S59)]
- [Mixerunit (S35),Cabin (S33),ZoneController (S50),TrimAirValve (S34),CompressorTemperatureSensor (S44),PackFlowControlValve (S48),TemperatureSensor (S45),TurbineBypassValve (S43),PackController (S51),FanInlet (S57),Precooler (S54),PressureRegulatorValve (S53),HighPressureInlet2 (S63)]
- [Mixerunit (S35),Cabin (S33),ZoneController (S50),TrimAirValve (S34),CompressorTemperatureSensor (S44),PackFlowControlValve (S48),TemperatureSensor (S45),TurbineBypassValve (S43),PackController (S51),Precooler (S54),FanInlet2 (S64),PressureRegulatorValve (S53),IntermediatePressureInlet2 (S62)]
- [Mixerunit (S35),Cabin (S33),ZoneController (S50),TrimAirValve (S34),CompressorTemperatureSensor (S44),PackFlowControlValve (S48),TemperatureSensor (S45),TurbineBypassValve (S43),PackController (S51),Precooler (S54),FanInlet2 (S64),PressureRegulatorValve (S53),HighPressureInlet (S59)]
- [Mixerunit (S35),Cabin (S33),ZoneController (S50),TrimAirValve (S34),CompressorTemperatureSensor (S44),PackFlowControlValve (S48),TemperatureSensor (S45),TurbineBypassValve (S43),PackController (S51),Precooler (S54),FanInlet2 (S64),PressureRegulatorValve (S53),HighPressureInlet2 (S63)]

Minimal Path Sets Filtered

- [IntermediatePressureInlet (S58),Mixerunit (S35),Cabin (S33),ZoneController (S50),TrimAirValve (S34),CompressorTemperatureSensor (S44),PackFlowControlValve (S48),TemperatureSensor (S45),TurbineBypassValve (S43),PackController (S51),FanInlet (S57),Precooler (S54),PressureRegulatorValve (S53)]
- [Mixerunit (S35),Cabin (S33),ZoneController (S50),TrimAirValve (S34),CompressorTemperatureSensor (S44),PackFlowControlValve (S48),TemperatureSensor (S45),TurbineBypassValve (S43),PackController (S51),IntermediatePressureInlet (S58),FanInlet (S57),Precooler2 (S65),PressureRegulatorValve2 (S67)]
- [Mixerunit (S35),Cabin (S33),ZoneController (S50),TrimAirValve (S34),CompressorTemperatureSensor (S44),PackFlowControlValve (S48),TemperatureSensor (S45),TurbineBypassValve (S43),PackController (S51),IntermediatePressureInlet (S58),Precooler2 (S65),FanInlet2 (S64),PressureRegulatorValve2 (S67)]
- [Mixerunit (S35),Cabin (S33),ZoneController (S50),TrimAirValve (S34),CompressorTemperatureSensor (S44),PackFlowControlValve (S48),TemperatureSensor (S45),TurbineBypassValve (S43),PackController (S51),FanInlet (S57),Precooler2 (S65),PressureRegulatorValve2 (S67),IntermediatePressureInlet2 (S62)]
- [Mixerunit (S35),Cabin (S33),ZoneController (S50),TrimAirValve (S34),CompressorTemperatureSensor (S44),PackFlowControlValve

(S48),TemperatureSensor (S45),TurbineBypassValve (S43),PackController (S51),FanInlet (S57),Precooler2 (S65),PressureRegulatorValve2 (S67),HighPressureInlet (S59)]

- [Mixerunit (S35),Cabin (S33),ZoneController (S50),TrimAirValve (S34),CompressorTemperatureSensor (S44),PackFlowControlValve (S48),TemperatureSensor (S45),TurbineBypassValve (S43),PackController (S51),FanInlet (S57),Precooler2 (S65),PressureRegulatorValve2 (S67),HighPressureInlet2 (S63)]
- [Mixerunit (S35),Cabin (S33),ZoneController (S50),TrimAirValve (S34),CompressorTemperatureSensor (S44),PackFlowControlValve (S48),TemperatureSensor (S45),TurbineBypassValve (S43),PackController (S51),Precooler2 (S65),FanInlet2 (S64),PressureRegulatorValve2 (S67),IntermediatePressureInlet2 (S62)]
- [Mixerunit (S35),Cabin (S33),ZoneController (S50),TrimAirValve (S34),CompressorTemperatureSensor (S44),PackFlowControlValve (S48),TemperatureSensor (S45),TurbineBypassValve (S43),PackController (S51),Precooler2 (S65),FanInlet2 (S64),PressureRegulatorValve2 (S67),HighPressureInlet (S59)]
- [Mixerunit (S35),Cabin (S33),ZoneController (S50),TrimAirValve (S34),CompressorTemperatureSensor (S44),PackFlowControlValve (S48),TemperatureSensor (S45),TurbineBypassValve (S43),PackController (S51),Precooler2 (S65),FanInlet2 (S64),PressureRegulatorValve2 (S67),HighPressureInlet2 (S63)]
- [IntermediatePressureInlet (S58),Mixerunit (S35),Cabin (S33),ZoneController (S50),TrimAirValve (S34),CompressorTemperatureSensor (S44),PackFlowControlValve (S48),TemperatureSensor (S45),TurbineBypassValve (S43),PackController (S51),Precooler (S54),FanInlet2 (S64),PressureRegulatorValve (S53)]
- [Mixerunit (S35),Cabin (S33),ZoneController (S50),TrimAirValve (S34),CompressorTemperatureSensor (S44),PackFlowControlValve (S48),TemperatureSensor (S45),TurbineBypassValve (S43),PackController (S51),FanInlet (S57),Precooler (S54),PressureRegulatorValve (S53),IntermediatePressureInlet2 (S62)]
- [Mixerunit (S35),Cabin (S33),ZoneController (S50),TrimAirValve (S34),CompressorTemperatureSensor (S44),PackFlowControlValve (S48),TemperatureSensor (S45),TurbineBypassValve (S43),PackController (S51),FanInlet (S57),Precooler (S54),PressureRegulatorValve (S53),HighPressureInlet (S59)]
- [Mixerunit (S35),Cabin (S33),ZoneController (S50),TrimAirValve (S34),CompressorTemperatureSensor (S44),PackFlowControlValve (S48),TemperatureSensor (S45),TurbineBypassValve (S43),PackController (S51),FanInlet (S57),Precooler (S54),PressureRegulatorValve (S53),HighPressureInlet2 (S63)]
- [Mixerunit (S35),Cabin (S33),ZoneController (S50),TrimAirValve (S34),CompressorTemperatureSensor (S44),PackFlowControlValve (S48),TemperatureSensor (S45),TurbineBypassValve (S43),PackController (S51),Precooler (S54),FanInlet2 (S64),PressureRegulatorValve (S53),IntermediatePressureInlet2 (S62)]
- [Mixerunit (S35),Cabin (S33),ZoneController (S50),TrimAirValve (S34),CompressorTemperatureSensor (S44),PackFlowControlValve (S48),TemperatureSensor (S45),TurbineBypassValve (S43),PackController (S51),Precooler (S54),FanInlet2 (S64),PressureRegulatorValve (S53),HighPressureInlet (S59)]
- [Mixerunit (S35),Cabin (S33),ZoneController (S50),TrimAirValve (S34),CompressorTemperatureSensor (S44),PackFlowControlValve (S48),TemperatureSensor (S45),TurbineBypassValve (S43),PackController (S51),Precooler (S54),FanInlet2 (S64),PressureRegulatorValve (S53),HighPressureInlet2 (S63)]

Appendix C

Evaluation Questionnaire

Building safety into the conceptual design of complex systems. Evaluation Questionnaire

Sergio Jimeno Altelarrea

Email: s.jimeno@cranfield.ac.uk
Telephone: 07341 957 208

* Required

Personal Data

1. Name (Optional).

2. Participant id number. *

Number must be between 100000 ~ 999999

3. Job title and department (Optional).

4. Years of relevant experience. *

Number must be between 0 ~ 100

5. I confirm that I have read and understand the information provided on the consent form and give my consent to taking part in this research.

Check this item to confirm	○

STPA Enabler

6. Please indicate to what extent you agree or disagree with the following statements by marking the appropriate box. *

	Strongly disagree	Disagree	Neutral	Agree	Strongly agree
The proposed enablers enhance the interactivity of the safety assessment process.	○	○	○	○	○
The proposed enablers result in a tighter integration between the architecture definition (in particular, the logical view) and the safety assessment models.	○	○	○	○	○
The approach of STPA combined with RFLP satisfy the industrial need for safety assessment capabilities.	○	○	○	○	○

Safety architecting enablers

7. Please indicate to what extent you agree or disagree with the following statements by marking the appropriate box. *

	Strongly disagree	Disagree	Neutral	Agree	Strongly agree
The proposed enablers enhance interactivity while architecting physical and functional redundancy, and containment.	○	○	○	○	○
The proposed enablers reduce the number of time-consuming manual activities required for architecting physical and functional redundancy, and containment.	○	○	○	○	○
Physical and functional redundancy, and containment cover to a sufficient extent the industrial requirement for safety principles.	○	○	○	○	○

FTA enabler

8. Please indicate to what extent you agree or disagree with the following statements by marking the appropriate box. *

	Strongly disagree	Disagree	Neutral	Agree	Strongly agree
The proposed enablers reduce the number of time-consuming manual activities for FTA creation.	○	○	○	○	○
The proposed enablers enhance interactivity of the exploration of FTA results.	○	○	○	○	○
The level of detail of the FTA created by this technique is commensurate with conceptual design.	○	○	○	○	○

Sizing enabler

9. Please indicate to what extent you agree or disagree with the following statements by marking the appropriate box. *

	Strongly disagree	Disagree	Neutral	Agree	Strongly agree
The proposed enablers reduce the number of time-consuming manual activities for sizing workflow creation.	○	○	○	○	○
The level of detail regarding context definition (configuration/scenarios and environmental conditions) is appropriate for conceptual design.	○	○	○	○	○

Overall framework (Efficiency and effectiveness)

10. Please indicate to what extent you agree or disagree with the following statements by marking the appropriate box. *

	Strongly disagree	Disagree	Neutral	Agree	Strongly agree
The techniques presented allow the exploration of more design alternatives during conceptual design, specifically regarding different ways of complying with safety requirements.	○	○	○	○	○
The additional exploration capabilities will contribute to a higher level of understanding regarding how to achieve safety and the effects on system performance.	○	○	○	○	○
An interactive software tool similar to the one presented, integrated with existing tools used in the company, would be useful and add value to the company.	○	○	○	○	○

Open questions

11. Is there anything that you particularly liked about any of the presented methods? Which one of them do you believe would be the most useful? How do you think the methods could add value in the conceptual design stage?

12. Is there anything that you did not particularly like about any of the presented methods? Any particular capability that the methods fail to support?

13. Regarding the certification process, how relevant is the early consideration of safety? How could these methods contribute towards the certification of aircraft in the future?

Appendix D

Informed Consent Form

Appendix E

Publications by the Author

The research in this thesis led to the following publications by the author:

- Sergio Jimeno, Arturo Molina-Cristobal, Atif Riaz and Marin Guenov. "Incorporating Safety in Early (Airframe) Systems Design and Assessment". *AIAA Scitech 2019 Forum*. San Diego, California. January 2019.

  DOI: 10.2514/6.2019-0553

- Sergio Jimeno, Atif Riaz, Marin Guenov and Arturo Molina-Cristobal. "Enabling Interactive Safety and Performance Trade-offs in Early Airframe Systems Design". *AIAA Scitech 2020 Forum*. Orlando, Florida. January 2020.

  DOI: 10.2514/6.2020-0550

Glossary

Accident Causation Model Model that explain how accidents happen and therefore determine how accidents are investigated, how the risk associated with existing products is assessed, and how safer systems are designed.

Barrier Component that stop the propagation of disturbances, such as a valve or a circuit breaker.

Computational Model Executable piece of computer code that describes part of the physical behaviour and other relevant characteristics (e.g. weight or cost) of a solution in the architecture.

Computational Workflow Ordered set of computational models, which are executed according to their order.

Containment Safety principle that states that the system should assure that failures cannot propagate from node to node.

Element Term that encompasses the fundamental parts of an architecture such as requirements, functions and solutions.

Fault Tree Analysis Inductive safety analysis technique based on fault trees. Analysis of the trees provides information regarding the probability of failure of the top event and the minimal cut sets.

Fault Tree Failure model that explains a particular system failure mode, called the top event, in terms of lower-level events.

Function What the system or parts of the system must do to meet the requirements

Functional Basis Formal function representation that describes functions as a combination of a flow and an operation on the flow. It proposes two vocabularies, flow and operation, from where the terms that describes the functions are obtained.

Functional Modelling The process of creating the functional model of a system, which describes the system in terms of the elementary functions that are required to achieve its overall purpose.

Functional Redundancy Safety principle that states that there should be two or more independent and physically different ways to perform a critical task.

Graph A graph G is a pair (V,E), where V is a finite set and E is a binary relation on V. The set V contains the vertices and the set E contains the edges. Vertices can be used to represent elements of an architecture and edges can be used to represent relations between the elements.

Graph Cut Subset of edges that, if removed, separates the graph that contains it into two disjoint sets as one set of vertices cannot be reached from the other.

Hazard Asesessment The examination of the system to identify safety-related risk and propose safety requirements.

Hierarchycal Control Structure Model of a system in terms of feedback control loops including controllers, control actions, feedback signals, controlled processes and other inputs and outputs that is used in STPA.

Minimal Cut Sets Minimal sets of components that, when all fail together, cause the system to fail.

Minimum Cut Cut whose capacity is minimum over all cuts of the grpah.

Minimum Cut Problem The problem of finding the minimun cut of a graph.

Model Based Systems Engineering The application of modelling to support system engineering activities.

Reliability The ability of the system to perform its required functions under expected conditions for a specified period of time, generally expressed as a probability.

Requirement Stakeholder need that a particular system aims to satisfy.

Resilience The ability of a system to maintain its functionality in the face of disruptive events. It is a dynamic capability, enabled by three sequential phases: survival phase, reconfiguration phase and recovery phase.

RFL Paradigm Systems architecting paradigm that assumes that systems engineering is distributed over four notional domains: Requirements, Functional, Logical and Physical.

Physical Redundancy Safety principle that states that the system should possess two or more independent and identical legs to perform critical tasks.

Safety The ability of a system not to cause, under given conditions, critical or catastrophic events. Catastrophic event are defined as those that can cause death, injury, occupational illness, damage to or loss of equipment or property, or damage to the environment.

Safety Analysis The analysis of the architecture of a system to verify that safety requirements are met.

Safety Principle General design strategy that is expected to result in a safer system and that can be applied to the designs of diverse systems.

Solution Component of the architecture that implements one or more functions of the architecture.

STPA Hazard asessment method based on systems and contorl theory that includes additional causes for accidents such as system design errors, human error (in more detail than just random failure) and various types of systemic accident causes

SysML General-purpose architecture modeling language for systems engineering applications. SysML is a dialect of UML.

Systems Engineering Transdisciplinary and integrative approach to enable the successful realization, use, and retirement of engineered systems, using systems principles and concepts, and scientific, technological, and management methods.

Template Objects that contain the information that allows the the consistent instantiation of architectural elements acoording to the type of element described in the template.

Traversal The systematic exploration of the edges of a graph to discover every vertex that is reachable from an initial vertex.

UML Modelling language whose purpose is to specify, visualize, and document models of software systems, including their structure and design, in a way that the system requirements can be met.