THE UNIVERSITY OF HULL

Generation of Model-Based Safety Arguments from Automatically Allocated Safety Integrity Levels

being a Thesis submitted for the Degree of

Doctor of Philosophy

in the University of Hull

by

Ioannis Sorokos BSc., MSc.

June 2017

For Pia, Nasos, Alexis, Oi and Carys…

Abstract

To certify safety-critical systems, assurance arguments linking evidence of safety to appropriate requirements must be constructed. However, modern safety-critical systems feature increasing complexity and integration, which render manual approaches impractical to apply. This thesis addresses this problem by introducing a model-based method, with an exemplary application based on the aerospace domain.

Previous work has partially addressed this problem for slightly different applications, including verification-based, COTS, product-line and process-based assurance. Each of the approaches is applicable to a specialised case and does not deliver a solution applicable to a generic system in a top-down process. This thesis argues that such a solution is feasible and can be achieved based on the automatic allocation of safety requirements onto a system's architecture. This automatic allocation is a recent development which combines model-based safety analysis and optimisation techniques. The proposed approach emphasises the use of model-based safety analysis, such as HiP-HOPS, to maximise the benefits towards the system development lifecycle.

The thesis investigates the background and earlier work regarding construction of safety arguments, safety requirements allocation and optimisation. A method for addressing the problem of optimal safety requirements allocation is first introduced, using the Tabu Search optimisation metaheuristic. The method delivers satisfactory results that are further exploited for construction of safety arguments. Using the produced requirements allocation, an instantiation algorithm is applied onto a generic safety argument pattern, which is compliant with standards, to automatically construct an argument establishing a claim that a system's safety requirements have been met. This argument is hierarchically decomposed and shows how system and subsystem safety requirements are satisfied by architectures and analyses at low levels of decomposition. Evaluation on two abstract case studies demonstrates the feasibility and scalability of the method and indicates good performance of the algorithms proposed. Limitations and potential areas of further investigation are identified.

Acknowledgements

Reflecting on the path that led to the creation of this thesis, I distinctly remember the times I found myself questioning whether I should continue pushing forward towards its conclusion. In those difficult moments, I turned to my friends and family for support and, as always, they lent me inspiration, encouragement, cheer and love that enabled me to complete this work.

The journey towards this thesis began when I first met Prof. Yiannis Papadopoulos in 2012. We quickly developed a level of communication and trust which I had seldom found in my previous academic experience. Yiannis introduced me to his field of research and captured my interest in it as well. Upon completion of my MSc, he offered me a chance to pursue a PhD, an idea which I had not considered before. Despite that, his enthusiasm for his work infected me and I gladly accepted the invitation. Yiannis has guided me wisely throughout this project, advising in matters both academic and personal. After each meeting we have had, I have always departed with uplifted spirits and renewed confidence in pursuing my research. For all his ongoing support, I am forever grateful.

I was also fortunate to be supported by Dr. David Parker. David's work has never failed to impress me, whether it was regarding the quality of his writing or application development. His attention to detail and reliability has been motivational and of great help throughout.

I would also like to thank some of my university friends and colleagues which I became more familiar with over the course of these past years. Ivan Hristov, Manos Founariotakis, Johnny McCullough, Angel Munoz, Dr. Sohag Kabir, Dr. Martin Walker, Dr. Septavera Sharvia, Dr. Zhibao Mian, Sahar Arsi, Ali Sa, Dr. Francisco Inacio, Dr. Martin Vojtisek, Radu Diaconesea, Luis Torrao, Dr. John Dixon, Dr. Youcef Gheraibia, Becky Lavelle and John Stamford have all helped make the department a positive, friendly environment I am always happy to return to.

As I began my research, I met Dr. Luis Azevedo, Dr. Kreshnik Hoxha and Dr. Flore-Anne Poujade during their PhD studentships. They supported me with their friendship and inspired me to join them in their pursuit. In particular, Luis' work on automatic ASIL decomposition has been essential to the current thesis. However, it is his strength of character and dedication to his friends and family that makes me feel privileged to be counted among those. Luis', Kreshnik's and Flore-Anne's joie-de-vivre has been a source of many joyful moments, travels and experiences we shared which helped make Hull feel like a second home.

I have been lucky to have been joined during my stay in Hull by my dear friends Kostas and Anna-Maria as well as my sister Pia and her partner Nasos. They brought with them a part of Greece, of home and of family and shared it with me. Together, we took part in many happy memories and unique moments that I shall cherish for years to come.

My friends from Greece have been providing moral support throughout my studies. Kostas, Nikos, Vagelis, Lefteris, Michalis, Pavlos and Harry always cheer me up and make me feel appreciated when I visit home.

My parents, Alexis and Oi, have always been eager to support me and push me towards a better future. I have not known more powerful love than theirs. My hope is that I can reward their unconditional love by achieving the happiness they wish for me and my sister.

The final person I would like to thank also happens to be the one that supported me the most during my final year of writing. She always finds a way of bringing a smile to my lips, caring for my well-being and happiness. In her I found not only a person to love but a reason to love myself as well. Carys, I can never repay you enough.

Contents

Table of Figures

Tables

Glossary

Acronym	Meaning	Page w/ definition
ASIL	Automotive Safety Integrity Level	62
CAE	Claims Arguments Evidence Notation	23
DAL	Development Assurance Level	95
FAA	Federal Aviation Administration	63
FFS	Functional Failure Set	98
FHA	Functional Hazard Analysis	62
FMEA	Failure Modes and Effects Analysis	72
FTA	Fault Tree Analysis	69
GSN	Goal-Structuring Notation	23
HiP-HOPS	Hierarchically Performed Hazard Origins Propagation Studies	29
MCS	Minimal Cut Set	98
SAE	Society of Automotive Engineers	64
SIL	Safety Integrity Level	26

Chapter 1: Introduction

1. Motivation

Electronic and computer based systems have contributed significantly to the improvement of our quality of life. To justify their continued application and extend it to new areas, we need to provide confidence that any significant dangers associated with their operation have been dealt with appropriately. Major accidents of the past have shown that despite best engineering efforts, the possibility of a life-threatening accident can never be eliminated. We need systems to be safe to use, to protect their users, society and the environment against such accidents.

2. Safety Assessment and Assurance Fundamentals

Safety-critical systems can be defined as systems whose malfunction presents a considerable danger to the public interest. Depending on the category of system, its field of application and scale, the potential effect of the potential dangers can vary. A danger could involve, for example, injury or loss of human life or catastrophic destruction of the environment. Examples of safety-critical systems include those found in aircraft, cars, nuclear power plants and healthcare systems. In all of these systems, negative events that are possible to occur during operation, referred to as **hazards**, need to be addressed. In the domains mentioned earlier, examples of hazards would include aircraft and car crashes, radioactive pollution due to the fallout from a nuclear power plant failure and threats to patient well-being respectively.

Some hazards are more likely to occur than others. For this reason, hazards are treated differently based on the combination of the impact they can cause, the likelihood of their occurrence and, depending on the domain, the means immediately available to the user to mitigate them. This combination is defined as a hazard's **risk**. Based on its definition, risk, and by extension, its associated hazard, can be mitigated by reducing its likelihood and lessening the impact of its occurrence. Significant experience is needed to identify the possible dangers that lie with each of the system's functionalities, to determine what the exact effects of each hazard might be and to quantify its probability of occurrence (Wilkinson & Kelly, 1998).

Further complications are introduced when developing the system against hazards. The functionality associated with each identified hazard is typically delivered as the combined result of multiple systems cooperating. In turn, each of these systems might contain more subsystems internally, forming an architecture which spans all the way down to component elements. Therefore, to apply any mitigation measures, the underlying architecture needs to be assessed in-depth. The scale and complexity featured in many contemporary systems makes it considerably harder to apply **safety assessment** processes across them. Safety assessment processes evaluate what kind of safety requirements systems need to fulfil to mitigate hazardous risk and whether the eventual system implementation meets them correctly. Safety requirements are defined to address identified hazards associated with the system's high-level functionality and dispersed progressively throughout the lower subsystems. The process of safety requirement distribution is tightly coupled to the system's architecture and, as the architecture grows larger and more complex, significantly more time and effort is required to perform it.

Safety assessment activities are not sufficient on their own to provide confidence that hazards have been addressed. The possibility of oversights, design errors and implementation mistakes at all stages of development needs to be sufficiently addressed. This means that the methodology employed must itself be evaluated, as well as the safety rationale and management decisions involved. **Safety assurance** encompasses activities which aim to establish confidence in the system's safety across the system's lifecycle. The progress of assurance activities performed over the course of system development and operation is captured in the **safety case**, a living documentation of the system's safety. The aim of the safety case is to provide a clear, comprehensive and convincing argument that the system is safe to the system's stakeholders (Kelly, 1998:22). System stakeholders can include the system's developers, owners, relevant certification bodies and users.

3. Role of the Safety Case

The need for establishing and maintaining the safety case rationale became apparent through a series of major accidents which occurred starting from the 60s up to the early 90s (explained in more detail in Chapter 2). Investigations into the contributing factors of these accidents highlighted the need for a more flexible approach to safety assurance. The earlier approach to safety assessment relied on the idea of all-encompassing domain-specific safety standards. These standards would prescribe in as much detail as possible the procedures, material evidence, established tests and other known practices that were considered to contribute towards system safety. This paradigm aimed to maximize coverage and safety by applying as much known best practice available onto the development of relevant systems. However, this approach adjusted poorly when considering, for example, novel systems, unanticipated scenarios during operation or poor comprehension of standard regulations. These examples are just some of the sources of potential and, in certain cases, catastrophic accidents owing to this earlier safety standard paradigm. The weaknesses highlighted following the accidents eventually resulted in the invention of the modern safety case.

In contemporary safety standards, the onus of addressing safety considerations has shifted to lie more with the developers of systems rather than the regulatory and certification authorities. The safety case is meant to be used as a means for developers to present an argument of safety that is appropriately adjusted to the risks of their particular system. For instance, novel features are addressed by introducing new supporting arguments to a previously established argument. Furthermore, interpretation of regulations is explicitly incorporated in the argument rationale. This allows deviations or alternative interpretations to be clearly annotated, leaving less room for errors and ambiguity.

Safety assessment is applied based on a process framework composed through sets of industry regulations, standards and guidelines. Safety assurance is derived based on reflection of the goals, rationale and evidence of the assessment and guided by the material mentioned above as well. In industry practice, it is often necessary for systems to be certified for their safety to be legally allowed to be put into operation. Even in cases where certification is not required by law, it is often highly desirable as it establishes consumer confidence in the quality of a system. Typically, independent government or industry regulatory bodies provide certification to systems which have been shown to comply with the relevant regulations. The safety case, while originating from the nuclear (Bishop & Bloomfield, 1995) and defence sectors ((MoD, 1997) quoted in (Kelly, 1998:24)), has been adopted across a plethora of industry standards and is now practiced by notable organizations such as NASA.

The safety case is meant to be developed in parallel to the system itself, promoting a feedback cycle between the two development flows. The progress on the system informs the state of the safety case, which in turn highlights weak points in the system's safety. This allows potential safety weaknesses to be identified as early as possible during development (ideally during the design stage), minimizing the impact of the changes necessary to address them. During operation, as new information becomes available and the system undergoes maintenance, the safety case should be updated appropriately and guide future evaluations.

4. Safety Case Management Systems

Safety case construction initially relied on plain text approaches, which suffered from a plethora of weaknesses. Structure through text alone was hard to maintain, ambiguity was easy to be inadvertently introduced and the argument rationale was overly difficult to both construct and follow. To facilitate the production and maintenance and improve the quality of safety cases, safety argument notation systems were introduced. These are known as the Goal Structuring Notation (GSN) and Claims Argument

Evidence (CAE) notation. These notation schemes provide structure to arguments by allowing users to explicitly define and connect graphs consisting of claims, argument rationale and evidence. These three elements are considered the fundamental building blocks for all arguments related to safety. Claims involve aspects of the system's safety and supporting properties. Each claim is supported by subsequent claims through reasoning captured in the argument structure itself. At the base of the structure lies evidence, typically produced from safety assessment processes, which lends support towards the claim structure above it. The representation facilitates construction and understanding of the argument, as the inferential logic is explicated and depicted concisely. Through this graphical representation, the reader can more easily follow the connections between elements and trace the logic underlying it.

Further means for managing argument structures are provided through concepts such as argument patterns and modules. The former, based on software design patterns, allow details of particular arguments to be abstracted. Argument patterns are reusable across multiple arguments both within the same and different safety cases, simplifying their construction and management. Argument modules isolate subsets of the overall argument structure into separate elements, connected to other elements via explicitly defined interfaces. Thus, entire subgraphs can be encapsulated within modules and be inspected separately in a much more compact and comprehensive way. This concept further facilitates management of the safety case.

5. Challenges in Safety Case Construction

Despite these and other facilities introduced over the years, constructing safety cases remains a tedious process, requiring significant manual effort to put together. Safety case engineers continually make safety design decisions over the course of the project which they must then accurately record and justify in the safety case. They must apply these decisions manually, reconstruct parts of the argument structure affected and provide feedback to the development iteration cycle. Furthermore, due to the dependency between the system architecture and the safety case, changes to the former need to be synchronized to the latter. Otherwise, inconsistency and potential for errors in the process can be introduced. In contemporary systems development, the majority of development approaches advocate iterative cycles of design-implementation-evaluation. Repeated delays in a particular cycle stage have the potential to accumulate over time, presenting a considerable problem towards meeting development deadlines reliably. For development of systems of smaller scale, the aforementioned issues are not a major deterrent towards project completion. However, as mentioned earlier, as system scale increases, these issues become further exacerbated. This fundamental dependence between the argument structure and the architecture of the system means that scalability of the safety case construction process is at the heart of this problem. For instance, a preliminary safety case for the EOSAN project is 200 pages long and expected to grow as development progresses (EUROCONTROL, 2011).

In order to address the issue of constructing and maintaining a safety case, we need to examine how standards propose it should be structured. To begin this investigation, repetitive processes in safety case construction that can be directly mapped to argument structures need to be identified. From this mapping, information generated by these processes can be compiled towards the production of safety arguments with minimal user input. This would allow the production to effectively address the scalability challenge. Additionally, safety standards instruct developers to apply safety assessment processes in a top-down fashion. Therefore, the most intuitive approach to translating the assessment process hierarchy in safety arguments should follow this paradigm, meaning it should also be top-down. In short, a set of repetitive processes which are applied from the highest to the lowest levels of the system architecture hierarchy must be identified.

Another aspect to consider is the content of the arguments supplied. Safety assessment during development can be broadly separated into two major phases, the requirements validation and the requirements verification phase. Validation is applied mainly in the earlier stages of design, before implementation begins. The aim of this stage is to confirm that the requirements that have been selected for the system architecture design at a given level contribute to the higher-level safety objectives. During the verification phase, the behaviour of the implementation is evaluated, in order to determine whether it meets the previously set requirements. Safety case construction varies depending on the phase it is performed at. In the validation phase, given the absence of an implementation, the arguments are expected to focus on the correctness of the requirements chosen. In the later phases of development, arguments regarding the correctness of the system behaviour are added to the former arguments. The overall rationale followed can be summarized as 'the system correctly meets the correctly identified requirements'.

For safety arguments concerning the system's requirements, another distinction in the types of arguments put forth can be made. Product-based requirements arguments focus on the particular properties the developed system must fulfil to be safe. Process-based requirement arguments focus on the planning, management and other procedural steps taken during development, to provide confidence they contribute to safety via their correctness. Based on these two categories of requirements, different sets of arguments can be produced. The former would be produced based on safety assessment of the system's properties, whereas the latter from documentation and evaluation of the processes' correctness employed during development.

6. Safety Integrity Levels

For software and hardware systems, due to their particular nature, another form of safety requirements has been introduced. Safety Integrity Levels, known as SILs, can be described as meta-requirements, as they can provide a common means to describe safety requirements across different systems and components.

Most safety standards use SILs to address safety-critical functionality or safety goals of the system. Initially, SILs are derived at the system level, are then progressively further refined and assigned to components as specific integrity requirements. One of the major goals of certification via safety standards is to determine that system SILs have been met via the satisfaction of component SILs.

SILs were introduced to address concerns regarding the development of software and hardware systems, which are unique compared to more traditional engineering systems in their safety evaluation needs. Electrical and mechanical systems fail randomly, due to material degradation and wear through usage. Software (and to the extent of their programmed logic, hardware) systems do not fail randomly, but systematically, when the input and control data that cause the system failure are present. In the former case, statistical analysis of materials and systems testing can usually determine reliable probabilistic failure rates for electromechanical systems. In the case of the latter, software testing and formal methods are used to address the possibility of software (and firmware) errors. Software testing methods evaluate the behaviour of a program to determine whether it meets its functional and performance requirements. Some formal methods express the program as a mathematical theorem which can then be proven via automatic solvers. Others simulate the system's behaviour based on its formal specification to ensure it does not violate safe conditions.

There are trade-offs to consider when choosing to apply either approach. With software testing, code behaviour can quickly be evaluated for significant portions of the program. However, complete or even extensive testing coverage of the program's behaviour for any input can be time-consuming to achieve, often prohibitively so. On the other hand, proof via formal methods addresses the issue of input coverage, however they can be difficult and expensive in development time to apply.

SILs allow developers to target particular sections or subsystems of an overall system and determine the type and the rigor of the assessment methods applied. Higher SILs imply more rigorous evaluation methods and careful documentation, whereas at lower levels evaluation can be even omitted. Furthermore, SILs are allocated following the system architecture hierarchy, i.e. top-down. When failure-tolerant design is present, e.g. via component redundancy, the SILs of lower architectural levels can be reduced. The particular rules which govern the options for reduction are domain-dependent and are usually defined in corresponding safety standards or guidelines. Thus, SILs allow the potential for optimisation of a system’s safety versus its production cost, given an appropriate allocation of SILs.

In practice, allocating SILs optimally across a system architecture manually shares similar problems with safety case construction and maintenance. Namely, the larger the overall system grows and the more numerous the subsystems and components become, the more difficult it can become to identify which of the potentially millions of SIL allocations is the best. The potential choices form a design space through which architect designers need to navigate. With every SIL being allocated, other SILs of supporting elements are potentially constrained. In effect, each allocation decision sets the designer potentially on a separate path of consequent allocations, the net effect of which can be very hard to determine beforehand across all choices.

This allocation problem can be defined as an optimisation problem, where the objective is to minimize the total development cost of the system and the constraints are the rules for allocating SILs. Previous research has shown it is possible to automatically allocate SILs for the automotive domain. Some approaches employed integer programming techniques, which optimise by methodically investigating the space of viable solutions. Other approaches used metaheuristic techniques, which offer a trade-off by gaining in time complexity and execution time while not guaranteeing optimal solutions. Some research has shown the problem automatically solvable for the aerospace domain as well, although with limitations, which will be explained further in Chapter 3.

7. Model-Based Safety Analysis

Safety assessment methods have traditionally been applied using informal models of the system, based on knowledge and documentation available at the time of the assessment. This approach often contributed towards analyses which were highly influenced by the assessor's personal view, requiring further consensus by other staff to be accepted. Further, the manual nature of the approach meant that additional effort was necessary to assure that errors or omissions were not introduced in the analyses.

Model-based safety analysis is a safety-critical system development paradigm. Under said paradigm, the processes supporting system development are linked via structural models to those supporting safety analysis. This link helps maintain information flow across the two process cycles with less effort and promotes more effective feedback between them to be available earlier, where it is most useful. The thesis will further explore how the use of model-based safety analysis methodology can support the generation of safety arguments.

In particular, the state-of-the-art reliability analysis method Hierarchically Performed Hazard Origins Propagation Studies (HiP-HOPS) will be investigated and extended. HiP-HOPS can be classified as a model-based safety analysis technique which supports the generation of safety assessment and assurance artefacts. By exploiting recent work on automatically allocating SILs via HiP-HOPS, the relevant artefacts it produces can be used to inform the construction of safety arguments.

8. Thesis Outline

In this work, we will focus on the aerospace domain, for several reasons. First of all, the problem of automatically allocating SILs in this domain is still relatively unexplored, as metaheuristics have not been employed until now. Furthermore, this offered an opportunity to explore the cross-domain potential of the technique. Next, employing a model-based technique to generate safety arguments would allow maintaining such arguments up-to-date, consistently useful and available throughout development. Therefore, an appropriate model-based method for undertaking this task is needed. By extending HiP-HOPS, the benefits are two-fold. First, the need for automatically generating safety arguments from automatically allocated SILs is addressed. Second, the user obtains the means to rapidly reconstruct the reliability analysis, SIL allocation and argument generation on demand.

In the following chapter, research into the construction and management of safety cases will be presented. The chapter will then provide insight into the established methods of safety assurance advocated by safety standards. Finally, the chapter will summarize contemporary methods that support safety assurance and assessment.

In Chapter 3, a summary of the state-of-the-art research underpinning the methodology presented later in the thesis will be presented. In chapters 4 and 5, elements, ideas and concepts from the optimization and safety argument generation methods presented in Chapter 3 will be selected and extended to address the objectives of the thesis. The reasoning regarding the choice of methods will be made clear following the state-of-the-art summary in Chapter 3.

In Chapter 4, a novel method for allocating safety requirements automatically to a system architecture for the aerospace industry will be presented. Furthermore, a case study demonstrating the effectiveness of the method on a wheel braking system will be shown.

In Chapter 5, the method presented in Chapter 4 will be extended to allow generation of a safety argument structure from the allocated safety requirements. In Chapter 6, a case study showing the method in practice will demonstrate and evaluate its effects.

In Chapter 7, the methods shown will be evaluated and discussion on their significance will be provided. Finally, further avenues of research based on the work shown here will also be discussed.

9. Hypothesis and Thesis Objectives

Current safety case support provided by GSN or CAE still require users to manually construct and maintain major parts of the safety case by hand. A model-based method for construction and management of safety cases based on allocation of safety requirements will facilitate their maintenance.

The objectives outlined below were identified, with an aim to investigate the hypothesis.

1. Investigate relevant literature to determine:
    a. How contemporary safety standards propose safety case construction.
    b. What kind of processes are involved in the construction.
    c. The information flow between these processes.
    d. Means of improving the scalability of these processes.
2. Develop a novel concept which supports model-based automatic generation of safety arguments.
3. Demonstrate the feasibility of this method.
4. Evaluate the effectiveness of the method via appropriately identified criteria.

The first objective aims to explore relevant literature and safety standards to identify key contemporary methodologies in safety assessment and assurance. Where possible, the potential for reducing the impact of model complexity and scale on the application of said methodologies will be investigated. Previous work that aims to address this challenge will also be evaluated. Based on this investigation, a novel method will be developed as part of the second objective. The method’s feasibility and effectiveness in addressing scalability will be demonstrated via appropriate case studies, as part of the two final objectives. The specific criteria identified to evaluate the method are explained in Chapters 2, 3 and 6.

10. Thesis Summary and Contributions

To improve accessibility of the material within the thesis, a short summary of the progress made against each of the thesis objectives and its overall research contributions is provided as follows.

With respect to the first objective, a literature review of the theory underpinning safety case management and model-based safety analysis was undertaken. This review identified some of the significant issues associated with the development of safety assessment and assurance artefacts for safety-critical systems. Evaluation criteria that potential solutions to this problem were also derived. Furthermore, specific methods which could address separate aspects of the above problem were identified and from them, an approach for each aspect was adopted for the thesis while considering the evaluation criteria.

For the second objective, methods were developed to address the allocation of SILs and generation of safety arguments from MBSA artefacts. These methods extend the HiP-HOPS reliability analysis method and satisfy the fundamental thesis objective.

Method feasibility, for the third objective, is demonstrated using small examples and two case studies, based on an established example from an aerospace safety standard (ARP4761) as well as an abstract version of the Flight Control System from the Airbus A320.

Finally, the evaluation criteria identified in the first objective were used to evaluate each method, using the results from the case studies as well.

In summary, the thesis contributions are as follows:

- Adapted Optimal Allocation of SILs using metaheuristics (Tabu Search) for the aerospace (ARP4754-A) domain. A pre-optimisation reduction step which can greatly reduce search space was identified. Finally, the understanding of the implications of the choice of SIL cost function on the optimization was advanced.

- Integrated above method with HiP-HOPS to support model-based optimal SIL allocation.
- Extended the concept of argument patterns to incorporate model-based and safety analysis elements.
- Integrated above extension with HiP-HOPS and first method to enable automatic generation of safety argument fragments for a (preliminary) safety case.
- Demonstrated application of above methods in abstract case studies based on widely-known systems.

11. Published Material

Some of the material included in this dissertation has been published previously, as follows:

- Sorokos, I., Papadopoulos, Y., Azevedo, L.S., Parker, D. & Walker, M. (2015) Automating Allocation of Development Assurance Levels: an extension to HiP-HOPS. *IFAC-PapersOnLine*, 48(7), 1 January, 9–14. Available online: http://dx.doi.org/10.1016/j.ifacol.2015.06.466.
- Sorokos, I., Papadopoulos, Y., Walker, M., Azevedo, L.S. & Parker, D. (2015) Driving design refinement: how to optimise allocation of software development assurance or integrity requirements. In Mistrik, I., Soley, R.M., Ali, N., Grundy, J. & Tekinerdogan, B. (eds) *Software Quality Assurance: In Large Scale and Complex Software-intensive Systems*. Morgan Kaufmann, 237–250.
- Sorokos, I., Azevedo, L.S., Papadopoulos, Y., Walker, M. & Parker, D.J. (2016) Comparing Automatic Allocation of Safety Integrity Levels in the Aerospace and Automotive Domains. IFAC-PapersOnLine, 49(3), 1 January, 184–190. Available online: http://dx.doi.org/10.1016/j.ifacol.2016.07.031.
- Sorokos, I., Papadopoulos, Y. & Bottaci, L. (2016) Maintaining Safety Arguments via Automatic Allocation of Safety Requirements. IFAC-PapersOnLine, 49(28), 1 January, 25–30. Available online: http://dx.doi.org/10.1016/j.ifacol.2016.11.005.

Chapter 2: Background

In this chapter, the necessary context for understanding the following chapters of the thesis is provided. Initially, formal definitions for key concepts related to the subject matter are presented. Following this, the reader is introduced to the idea of safety cases and their role in systems development. Next, the chapter describes the processes involved with certifying the safety of civil aircraft. Finally, safety assessment processes widely employed are briefly explained.

1. Useful Definitions

A series of definitions relevant to the key issues the thesis addresses will follow. These should help familiarize the reader with the fundamental concepts, improve comprehension and reduce ambiguity.

First of all, arguably the most important concept to understand is **safety**. Safety is a concept whose precise definition can be difficult, as it is highly dependent on the context it is used in. The definitions that follow will be drawn from safety standards from various industrial domains. Although alternative sources regarding the definition of safety are available, given the subject at hand, the former were deemed more relevant.

- In the ISO 26262 automotive safety standard, safety is defined as the "absence of unreasonable risk" (ISO, 2011:14).
- In ARP4754-A, an aerospace domain standard, it is defined as "the state in which risk is acceptable" (SAE, 2010:13).
- In the domain-neutral IEC 61508, safety is "**freedom from unacceptable risk**" (IEC, 2010:5).
- In Defence Standard 00-56, a standard developed by the UK Ministry of Defence, it is defined as a state where "risk has been demonstrated to have been reduced to a level that is [as low as reasonably practicable] and broadly acceptable or tolerable, and relevant prescriptive safety

requirements have been met, for a system in a given application in a given operating system" (MoD, 1996:10). This is an indirect definition; the standard assumes previous understanding of the concept of safety itself.

Despite the difference in wording and detail, overall, there seems to be consensus. We will adopt the definition from IEC 61508, being the most succinct.

Given the choice of safety definition, to understand safety, one must understand **risk**. Once more, by reviewing various safety standards and other sources, consensus seems to be reached regarding the definition of risk. The definition we will adopt for risk is "**combination of the probability of occurrence of harm and the severity of that harm**" (IEC, 2010:4). Under this definition, **harm** is defined as **damage to the health of people, directly or indirectly**. This definition is frequently employed in domains where safety is addressed through risk management. The essence of managing risk revolves around identifying sources of risk, quantifying their negative effects and likelihood, and identifying means of mitigation, if needed (ISO, 2009:1). In the context of the thesis, **events during which the harmful effects are realized** are referred to as **hazards**. Thus, by reducing the likelihood of hazards or their negative effects, risk can be reduced.

Providing proof of safety is often an intractable problem, in most cases involving non-trivial systems. This is because the causes of the potential hazards that might affect the system are, in general, unknown and there is no known general way of identifying and addressing all of them. Thus, **safety assurance** is the term used to represent the "**planned and systematic actions necessary to provide adequate confidence and evidence that a product or process satisfies given requirements"** (SAE, 2010:10).

Choosing the appropriate measures to reduce risk is yet another issue of importance. Since development resources are limited, this necessitates establishing priorities among sources of risk and measures of addressing them. Obviously, the general principle is that measures should be selected with higher priority for addressing hazards of relatively higher risk. Thus, choosing the appropriate level of risk mitigation follows some form of cost-benefit analysis. A principle which reflects this view is reducing risk to a level being **As Low As Reasonably Practicable** (**ALARP**) (HSW, 1974:2). In (Redmill, 2010), the author further examines the reasoning underpinning the ALARP principle.

The thesis will investigate safety in the context of systems development. By **system**, we refer to a "**collection of inter-related [elements] arranged to perform a specific function(s)**" (SAE, 2010:13). The elements mentioned in the former definition refer to singular hardware or software modules with well-defined interfaces. The thesis aims to address a particular class of systems, known as safety-critical systems. These can be defined as systems where "safety concerns" are present (Bowen & Stavridou, 1993:2). In other words, when these systems fail, one or more hazards can occur.

The final fundamental concept related to the thesis is **reliability**. It is defined as "**the ability of [a system] to perform as required, without failure, for a given time interval, under given conditions**" (IEC, 2017). Although often assessed in tandem with safety, the two concepts have been known to be incorrectly causally linked. In (Leveson, 2011a:8), examples of systems which are safe but not reliable and vice versa can be found.

2. Introduction to Safety Cases

The landscape of safety assurance methodologies has seen a significant degree of change over the years. Early approaches focused on investigation of previous accidents or incidents, leading to knowledge and best practice being established from this basis. Particular hazards and appropriate countermeasures were also identified. In (Rushby, 2015:14), the steam boiler codes in the USA and UK are mentioned as examples of this approach. The government regulations that effectively enforced assessment according to the codes proved quite effective. Both explosions and deaths of individuals were severely reduced in the following decades.

Unfortunately, this prescriptive approach was not foolproof. As systems grew more complex and their application more widespread, the codes became increasingly larger. For example, the American Society of Mechanical Engineers boiler codes eventually included more than 16,000 pages (Rushby, 2015:15). Another issue was that this regulatory approach established a checklist mentality in safety evaluators, whereby the codes replaced safety as the end goal rather than being the means to assure it. These concerns were captured in a UK government report, known as the 'Robens Report' (Robens et al., 1972). A collection of recommendations within the report seems to have instigated significant change to the then-contemporary approach to safety. One of the major ideas is that the responsibility for managing risk should lie with the developers of systems rather than regulators. Instead, regulations should establish goals that developers and systems should meet, rather than prescribe specific methods to be applied. The ALARP principle mentioned earlier in the chapter was also established in the report.

A series of notable accidents in the years following the Robens Report further motivated the adoption of its ideas. These accidents include the 1974 Flixborough chemical plant explosion, the 1988 Piper Alpha oil platform explosion, the 1988 Clapham train collision (Kelly, 1998:19–22) and the 1976 Seveso chemical plant explosion (Rushby, 2015:19). The above cases share an important common characteristic. In each case, operators followed the established safety protocols, disregarding the context of the system or assumptions supporting the procedures. A later investigation led by Lord Cullen in the UK corroborated the concerns from the Robens report, relative to the Piper Alpha accident. The investigation formally recommended the adoption of the goal-based approach towards safety assurance of offshore platforms (DPIE, 1991:3).

Following evaluation of the accidents, regulatory authorities eventually adopted a novel concept to address the shortcomings of prescriptive regulation. The new approach involved the production of a document which would provide goal-based assurance of safety, known as the safety case. The **safety case** has been defined as both a (collection of) document(s) as well as the argument and supporting material contained within the documentation. In this thesis, the latter definition will be adopted. Specifically, it will be defined as a "**clear, comprehensive and defensible argument that a system is acceptably safe to operate in a particular context**" (Kelly, 1998:22). The thesis will emphasize on the construction of argument structures for safety cases, so this definition is more appropriate.

The fundamental elements that drive contemporary safety case development are **claims**, **argument strategies** and **evidence**. Under the context of system safety, claims are propositions regarding safety-related properties of the system. Claims can be supported by other, more refined claims or evidence. The reasoning process linking claims with sub-claims or other evidence can be captured within argument strategies. The choice of fundamental elements was inspired by research originating from the field of philosophy. The work of British philosopher Stephen Toulmin in particular has been highlighted in its role towards identifying the fundamental argumentation elements of safety cases (Kelly, 1998:62–63). In "The Uses of Argument" (Toulmin, 2003), the author identifies what he considers to be fundamental elements of any argument. Beyond the elements mentioned earlier, an additional three are defined. These are **backing**, **rebuttals** and **modality**. Backing includes underlying assumptions related to the argument strategy. Rebuttals explain conditions where the argument does not hold. Finally, modality represents the level of confidence the person presenting the claim has in favour of the claim. Although Toulmin's method contributed significantly towards establishing the building blocks of arguments, constructing arguments solely on that basis proved to be ineffective. The Toulmin elements restricted the forms of argument that were constructible and could not represent some of the usual arguments found in actual safety cases (Kelly, 1998:41–42). These were some of the drawbacks of the early approach for constructing safety cases that led to the development of better structured construction and representation methods. Before proceeding with further details regarding the construction methods, an overview of the benefits and limitations associated with the use safety cases towards safety assurance will be presented.

2.1. Benefits and Limitations of Safety Cases

In numerous industry safety standards, safety cases are part of the regulatory framework and feature explicit requirements. Examples of such standards include:

- UK Shipping Regulations for Safety and Environmental Protection DSA02-DMR (MoD, 2016:13)
- UK The Railway (Safety Case) Regulations (TRR, 1994:10–12)
- UK Defence Software Systems DS 00-55 (MoD, 1997)

The contents of the safety case prescribed in such standards are similar, for example in the Defence Standard 00-55:

- System and Design Safety Aspects
- Software Safety Requirements
- Software Description
- Safety Arguments
- Safety Related System Development Process
- Current Status
- Change History
- Compliance with Safety Requirements
- In-Service Feedback
- Software Identification

Some of these content sections prescribe categories of arguments to be included in the safety case, adequately demonstrating the corresponding claim. For example, this would include the 'Compliance with Safety Requirements' section. Others are included to provide the necessary system and contextual information to sufficiently describe the system, its role, its development and operation.

Not every domain features safety standards where safety cases are explicitly required for certification purposes. Even in cases where a basis for certification cannot be provided, there are other **benefits** to be had from maintaining a safety case:

- It provides an overview of the current state of the system's safety. On the one hand, this helps clarify and highlight the focal points of the safety argumentation. On the other hand, it can also help identify weaknesses or omissions in the reasoning. Missing evidence, arguments in need of further development and even assumptions which have been disproven through simulation and analysis can all be brought to attention through the safety case.
- It adds confidence to the development process leading up to certification. Efforts have been made to implement confidence quantification mechanisms in safety cases, notably in (Denney et al., 2011) and (Weaver et al., 2006). Even without such extensions, merely expressing the rationale motivating the safety assessment can help clarify misconceptions, establish consensus and confidence within a development team. This last point is significant; after all, the development team is one of the most important stakeholders of the system. Therefore, maintaining the team's confidence in the system's safety is essential for ensuring development proceeds smoothly.
- It establishes the connection between the safety assessment evidence and the safety requirements or objectives. Without this connection, it can be unclear to stakeholders – especially those less familiar with safety engineering – whether the standard guidelines have

been met. Furthermore, in cases where novel technology is proposed, alternative methods of safety assessment may be needed during development. Providing standard safety assessment evidence on its own can be insufficient in lieu of the added risks; additional argumentation could help explain how the evidence is adequate.

- A plethora of other side-benefits identified through the use of safety cases have been summarized in (Pfitzer et al., 2013:8–9).

Naturally, safety cases are not a perfect solution. In (Leveson, 2011b:3–4), the author expresses her concerns regarding their **limitations**. The major criticism is that goal-based safety assurance using safety cases risks instilling a false sense of safety in the people involved in the system's development. The danger being that this mindset can contribute towards various forms of cognitive bias – most importantly, confirmation bias. Safety cases aim to instil confidence but (can)not prove that a given system is safe. Thus, when confirmation bias is present, engineers are at risk of favouring arguments or evidence that supports their case, or inadvertently omitting to explore counter-scenarios. The second criticism levied against safety cases is that their use could be misinterpreted based on their naming. In other words, instead of using 'safety case', the term 'risk case' should be used instead. In the author's view, this would help clarify its role and encourage developers to consider their system's vulnerabilities as well. This view is supported by that of the committee that investigated the 2006 UK Royal Air Force Nimrod crash (Haddon-Cave, 2009). The next criticism the author highlights involves the usual choice of risk analysis criteria. In particular, Leveson notes the underuse of worst-case analysis (Leveson, 2011b:3), in favour of evaluating less severe but more likely outcomes, referencing (Houck, 2010).

The Nimrod crash review offers a number of critical points against the aircraft's safety case. Although some of the criticism is focused on the weaknesses of this particular safety case, more general points of contention are also raised:

- The safety case developed for the Nimrod was very poorly developed. It included numerous errors and was underpinned by an assumption that the aircraft was de facto safe due to its 30-year successful flight record. Thus, constructing the safety case was treated as a bureaucratic paperwork exercise instead of an engineering process.
- When the safety case is developed under such conditions, it adds no value to the development process and distracts from making progress towards actual safety.
- Safety cases are treated as an end themselves instead of being a mean in supporting safety. Furthermore, they are written with an aim to satisfy regulations and do not actively contribute towards the discovery of safety vulnerabilities.
- Safety cases can grow too large, too difficult to follow, unbalanced in focus and their quality is tightly coupled with the skill of their developer.

Despite their drawbacks, the benefits safety cases offer when employed properly are considerable. To maximize the contribution towards system safety, proper training and awareness of the engineers responsible for the safety case is necessary. To assist in this effort, model-based (see Section 7) tool support can be employed to reduce the encumbrance of safety case production. By facilitating safety case production, focus can shift towards major argument design choices instead of dealing with minutiae.

3. Safety Case Notation Systems

The initial approach to safety case construction was **text-based**, with structure loosely established at the author's discretion and ability. This approach suffered from multiple weaknesses; ambiguity, errors and excessive cross-referencing were easy to introduce. This made both writing and reading the safety case a significant, time-consuming challenge (Kelly, 1998:48–49).

Another idea was to represent information in **tabular structures**, thereby containing the text that comprised the safety case into a rigid structure of fundamental elements. Tables separated the text into columns of claims, argument and evidence/assumptions. The method was simple to apply and helped dilute the argument into its fundamentals. The drawback of this approach was that tables were too restrictive and failed to adequately represent multiple stages of argumentation. Another hindrance is that there was limited guidance regarding what exactly the contents of the table should be (Kelly, 1998:49–50).

A different concept which aims to address safety case construction are **claim structures**. They can be found in Annex H of the DS 00-55 safety standard (MoD, 1997). Claim structures represent safety cases graphically, as a tree graph. Each node of the graph represents a claim, linked with other nodes through logical AND and OR gates. AND gates are used to decompose higher claims into constituent ones, whereas OR gates express support through independent claims. At the top of the graph lies the initial claim, with successive claims decomposing it further until base or undeveloped claims are reached. Claim structures are a precursor to the notation systems introduced in the following sections. Although they were able to represent the fundamental structure of the argument, they lacked detail. Specifically, the reasoning, assumptions, context and evidence was not represented.

Bayesian Belief Networks (BNNs) is an idea used in various domains including safety engineering and computer science. In safety cases, BNNs were used in a research project, Safety and Risk Evaluation using Bayesian Networks (Marsh, 1999). They are directed graphs that interconnect variables linked via probabilistic causal relationships, forming networks. Each node in the graph represents a variable, whereas connections represent causal dependencies. Figure 2.1 shows an example of such a graph.

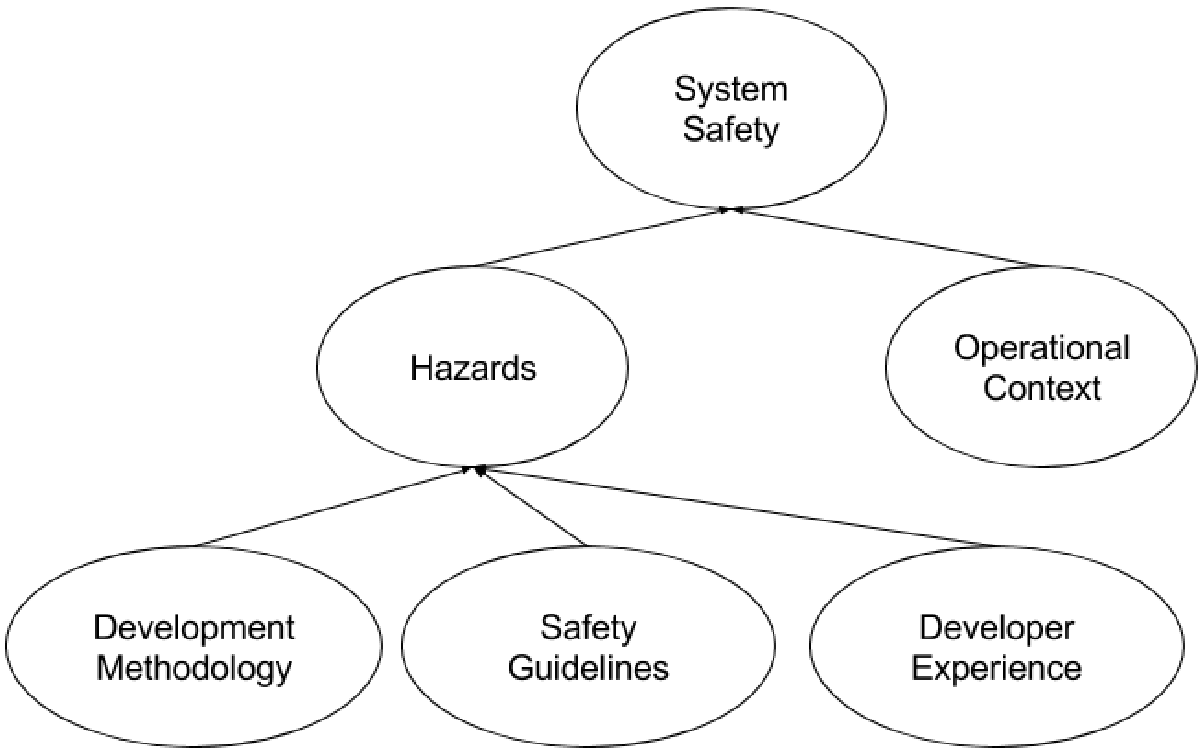


Figure 2.1 - Example Bayesian Belief Network

In the example, a system's safety is dependent on its operational context and the hazards that can be caused by its operation. The occurrence of hazards further depends on the development methodology used, whether the guidelines of a safety standard have been applied and on the development staff's experience. Each of the causal relationships would be assigned a conditional probability to express the extent to which each variable can be caused by each of its dependent factors. Given the above BNN, the joint probability for the System Safety can be given by:

$$\begin{aligned}&\Pr(SS, H, OC, DM, SG, DE)\\ &\qquad = \Pr(SS \mid H, OC) * \Pr(H|DM, SG, DE) * \Pr(OC) * \Pr(DM) * \Pr(SG) * \Pr(DE)\end{aligned}$$

Where,

- *SS* is System Safety
- *H* is Hazards

- *OC* is Operational Context
- *DM* is Development Methodology
- *SG* is the use of Safety Guidelines
- *DE* is the level of Developer Experience
- Pr is the probability mass function of the corresponding variable for single variables, the joint probability function for multiple variables and the conditional probability function accordingly

BNNs offer the capability of expressing claims regarding a system's safety and can further quantify those claims by computing the conditional probability of corresponding nodes in the network. However, there is no general or objective methodology for determining the conditional probabilities between variables. Effectively, the network is constructed ad-hoc and its accuracy depends heavily on its creator's knowledge, skill and experience. For particular variables, which happen to be objectively measurable properties, the process can provide more accurate results over time. Another drawback of BNNs is the lack of elements which explicitly capture argumentation rationale. The argument embedded in a BNN is implicit, captured within the relationships between variables. For these reasons, BNNs are more useful as safety evidence or quantifying the confidence in a safety case rather than a vehicle for expressing the safety case.

Traceability matrices are another feature employed in safety cases. These are tables that link requirements to elements of argumentation. This type of matrix is commonly used in various engineering disciplines (Cleland-Huang et al., 2004:2). Traceability matrices are useful in recording directly the link between pairs of requirements and elements. However, like tabular structures, they can only depict one level of connection at a time. They also lack the capacity for expressing the reasoning behind each link. Thus, they are more appropriate to be used as assessment artefacts rather than structuring the safety case around them.

All of the above approaches each suffer from a number of limitations. The safety case notation systems introduced in the following sections address many of the shortcomings and have gained wide acceptance in industry practice (Pfitzer et al., 2013:10).

3.1. Goal Structuring Notation

The Goal Structuring Notation (**GSN**) is a graphical notation that can be used to represent safety cases. It was developed as part of a series of research projects, described in (McDermid, 1994), (Wilson & McDermid, 1995) and (Wilson et al., 1997). The objective of the research was to develop a "structured method and comprehensive tool support for the production of safety cases" (Kelly, 1998:42). The GSN views arguments as goal hierarchies, where the constituent elements support an overlying goal. In safety cases, this is often a safety objective e.g. that the system is considered safe. The elements that can comprise a GSN goal structure are:

- Goals, which represent claims regarding a safety-related property of the system
- Strategies, which explain how goals are supported by sub-goals
- Solutions, which refer to evidence that support goals
- Context, which allow reference to useful information regarding a particular element to be linked
- Justifications, which explain why a particular strategy was chosen
- Assumptions, which record important hypotheses elements of the argument rely upon

Goals and strategies can also be deemed 'undeveloped', represented via a rhombus under the corresponding element. An undeveloped element presents a claim or strategy which will require further explanation before the safety case is considered to be complete. Directed links bind together GSN elements to form safety cases. The direction of the links determines the nature of the relationship between two elements. For example, a goal with a link with directed towards a strategy is usually in a

‘SolvedBy’ relationship with supporting elements, strategies and solutions. Instead, context elements, including assumptions and justifications are usually found in ‘InContextOf’ relationships.

Figure 2.2 depicts all of the above elements of the GSN in their standard graphical representation (GSN Working Group, 2011:8). In practice, each of the elements should feature a unique title and description of their role in the argument.

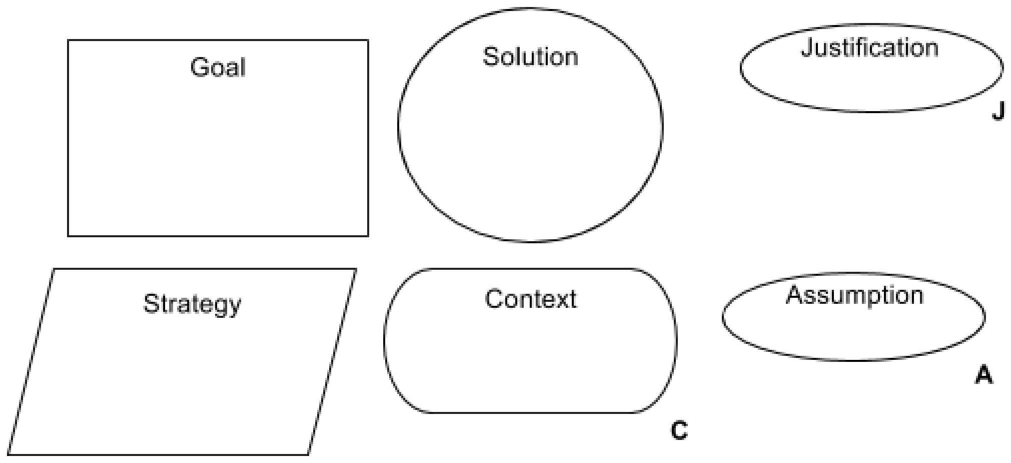


Figure 2.2 - GSN Elements

In Figure 2.3, a simple example is shown. The example shows how a system is argued to be safe, as expressed in SystemGoal of the figure. SystemContext would provide a link to the documentation describing the system’s specification. SystemStrategy supports SystemGoal by arguing how all of the hazardous failures have been addressed by design or implementation. The underlying assumption is that there are no hazardous failures that have not been identified, captured in the associated Assumption. Two failure modes that are hazardous are claimed to be addressed in OmissionGoal and LowOutputGoal. OmissionGoal offers OmissionSol as a solution to its goal, whereas LowOutputGoal is yet undeveloped and will need further work for the argument to be complete. Finally, OmissionSol is provided as evidence that the system cannot fail to output, given its fail-safe mode feature. Presumably, this measure would provide a default, safe output in all scenarios accounted for.

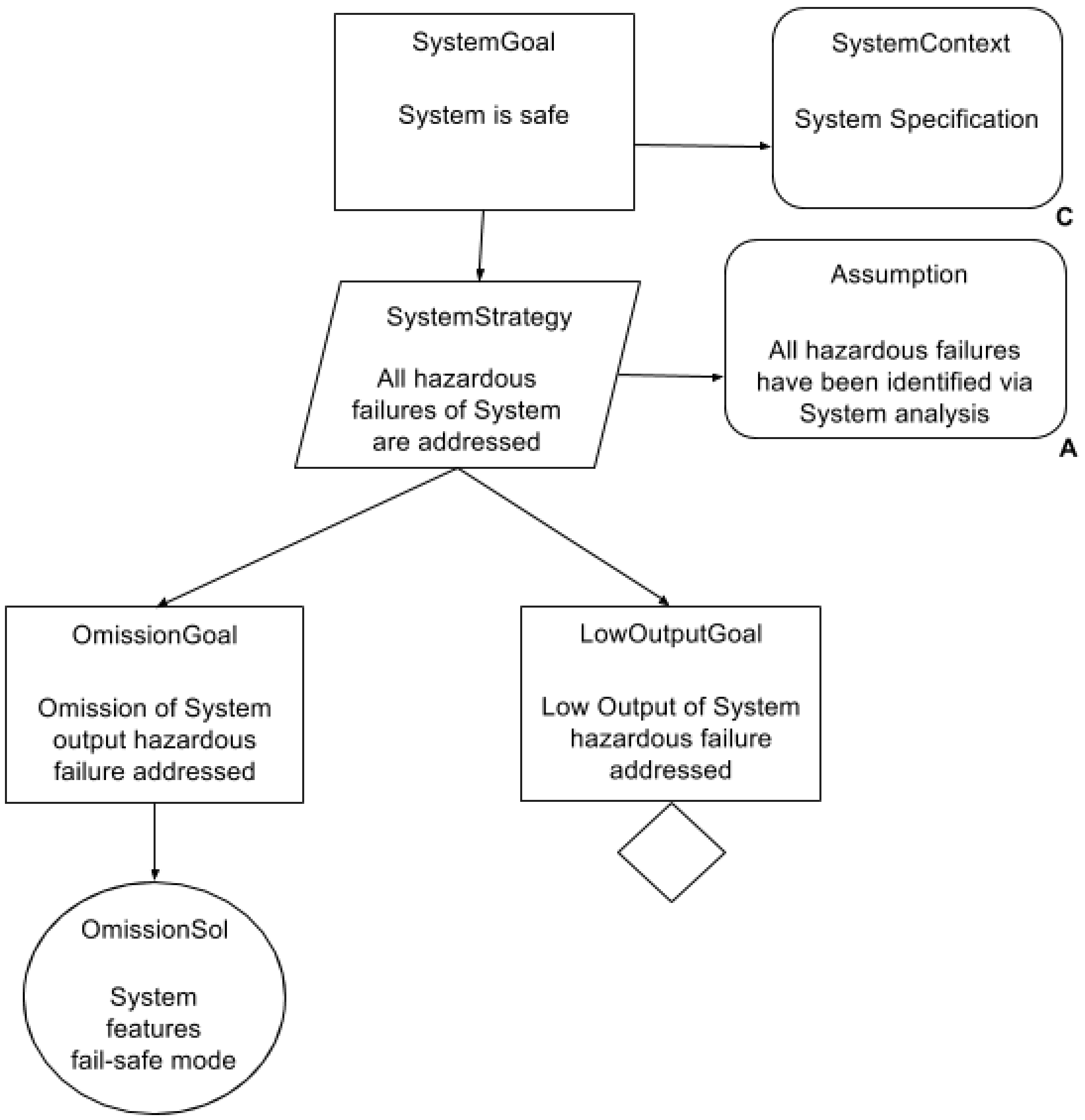

SystemGoal
System is safe
SystemContext
System Specification
C
SystemStrategy
All hazardous failures of System are addressed
Assumption
All hazardous failures have been identified via System analysis
A
OmissionGoal
Omission of System output hazardous failure addressed
LowOutputGoal
Low Output of System hazardous failure addressed
OmissionSol
System features fail-safe mode

Figure 2.3 - Simple GSN Example

GSN was chosen in (Kelly, 1998:56–57) as the preferred approach for structuring arguments due to its identified strengths:

- It captures the logical flow of the safety argument via the SolvedBy relationships linking the GSN elements
- It incorporates all elements of argumentation as defined by Toulmin’s method
- The rationale of the argument is embedded within strategy and justification elements

In (Kelly, 1998:81–86), a six-step method for constructing safety arguments via GSN is provided. The steps can be seen in Figure 2.4.

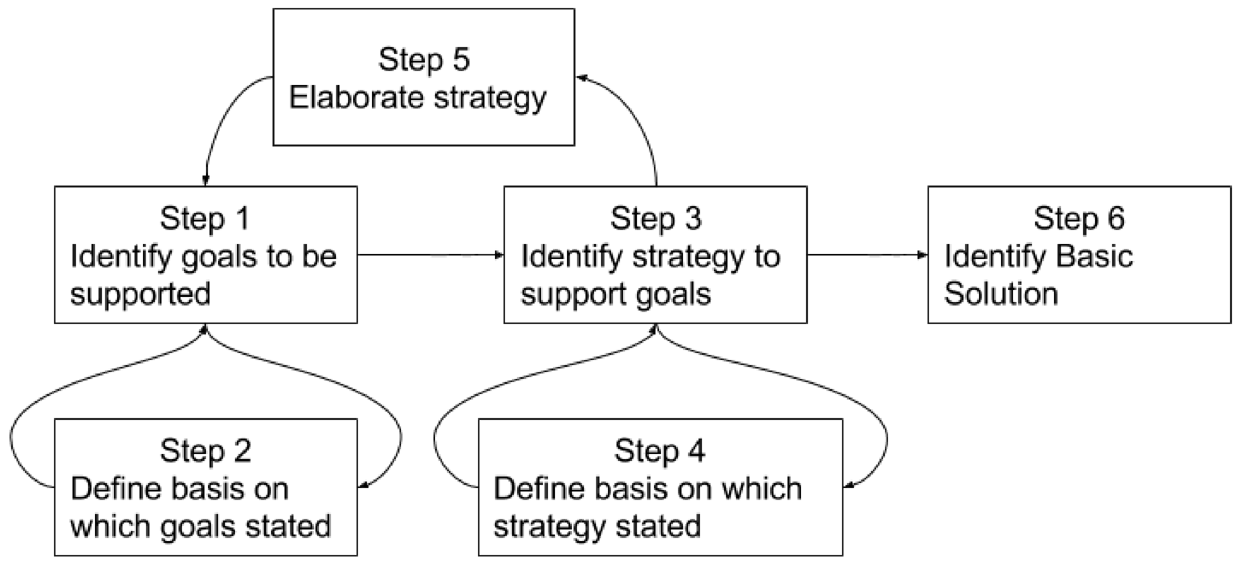

Step 5
Elaborate strategy
Step 1
Identify goals to be supported
Step 3
Identify strategy to support goals
Step 6
Identify Basic Solution
Step 2
Define basis on which goals stated
Step 4
Define basis on which strategy stated

Figure 2.4 - GSN Six-Step Construction Method, reproduced from (Kelly, 1998: 81)

First, the higher-level goals that need support are identified. For a safety argument, this involves clearly stating what the safety objective is. A 'Noun-Phrase Verb-Phrase' form is recommended for the description of the goal. The Noun-Phrase is the part of the description referring to the subject of the goal e.g. the system in question. The Verb-Phrase is the remainder of the description, preferably containing a verb which appropriately describes the function of the subject in the whole phrase. For example, "Car wheel is acceptably safe to be operated in car" is an acceptable form. The Noun-Phrase is "Car wheel" and the Verb-Phrase is "is acceptably safe to be operated in car". The second step is to define the context related to the goals from Step 1. This usually associates each goal with references to the subject in its description. For example, in the earlier example, the car wheel specification, the car's specification and an appropriate definition of what is considered 'acceptably safe' could be referenced via context elements. The next step is to identify how the goals set in Step 1 are to be supported. This reasoning is captured in a strategy element, linked to each goal supported respectively. If a goal can be supported directly by a solution instead, the final step is applied. Otherwise, the process moves on to the following step. The step that follows repeats the second step, establishing context elements but this time for the strategies defined in Step 3. Following this, if there any strategies that have not been further elaborated, this is done so at this point. The process essentially repeats at this point, returning to Step 1 for each strategy. The final step involves the identification and linking of a solution to goals which require no further strategy for support.

3.2. Adelard Claims Arguments Evidence Notation

Claims-Arguments-Evidence (**CAE**) notation is an alternative to GSN, featuring a more direct interpretation of elements from Toulmin's method. Claims are featured once again, along with sub-claims and arguments. However, CAE explicitly includes side-warrants, backing and system information. The CAE argument conflates the concept of an argument with that of the justification and the warrant in Toulmin's method. This combination is supported by side-warrants, which essentially represent assumptions or further justification associated with an argument. Side-warrants are different than normal arguments in that they serve a relatively secondary role to the latter. Backing serves the standard purpose; it provides arguments and side-warrants with additional support which cannot be further refined. Finally, system information can also be linked to arguments and side-warrants, providing contextual information about the system in question. In Figure 2.6, the elements mentioned above are presented in their standard graphical form. Whereas GSN was developed as part of research projects involving the University of York, CAE was developed by the Adelard company. The official term for Adelard's methodology is Adelard Safety Case Development (ASCAD) (Bishop et al., 2004), however the term for referring to the notation scheme CAE is more frequently encountered in the material reviewed by the author, as seen for example in (Rushby, 2015:55). The standard method of constructing safety cases via CAE is described in (Bloomfield & Netkachova, 2014). The concept of a 'stack' is offered as a high-level guide for developing CAE resources, reproduced in Figure 2.5.

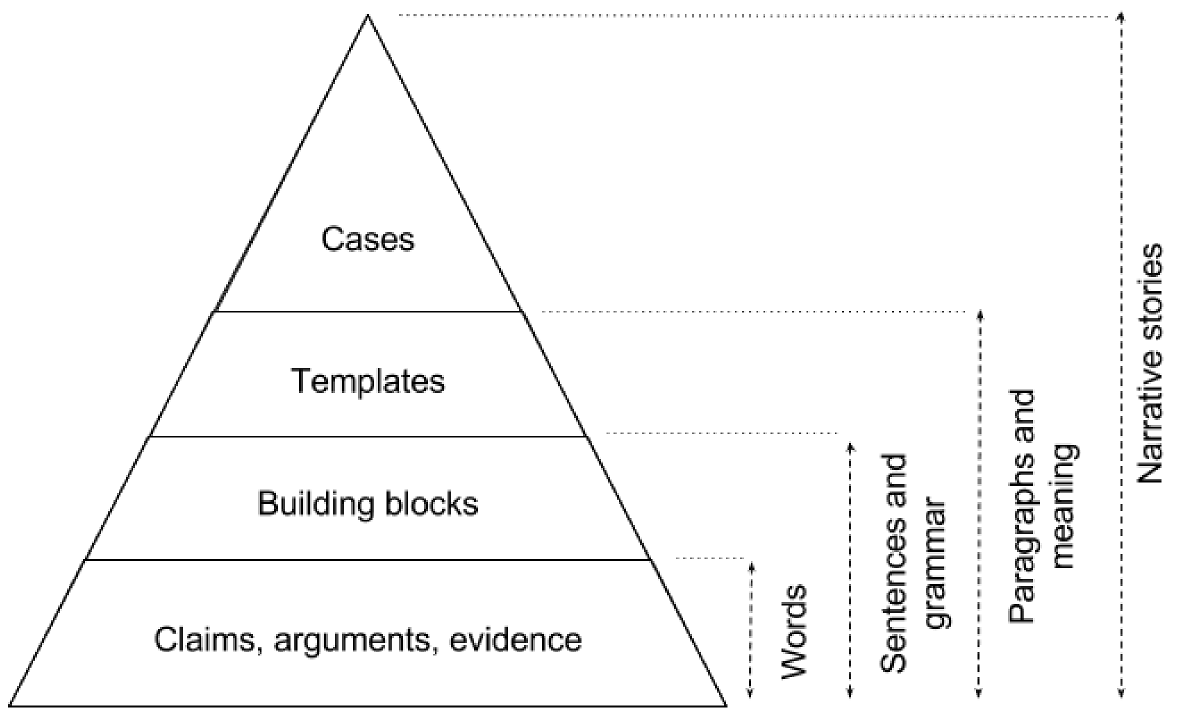


Figure 2.5 - CAE Stack, reproduced from (Bishop et al., 2004)

At the bottom of the stack, the fundamental elements as defined in CAE are found. They are labelled 'words' in the sense that they represent the very base elements with which safety cases are meant to be constructed. This analogy is continued at each higher level of the stack, eventually reaching 'narrative stories', representing entire safety cases. At the level immediately above the fundamentals, building blocks are formed from groups of base elements. An example block can be seen in Figure 2.6. The direction of the arrows is reversed compared to the standard GSN representation, however this difference is relatively minor. CAE also utilizes colour to accentuate the different types of elements it uses and the connections between elements.

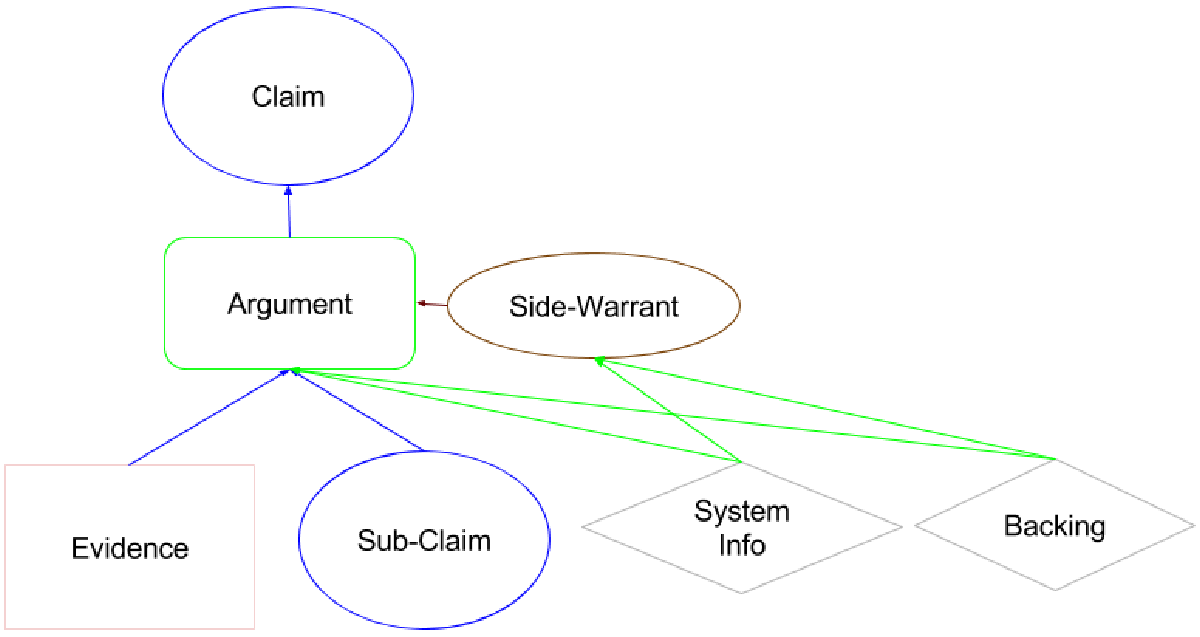


Figure 2.6 - CAE Block

Block construction revolves around the argument element. The argument is chosen to justify a top-level claim and, in the case of the example, is supported by a yet undefined number of sub-claims. As per the earlier description of the fundamentals, the argument is supported by a side-warrant, contextualized by system information and external backing grounding the rationale. A number of basic blocks are identified in (Bloomfield & Netkachova, 2014:3–5). These basic blocks have emerged by empirical examination of actual safety cases, essentially capturing best practice. They include:

- The decomposition block. This block captures the scenario where the satisfaction of a safety property of an abstract object is dependent on its satisfaction for each of the system's constituent elements, under a given context. A distinction is made between 'double' and 'single' decomposition. In the former case, the property can be satisfied by constituent elements' properties which can be heterogenous to the former. In the latter case, satisfaction is achieved by decomposition of either the property or the system but not both.

- The substitution block. This block enables the expression of equivalency between heterogenous properties of abstract objects.
- The evidence incorporation block. This block captures the standard way of including evidence into the safety case.
- The concretion block. This block is used when a general property or abstract object can be specialized under a particular context. This enables arguments which, when argued independently might seem vague but are clarified when applied to a specific instance.
- The calculation block is used to argue that a property of an abstract object is computable from relevant properties of related abstract objects in a particular context.

A number of ‘composite’ blocks have also been identified by combining specific pairs of basic blocks, such as the substitution and decomposition. The combination of the two results in a block which establishes a claim regarding an object that depends on a set of claims regarding an equivalent object. Other permutations and combinations are possible, as described in (Bloomfield & Netkachova, 2014:5).

CAE and GSN share much of the common philosophy that defines the fundamental elements of the Toulmin approach to argumentation. Effectively, the choice of notation depends on the personal choice of the safety case designer, depending on which notation approach he or she is most comfortable with. As mentioned in (Rushby, 2015:55) and (Linnosmaa, 2016:38–51), many of the notation software tools support at least GSN and some CAE as well. This is further supported by an OMG standard; the Structured Assurance Case Metamodel (SACM) that enables support for both notations by tools that implement it (OMG, 2016). For the purposes of this thesis, a GSN-style notation will be used.

4. Safety Case Management Facilities

4.1. Safety Case Patterns

In (Kelly, 1998:159–196) the concept of Safety Case Patterns is introduced, based on the ideas of architectural and software design patterns from (Alexander et al., 1977) and (Gamma et al., 1994). In each case, patterns are used to abstract the details of a particular instance of an architectural, software design or safety argument structure in order to establish a useful, flexible and reusable blueprint. In many cases, patterns can also function as counter-examples of design best avoided, known as 'anti-patterns'.

Patterns typically emerge as practitioners of the corresponding discipline identify commonly re-occurring problems or scenarios whose solutions can be made to share a similar structure. When this structure is identified, a pattern is established and includes details such as its description, purpose, and advice on when to use or when to avoid. Pattern catalogues consist of collections of patterns which are often grouped into categories usually based on similar pattern purposes. The initial catalogue for safety argument patterns in (Kelly, 1998:285–332) was extended in (Weaver, 2003) and (Hawkins et al., 2011) with amended and software-specific patterns.

Initially, architectural patterns were accompanied by information describing each pattern's problem, context and solution. Additionally, graphical representation of the patterns, appropriate for the corresponding domain were also used, e.g. architectural diagrams for architectural patterns and class, object and interaction diagrams for software design patterns. Further investigation of patterns led to more extensive documentation schemes. The scheme found in (Gamma et al., 1994:16–18) presents arguably the most extensive one, describing 12 aspects of the pattern ranging from its basics, such as its intent and description, to even sample code for direct guidance in implementation of the pattern. For our purpose, such extensive documentation is not necessary. We use our patterns not as guides for manual construction of arguments but as blueprints for algorithmic implementation. As a result of the above concerns, the following attributes of documentation have been identified to be potentially useful for the purposes of the thesis:

- Intent, to document the rationale behind the pattern selection.
- Structure, to document the behaviour of the generation process.
- Related patterns, to document alternatives to the chosen pattern.

Figure 2.7 shows an instance of a GSN pattern, the Hazardous Contribution Software Safety Argument Pattern (Hawkins & Kelly, 2013:21).

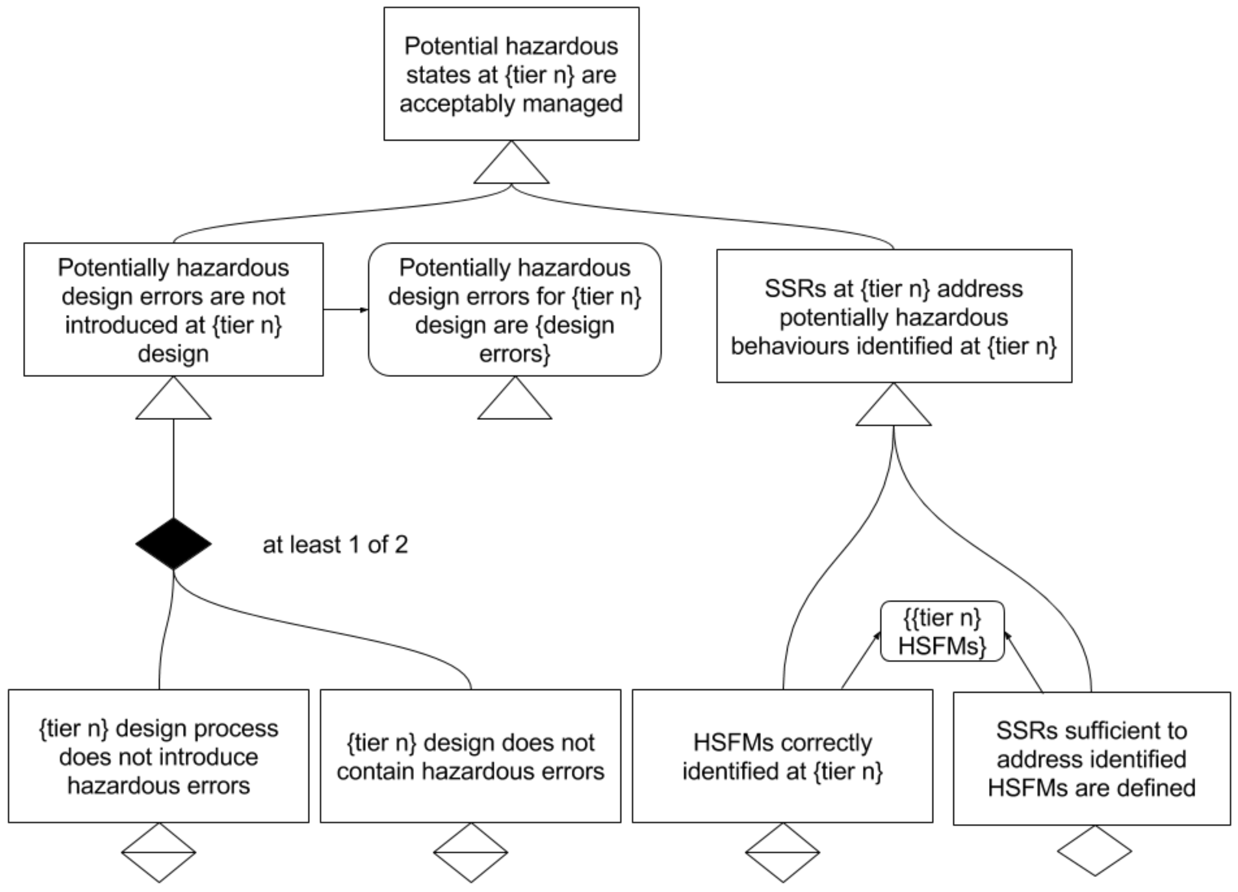

Potential hazardous states at {tier n} are acceptably managed
Potentially hazardous design errors are not introduced at {tier n} design
Potentially hazardous design errors for {tier n} design are {design errors}
SSRs at {tier n} address potentially hazardous behaviours identified at {tier n}
at least 1 of 2
{{tier n} HSFMs}
{tier n} design process does not introduce hazardous errors
{tier n} design does not contain hazardous errors
HSFMs correctly identified at {tier n}
SSRs sufficient to address identified HSFMs are defined

Figure 2.7 - Hazardous Contribution Software Safety Argument Pattern

The pattern provides an argument structure to address the claim that potential hazardous failures for a given level of the system architecture hierarchy have been acceptably managed. The claim is supported two-fold; first, by arguing that no design errors which can lead to hazardous failures are introduced at the given level; secondly, by arguing that system safety requirements (SSRs) address the hazardous failure modes identified. The pattern features a number of differences in its elements compared to the base GSN elements. When the pattern is applied onto a particular system context, it is referred to as being 'instantiated'. Elements of the pattern, including the pattern itself, which will be altered upon instantiation are denoted by a triangle underneath them and are referred to as 'uninstantiated' (GSN Working Group, 2011:16). If they also happen to be undeveloped, the rhombus usually representing this fact features a separating line. Parameterised elements control the resulting argument similarly to how a software program's code controls the output for a given set of input variables. Continuing with this analogy, the pattern itself would serve as the control variables of a program. Finally, the developer selecting and instantiating the pattern would serve as the program. Both the text in each element, as well as the structure of the argument can be controlled to adapt to the context on instantiation. Throughout the pattern, text in brackets e.g. {tier n} represents parameterized arguments. Such arguments allow the user to control the relevant part of the goal structure or its content based on the subject system and its context. In the D-Case editor (Matsuno, 2011) the types of these arguments can be type-checked for consistency.

Safety argument patterns in GSN introduce two forms of abstraction into safety cases; structural and entity abstraction. Structural abstraction allows argument structure to be decided when the pattern is instantiated. Entity abstraction allows the details of individual elements i.e. entities to be decided on instantiation. In turn, both forms of abstraction are expressed via extensions to the GSN.

Structural abstraction is supported via the multiplicity and optionality extensions. Multiplicity allows n-ary relationships between pattern entities to be defined. Optionality defines a cardinality requirement for a relationship, either as a 1-of-many or multiple-of-many selection. Each of the extensions is diagrammatically shown in Figure 2.8.

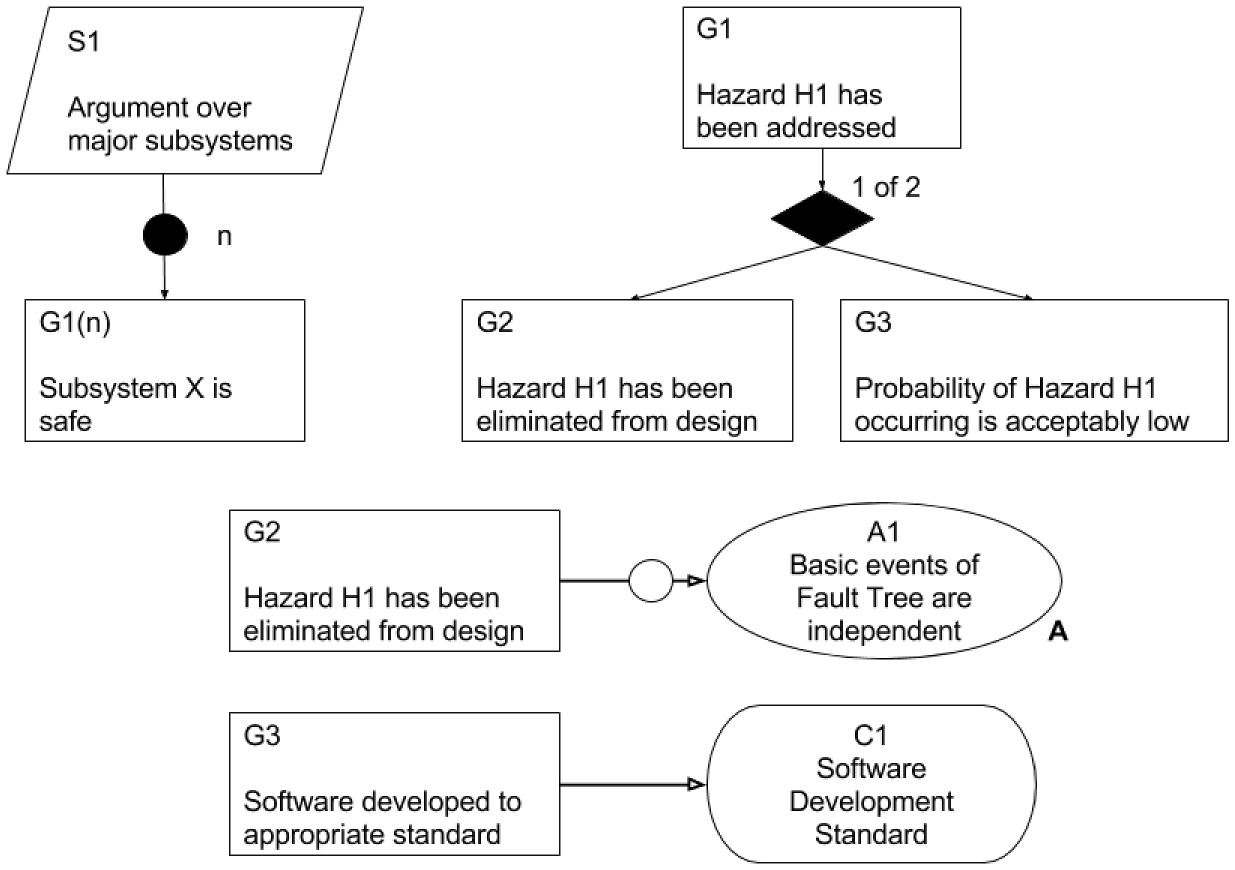

S1
Argument over major subsystems
n
G1(n)
Subsystem X is safe
G1
Hazard H1 has been addressed
1 of 2
G2
Hazard H1 has been eliminated from design
G3
Probability of Hazard H1 occurring is acceptably low
G2
Hazard H1 has been eliminated from design
A1
Basic events of Fault Tree are independent
A
G3
Software developed to appropriate standard
C1
Software Development Standard

Figure 2.8 – GSN Multiplicity & Optionality Extensions (Kelly, 1998:168–169)

In the figure, multiplicity is shown on the top left via an argument consisting of a strategy S1 and a claim G1. S1 iterates by repeating the G1 over each major subsystem of (presumably) a system. Optionality is shown on the top right by supporting claim G2 with either G3 or G4. Near the bottom of the figure, the optional inclusion of an abstract assumption and the inclusion of an abstract context to goals G2 and G3 are shown.

Entity abstraction links entities in argument patterns to entities in the instantiated arguments by abstracting the details of the latter. The usual way of establishing this link is through the use of 'Is_A' relations, a concept similar to that of objected-oriented inheritance. An example of such a relationship can be seen in Figure 2.9.

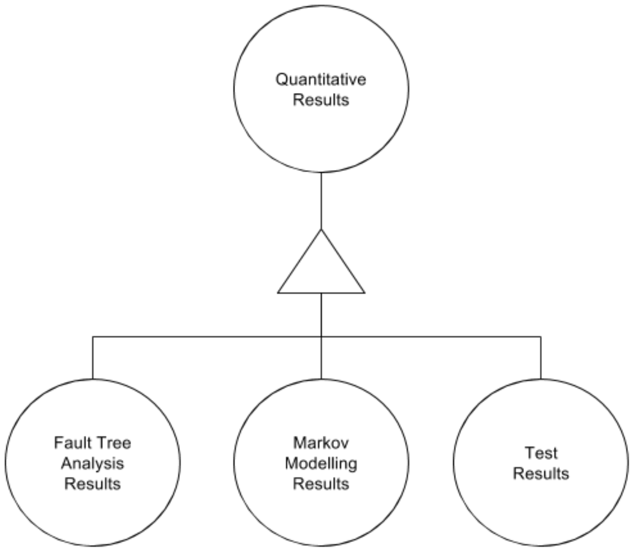


Figure 2.9 - GSN Entity Abstraction (Kelly, 1998:170)

In the figure, entity abstraction is demonstrated by setting three alternative types of analysis results to be equivalent to ‘quantitative results’. This allows either of the three types to replace the ‘quantitative results’ when used in a concrete safety case.

The advantages that argument patterns provide are manifold. Patterns allow for more consistent production of safety cases, as all of the generated instances share a pattern’s common features. Further, patterns typically capture what is considered to be ‘good practice’ in argument formulation. In processes largely left by standards to the discretion of the safety engineer to argue, the guidance captured in such patterns can be of significant utility. Patterns can also serve as a means to summarize argument structures more succinctly. Specifically, if we consider a scenario where a particular pattern is applied multiple times over the body of a safety case, those instances can be abbreviated by denoting the pattern application and the input to the pattern instead. Not only does this reduces the size of the parent argument structure, but it makes the safety case more comprehensive as well; understanding the pattern once allows the reader to easily understand its repeated application across the safety case.

Unfortunately, the notation systems do not provide strict rules for the parameters that control the patterns. For instance, in the example in Figure 2.7, when should the claim referring to the design introducing hazards be preferred versus that referring to the design containing hazards? Some of the reasoning is captured in the material accompanying the pattern e.g. pattern intent. However, the final decision and interpretation is left to the engineer's discretion. In the original evaluation of safety case patterns in (Kelly, 1998:219), they are evaluated to be closer to 'advisory material' rather than 'definitive solutions'. In this sense, the lack of detail noted in the pattern description such as in the example above is consistent with this view. That being said, when an argument pattern is used not as guidance material but a formula for constructing and understanding a safety case algorithmically constructed, more detail is mandated.

4.2. Safety Case Modules

Another means of managing safety arguments is the concept of argument modules. An argument module isolates an argument subgraph within it, including a reference to the top element within the subgraph (usually a claim). Through effective organization of a safety case into modules, the overall structure of the safety case can be summarized more succinctly. Even better, argument structures that are reused throughout a safety case can be formed once and then invoked by reference, reducing the potential for errors in reproduction. Finally, separating the safety case into modules allows for concurrent development of multiple strands of argumentation. This can be a useful advantage when multiple parties collaborate towards the construction of a safety case, as the overall progress can be maintained independently of delays in particular parts of the case.

In Figure 2.10 we can see an example of an 'away goal', which represents a claim residing in more detail in a different module. Strategies that refer to this claim for support can do so without immediately expanding upon the claim. Instead, the reader is referred to the module with the given identifier if he wishes to inspect the claim in-depth.

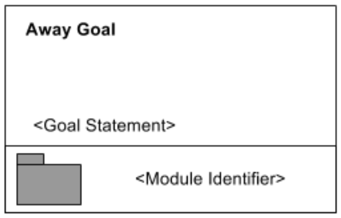


Figure 2.10 - GSN Module Away Goal, reproduced from (GSN Working Group, 2011)

In the GSN standard (GSN Working Group, 2011:17–24), further provisions for defining module-related GSN element interactions are described. For the purposes of this thesis, the inter-module notation will be presented in Figure 2.11.

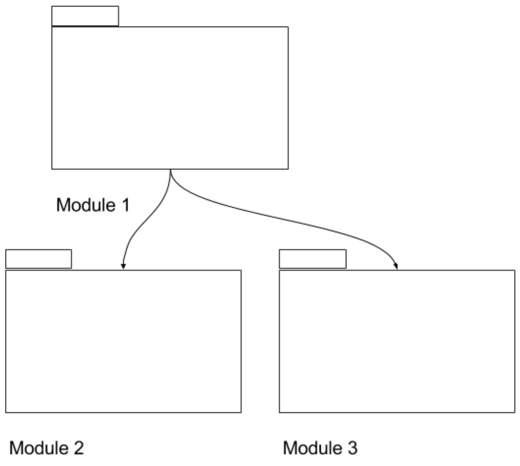


Figure 2.11 - GSN Inter-Module Notation, reproduced from (GSN Working Group, 2011)

The concept is relatively straightforward; in the figure, GSN elements from Module 1 are supported by one or more elements from Modules 2 and 3. This is a higher-level view of arguments featuring module hierarchies. We should also note the possibility for a module to both support and be supported by another module at the same time. However, this is restricted to cases where argument cycles are not created across modules by the inter-module support.

5. Safety-Critical Systems Development in the Civil Aircraft Domain

Aircraft transportation safety is an issue which deeply concerns both the general public as well as the manufacturers and operators of aircraft. Fortunately, despite the significant, potentially catastrophic impact that an aircraft accident can have, air transportation is still demonstrably safe compared to other modes of travel (Savage, 2013:6). Although this is certainly an achievement to be heralded, development of safe aircraft is certainly not an accomplished fact. As the complexity of the underlying systems becomes harder to manage, concerns that aircraft system developmental errors might not be properly addressed have risen (SAE, 2010:22). The Asiana Flight 214 crash, a Boeing 777 commercial aircraft in 2013, is an example of the complexity of newly introduced and hard to understand by the crew systems being the cause of the accident ("need for … Reduced design complexity and enhanced training on the airplane's autoflight system") (NTSB, 2014). Therefore, the need to maintain and improve safety assessment methodology is still very much present. In the following section, the concept of functional safety will be introduced, in order to establish the fundamental view of safety in systems.

5.1. Functional Safety

In (Bozzano & Villafiorita, 2003:22–23), system safety is defined to have three aspects:

- Primary Safety, covering risks directly influenced by hardware malfunction such as electrical burns from short circuiting.
- Functional Safety, covering risks associated with the operation of systems subject to risk-reduction measures. For example, incorrect detection of smoke leading to inadvertent activation of a fire suppression system.

- Indirect Safety, covering risks associated with the incorrect operation of a system, such as a pressure valve meter displaying an incorrect measurement of pressure.

At the system level, standards primarily focus on addressing issues related to functional safety, delegating discussion of primary and indirect safety to sub-system and component level views. Functional safety can further be separated into the categories of failures it attempts to address (Capelle & Houtermans, 2006:2–3):

- Random Hardware Failures, caused by material degradation over time. Some of them have a permanent effect, whereas the effect of others can be repaired. In either case, safety analysis and appropriate measures are recommended by standards in anticipation of eventual random failures.
- Common Cause Failures, caused by the coincidental failure of two or more separate sub-elements of the system, leading to total system failure. Common cause failures are defined to always stem from a common environmental cause. Standards typically prescribe the use of redundancy and multiple, independently developed components to mitigate them. This concept is referred to differently depending on the standard such as ensuring 'Diversity' in IEC 61508 (IEC, 2010:9) and 'Independence' in ARP4754-A (SAE, 2010:12).
- Systematic Failures, caused by errors in the design, implementation or documentation of a system. Unlike the previous types, systematic failures can include both software and hardware elements. To address these failures, a shared concept across standards is used under a different name, depending on the standard:
  - In IEC 61508 and EN 50126 (CENELEC, 1999), it is referred to as 'Safety Integrity Levels'.
  - In ISO 26262, 'Automotive Safety Integrity Levels'.
  - In ARP4754-A, 'Development Assurance Levels'.

The usage of SILs coincides with the transition from prescriptive standards to risk-management standards. This is indicated by the initial method of SIL assignment, as it is performed on the basis of a given system's risk assessment (see FHA in Section 5.5). SIL usage arguably drives the rest of the safety assessment activity, as it is used to control the rigor with which the various assessment activities are meant to be performed. The reason behind centralizing safety standards around this concept are systematic failures. To explain further, we should consider that traditional failure assessment techniques rely upon statistical models of failure. The assumption supporting the use of such techniques is that the system will eventually succumb to material degradation, leading to its failure. Given previous knowledge and research of material performance, the likelihood of failure can be quantitatively predicted. While this can be effective for non-electronic systems, software and hardware introduce complications.

For the physical aspect of hardware, statistical analysis of failure can still be applied, following the rationale stated above. However, the logical aspect, which software encompasses completely, does not fail based on statistical processes. Software fails systematically, when the conditions in the program result in an inadvertent state. SILs are employed as a rough indicator of the level of rigor safety assessment activities need to be conducted for a particular part of an aircraft architecture.

5.2. Safety Integrity Levels

Safety Integrity Levels (SILs) were introduced in 1996 in the Instrumentation Systems and Automation Society's 84.01 standard (ISA, 1997), which addressed Safety Instrumented Systems for the process industry. The concept was then included in an industry-neutral standard, the International Electrotechnical Commission's (IEC) 61508 and eventually gained widespread application in standards of specific industries in adapted forms.

In IEC 61508, SILs are defined as 'a discrete level (one of four) for specifying the safety integrity requirements of safety functions', where safety integrity is defined as 'the likelihood of a safety-related system satisfactorily performing the required safety functions under all the stated conditions, within a stated period of time' (Redmill, 2000:4).

Both the number of levels and the essential role of SILs across standards remain similar. Depending on the standard, there are differences in certain qualitative and quantitative properties which alter the context of their application, meaning that comparison of SILs (and similar levels) can only be meaningful under the same context, i.e. comparing SILs for systems of different industries is pointless.

Given the focus of the thesis on the aerospace domain, the role of DALs in certification needs to be examined in further detail. However, safety assurance encompasses not only DALs but the entire reasoning supporting the choice and implementation of safety assessment activities throughout development and operation. For this reason, a top-down view of how certification for civil aircraft via safety regulations is performed internationally will be provided in the following section.

5.3. Regulations for Civil Aircraft Systems Development

The framework for producing the evidence necessary for safety assurance is typically provided through the guidelines of one or more safety standards relevant to the subject system. Depending on the industry, the system may need to be certified by independent authorities and/or official regulatory bodies. In the case of civil aircraft, the standards set by the International Civil Aviation Organization (ICAO), the Federal Aviation Administration in the USA (FAA) or the European Aviation Safety Agency (EASA) are typically involved in the certification process, depending on the country where the certification in question is to be conducted. The organizations share many of their regulations, safety standards and overall philosophy, so the information in the following sections should be transferrable across all of them with minor differences.

For software and hardware systems found in civil aircraft, the applicable FAA regulations are found under the USA's Code of Federal Regulations (14/I/C/25/F/25) also referred to as Federal Aviation

Regulations / Joint Aviation Requirements (FAR/JAR) Part 25 (USA Government, 1964). Equivalent regulations are established by the EASA in Certification Specification (CS) CS-25 (EASA, 2011).

The requirements defined in Part 25 are very specific in some cases e.g. JAR 25.31 'Removable ballast': 'removable ballast may be used in showing compliance with the flight requirements'. In other cases, they are quite vague e.g. JAR 25.33 'Propeller speed and pitch limits': … '(1) Safe operation under normal conditions…'. Presumably, the purpose of this is to allow a degree of freedom to developers to pursue their own strategy to follow regulations.

However, via Advisory Circular documents such as AC 25.1309 more guidance and details are provided on "acceptable means for showing compliance with the requirements of FAR 25.1309" (FAA, 1988:2). AC 25.1309 in particular provides information regarding system design and analysis, which is essential in developing software and hardware systems for civil aircraft. The Society of Automotive Engineers (SAE) published a series of documents titled Aerospace Recommended Practice (ARP). ARP documents provide even further support and detailed guidance for developing systems by expanding upon AC 25.1309-1A. The central ARP document which directly provides guidance for systems development is ARP4754-A. Although these are technically guidelines, they are often referred to as safety standards in the literature due to their widespread acceptance in practice. Equivalent guidelines have been published for the European regulation counterparts, by the European Organization for Civil Aviation Equipment (EUROCAE). These documents are referred to as EUROCAE Documents (ED). For instance, the counterpart for ARP4754-A is ED-79A (EUROCAE, 2010). Although the thesis will focus on the ARP series, the methods developed should be directly applicable to the ED series as well.

5.4. Certification via ARP4754-A / ED-79A

The ARP4754-A or "Guidelines for Development of Civil Aircraft and Systems" (SAE, 2010) is part of a set of documents that are responsible for guiding developers through system development. Ideally, systems developed via recommended practice should result in certification according to the FAR 25.1309 regulations. The ARP documents supporting 4754-A and their relationship to it are presented in Figure 2.12.

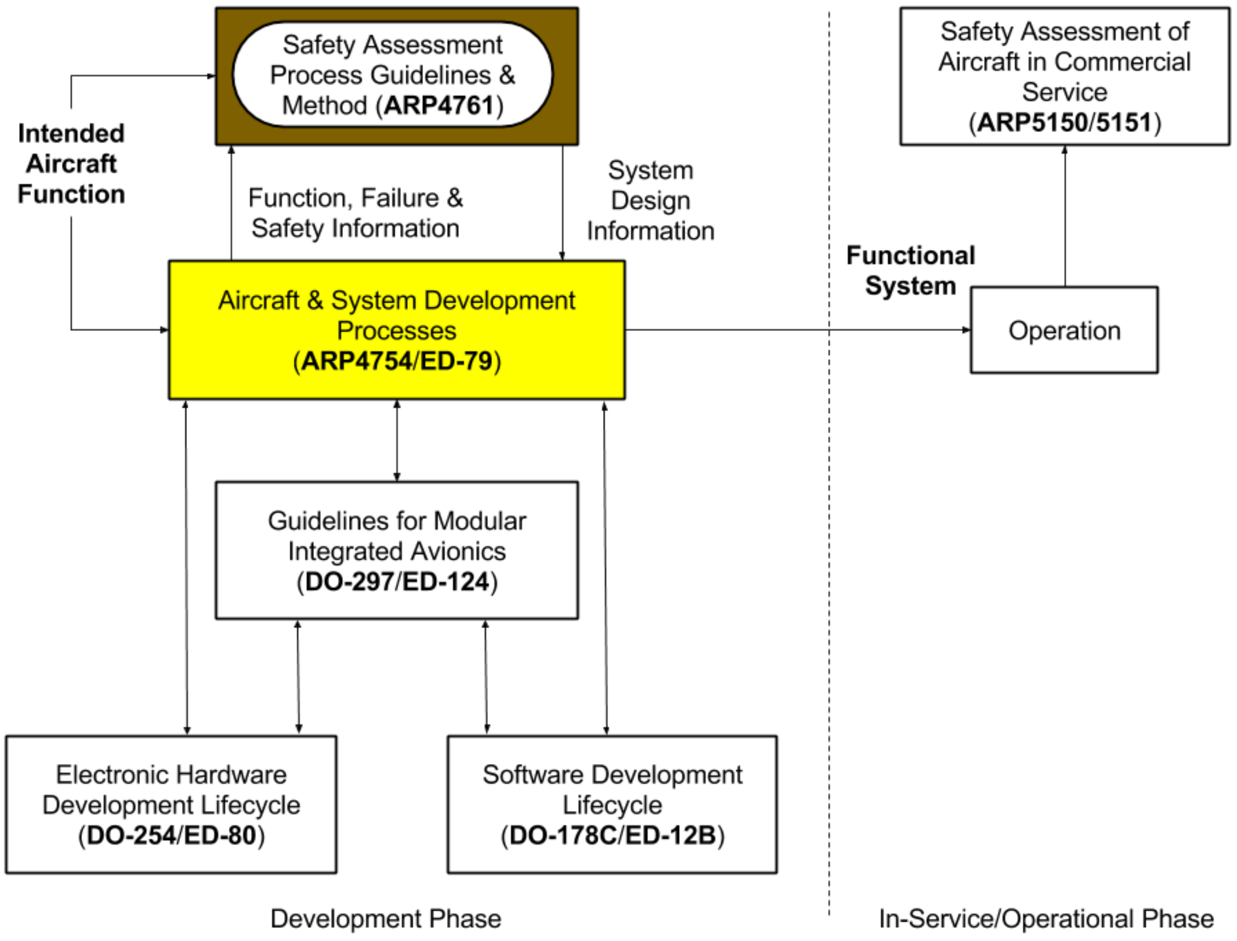

Safety Assessment Process Guidelines & Method (ARP4761)
Intended Aircraft Function
Function, Failure & Safety Information
System Design Information
Safety Assessment of Aircraft in Commercial Service (ARP5150/5151)
Functional System
Aircraft & System Development Processes (ARP4754/ED-79)
Operation
Guidelines for Modular Integrated Avionics (DO-297/ED-124)
Electronic Hardware Development Lifecycle (DO-254/ED-80)
Software Development Lifecycle (DO-178C/ED-12B)
Development Phase
In-Service/Operational Phase

Figure 2.12 - ARP Documentation, reproduced from (SAE, 2010:6)

ARP4754 is responsible for providing the central framework for development of systems for civil aircraft with regards to safety. The framework is composed of a collection of safety assessment processes, which are meant to be applied across an aircraft's software and hardware architecture. Specific guidance in applying the former processes is provided in ARP4761. For assessment of specific components, the Radio Technical Commission for Aeronautics (RTCA) organization has developed another series of documents, each referred to as Document Object (DO). These provide particular guidance depending on the nature of the component in question. As seen in Figure 2.12, for hardware and software, DO-254 (RTCA, 2000) and DO-178B (initially, now updated to DO-178C) (RTCA, 2011) and their ED counterparts, provide the appropriate guidance. When the aircraft has entered operation, ARP5150 provides guidance for its assessment instead. Given the emphasis the thesis will place on ARP4754-A, it will be referred to as 'the standard' from this point forward.

5.5. The ARP4754-A Development Lifecycle

The standard advocates an iterative, waterfall-like development lifecycle model, represented in Figure 2.13.

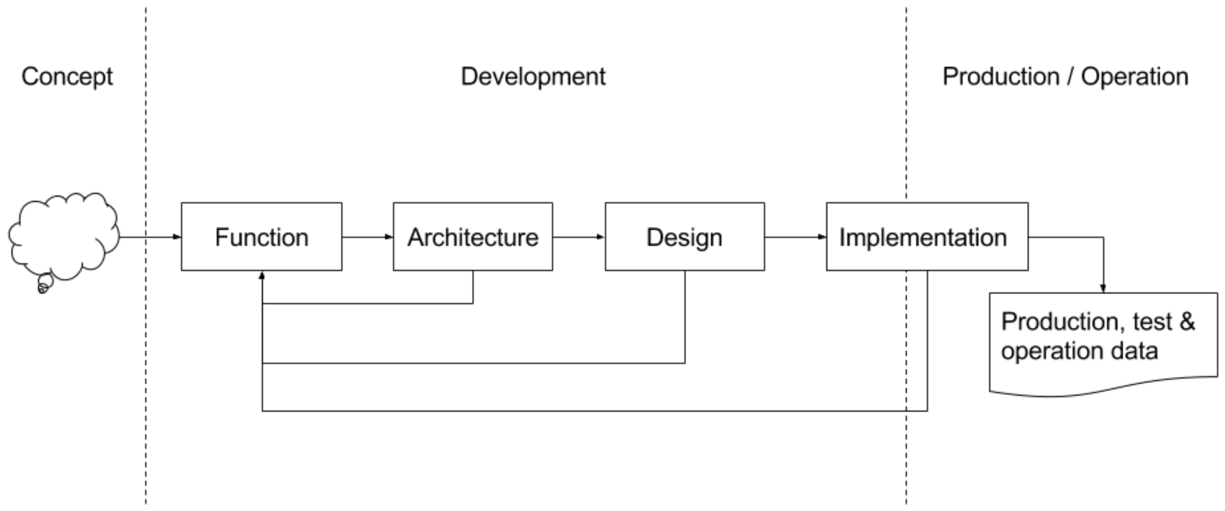


Figure 2.13 - ARP4754-A Development Lifecycle, reproduced from (SAE, 2010:20)

Initially, the concept of the aircraft is formed and fundamental aircraft requirements are determined. For instance, the aircraft size, its operational range, its passenger capacity, number of engines and other basic properties are determined at this point. Once the concept is finalized, the development stage initiates. During this stage, the aircraft's electronic systems architecture is designed, evaluated and implemented. The standard defines three broad types elements of the architecture can fall under:

- Functions, representing high-level functionality of architectural elements of the aircraft. Functions are defined as “intended behaviour of a product based on a defined set of requirements regardless of implementation” (SAE, 2010:11). The functionality refers to the architectural level being discussed. For instance, aircraft functionality is provided by so-called ‘aircraft functions’.

- Systems, which separate the overall aircraft architecture into particular units. The definition provided in Section 1 of the current chapter applies. One or more systems can combine their output to provide a particular aircraft or system function. Systems can be composed of other systems and items.

- Items, which represent architectural elements that cannot be further refined. The standard defines them as “a hardware or software element having well-bounded interfaces” (SAE, 2010:12).

Based on this aircraft architecture terminology, development initiates by identifying aircraft functions. Following this process, systems that support said functions are then designed during the ‘architecture’ step. Further systems that are internal to the former can also be identified during this step. The ‘design’ step in the figure refers to the development of items, perhaps an unexpected choice of description by the standard. The final step is the implementation of all of the items. The links that recur are meant to imply that within each step, necessary changes can be identified to elements identified or designed in earlier steps. Once implementation is complete, the aircraft can enter the ‘production/operation’ stage. Over the course of production and during operation, data collection and testing is performed for evaluation purposes.

The standard focuses on the development stage above all, with much of its safety assessment framework closely following the individual development stage processes. Figure 2.14 presents a more detailed view of the process framework, as viewed by the standard. The v-model, as it is known, represented in the figure, features two sides of processes. The left side is involved in the validation of the aircraft’s design, whereas the right side in its verification. Validation activities aim to confirm that the entire set of requirements identified for a system are complete and correct. Verification activities aim to confirm that the implemented system meets the requirements identified in the earlier stages of development. Development guidance from the standard focuses on a top-down approach for validation and bottom-up for verification.

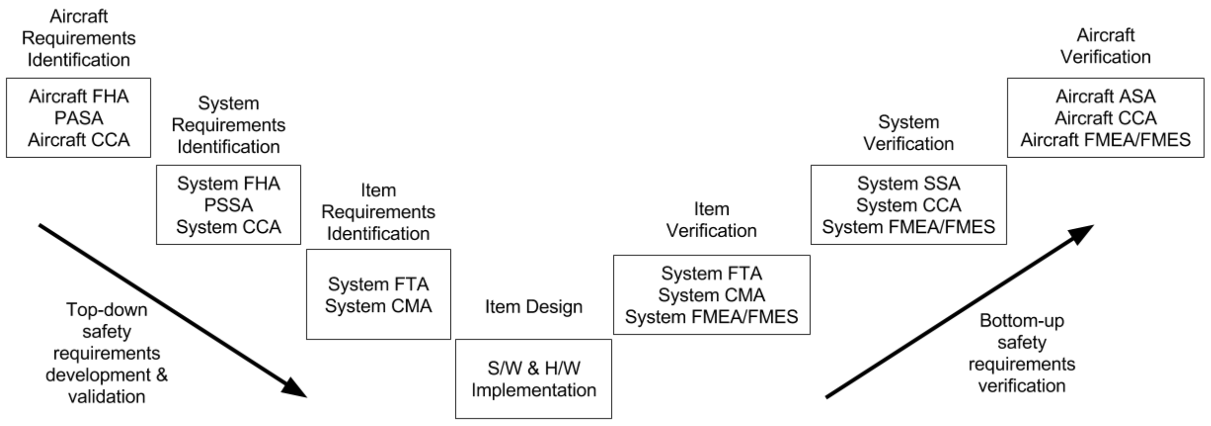


Figure 2.14 - ARP4754-A Safety Assessment Processes

The safety assessment framework begins during the aircraft requirements identification stage. During this stage, aircraft functions are defined, with additional requirements and their interface as well. To determine safety requirements associated with the aircraft functions, Functional Hazard Assessment (FHA) is conducted (SAE, 1996:16). The aim of each function's FHA is to identify potential hazards related to the former that can occur during operation, as well as each hazard's severity and likelihood of occurrence. Based on the hazards and their risk, particular safety objectives may be assigned from the FAR regulations. DALs are also assigned at this point, based on the hazard severity. Each function receives the highest DAL of the hazards associated with it.

Following the Aircraft FHA, the Preliminary Aircraft Safety Assessment (PASA) is conducted (SAE, 1996:17). The PASA evaluates proposed architectures meant to provide the aircraft functionality to determine how their failures could cause hazards identified by the FHA. The aim of the PASA is to identify any further safety requirements that need to be satisfied in order to meet the objectives identified by the FHA. During the PASA, function DALs are allocated to systems that provide their functionality. If multiple systems support a function and they provide failure redundancy for the function, DALs inherited by those systems can be reduced following certain rules. Further details can be seen in Section 2 of Chapter 3.

After the PASA, an aircraft-level Common Cause Analysis (CCA) is applied (SAE, 1996:26). CCA is a process that investigates whether earlier assumptions regarding independence between architectural elements of the system hold. This is particularly important to determine whether the DALs were allocated correctly from functions to systems. If systems are assigned lower DALs based on an assumption of independence and the assumption is invalidated by the CCA, a new allocation is needed.

Once the above processes have been adequately completed with regards to aircraft functions, they are applied again on the systems architecture that supports each function. System FHA, PSSA and System CCA reflect this application. Safety objectives, requirements and DALs are produced for lower-level systems and items. This cycle repeats until the level of individual systems (containing only items) is reached. At that point, basic failure and independence analyses are applied to identify the item requirements and DALs needed to complete the design step. The figure lists Fault Tree Analysis (FTA) (see Section 6.1) and Common Mode Analysis (CMA) although again, the standard is not restrictive to just those. CMA is one aspect of the CCA, focusing on failure modes that are shared across elements. Over the course of the design step, the validity of new requirements emerging from the development is evaluated. The standard suggests three structured methods of validation, although it also allows developers to apply their own techniques as well. The validation methods suggested include (SAE, 2010:62–63):

• Traceability, a method of directly linking lower to higher level requirements. Through it, validity is established by following the chain of requirements upwards to the highest level. For requirements that originate from lower levels of architecture, known as derived requirements (SAE, 2010:11), their design rationale should be captured instead.

- Various analysis methods, described in further detail in Section 6.
- Models of systems/items can validate requirements.
- Testing, simulation or demonstration. Item verification tests can also be used for validation.
- Similarity, a method of validating by comparison to similar, previously certified systems or items. The hazard severity of the functionality must match, as well as the environment and purpose of the systems or items. Further, the actual functionality must be similar in equivalent environments.
- Engineering review, where development staff inspects the requirements and determines their completeness and correctness, documenting the process itself and the rationale followed. The review is highly dependent on personal experience and similarity of the systems or items to previous cases.

If new requirements or changes to the earlier designs are needed, they are incorporated. Affected parts of the design must have the appropriate processes reapplied for them, in the standard top-down approach. If no further modification is needed, implementation can then begin. As individual items are implemented, the verification stage commences for each item. The verification stage follows the reverse flow compared to validation; it is bottom-up, verifying items, systems and finally functions.

Verification methods for items can be broadly grouped into testing and formal verification. Testing involves evaluating a program module for a set of input data (Ammann & Offutt, 2008). If the objective of the evaluation is to maximize certainty that a program will execute correctly in all cases, all possible inputs need to be tested. For non-trivial programs, this can become an intractable problem, which is computable in theory but impractical to execute due to enormously large run-times. The usual compromise is to identify intervals and particular input values based on requirements and then evaluate the program repeatedly by sampling this set. When safety requirements are relatively low or not applicable, this is arguably the default approach to testing. However, when safety risks are applicable, testing might not be sufficient to address them.

Formal verification methods express a program's behaviour as a formal specification and then confirm that the specification meets the requirements, also expressed formally. There are a number of approaches that can fall under formal verification; the most common in the context of safety-critical systems is model checking. Model checking involves systematic and, often, exhaustive investigation of the program's behaviour. A highly-regarded approach uses a branch of formal logic known as Computational Tree Logic (CTL) to express the program's potential states (Clarke & Emerson, 1981). The result is a CTL formula which can be parsed algorithmically to determine when a given property of the program is true, given an initial state of the program. This is possible, for example, by examining all possible transitions from the initial program state. Depending on the type of model checking method used, the execution can be exhaustive, in which case the results are definitive but might also include cases where the checking cannot terminate. Otherwise, the method can offer a trade-off in the interest of always terminating and provide a weaker answer. The more rigorous verification methods are usually applied towards items with higher DALs.

Once all items belonging to a system have been verified, verification is applied to the containing system. The item requirements are confirmed to be contributing to the system requirements by re-evaluating the system with failure analysis methods such as FTA, similar to the validation step. Failure Modes and Effects Analysis (FMEA) (see Section 6.5) is also employed during these stages to assist in confirming the correctness of requirements in the implementation. System Safety Assessment (SSA) and Aircraft Safety Assessment (ASA) encapsulate the verification activities for the system and aircraft function level, mirroring the PSSA and PASA respectively. Once the ASA is complete, the development stage concludes, with the system or aircraft proceeding to the production/operation stage.

5.6. Safety Cases in ARP4754-A

Traditionally, safety cases used to be produced at the end of the development stage. They functioned as a final report which supported the argument in of the system's safety, primarily aimed to convince regulatory authorities. In (Cullen, 1996), a number of drawbacks when following this approach have been identified, including:

- Significant redesign when a safety case does not satisfy authorities. This is particularly problematic when it is discovered that assumptions held during development are proven invalid, which can even endanger the success of a fully implemented system.
- The arguments presented tend to lack confidence, as they are referencing a fixed design instead of providing feedback for it. Such arguments tend to rely on probabilistic assumptions that invite challenge and doubt.
- The safety rationale followed during development is largely lost. This is problematic as an otherwise sound design decision might be argued inadequately, raising unneeded concern.

The above considerations were taken into account in more recent safety standards that require safety case construction. For instance, DS 00-56 (MoD, 1996) requires the safety case to be initiated as early as possible to address hazards immediately. Similarly, DSA02-DMR requires the safety case to be updated parallel to the project's development milestones. In DS 00-55, three versions of the safety are recommended to be produced:

- Preliminary Safety Case, following the system requirements identification stage
- Interim Safety Case, following the initial system design and preliminary requirements validation stage
- Operational Safety Case, immediately prior to operation

These three versions coincide with the v-model's vertices, as seen in Figure 2.15. In terms of the ARP4754-A, the Preliminary Safety Case is produced once the PASA is completed. The Interim Safety Case is produced following the PSSA's completion. Finally, the Operational version is produced once the ASA is completed.

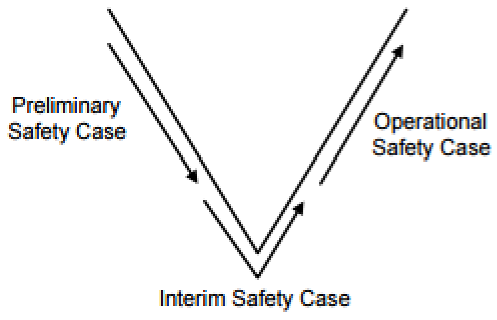


Figure 2.15 - Safety Case V-Model

Based on this view of maintaining the safety case up-to-date with development, generic milestones for safety cases from ARP4754-A can be identified. A direct way of choosing these would be to adopt the 3 versions from Figure 2.15 above. This means a preliminary safety case for the standard would need to capture all safety-related arguments once PSSA for all systems are complete. An interim safety case would include all information available immediately before implementation commences i.e. once all validation of requirements from the development stage is complete. Finally, an operational safety case would include all information available once SSA for all functions are complete. It would now be useful to review what advice the standard itself provides on the matter.

Constructing a safety case following the ARP4754-A guidelines involves documenting the results of the safety assessment processes applied. The production of the safety case is assigned by the standard to be the responsibility of the Aircraft Safety Group (ASG). The ASG is defined by the standard as an organizational unit whose responsibility is to coordinate with most of the other development teams to produce numerous deliverables, including the safety case. Unfortunately, beyond this, few details regarding the construction method, the expected content or the structure of the safety case are provided by the standard. In fact, only a base definition of the safety case is provided (SAE, 2010:35). This freeform approach is consistent with the view that developers should be allowed to choose their own means and rationale of safety assurance. However, the exact structure and contents of the safety case in the ARP4754-A are not as well defined as per other standards. This point is criticized in (Linling et al., 2011), where the authors note that additional guidance is needed to explicate the rationale of the standard, despite the standard's widespread acceptance.

Partially in order to address this shortcoming, effort has been made to explicate the rationale of the DO-178C guidelines into argument structures in (Holloway, 2015). Through this approach, the guidance provided within that standard can be translated into explicit arguments which explain how one would argue system safety in a certifiable way. This way, the documentation, safety assessment and assurance artefacts can be used directly within semi-official interpretation of the guidelines. Whether that interpretation provides an acceptable argument for certification via safety case would be the subject of negotiation between the argument provider and the certification body.

In (Kelly, 1998:33) the requirements for safety cases over several comparable standards are summarized. There are many similarities between the ARP4754-A and other standards, such as DS 00-55 and ISO 26262. Thus, the argument rationale could be adapted from the other standards, while incorporating the assessment methodology and recommended evidence from ARP4754-A.

Based on the findings in (Pfitzer et al., 2013:2), across a plethora of standards, it is advised that a process of risk analysis, mitigation and acceptance is an essential part of a safety case. Thus, both in the case of the ARP4754-A as well as in the general case, the above concepts form a fundamental block upon which a safety case can be structured. This block consists of an argument which explains how the risks discovered through investigative processes have been mitigated to acceptable levels via proposed measures. All of the above claims should be supported with adequate evidence for both the processes applied as well as the various assessment findings. Furthermore, the processes themselves should be chosen and documented to be correct and compliant with the guidelines. As mentioned, this would constitute only a part of the safety case, albeit a crucial one. Lower-level argument sections would explain how the evidence was established and delve into lower parts of the system architecture hierarchy. More specialized documentation, such as DO-178C for software component development, would guide this part of the argument.

6. Safety Assessment Methods

6.1. Fault Tree Analysis

Fault Tree Analysis (FTA) is a traditional safety analysis method that determines the base causes leading to an unwanted event, typically the failure of a given system. The technique was invented in 1961 to aid in the design of the Minuteman missile for the Boeing Corporation (Lee et al., 1985:1). Failure analysis is achieved by traversing a tree structure, referred to as a ‘fault tree’, where the top node is the unwanted event and the leaf nodes are the base events, linked via a set of logical gates such as AND and OR gates. An example of a fault tree can be seen in Figure 2.16. The example is abstract, with A, B, C and D representing component failures.

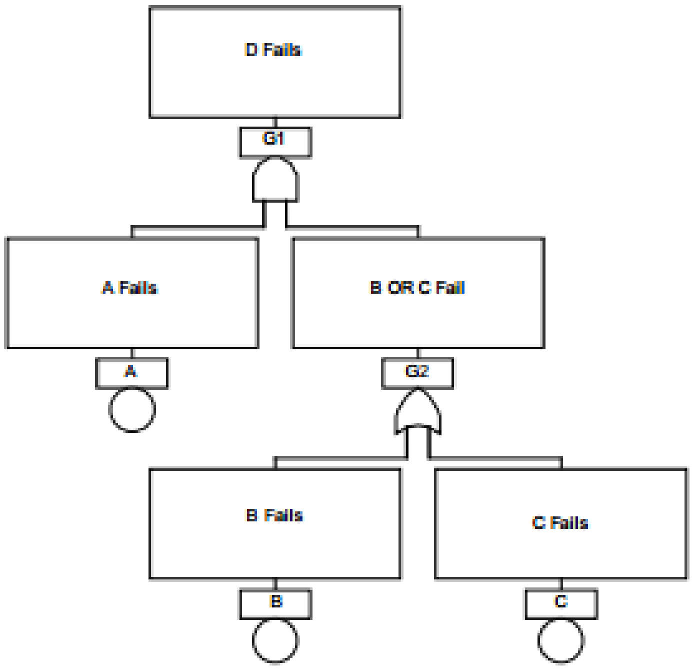


Figure 2.16 - 'A Simple Fault Tree' (Stamatelatos et al., 2002:49)

Analysis of the fault tree can yield critical information for the safety assessment process, both qualitative and quantitative. In the former case, the qualitative analysis of the fault tree can determine the combinations of base causes whose occurrence is necessary and sufficient to cause the top undesired event to occur, i.e. the system's failure. Identifying these combinations allows weak points with regards to safety to be recognized. For example, if a single base cause – a particular mode of failure of a component for instance - can cause the failure of the system, then this constitutes a single point of failure, which is significant cause for concern and potentially a motive for re-evaluation of the system's design. These combinations are referred to as 'minimal cut sets'. For quantitative analysis, the logical gates allow computation of the top event probability by combining the probabilities of the base and intermediate events. The benefits of this analysis are straightforward; knowledge of the system's likelihood of failure is essential in making any evaluation of its safety.

Construction and analysis of the fault tree is performed from the top of the tree to its leaves, aka ‘top-down’, which renders this a deductive technique. Its construction has traditionally been performed manually, however several tools that support automatic construction and analysis of fault trees have been developed and will be summarized in following sections.

6.2. Component Fault Trees

In traditional FTA modularisation is possible but only with regards to the structure of the tree and not the underlying system. Therefore, ‘there is no way to assign to each technical component a separate and reusable entity in FTA’ (Kaiser et al., 2003:1). In said publication, an extension to fault trees using the notion of components is proposed to address this weakness. Each component corresponds to a real component from a given system and the interconnections are represented as ‘ports’. An example of the partitioning of the fault tree can be seen in Figure 2.17. In it, a fault tree for a Controller System composed of a Main Controller, Aux Controller and Power Unit is shown. The AND gate is represented with a ‘&’ symbol and the OR gates with a ‘>=1’, as at least one of the base causes needs to occur for the gate to activate.

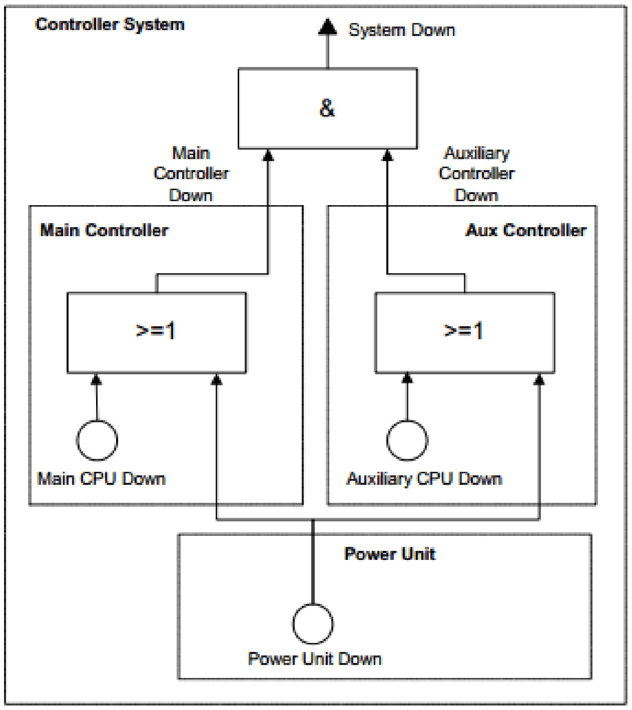


Figure 2.17 - Partition into Components (Kaiser et al., 2003:4)

The technique also extends the structure of fault trees into directed acyclic graphs, which it refers to as 'cause effect graphs' (CEGs). An example of such a graph can be seen in Figure 2.18. In the figure, the top event relies on the occurrence of both of the intermediate failure events; 'Main Controller Down' and 'Auxiliary Controller Down'; both of which can be caused if at least one of the basic causes linked to them occurs. In a traditional fault tree, the common basic cause 'Power Unit Down' would be repeated in this structure twice. CEGs also allow multiple top-events to be contained in the fault tree, allowing different modes of failure to be addressed simultaneously.

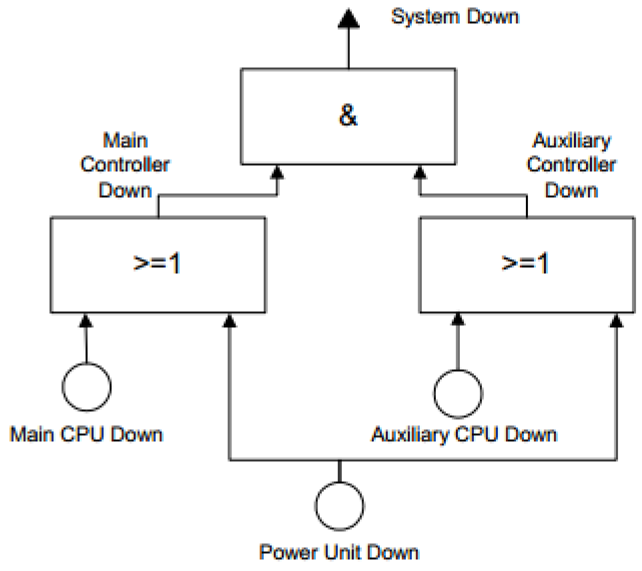


Figure 2.18 - Cause-Effect Graph (Kaiser et al., 2003:3)

6.3. Dynamic Fault Trees

Dynamic Fault Trees (DFTs), introduced in (Dugan et al., 1992), extend fault trees with the following types of gates:

- Functional Dependency Gates, which link a trigger event (a basic event or the output of another gate in the tree) and one or more basic events of the fault tree dependent on that trigger.
- Spare Gates, which link a primary unit with a set of redundant units. In the case of failure of the primary unit, the redundant units activate in its stead, in a defined sequence.
- Priority And Gates, which allow representation of failures which depend on a sequence of underlying events to occur.

Figure 2.19 depicts graphical representations of each of the above types of gates, respectively in the above order.

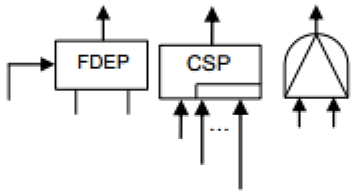


Figure 2.19 - From left to right: FDEP, Spare and PAND Gates (Li et al., 2012:3)

DFTs aim to address traditional FTA's inability to describe sequenced systemic behaviour, while still providing analytical capabilities. Analysis of DFTs is typically performed using Markov chains (Norris, 1998). Markov chains model a system as a state transition graph, assigning a probability of transition between states. Given this model, it is then possible to compute the probability of the system eventually being in a given state e.g. a failure state. The drawback of this technique is that the number of potential states in the chain can quickly become too many to practically compute as the input system size increases. Recent research on DFTs is summarised in (Walker, 2009:58–63), where alternative ways of performing analysis on DFTs and a new temporal gate, the Priority-Or Gate are presented.

6.4. State Event Fault Trees

Another weakness of traditional FTA is its lack of accounting for system behaviour, instead only examining an abstract, static state where the system can either fail or not. In (Kaiser & Gramlich, 2004) this weakness is addressed with the proposal of an extension to FTA, State Event Fault Trees (SEFTs). This extension adds the concepts of states and events to FTA, the former describing conditions with a timed duration and the latter representing instant occurrences or state transitions. Each type uses its own set of logical gates to propagate failure logic, for example state AND gates and event OR gates. As with CFTs, SEFTs also feature separation based on components and use of ports for communication between components. In the case of SEFTs, ports are also differentiated between states and events. In Figure 2.20, a fragment of a SEFT is shown.

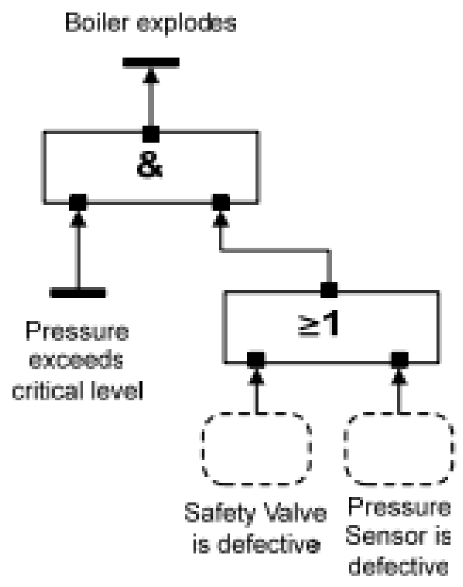

Boiler explodes
&
Pressure exceeds critical level
≥1
Safety Valve is defective
Pressure Sensor is defective

Figure 2.20 - SEFT Fragment (Kaiser & Gramlich, 2004:10)

SEFTs require a different approach in their analysis, via component-wise translation into Deterministic and Stochastic Petri Nets, which existing analysis tools can then assess. Petri Nets are a modelling method appropriate for discrete state systems that exhibit concurrency. The inclusion of both deterministic and stochastic variants accounts for the presence of time in the system, allowing probabilistic modelling of time-sensitive attributes such as the system state at a particular time interval.

6.5. Failure Modes and Effects Analysis

Failure Modes and Effects Analysis (FMEA) is a bottom-up or inductive analysis (SAE, 1996:135). Its purpose is to identify failure modes of a function, system or item and to determine the effects of each mode on the higher architectural levels. The analysis can be both qualitative and quantitative, as per FTA. In the latter case, each failure mode has an expected failure rate associated with it. In ARP4754-A, a summary of FMEAs, the Failure Modes and Effects Summary (FMES) is compiled. Its purpose is to assist with performing other assessment processes and methods such as the SSA or the FTA. Table 2.1 shows an example entry from an FMEA table.

Function	System	Failure Mode	Failure Rate	Failure Effect	Detection Method
Flight Control	Flight Control System	Omission	10e-9	Loss of Flight Control	Pilot Feedback via Flight Display

Table 2.1 - FMEA Example

In the example, the Flight Control function is determined to potentially suffer an omission. The failure rate is determined to correspond to feature a very low occurrence likelihood per flight hour, at 10e-9 (i.e. once per one billion flight hours). The effect of the failure mode is the loss of flight control. Because the high-level function happens to coincide with the system's output function, the effect is trivial in this case. In general, the relationship might be more complex and this entry would be more meaningful. Finally, the detection method is determined to provide feedback of the loss back to the pilots via the flight display in the cockpit.

6.6. Dependence Diagrams

Dependence Diagrams (SAE, 1996:104–107) are also known as Reliability Block Diagrams (RBDs) (Modarres et al., 1999). RBDs are used to represent the propagation of failure across elements of the system architecture. In Figure 2.21, an RBD for a system composed of 5 elements is shown.

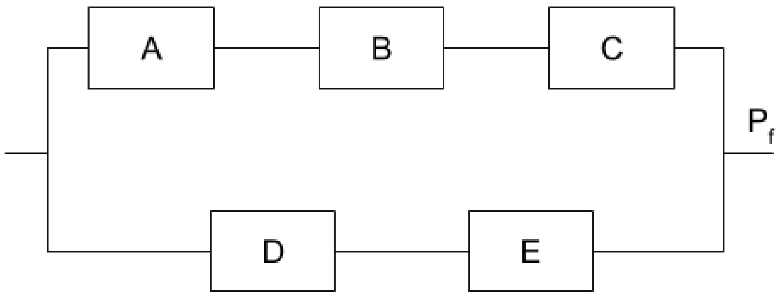


Figure 2.21 - RBD Example (SAE, 1996:104)

The failure propagation is meant to flow from left to right. On the left side is some input to the current slice of the architecture being viewed, while on the right, an output labelled Pf. The top 3 elements, A, B and C are said to be ‘in series’ configuration, meaning that faults propagate through them successively. If A fails, then both B and C fail. If B fails, then C fails. The same applies to the bottom elements, D and E. The two groups of elements are said to be ‘in parallel’ configuration, meaning that failure of the output can be triggered by either the top or bottom group’s failure independently.

RBDs also allow for quantitative analysis. For elements in failure series, the probability of failure is the sum of each element’s probability of failure. For elements in parallel, it’s the product of the two element groups. All of the above assume of course that the elements fail independently.

6.7. Markov Analysis

A Markov Analysis (MA) (SAE, 1996:108) is primarily focused on quantifying the probability of a system being in a possible state. It is most useful in safety analysis when a system features dynamic states, typically encountered when a system can recover from a failure state via repair. MA is performed by constructing a Markov ‘chain’, i.e. a model representing various states of the system, connecting each state to others via transition relationships. Each state can be characterized as operational or not (implying it has failed) and transitions are also characterized as functions of failure or repair rate. An example of such a model can be seen in Figure 2.22.

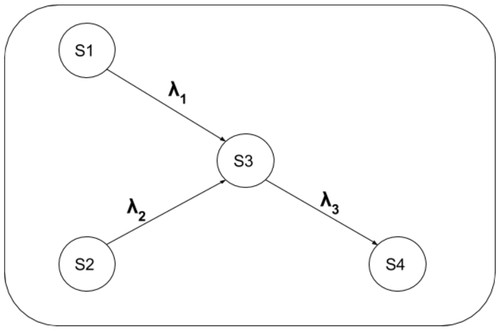


Figure 2.22 – Markov Model Example, reproduced from (SAE, 1996:111)

In the example, enclosed within the system boundary, are four system states, S1 to S4. Transitions to S3 are labelled as $\lambda 1$ and $\lambda 2$, whereas out of it with $\lambda 3$. These labels represent the probability of the system transitioning from a given state to another, e.g. $\lambda 1$ is the probability of transitioning from state S1 to S3. To determine the probability of the system being in a particular state, the rate of change of state probability over time for each state needs to be determined. In the example, the rate for S3 ($dP_3(t)$) over time can be described through the differential equation:

$$\frac{dP_3(t)}{dt} = \lambda_1 P_1(t) + \lambda_2 P_2(t) - \lambda_3 P_3(t)$$

Where $P_i(t)$ is the function of the probability of the system being in state Si at time t.

In other words, the rate of change of the system's probability to change from state S3 is the sum of the linear combination of the probability of finding itself in a state 'incoming' to S3 (S1 and S2 in the example) and the probability of transitioning to S3 minus the same linear combination for the states 'outgoing' from S3 (S4 in the example).

Similar differential equations can describe the rate of change for the other system states. By combining all the equations and solving this system, the probability of the system being in each state can be eventually computed. Due to the inherent difficulty in generating the model as well as solving the system of equations, software tools are often employed to apply the process.

In (SAE, 1996:109), MA is recommended over FTA and DDs in cases where the system includes dynamic characteristics such as having a failure or repair rate which is variable, dependent on the system state. MA also facilitates the description of systems with user-controlled modes of operation, such as flight modes on aircraft, as well as dynamic failure behaviour variability dependent on system state.

Despite those useful attributes, other analyses are more often applied compared to MA for several reasons. First of all, the method requires in-depth knowledge on both the construction of the model as well as in solving the equation system. Thus, the method compares unfavourably versus FTA and other techniques that are less difficult to use. Further, MA is limited by a lack of a systematic process when constructing its states and transitions. The scope of those elements is left to the discretion of the safety engineer and can often be inconsistent. Finally, MA scales badly as the system size increases, requiring a more simplistic modelling of the chains to effectively solve the resulting equations (Ericson, 2005:317–333). Thus, MA's most important benefit, the flexibility in accurately determining the probability of a dynamic system state is limited in large systems.

6.8. Failure Propagation and Transformation Notation

The Failure Propagation and Transformation Notation (FPTN) is a graphical technique for modelling failure behaviour of systems, by complementing existing failure analysis methods, including FTA and Failure Modes and Effects Analysis (Fenelon & McDermid, 1993). Like CFTs and SEFTs, FPTN also places emphasis on modularization of a given system into components, for which it defines input, output and internal behaviour failures. An example of FPTN can be seen in Figure 2.*23*.

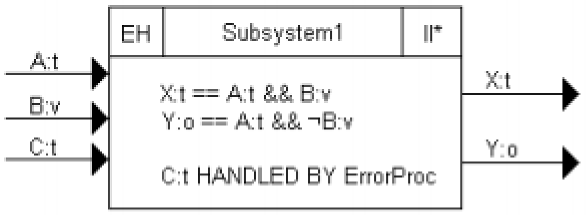


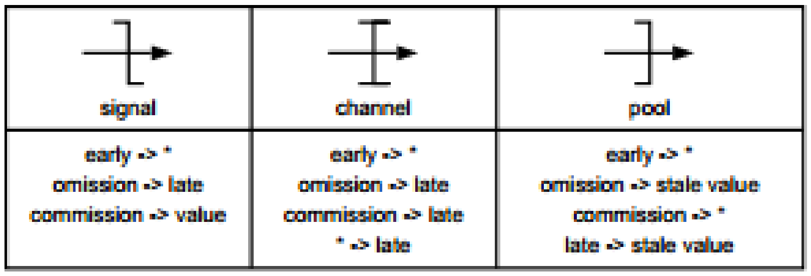
Figure 2.23 - *An example FPTN graphical description (Mahmud, 2012:56)*

In the figure, the arrows represent inputs (on the left) and outputs (on the right) of a subsystem named 'Subsystem1'. Each arrow is annotated with the type of failure behaviour it is representing, for instance, arrow 'A' is a 'timing' failure, caused on the arrival of input into the subsystem at an incorrect time interval. Also, note that each arrow is meant to propagate the corresponding failures from or towards the other components it is linked to respectively. The failure behaviour of the subsystem is described within the rectangle text, describing how each of the output failures are caused by the input failures.

'EH' refers to the subsystem's error handling mechanism, which addresses input timing failures represented by arrow 'C'. Finally, the subsystem's criticality class is 'II*', seen in the top right of the rectangle. FPTN's greatest weakness was the fact that its failure model is decoupled from the actual system model, therefore allowing discrepancies to occur as development progresses.

6.9. Failure Propagation and Transformation Calculus

The Fault Propagation and Transformation Calculus (FPTC) attempted to address the weaknesses of FPTN by connecting its failure model with the system model. The different types of failures are annotated directly on each system component under FPTC together with the failure propagation logic and the error-handling mechanisms are included as part of the non-failure behaviour as well. Figure 2.24 demonstrates the symbols used for denoting the propagation logic in a given system. Each symbol defines transitions for each type of failure as they propagate through.



signal	channel	pool
early -> * omission -> late commission -> value	early -> * omission -> late commission -> late * -> late	early -> * omission -> stale value commission -> * late -> stale value

Figure 2.24 - FPTC Expressions (Wallace, 2005:5)

6.10. Failure Propagation and Transformation Analysis

Failure Propagation and Transformation Analysis (FPTA) extends FPTN and FPTC by providing automatic quantitative analysis (Ge et al., 2009). FPTA introduces the concept of 'modes' for the connecting links between components. FPTA modes combine the failure types annotating the connections in FPTN (see Figure 2.25) and can be annotated with failure probabilities that can then be used to perform quantitative probabilistic failure analysis. This change essentially aggregates the each set of connections between components into a single entity, with modes and their probabilities as attributes. An example of failure propagation described in FPTA terms can be seen in Figure 2.25.

$$\begin{array}{ccl}
input.omission & \longrightarrow & output.omission\ ,0.0001 \\
input.value & \longrightarrow & output.omission\ ,0.0001 \\
input.normal & \longrightarrow & output.omission\ ,0.0001 \\
input.omission & \longrightarrow & output.omission\ ,0.9999 \\
input.value & \longrightarrow & \quad output.value\ \quad ,0.9999 \\
input.normal & \longrightarrow & \ \ output.normal\ \ ,0.9999
\end{array}$$

Figure 2.25 - Failure behaviour in FPTA (Ge et al., 2009:5)

7. Model-Based Safety Analysis

7.1. Introduction to MBSA

Model-Based Safety Analysis (MBSA) is introduced in (Joshi et al., 2006) in order to address the lack of a common structural model i.e. system specification, linking development and safety assessment processes. When using informal models, there is considerable risk of introducing errors during the flow of information from one process flow to the other. Many of the traditional safety analyses such as FHA and FTA are highly dependent on the skill and experience of the practitioner. When reviewing such analyses, consensus across the safety engineers must be reached to accept the results. The need for consensus places an even greater emphasis on maintaining an objective, correct and clear view of the information regarding the system's status and development. Furthermore, including structural models unlocks the potential for providing safety assessment automation support. In the following sections, safety tools and methodologies that support this paradigm. In (Sharvia & Papadopoulos, 2011a), the MBSA methodologies are distinguished into compositional and behavioural. In the former category, methods utilize safety information garnered from the system items to compose and analyse views of the system's failure behaviour. In the latter, methods use a variation of automated formal verification techniques to conduct the analysis.

7.2. MBSA Tools & Methodologies

7.2.1. Altarica

The Altarica tool framework provides a language which allows users to formally specify both the nominal and failure behaviour of a system (Arnold et al., 1999), (Bieber et al., 2004). This specification can then be analysed using a provided tool set in several ways, including FTA.

Describing a system under the Altarica language involves disassembling a system to its components and isolating their behaviour. Each component can then have its possible states, state transitions and interface boundaries with the rest of the components defined. Once this process is complete, the union of the possible states for each of the system components is produced and evaluated to determine how the system can transition from one combined state to another. An abstract example of this process for a system of three components can be seen in Figure 2.26. The system described in the figure contains three components, A, B and S, corresponding to the primary, backup and monitoring sensor components of the system respectively.

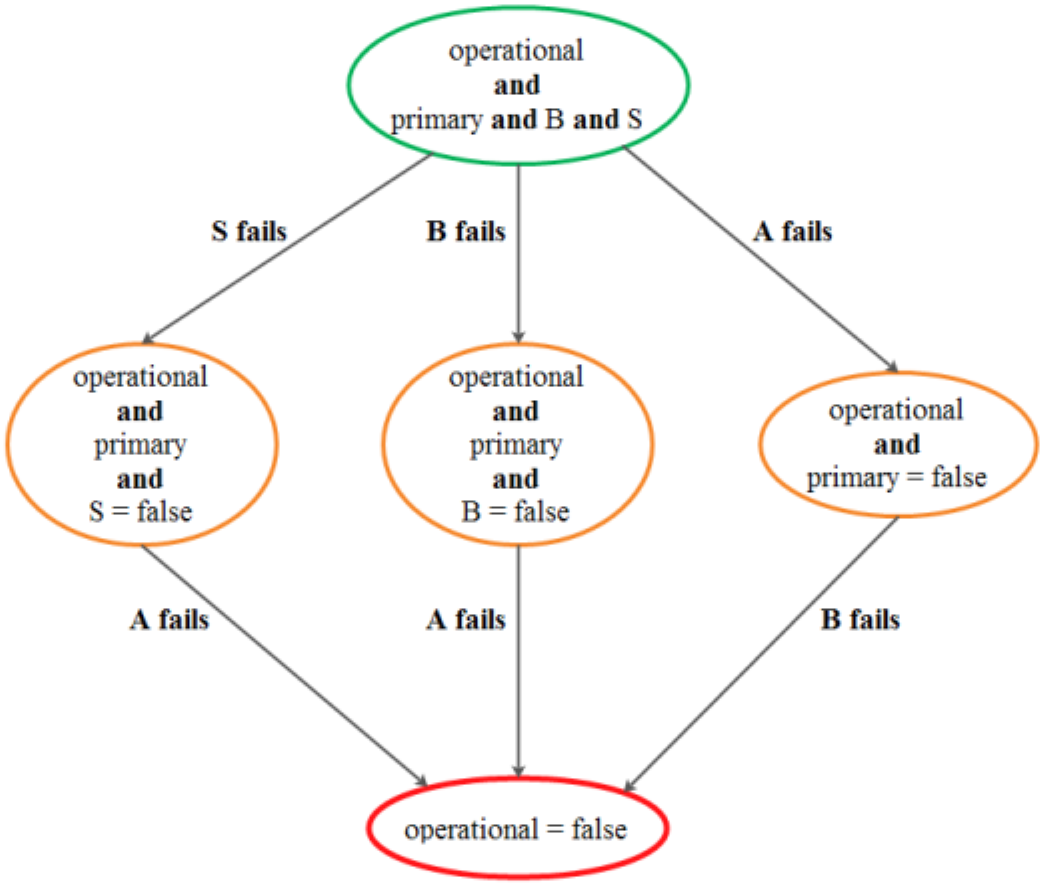


Figure 2.26 - Example of State Transitions in Altarica (Mahmud, 2012:57)

Once the specification is defined in the Altarica language, tools supporting the language can then analyse the model. In (Bieber et al., 2004) some key benefits and limitations of using Altarica tools are discussed. The authors mention that, given a component library, system modelling is effortless and rapid, due to its effective graphical modelling interface. While hierarchies of abstract systems can be built efficiently, capturing the failure propagation through a physical system model proved to be challenging. The language does not fully support the presence of failure propagation loops, meaning that syntactic workarounds need to be employed when they are a necessary feature of the system. A fault tree generator and analyser is also available for FTA, providing minimal cut set identification and quantitative analysis. However, the tools used do not support temporal FTA, which would allow event ordering to be a factor in the analysis. A sequence generator is provided instead, allowing limited, bounded search of the potential sequence space. Formal verification via model-checking is also available, applying an exhaustive simulation of the specification. The Altarica methodology shares the same major weakness as other state-based techniques. In particular, the increased number of options that need to be evaluated when determining the system's possible states can severely hinder the speed with which the state transition graph is formed. This can cause the rest of the analysis process to be delayed as well.

7.2.2. FSAP/nuSMV

The Formal Safety Analysis Platform (FSAP) (Bozzano & Villafiorita, 2003) is an extension of the New Symbolic Model Verifier (Cimatti et al., 2003). FSAP-NuSMV provides various forms of analysis of system models including model checking for user-defined properties, fault tree generation and analysis as well as generation of analysis artefacts such as simulation results, minimal cut sets and property counterexamples.

The tool operates following these main stages:

- Model Capturing, where the system's model is expressed in the NuSMV input language.
- Failure Mode Capturing and Model Extension, where the user defines modes of failure for the system and then injects them into the model, thereby extending it.
- Safety Requirement Capturing, where the user defines safety requirements for the model using temporal logic. A library of safety requirement patterns is available to facilitate this process.
- Model Analysis, where simulation of the extended model is performed. During this stage, the validity of the defined safety requirements is evaluated and counterexamples are generated when found. Additionally, fault trees are generated during this stage.

- Result Extraction and Analysis, during which the results of the analysis and simulation are compiled and made available in various commercially popular formats.

Several limitations of the tool have been noted in (Joshi et al., 2006). Specifically, the generated fault trees are ‘flat’ i.e. have a very limited depth of two levels only, causing them to be very broad given a sizeable input system model. This renders the produced fault trees harder to review and evaluate. Definition of the model’s failure behaviour is also problematic, as the specification of failure propagation or concurrent dependent failures is constrained. Being a state-based technique, the method also suffers from the weakness of state explosion mentioned earlier for Altarica.

7.2.3. Hierarchically Performed Hazard Origin Propagation Studies

Hierarchically Performed Hazard Origins and Propagation Studies (HiP-HOPS) (Papadopoulos et al., 2011) is a state-of-the-art MBSA software tool and methodology that largely automates the synthesis of fault trees and failure modes and effects analyses (FMEA) from system models. The method has been extended to support temporal FTA (Walker, 2009) and multi-objective optimisation of safety-critical architectures (Parker, 2010).

The overall process of HiP-HOPS can be seen in Figure 2.27. The process is broken into three main phases:

- Modelling phase, during which the system model is annotated with failure behaviour information. A modelling tool or software package such as SimulationX or MatLab/Simulink compatible with HiP-HOPS can be used to perform this procedure. The main benefit of this process is that it allows use of the same model as in the main development effort, ensuring that safety assessment can be performed concurrently.
- Synthesis phase, during which the fault trees are constructed from the annotated model. Using the component failure data, the failure propagation logic is determined. A network of fault trees

is produced, which links the failure of system outputs to failure modes of individual components.

- Analysis phase, during which the produced fault trees are analysed and a FMEA analysis is produced. The network of fault trees is minimized to compute the sets of combinations of elementary component failures which are necessary and sufficient of causing any system failure. These sets are used to perform a quantitative probabilistic analysis of the system's likelihood of failure (for each of the ways the system can fail) and a FMEA, which illustrates the effects of component failures on the system's behaviour.

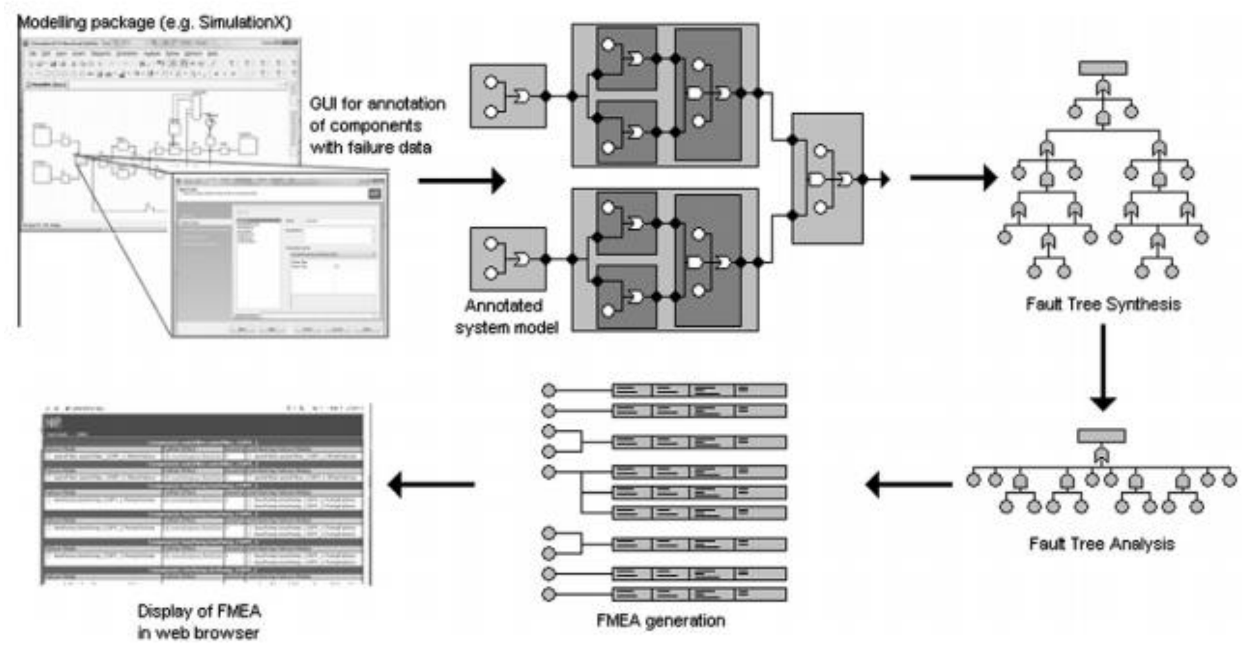

Modelling package (e.g. SimulationX)
GUI for annotation of components with failure data
Annotated system model
Fault Tree Synthesis
Fault Tree Analysis
FMEA generation
Display of FMEA in web browser

Figure 2.27 - Overview of HiP-HOPS Process (Papadopoulos et al., 2011:7)

HiP-HOPS features a number of extensions that have accumulated over years of research that provide an expansive feature set. For the purposes of this thesis, the automatic safety requirements allocation (Azevedo, 2015) extension is of particular interest and will be further presented in Section 6.1 of Chapter 3.

7.2.4. Galileo

Galileo (Sullivan et al., 1999) is a tool that supports FTA via the Dynamic Innovative Fault Tree (DIFTree) methodology (Dugan et al., 1997). The tool provides means for analysing and editing fault trees, however it does not support fault tree generation. Fault trees generated from other compatible software must be provided to it for analysis instead. Regarding DIFTree itself, it is a hybrid technique that allows analysis of both standard and temporal fault trees. For the former, the method translates the fault trees into equivalent Binary Decision Diagrams (Doyle & Dugan, 1995). For the latter, the fault trees are instead converted into equivalent Markov models (Dugan et al., 1992).

7.2.5. AADL Error-Model Annex

In (Delange & Feiler, 2014), the Error Model Annex of the Architecture Analysis & Design Language (AADL) safety standard is revised. The revision aka EMV2 enables the modelling of system failure behaviour and propagation. Further, analysis tools that directly support relevant safety standards from the civil aircraft domain such as DO-178C and ARP4761 are also described. AADL is used to model the system and relevant aspects of its failure behaviour. The language is extensible so additional user-defined properties or even specialized languages can be incorporated (Frana et al., 2007). System specifications can be expressed both graphically and textually. Given the model in AADL, the tools can generate FHA, FTA, MA and FMEA reports automatically.

8. Summary

In this chapter, a review of the literature surrounding safety case production and maintenance, civil aircraft safety certification and model-based safety analysis (MBSA) was presented. The GSN and CAE are found to be important methodologies in the construction, representation and management of safety cases. The concepts and approach supporting these notation systems are employed in Chapter 5 and 6 to develop and illustrate the proposed method for safety case generation. The framework found in ARP4754-A promotes the extensive use of DALs, so a mechanism for automatically allocating them across the aircraft architecture will be presented in Chapter 4. This mechanism provides a basis for generating the safety case.

Chapter 3: State-of-the-art Review and Method Selection

1. Introduction

To achieve the central objective of the thesis, two methods are proposed in chapters 4 and 5. The methods are directly based on the literature reviewed in this chapter. From this body of work, specific ideas, methods and techniques were later employed in the thesis' proposed methods. The reasoning which supported the selection of certain approaches over others is also presented. A more in-depth view of the issues surrounding SIL (DAL) allocation and safety argument generation is provided, preceding discussion of the specific approaches for each of the above problems. Later, in Section 8, issues regarding the construction of safety cases are also presented, after which current methods that aim to address them are also discussed.

2. DAL Allocation in Detail

As mentioned in Section 5.5 of Chapter 2, initial allocation of DALs is performed during the Functional Hazard Analysis (FHA). As each hazard's attributes are identified, it is assigned a DAL based on its worst-case severity. The assignment follows the correspondence in Table 3.1.

DAL	**A**	**B**	**C**	**D**	**E**
Hazard Severity	Catastrophic	Hazardous	Major	Minor	No Safety Effect

Table 3.1 - DAL to Hazard Severity Correspondence

A description of each category can be found in the FAR regulations, included in Table 3.2 for reference.

Severity	**Description of Potential Effects**

Catastrophic	Multiple fatalities, loss of airplane
Hazardous	• Large reduction of safety margins / functional capability • Flight crew overburdened / physically distressed; cannot be relied upon to manage workload effectively • Serious / fatal injury to relatively small number of passengers
Major	• Significant reduction of safety margins / functional capability • Flight crew workload significantly increased • Physical discomfort in crew or passengers
Minor	• Slight reduction of safety margins / functional capabilities • Slight increase in crew workload • Some physical discomfort to crew or passengers
No Safety Effect	No effect on safety

Table 3.2 - Hazard Severity and Effects (RTCA, 2011:13)

Every hazard identified by the FHA is assigned with an allowable average probability of occurrence per flight hour, as per Table 3.3. These probabilities are assigned as quantitative requirements to associated architectural elements, whereas DALs are qualitative requirements.

Severity	**Maximum likelihood of occurrence**
Catastrophic	At most 10e-9
Hazardous	More than 10e-9 & lower than 10e-7
Major	More than 10e-7 & lower than 10e-5

Minor	More than 10-e5
No Safety Effect	No requirement

Table 3.3 - Maximum Allowed Average Probability Per Flight Hour per Severity Level (EASA, 2011:8)

Once DALs have been assigned to functions, and an architecture for the system has been defined, the standard's rules for allocating DALs to systems/items are applied. These rules utilize the concept of Functional Failure Sets (FFS); these are sets that contain the minimum combinations of Functional Failures (FF) that are necessary and sufficient to cause an overall system failure (SAE, 2010:11). FFS are equivalent to Minimal Cut Sets (MCS), mentioned in Section 6.1 of Chapter 2. The items whose failures are members of these sets will be allocated DALs. For allocation purposes, we can assign each DAL a number, which helps express the allocation rules more succinctly. We assign DAL A to E numbers 4 to 0 respectively. Given a FFS with a DAL of **k**, its members can be allocated a DAL based on two options (SAE, 2010:44):

- Option 1: a singular FFS member is assigned a DAL of at least **k** and the other members DALs of at least **k-2**.
- Option 2: two FFS members are assigned a DAL of at least **k-1** and the other members DALs of at least **k-2**.

Note that in the case of a FFS with only one member, option 1 is always taken. Additionally, reduction past 0, e.g. DAL of -1 and lower, means the level allocated is the lowest, effectively DAL E.

The allocation rules effectively state that in a set of items which, by failing together, cause a system failure, either one of the elements must be developed at the hazard DAL with the rest developed at two DALs below, or two elements must be developed one DAL below the hazard DAL with the rest developed two DALs below the hazard integrity level. These options are applicable under the assumption that the members are independent, a concept explained further in Section 2.3 of Chapter 4.

Figure 3.1 illustrates an example of DAL allocation on an abstract system of DAL B.

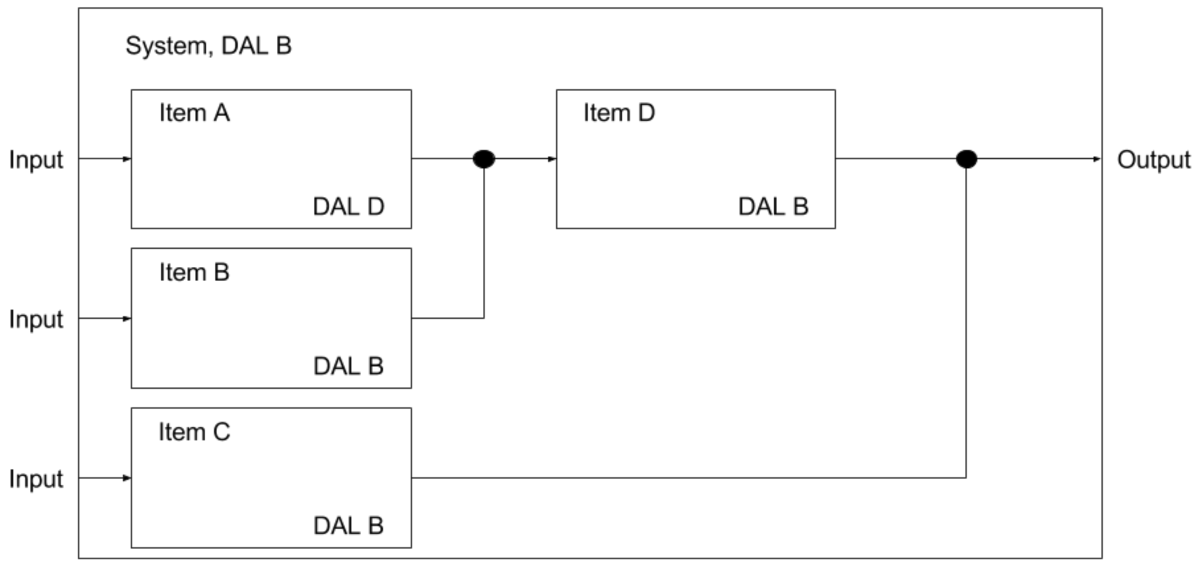


Figure 3.1 - Example System with DAL allocation

The arrows in the diagram represent the flow of information from the system inputs to the system output. For the purposes of the example, we disregard the input from our failure analysis.

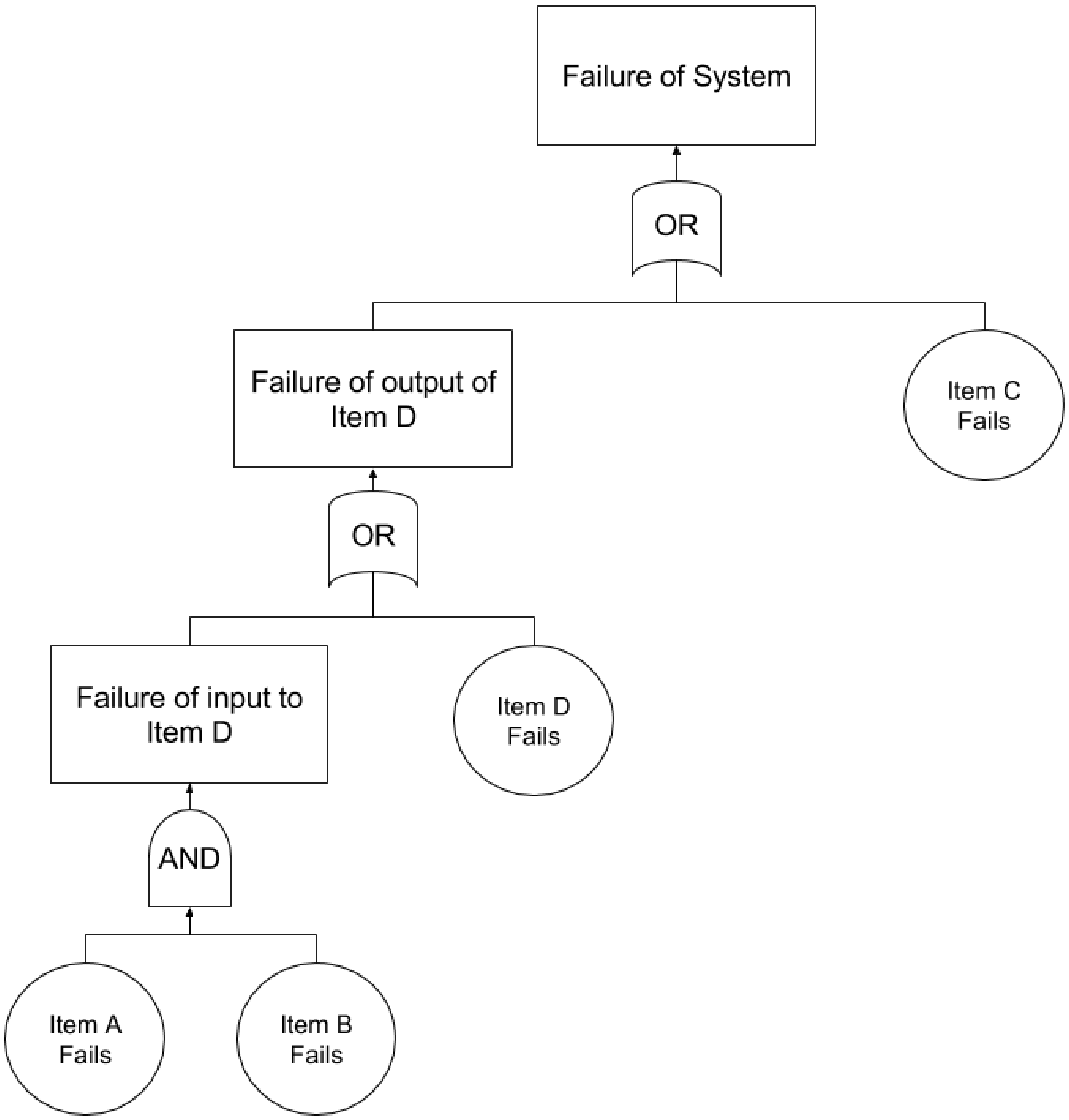


Figure 3.2 - Example System Fault Tree

The system's fault tree, depicted in Figure 3.2, depicts its failure behaviour. Specifically, its output depends on correct output from both Items C and D. Correct output for Item D depends on correct output from Items A and B as well.

Based on this simplified model, it is evident that the system can fail to produce a correct output when a failure in either item C or D occurs. However, for item D, as its input depends on items A and B, it is also possible that their combined failure can also cause D to fail.

Based on the above reasoning, the following FFSs can be constructed, and can more formally be derived through fault tree analysis:

- FFS 1: Failure of C
- FFS 2: Failure of D
- FFS 3: Failure of A and B

Given these FFSs, there are 3 possible options of allocation that satisfy the minimum necessary DAL for each item. Figure 3.1 presents only one of these options; the second option would allocate DAL B to item A and DAL D to item B; the third would assign Items A, B and D DAL C while Item C would then receive DAL A. This raises the question: which of these allocations is preferable? They all fulfil the integrity requirements of the system, so the decision must be based on another criterion. As will be shown at a later point, this criterion would be 'cost'.

At this point, it should be noted that this example is rather simplistic; in practice, the failure behaviour of both the items and the system can be much more complex, typically featuring multiple modes of failure for every output with different severity of failure and hence requirement for integrity. Each of those system failures may also have different causes and propagation of failure though the system which in turn will require application of different rules for DAL allocation.

3. DAL Allocation Development Cost Impact

The allocation of a specific DAL to a function or item typically implies a development cost which increases as the DAL increases. Higher DALs imply a higher level of rigor, more expensive development, safety assessment and assurance activities. This is clear in the standards where one can see, for instance, that the higher the DAL for a software item, the higher the number of assurance objectives that must be achieved, as shown in Table 3.4, which is based on the guidelines given in DO-178C/ED-12C (SAE, 2010; EUROCAE, 2012). Note that the objectives 'with independence requirement' are required to be assessed twice by separate teams of safety developers, necessitating even more time and effort to be expended towards the independent assessment.

Item DAL	A	B	C	D	E
Number of Objectives	71	69	62	26	0
Number of Objectives with independence requirement	30	18	5	2	0

Table 3.4 - Item DAL Objective Count (Nordhoff, 2012:7)

It is apparent that allocation schemes which can achieve the required integrity for the system by assigning lower DALs to more items would be more economical and translate into less effort and time spent on assurance activities. It is precisely those cost-optimal allocations that one is interested to find during the refinement of the architecture of a system under design.

4. DAL Allocation as a Constraint Optimization Problem

Now that the issue of cost has been introduced, the problem can be reframed. Specifically, the problem can be more formally defined as a constrained optimization problem, with the decision variables being the DALs of each item, the constraints being the rules of allocation defined in the standard and the objective of the optimization to minimize the overall cost of the allocation on the development.

This description can be summarized in the following expressions:

$$\underset{X}{\operatorname{argmin}} \sum_{i=0}^{n} Cost(X_i)$$

X_i: the i-th allocated item DAL across all functions

Cost: the cost function, assigning each DAL a specific cost

Therefore, we attempt to identify the allocation of item DALs across all functions which minimizes the total cost impact on the system.

Subject to either of the constraints:

$$\exists X_{ik} \in X_k : [X_{ik} \geq k \wedge (\forall X_{jk} \in X_k \geq k-2, i \neq j)]$$

Or

$$\exists X_{ik}, X_{jk} \in X_k : [[(X_{ik} + 1 \geq k) \wedge (X_{jk} + 1 \geq k)] \wedge (\forall X_{mk} \in X_k \geq k-2, i \neq j \ \neq m)]$$

Where

n: the number of item DALs in the allocation

X_{ik}: the i-th allocated item DAL contributing to a function with a DAL of k

X_k: the set of DALs for items of a function with DAL of k

The two inequalities used as constraints represent the two options available when allocating DALs. The first option allocates one member of the set of items contributing to the system failure with at least the system's DAL and the rest with at least two levels lower. The second option assigns two members of the aforementioned set with at least one level lower and the rest with at least two levels lower.

5. Optimization Techniques

Most optimization techniques roughly fall into two categories; mathematical (linear/non-linear) programming and metaheuristic techniques. The former approach constrained optimization problems by formulating them under an appropriate mathematical model and computing a solution for it. The latter category features techniques which apply a combination of local and global search strategies to navigate through a space of possible solutions.

5.1. Linear Programming

Linear programming (LP) is a family of optimisation techniques which solve optimisation problems by modelling them as sets of linear equations (Sierksma, 2001). A DAL allocation problem can be solved by such techniques because its objective function and constraint functions can all be described using a system of linear equations. The problem can be further specialised as all the variables involved are integers, allowing Integer Linear Programming (ILP) techniques to be applicable.

In general, LP techniques approach the optimisation by attempting to find solutions along boundaries of the problem's search space. This is possible because the constraints can form a search space geometrically described as a convex polytope. In this convex region, a transition from any point within the space towards any other point within it will necessarily pass through the space itself (and not exit the space). Thus, finding the minimum or maximum of the problem is equivalent to finding the 'highest' or 'lowest' point on this polytope.

LP techniques typically guarantee finding the optimal solution providing one exists for the problem instance. However, by doing so, the search is typically unbounded and can extend to include the entire search space. This means that it is possible that the search might not terminate within reasonable amounts of execution time.

5.2. Metaheuristics

When attempting to solve a constrained optimization problem, such as finding the optimal DAL allocation, an effective way of evaluating the possible candidate solutions is required, essentially 'searching' the space of potential solutions intending to find the best. When this search is performed over the entirety of the potential solutions, it is considered a 'global search', as opposed to a 'local search', which evaluates only a small subset of candidate solutions. Locally searching is typically rapid, but can often remain constrained around local minima and never reach the optimal solution.

The term metaheuristics is based on a concatenation of two Greek words, ‘meta-‘, meaning ‘after’, ‘above’ or ‘beyond’, and ‘heuristic’ which originates from the Ancient Greek verb ‘ευρίσκειν’ translated as ‘to find’. Metaheuristics are a class of search techniques which perform a global search of the solution space by employing one or more local search methods based on an overall strategy. In (Blum & Roli, 2003), it is noted that a dynamic balance between intensification, ‘the exploitation of accumulated search experience’, and diversification, the ‘exploration of the search space’, is required to identify regions with desirable solutions and avoid others with previously evaluated or undesirable solutions.

5.2.1. Genetic Algorithms

Genetic algorithms (GAs) were invented by John Holland in the 1960s (Holland, 1992), based on previous work on merging concepts of biological evolution and computer science with ‘evolutionsstrategie’ in (Rechenberg, 1989) and ‘evolutionary programming’ in (Fogel et al., 1966). Genetic algorithms are based on mimicking the process of genetic evolution over a population which undergoes a series of generational transitions. At each iteration, certain members of the population reproduce. The choice of which reproduce is referred to as the ‘selection operator’, which selects the best members of the population. Once selected, an operation referred to as ‘crossover’ takes place, where the genes of the parties involved are intermixed to produce offspring. ‘Mutation’ can also occur on a rare basis, causing members of the population to have their genes alter slightly. Each of these processes is algorithmically mimicked, typically expressing each solution as an encoding of characters or digits, referred to as the ‘chromosome’.

5.2.2. Particle Swarm Optimization

Particle swarm optimization (PSO) is inspired by the behaviour of schools of fish or flocks of birds in nature, developed in (Eberhart & Kennedy, 1995). In either case, members of the ‘swarm’, representing individual birds or fish, called ‘particles’, navigate the space of possible solutions. The swarm’s aim is to gravitate towards the better solutions in the search space, as it would with food sources or migration destinations. However, each individual particle moves independently. Its choice of trajectory will generally follow the direction of the currently best-known solution, with a certain amount of random deviation from it. As particles traverse the space, they update their best-known solutions. The best solution can be then found by determining the best solution found amongst all particles.

5.2.3. Ant Colony Optimization

Ant Colony Optimization (ACO) is a metaheuristic inspired by the natural foraging behaviour of ants, first published in (Dorigo et al., 1991). When ants depart from a colony to gather food for it, they have been observed to follow relatively efficient paths, seemingly optimizing their route between the colony and the food source. It has been determined that their success relies on the chemical they deploy on the ground as they traverse it, called pheromone, with which they mark the paths they travel through and can detect. Pheromone depletes over time, therefore paths less travelled will have less pheromone deposited on them over time. Initially, the ants first choose randomly which path to take. However, as shorter routes are discovered, these routes are preferred by the ants, leading to greater concentrations of pheromone, leading to more ants preferring these routes.

Structuring a problem to be solved with ACO involves encoding it with a construction graph, whose nodes are components of solutions and edges are connections between components. Finding a solution involves determining a feasible walk in the graph. Examples of ACO applications can be found in (Dorigo et al., 2006).

5.2.4. Simulated Annealing

Simulated Annealing is based on a model which, as its name suggests, simulates the physical annealing process, where particles of a solid material shift towards positions of thermal equilibrium. In physical annealing, metals or other materials subject to heat processing are heated to increase their ductility, making them more pliable and easier to shape. Once the desired shape is taken, the material is then cooled to solidify and make its shape permanent. The initial model generated sample states of a thermodynamic system, developed in (Metropolis et al., 1953) and the method itself was independently proposed in (Kirkpatrick et al., 1983) and (Černý, 1985).

Algorithmically, the method iterates by starting at some predefined or random solution and searches neighbouring solutions. It uses a control parameter, called 'temperature', to alter the probability of whether on the algorithm's next iteration, a worse solution is accepted or not. This allows the method to alternate between 'heat' phases. When the temperature is high, worse solutions are likely to be discarded, therefore the local minima are more likely to be found and therefore the method intensifies. When the temperature is low, worse solutions are likely to be accepted and the method diversifies.

5.2.5. Tabu Search

Tabu Search (Glover, 1986) is a technique which owes its name to the unique memory structures it features. The memory structures are used to store recently evaluated candidate solutions and avoid them temporarily. The candidates stored in these structures are not eligible for generation of further candidates and are thereby considered 'Tabu' by the algorithm. By using these memory structures, the technique trades space for time and therefore accelerates the search for the optimal solution. The technique was created in 1986 by Glover and has been applied to a wide variety of optimization problems such as the Travelling Salesman Problem (Knox, 1989), Graph Colouring (Hertz & Werra, 1987) and Machine Scheduling (Laguna et al., 1991), amongst others.

In Figure 3.3, an abstract example of the algorithm's process over three iterations can be seen. The black point signifies the current position in the search space, the light blue points are possible candidates and red are candidates recently selected and are thus 'Tabu'. After a parameter-defined number of iterations, the red candidates will be forgotten, allowing them to be selected again.

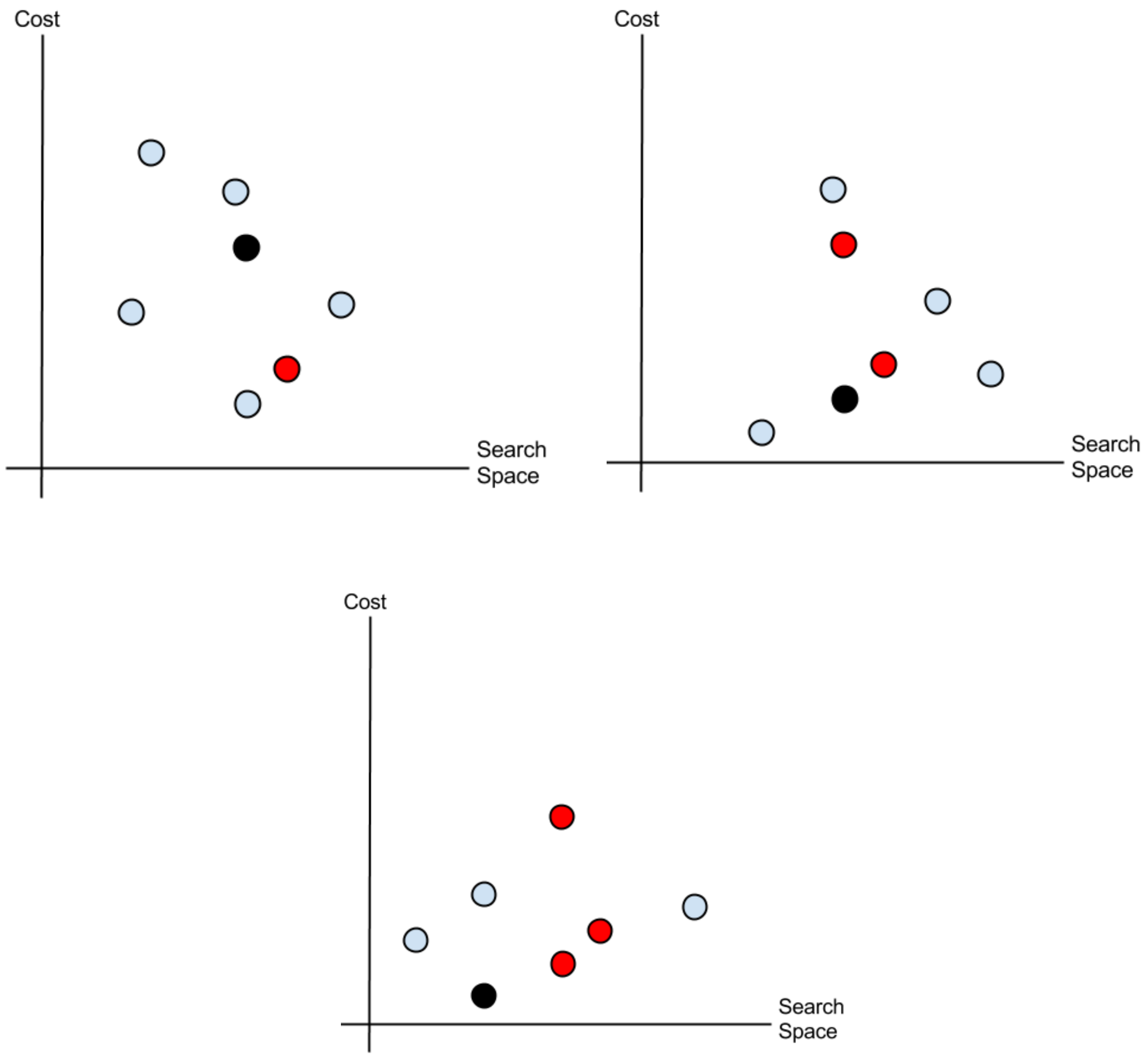


Figure 3.3 - 3 Steps of Tabu Search

Tabu Search operates by generating one or more candidate solutions at each iteration using an appropriate search method from a previous candidate, randomly generated at first. It then selects the best one out of the generated candidates. If this candidate has not been encountered in the recent past (see Tabu Tenure below), it is selected as the next iteration's candidate. If it has, the next available candidate is chosen.

The memory structure used to store recent previous candidates is called the Tabu Tenure. When a candidate is found to be present in the Tenure, it is said to be ‘Tabu’. A Tabu candidate can be selected for the iteration despite being Tabu if it meets certain Aspiration Criteria. These criteria are meant to allow exceptional candidates to be chosen more often than normal due to their desirable properties.

6. Previous Work in Automatic SIL Allocation

6.1. Metaheuristics for ASIL Allocation

In (Azevedo, 2015) a number of metaheuristics are investigated to evaluate their effectiveness in addressing ASIL allocation. ASILs are a concept similar to DALs for the automotive safety standard ISO 26262 (see Section 5.1 of Chapter 2). Tabu Search, Genetic Algorithms and Particle Swarm Optimization were identified as strong candidates for allocating ASILs. The Tabu Search technique employed is a variation based on (Hansen & Jaumard, 1990). The technique is referred to as Steepest Ascent Mildest Descent (SAMD). The term refers to the method’s strategy, which is to follow the ‘steepest’ or largest gain in objective function value until a local maximum is found. At that point, the ‘mildest’ or least-reducing options are used to move away from the local maximum. Because the ASIL allocation problem, as the DAL allocation problem, is a minimization problem instead, the term is reversed to Steepest Descent Mildest Ascent (SDMA). Based on the comparison across the three techniques, Tabu Search is overwhelmingly more efficient and equally as effective in identifying the optimal solution faster than the others (Azevedo, 2015:140–151). Specifically, it is found to be (roughly) “21 and 27 times faster than the ones presented by PSO and GA, respectively” (Azevedo, 2015:227), where PSO and GA refers to Particle Swarm Optimization and Genetic Algorithms. This approach is used to extend the HiP-HOPS methodology. Following the automatic FTA by HiP-HOPS, the produced minimal cut sets are used by the methods mentioned to optimally allocate ASILs.

6.2. Penguins Search Optimization Algorithm

In (Gheraibia & Moussaoui, 2013), an innovative nature-inspired metaheuristic is introduced. The Penguins Search Optimization Algorithm (PeSOA) is a population-based optimization approach based on the hunting behavioural patterns of penguins. The insight leading to this discovery is based on a view from an economic perspective of their hunting strategy. The aim of their hunting is to ensure that more energy is gained via the consumption of caught fish versus the energy spent hunting. This positive energy accumulation cycle is essential for their survival. PeSOA works by generating a number of 'penguin' groups. Each 'penguin' represents a solution e.g. an allocation of DALs for this problem instance. The algorithm then iterates over a fixed number of 'generations'. Each generation, every penguin adjusts its diving position via a combination of random movement and a cost-based stochastic correction. The correction accounts for the difference between the recently found 'best' solution and the previous solution. The penguins continue to make random movements and adjusting their positions until their predetermined oxygen levels are depleted. Once oxygen reserves for all penguins are depleted, the solutions found, expressed as 'fish eaten' in particular locations of the search space are recorded to influence the next iterations.

PeSOA has demonstrated satisfactory performance compared to other population-based approaches such as GAs and PSO, especially at lower number of iterations (Gheraibia & Moussaoui, 2013:8). The approach was also applied towards allocating Automotive Safety Integrity Levels (ASILs). ASILs are a concept similar to DALs, for the ISO 26262 safety standard (see Section 5.1 of Chapter 2). PeSOA delivered a superior performance in identifying optimal solutions faster compared to Tabu Search and Genetic Algorithms, seen in (Gheraibia et al., 2015:5). Once again, the method functions as an extension to HiP-HOPS by using the minimal cut sets it produces for the system being analysed.

6.3. ILP Solver with Papyrus Front-End

In (Mader et al., 2012), a technique for allocating ASILs using ILP is introduced. Once again, the technique requires the user to annotate a system model with component failure behaviour expressions which are used to compose fault trees. Fault trees are then analysed to establish causal relationship between component failures and targeted hazards. An integer linear problem specification is then constructed, with two sets of constraints. The first is the allocation rules for ASILs, which is of course fixed. The second is an optional, predefined allocation of ASILs for components that the user needs to exclude from the optimization search. The technique is implemented as a plugin to the Papyrus modelling environment (Gérard et al., 2010).

6.4. Exact Solvers

In (Murashkin et al., 2015), mathematical programming solvers are employed to find all of the solutions for optimally allocating ASILs to a system architecture. Three categories of solvers were investigated, Constraint Satisfaction Problem (CSP), Satisfiability Modulo Theories (SMT) and ILP solvers. The solvers were used to identify not just one but all cost-optimal ASIL allocations. Three versions of a system and four different cost heuristics were experimented with, using commercially available solvers from the categories mentioned above. The results show that the Z3 SMT (De Moura & Bjørner, 2008) and CPLEX ILP (Chang et al., 2001) solvers were very efficient in general regarding the processing times required. However, in some instances the optimization process required 'several hours or even multiple days' to complete the search of all solutions as reported in (Azevedo, 2015:243).

6.5. DALCulus

In (Bieber et al., 2011), an approach to allocating DALs automatically is presented, named DALCulus. The approach uses pre-calculated minimal cut sets and constructs a pseudo-Boolean integer program specification from them. It then proceeds to solve the optimization problem using two solvers, Sat4j (Le

Berre & Parrain, 2010) and Weighted Boolean Optimisation (WBO) algorithms (Manquinho et al., 2010). The results of the tests shown indicate the approach is very effective. The approach was able to allocate DALs consistently within milliseconds to a range of systems both abstract and from industrial sources. The tests were run using an associated tool named DALCulator.

The authors note the lack of minimal cut set management in their method compared to (Papadopoulos et al., 2011), which featured allocation of ASILs via HiP-HOPS. Furthermore, DALCulus produces an additional set of constraints specifically for ensuring independence for the DALs allocated. The authors interpret the standard to require minimal cut sets to contain a minimum number of independent items. The limit depends on the highest hazard severity classification the cut sets contribute to. For example, a minimal cut set contributing to a hazard of catastrophic severity 'should contain at least three items that are mutually independent' (Bieber et al., 2011:7). It is not clear how this rule is derived from the ARP standard. Section 2.3 of Chapter 4 addresses the issue of independence by the method presented in this thesis.

6.6. ASIL Allocation using System of Linear Equations

In (Dhouibi et al., 2014), the results of FTA are used to form systems of linear equations. The FTA proposed seems to be applied manually, producing minimal cut sets. Each minimal cut set is expressed as a sum of the ASILs of its members adding up to the ASIL of the hazard they contribute to. The equations are translated into a numerical matrix with columns dependent on the number of unique cut set members and rows dependent on the number of hazards. The matrix is reduced to its row echelon form and simplified repeatedly by the algorithm proposed, finding all optimal allocations eventually. The process also allows for fixing the ASIL of a particular member beforehand. The effectiveness of the process is demonstrated via experimental results on project examples from the Valeo technology company (Dhouibi et al., 2014:4). The algorithm proposed managed to identify all superior solutions within milliseconds.

The authors propose that all solutions that are not made redundant by other solutions should be available to developers. The basis for this view is that there is no established cost model for evaluating the impact of ASIL implementation on the development process. This approach is feasible for small to mid-scale systems; however, the lack of search bounds means the process might fail to address larger cases.

7. Optimization Method Selection

For the purposes of this thesis, Tabu Search was identified early as the most appealing avenue of further investigation. Its excellent performance for allocating ASILs, mentioned in Section 6.1, coupled with the application of HiP-HOPS indicated a high likelihood of success for developing an approach for DALs. Given the central aim of the thesis, if the results produced by the Tabu Search for DAL allocation were unsatisfactory, other metaheuristics would have been investigated further. However, that was not the case, as explained in Chapter 4.

Linear programming approaches (including integer programming) were considered as well. However, the concerns about the performance of these approaches regarding execution time and search space were sufficient to encourage prioritizing investigation of Tabu Search first. This rendered further investigation, for the purposes of this thesis, a moot point, as the extension to support DAL allocation was implemented successfully via Tabu Search. Of course, further work should investigate both other metaheuristic and linear programming approaches to compare and identify more effective and efficient methods for DAL allocation.

8. Challenges in Safety Case Construction

Earlier in Chapter 2, some of the extensions to the GSN notation were presented, namely argument patterns and modules. The importance of these and other facilities in effectively managing a safety case should not be understated. However, the production and maintenance of the safety case still remain largely manual processes. The user is still required to collect system architecture and failure behaviour information and then develop the safety case. The task of producing the safety case is relatively more structured compared to the past approach, however, it continues to demand a significant time and effort investment. At first glance, this fact might appear to be a minor inconvenience. However, if we consider the increasingly integrated nature of contemporary electronic systems, the implications should become apparent.

Towards this end, let us take into account a generic system's development lifecycle. Over the course of its lifecycle, the system architecture is likely to undergo numerous changes. This is especially true when iterative design approaches such as SCRUM (Schwaber, 2004), and other agile processes, are applied. As changes to the system take place, relevant changes to the safety case are also necessitated to maintain its correctness. As the subject system grows in scale, not only does it take longer to produce a safety case manually to begin with, it also becomes increasingly harder to keep it updated.

Even when major or minor architectural design evolution is not a concern, knowledge regarding the system's safety status is likely to change. For example, let us suppose that early during development, a safety requirement is established. This requirement establishes a minimum level of performance from the software system. Based on this requirement, a safety claim for the system is argued. At a later stage of development, it might become apparent that the requirement associated with the safety claim cannot be met while providing the required quality of service. As quality of service is deemed more important in this example by the development team, the performance requirement must now be discarded. Then, a new approach towards supporting the safety claim for the system needs to be established. For instance, providing additional verification for the software system might be deemed sufficient to mitigate its under-performance. This would replace the earlier, now discarded claim, that must be supported with the appropriate verification evidence. As mentioned in Chapter 2, the safety case should reflect – ideally at all times – the current state of the system's safety argument. Therefore, safety arguments can and should be susceptible to change throughout the system's lifecycle, even when its architecture remains stable.

The number of manual changes that the safety case developers must make can impede their progress, delay crucial feedback to the nominal design or maintenance process, and introduce errors. As an example, the preliminary safety case for the EOSAN project is about 200 pages long and 'expected to grow' (Denney & Pai, 2014:2) as development progresses (EUROCONTROL, 2011). In this case, major design changes could require rewriting large parts of the document, depending on the extent of the changes, in addition to reapplying the development process to implement those changes. This can represent a significant cost spent in time and manpower, and an incentive against maintaining the safety case throughout the system's lifecycle.

8.1.1. Previous Work in Safety Argument Generation

8.1.1.1. FLAR2SAF

In (Sljivo et al., 2014), an approach relevant to this thesis is presented, named FLAR2SAF (Failure Logic Analysis Results to Safety Case Argument Fragments). Failure Logic Analysis is performed via a plugin for the CHESS toolset (Mazzini et al., 2016), CHESS-FLA (Gallina et al., 2012), implementing Fault Propagation and Transformation Calculus (FPTC). FPTC allows a component's input/output failure behaviour to be specified via transformation rules and then analysed qualitatively and quantitatively, as mentioned in Section 6.9 of Chapter 2.

For example, a component with an internal fault-tolerance mechanism can transform an omission of an input into a late output from one of its outputs. This transformation can be annotated in FPTC and from it, an initial safety contract can be derived. Safety contracts define context-dependent or context-independent safety-relevant properties a given component needs (or promises) to hold during operation. Weak safety contracts are context-dependent, whereas strong safety contracts are context-independent (Sljivo et al., 2014:4). The initial safety contract has the FPTC analysis attached to it as evidence and can later be supplemented with additional validation/verification evidence as it becomes available over the course of development. From the safety contracts produced, combined with the supporting evidence provided, FLAR2SAF produces safety argument fragments based on argument patterns.

8.1.1.2. THRUST and related process-oriented methodologies

THRUST is a methodology presented in (Gallina et al., 2014b), which combines the ideas of the safety-oriented process line from (Gallina, 2014) and the process-based argumentation line from (Gallina et al., 2014a). Process lines model processes as sequences of tasks which can contain variation points or branch out into separate variants. Tasks are under the care of people in specific roles, who use tools to produce work products that satisfy the tasks. Finally, a process can feature multiple phases and applied based on particular guidance. Argumentation lines effectively constitute process-based safety arguments. The argumentation elements reference elements from a corresponding process line and the argument structure argues the soundness of the process line’s structure with regards to safety standard rationale.

Under THRUST, the safety process is modelled using the above elements, generically interpreted from safety standards or appropriately tailored to a particular development project. Extensions to the general-purpose language SPEM 2.0 (Bendraou et al., 2007), vSPEM (Martinez-Ruiz et al., 2011) for process lines and S-TunExSPEM (Gallina et al., 2014c) for argumentation lines can be employed to enable modelling. Process and argumentation lines are modelled in parallel, first by identifying relevant process elements and then specifying the details of the processes. The model from the process line can be shared via model-to-model transformation to the argumentation line, facilitating construction and reuse of information. The authors in (Gallina & Provenzano, 2015) provide their intuition with regards to how this transformation can be applied, in lieu of a formal meta-model linking the two types of lines. Effectively, this intuition constitutes a generic argument pattern, under which the relationship between processes and their fundamental elements (tasks, work-products, etc.) support an overarching claim. The claim made argues that each of the process's tasks are planned with rigor appropriate for their defined software integrity levels, supported by the structure of the process. These levels include DALs (see Section 2 of Chapter 3) and prescribe discrete levels of safety stringency with regards to the development processes and safety assessment activities followed.

8.1.1.3. Verification-Oriented Approaches

(Basir et al., 2010), develop a method for constructing a safety case (fragment) from code accompanied by formal verification information. The code, paired with safety requirements and analysis is analysed via Fault Tree Analysis (FTA) (see Section 6.1 of Chapter 2) to explore potential errors in its specification or the code itself. The code's behaviour is analysed by examining execution paths and coverage, to establish whether the claims and assumptions of the specification hold.

For each of the safety properties the code needs to satisfy, an argument pattern is used to argue satisfaction of the property through Hoare-style partial correctness of the code. Correctness is established by expressing the overlying safety policy as a set of Hoare-style rules.

For example, {y <= 43} y = y+1 {y<= 44} is a Hoare rule, with the elements in brackets being conditions before and after the execution of a command.

Using an automated theorem prover, code can be traversed backwards and apply the rules to determine earlier preconditions necessary for safe execution. Thus, the preconditions become the claims upon which the argument of satisfaction of the safety property relies. Further arguments with regards to each variable's satisfaction of the precondition overall, per each instance of use in the code and per each execution path are made.

In (Denney & Pai, 2014), formal software verification methods, in conjunction with safety analysis information, enable the production of arguments from argument patterns. In this case, the formal logic language PROLOG (Clocksin & Mellish, 2003) is employed to define required safety properties, upon which the goals of the safety case are based. Safety analysis evidence collected over the course of development is also linked to support the safety case goals and validate or verify earlier assessment. Development of the safety case flows from defining high-level hazards and relevant safety requirements to progressively lower-level hazards. Mitigation measures are incorporated to address hazards as they are identified. The approach departs slightly from the standard construction approach for GSN as in (Kelly, 1998:80–85). The higher-levels of the safety case are manually constructed in a top-down fashion and then linked with lower-level safety arguments automatically-generated from the verification information.

8.1.1.4. Generating Modular Safety Cases for Software Product Lines

In (Oliveira et al., 2015) a method for automatically constructing safety arguments for Software Product Lines (SPLs) is presented. SPLs are software-intensive systems whose development is based on variation over a core set of features. The problem of managing SPLs is further aggravated when safety considerations are introduced, as safety assurance must be provided across all system versions. The method introduced in the referenced thesis integrates SPL variability management, compositional MBSA and supports the automatic production and reuse of safety assessment, requirement and assurance artefacts.

HiP-HOPS and its extensions for automatic allocation of SILs are used to guide the generation of arguments via appropriate patterns. To manage SPLs, a significant methodological infrastructure is incorporated by this method. It involves an integrated functional, architectural and component failure metamodel which is linked with individual failure analysis tool metamodels to incorporate their functionality. Such tools include the OSATE AADL/Error Annex (The SEI AADL Team, 2005) and HiP-HOPS (Papadopoulos et al., 2011). The method also uses the Structured Assurance Case Metamodel (SACM) (OMG, 2016), which links the failure analysis metamodels and safety assurance metamodels. Through a 'weaving model' approach (Hawkins et al., 2015), the system models managed via the SPL variability tools are combined with the aforementioned SACM. The method then generates safety assessment and assurance artefacts, employable in relevant safety cases.

9. Safety Argument Generation Approach

Out of the previous work mentioned in Section 8 above, the most relevant to the objectives of this thesis is arguably found in Section 8.1.1.1 and Section 8.1.1.4, the FLAR2SAF and HiP-HOPS/SPL approaches respectively. Both methods share the concept of linking safety assessment artefacts with an argument pattern and using the latter as a guide for generating concrete safety cases. This idea will also be exploited in the method proposed in Chapter 5. However, each of the above methods is focused around a specialized context, COTS and SPLs respectively. The method proposed in this thesis aims to provide a generic approach, addressing systems development top-down and without product line considerations.

10. Summary

In this chapter, a review of the methodology most relevant to the objectives of this thesis was presented. The subject of optimization for SIL allocation was first explored, including a quick review of optimization techniques in general and earlier work specific to the research problem. Based on this investigation, Tabu Search was identified as the method of choice to pursue further for the thesis, as seen in Chapter 3.

Finally, the issue of safety argument generation was discussed and previous research into automatically generating safety arguments was reviewed. From this body of work, two methods close to the subject of the thesis were identified; the method proposed in Chapter 5 shares common points that will be discussed further in the latter chapter.

Chapter 4: Automatic Allocation of Development Assurance Levels

1. Introduction

In this chapter, a method for automatically allocating Development Assurance Levels (DALs) is presented. Given the importance of DALs in applying safety assessment within ARP4754-A, this method provides a model-based approach for addressing scalability of systems developed in that domain. Furthermore, it also provides a useful basis for supporting safety assurance via the extension described in Chapter 5.

The chapter is structured as follows; first, a more in-depth view of the rules and process governing DAL allocation per the standard. Next, the problem of allocation is framed as an optimization problem due to the associated development cost considerations. A number of optimization techniques which can address this problem are then presented. This information should supply the necessary background for presenting earlier work in addressing relevant problems. The method proposed is then explained in detail. Finally, an application and evaluation of the method on a small-scale abstract system concludes the chapter.

2. Automatic Allocation of DALs via Tabu Search

Based on the method described in Section 5.2.5 of Chapter 3, a Tabu Search extension to HiP-HOPS can be employed to allocate DALs. The difference in allocation rules between DALs and ASILs means a different implementation is required. However, the process up to the production of minimal cut sets via HiP-HOPS does not need to be altered. Therefore, the method introduced here will be described starting at the point where minimal cut sets for the system are known.

As explained in Section 5.5 of Chapter 2, to allocate DALs to lower elements of an architecture, the standard requires developers to identify which of the lower elements can contribute towards their system's failure. The methods of safety assessment presented in Sections 6.1, 6.5 and 6.7 of Chapter 2 (FTA, FMEA and Markov Analysis) are those recommended by the standard to perform this task. However, in the interest of flexibility, other methods are not explicitly restricted, assuming they are deemed acceptable by the certification authority.

The HiP-HOPS methodology, as shown in Section 7.2.3 of Chapter 2, is well-suited to identifying a system's FFS (referred to as 'minimal cut sets' outside the standard). The approach is compatible with the MBSA paradigm, as the safety analysis can be initiated and repeated automatically parallel to the system construction and modification over the course of development. In contrast, other approaches, such as constructing fault trees independently, require manual reconstruction of the fault trees with each iteration of design. Another benefit of using HiP-HOPS is related to the concept of Independence, explored further in Section 2.3.

2.1. Option Pruning Step

Consider the following scenario:

- FFS 1 with DAL of A: FF1
- FFS 2 with DAL of A: FF1, FF2
- FFS 3 with DAL of A: FF2, FF3
- FFS 4 with DAL of A: FF2, FF3, FF4

DAL	A	B	C	D	E
Cost	50	40	20	5	0

Table 4.1 - Cost function for Example

In this scenario, there is an interesting phenomenon occurring. Each of the sets contains at least one member from a previous set and one member from the next, apart from the first and last. Additionally, each set only contains one member not belonging to a previous set. The cost function itself seen in Table 4.1 is also interesting, as it is strictly increasing with regards to the DALs and non-linear.

Although there are multiple possible optimal solutions, finding one in particular in this case does not involve making any options, as seen in Table 4.2.

FF1	FF2	FF3	FF4
A	C	A	C

Table 4.2 - Optimal Allocation for Example

This solution, although not the only optimal solution, can be found using the following reasoning:

- FF1 was assigned A because it belongs to FFS1 and is the sole member, therefore inheriting its level.
- FF2 was assigned C because the other member of FFS2 is FF1 and has already been assigned level A, thus C is the lowest allowable level.
- FF3 was assigned A because the other member of FFS3 is FF2 and has already been assigned level C. Note that assigning FF2 and FF3 level B would result in a costlier allocation, due to the cost function, as C + A = 70 whereas B + B = 80. This is where the nature of the cost function chosen plays a particular role.
- Finally, FF4 was assigned C because another member of FFS4, FF3, has already been assigned level A.

Although this reasoning excludes the other possible optimal allocation, given in Table 4.3, it did not allow for any other options from the rules. Thus, the allocation was made without further investigation into other possibilities.

FF1	FF2	FF3	FF4
A	A	C	C

Table 4.3 – Alternative Optimal Allocation for Example

In practice, the options for allocation of function DALs to items of an architecture can be many, often too many to consider all exhaustively. An optimization algorithm that searches this space of potential allocations seeking a cost-optimal allocation would clearly benefit from a pre-processing step that reduces this search space, and this section demonstrates how such a reduction can be achieved.

Due to the high severity of aircraft hazards, the rules for DAL allocation are stricter than those found in other standards, allowing DAL reduction of only two levels at most. This allows, when a model and the cost function exhibit certain properties, to reduce the possible allocations significantly by removing inefficient options, in some cases even eliminating all options of allocation down to one. Even when there are still options remaining for optimization, the search space of the problem has been significantly reduced, thereby improving the effectiveness of any optimization technique employed afterwards.

These series of allocations can be applied when:

- the cost function of each element is non-linear and strictly increasing with respect to the DALs of its FFs (see below)
- there exist N FFSs for all of the architecture's effects that, when ordered in descending order of their effect's DAL, exhibit the following 'chain' property:

- Let FFS_i of size n be followed by FFS_{i+1}, FFS_{i+2} and so on. The chain property holds for these FFSs if:
    - there exists a common FF that belongs to both FFS_i and FFS_{i+1}, one that belongs to FFS_{i+1} and FFS_{i+2} and so on
    - $\left||FFS_{i+1}| - |FFS_{i+2}|\right| \leq 1$
- there exists a FFS amongst those satisfying the chain property with a single member

A non-linear, strictly increasing cost function is required in order for the algorithm to have only one option, the least expensive one, in cases where there are two variations of possible allocations. For example, let us consider the following scenario:

A FFS of level k can be decomposed to at least two members.

The possible combinations include:

- a variation of decomposition of one member by zero (one member is assigned k, the other(s) k-2)
- a variation of decomposition by one (two members are assigned k-1, the others k-2)

If the cost of the sum of two members having k and k-2 DALs is not equal to the cost of two members both having k-1 DALs then, provided the chain property applies, the algorithm can choose the least expensive of the two variations without another option. The rest of the members with k-2 are equal in cost between variations, so they don't affect the choice. In other words, the algorithm can always select optimally without option when either:

- $Cost(k) + Cost(k-2) < Cost(k-1) + Cost(k-1)$
- $\mathrm{Cost}(\mathrm{k}) + \mathrm{Cost}(\mathrm{k}-2) > Cost(k-1)$

Or equivalently when either:

- $Cost(k) - Cost(k-1) < Cost(k-1) - Cost(k-2)$
- $Cost(k) - Cost(k-1) > Cost(k-1) - Cost(k-2)$

In other words, when the cost function is non-linear.

The pseudo code for this reduction stage follows:

```
1) sizeCounter = 1
2) sort all FFSs in descending order of DAL
3) changesMade = true
4) while( changesMade )
    a) changesMade = false
    b) foreach FFS k
        i) if( sizeCounter = k.size ) then
            (1) if( there is just one Member in k unassigned ) then
                (a) assign it the lowest possible DAL
                (b) changesMade = true
            (2) end if
        ii) end if
    c) end foreach
    d) sizeCounter = sizeCounter + 1
5) end while
```

2.2. Tabu Search Algorithm

A basic version of Tabu Search has been implemented, as outlined in the previous section; each candidate is an allocation of DALs over all FFs and the best candidates are those with the lowest overall DAL cost. The Aspiration Criterion employed requires the candidate allocation to beat the Tenure's current best candidate in terms of overall DAL cost, thus being the best (i.e. cheapest) allocation in recent memory.

The search method used to generate the next set of candidates produces a new candidate for each allocation of the current one that can be changed and not violate DAL decomposition rules. This means that allocations assigned by the non-optional stage cannot be reduced in DAL under the level they were then assigned, only increased. An example of a single generation step is shown below.

- FFS 1: FF1
- FFS 2: FF2 AND FF3
- FFS 3: FF1 AND FF4

The current candidate allocation being examined (non-optional allocations annotated) can be seen in Table 4.4. For simplicity, DALs are assigned here to the FFs and not the elements they originate from.

FF1	FF2	FF3	FF4
B	E	C	A

Table 4.4 - Current Candidate Allocation

Next candidate allocations are seen in Table 4.5.

FF1	FF2	FF3	FF4
A	E	C	C
B	C	E	A
B	C	C	A
B	E	C	B

Table 4.5 - Next Candidate Allocations

All of the next candidates generated are allocations consistent with the decomposition rules. It should also be noted how the non-optional allocation on FF1 can still be altered due to its participation in the higher DAL FFS 3.

The pseudo-code for Tabu Search follows:

```
1) Generate a random allocation.
2) Set random allocation as the current choice.
3) Add the current choice to Tabu Tenure.
4) Repeat until iteration count or time limit are reached.
    a) Produce random alternative allocations from the current choice.
    b) Sort the produced allocations by DAL cost, ascending.
    c) Select the lowest cost allocation as the next choice.
```

    d) Repeat until a next choice has been selected or all alternative allocations have been examined.
        i) If the next choice is not Tabu, select it to be the next choice.
        ii) If it is Tabu but aspiring, select it to be the next choice.
        iii) Otherwise, examine the next produced choice.
    e) If none of the produced allocations is either non- Tabu or aspiring, set the lowest cost one as the next choice.
5) The next choice becomes the current choice.
6) Add the current choice to the Tabu Tenure.
7) Sort the Tabu Tenure by DAL cost, ascending.

Generating a random allocation in Step 1 involves selecting a random option from each Allocation Pack and then combining them with the non-optional allocations, as mentioned earlier. Sorting the generated allocations for the next iteration means that after each iteration the lowest cost - and ideally non-Tabu or aspiring - choice out of the produced candidates will have been made. To check if a candidate is Tabu, the fixed-size Tabu Tenure is parsed to see if there's an identical allocation on it. If so, the subject allocation is considered Tabu. An allocation is considered aspiring if its cost is lower than each member of the Tabu Tenure. Thanks to the sorting of the list in Step 4b, only a simple comparison with the first entry in the list is needed. To produce the next set of alternative allocations, the process described in the following pseudo-code is performed:

1) Add into list Optional all 'optional' partial allocations.
2) Add into list IncreasableNonOptional all 'non-optional' partial allocations of DAL lower than 4.
3) For every partial allocation in the IncreasableNonOptional list,
    a) Select a random Allocation Pack affecting that partial allocation.
    b) Select a random partial allocation from that Pack which assigns a higher DAL than the current one.

    c) If none exist, continue to the next partial allocation.
    d) Else, use that selection to generate a new allocation and add it to the resulting list.
    e) Repeat.

4) For every partial allocation in the Optional list,
    a) Select a random Allocation Pack affecting that partial allocation.
    b) Select a random partial allocation from that Pack which alters the current allocation.
    c) Use that selection to generate a new allocation and add it to the resulting list.
    d) Repeat.
5) Return the resulting list.

Listed in this way for reasons of simplicity, note that steps 3 and 4 of the above pseudo code are almost (except for sub-steps b and c) identical - simply applied to a different list. The purpose of the two lists is to find all partial allocations included in the allocation due to options taken from Allocation Packs which can be altered. Sub-step b illustrates the difference between the two lists. Whereas in Optional, any partial allocation which alters the current allocation can be selected, in the IncreasableNonOptional list only a partial allocation can be chosen, if any exist, which increases the DAL of the resulting allocation. As mentioned earlier, choosing such an option would not violate DAL allocation rules.

2.3. The issue of independence

In the standard, the concept of "Independence" is said to be 'a fundamental attribute to consider when assigning Development Assurance Levels' (SAE, 2010:41). The standard uses Independence as an attribute aiming to address the issue of common mode errors, which occur due to shared requirements amongst functions or development processes amongst items. Due to the apparent gravity that this concept garners in the standard (and the work mentioned above, amongst others) it is important to explain the views on this matter and how they support the methodology presented here.

The standard introduces two forms of Independence, Functional and Item. The former refers to the presence of common causes of failure between separate functions or systems of the architecture, while the latter between separate items. In both cases, identification of such common causes falls within the purview of the Aircraft/System Common Cause Analysis (CCA) process (see Section 5.5 of Chapter 2). The CCA process identifies such causes and includes them in the failure analyses performed at subsequent stages such as FTA.

Failures derived in this way are treated as any other failure by the HiP-HOPS methodology; if the failure analysis determines that they indeed contribute towards the failure of an item, system or function, they will affect the DAL allocation. In the model-based approach advocated in this work, common causes either explicitly propagate through a system model from a single source to many sink components (e.g. failure of a common power supply or data source), or they can be declared as implicitly causing simultaneous failure of more than one components (e.g. electromagnetic interference). Therefore, the method both captures dependencies and allocates DALs correctly while simultaneously addressing the issue of independence.

2.4. Region Shifting

Despite the presence of the Tenure, there are no guarantees that the algorithm will not become 'stuck' in a particular region of the search space for a given set of problem instances. To help alleviate this problem, a heuristic was added to the algorithm.

When a candidate is chosen, its composition is noted as a point of reference for later candidates. If the same candidate is repeatedly selected, up to a user-controlled number of times, the algorithm decides to 'shift' the space of search to a different, random region of the search space, using the same approach when generating the initial candidate, as mentioned above. The difference in this generation is that the elements comprising the new random candidate are chosen to be different than those of the point-of-reference-candidate.

This heuristic effectively restarts the search at a random point in the search space, hopefully away from the regions already explored. In the worst case, this feature contributes no benefit to the search, resulting in merely a different starting point. However, in the best case, it can lead to the exploration of a new region, with possibly better solutions, if the random relocation position happens to be appropriate.

3. Case Study on Abstract Wheel Braking System

3.1. Introduction to Wheel Braking System

The case study presented is based on an example aircraft Wheel Braking System (WBS) from ARP4761, adapted by (Sharvia & Papadopoulos, 2011b). The system is illustrated in Figure 4.1.

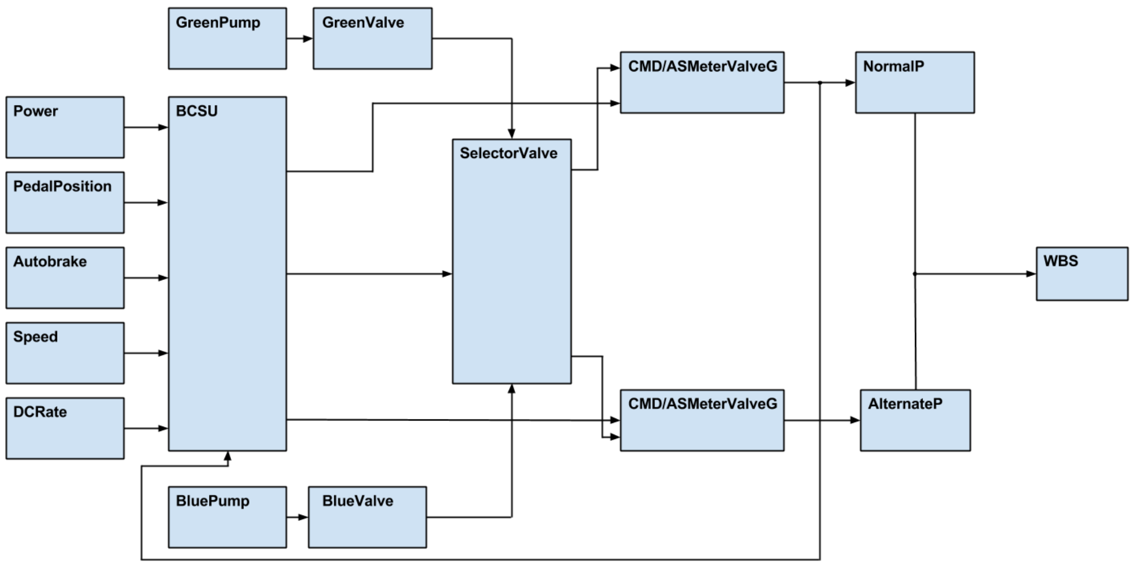


Figure 4.1 - Abstract Wheel Braking System

The purpose of the system is to provide safe braking during aircraft take-off and landing. It features two primary hydraulic pumps, GreenPump and BluePump. The Brake System Control Unit (BSCU) forwards input from the brake pedals to the brakes, monitors input systems and states for correctness and provides feedback to other systems. The SelectorValve receives a constant stream of pressure from both pumps, relaying the pressure from the appropriate one (see below) to the corresponding meter valve. The meter valves adjust the control valve, outputting the required amount of pressure based on BSCU's commands. The system features two modes of operation, Normal and Alternate. In Normal mode, GreenPump is used, sending pressure through the SelectorValve to ASMeterValveG (anti-skid meter valve). In Alternate mode, BluePump is used to send pressure through the SelectorValve to ASMeterValveB. Alternate mode is activated by BSCU when the pressure output falls under a certain threshold in Normal mode.

As mentioned, the case study is based on an example from the ARP4761, so it is well-known. While the scale of the WBS is relatively small (14 elements), it is large in terms of the potential space of feasible DAL allocations. A human designer would have to sift through potentially hundreds of thousands of possible allocations before identifying an optimal one. At the same time, given a DAL allocation for the WBS, the limited number of elements make it comparatively easy to confirm (or disprove) whether it is optimal. Thus, we can safely claim the WBS is appropriate for evaluating the proposed optimal DAL allocation method.

3.2. Applying Automatic DAL Allocation on WBS

If braking fails while take-off or landing, consequences could be catastrophic. The plane could fail to decelerate as expected or brake as it is about to take off, potentially causing the landing or take-off to fail, causing a crash. This reasoning was found to be sufficient to test the allocation by assigning the overall DAL of the system output (WBS) to be A. It should be noted that this assignment is primarily used as an example and in practice the actual DAL assignment could be lower. However, this would not impact the allocation process described; a lower allocation can only potentially lead to a smaller number of potential allocations that need to be evaluated. Therefore, in this sense, the worst-case scenario is selected based on maximizing number of potential candidates that need to be evaluated. For the purposes of the case study, the costs in Table 4.6 were used to approximate the cost each DAL would have on its component.

DAL	A	B	C	D	E
Cost	50	40	20	10	0

Table 4.6 - WBS Cost Function

Based on the FTA, the architecture's FFS are as follows:

- BSCU,
- Power,
- SelectorValve,
- AutoBrake AND PedalPosition,
- PedalPosition AND Speed,
- PedalPosition AND DCRate,
- BluePump AND CMD/ASMeterValveG,
- BluePump AND GreenPump,
- BluePump AND GreenValve,
- BlueValve AND CMD/ASMeterValveG,
- BlueValve AND GreenPump,
- BlueValve AND GreenValve,
- CMD/ASMeterValveB AND CMD/ASMeterValveG,
- CMD/ASMeterValveB AND GreenPump,
- CMD/ASMeterValveB AND GreenValve

3.3. Results

The resulting allocation when the parent FC (WBS) assigned a DAL of A is shown in Figure 4.2. This is one of numerous optimal allocations found overall.

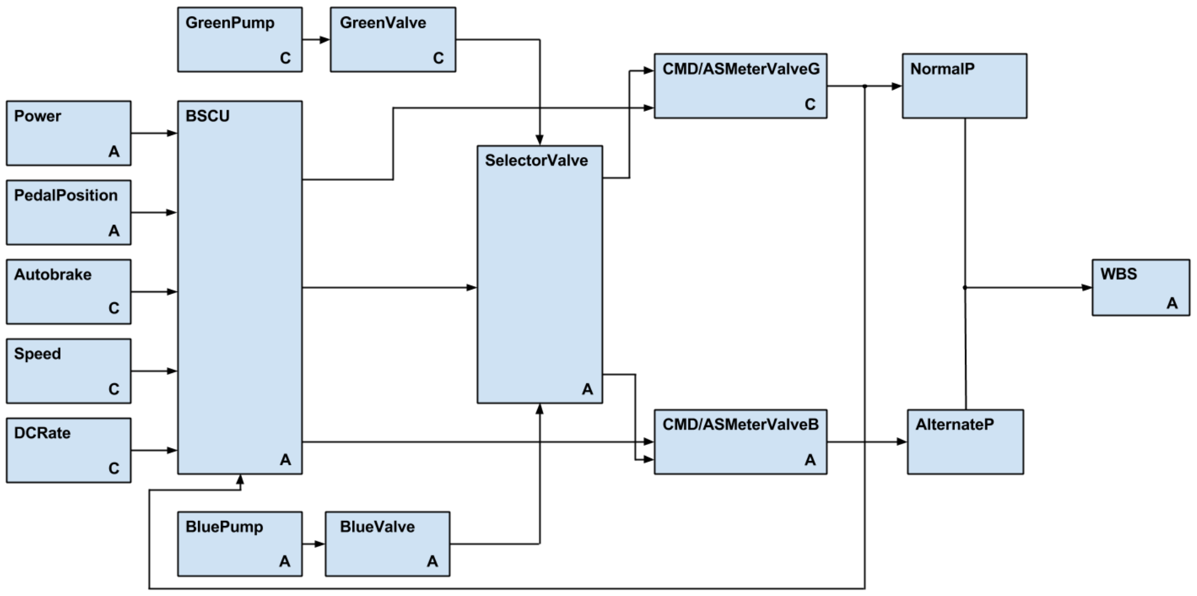


Figure 4.2 - WBS with DALs Allocated

The allocation algorithm was executed 100 times, with 1000 iterations per execution. Each time the optimization produced either the same or a permutation of the allocation shown above with the same overall DAL cost. This allocation was found to be optimal after exhaustively searching all possible combinations of DAL allocations for this model. It should be noted that this was only possible due to the relative small scale of the search space of 531.441 possible allocations. These include only allocations which would be compliant with the ARP4754-A DAL allocation rules. Larger-scale models might require days, months or even longer periods of time to exhaustively search for their optimal solutions.

The time required to execute the algorithm can be seen in Figure 4.3, whereas in Figure 4.4 the time required to find the optimal solution (before the search completion) is shown. The optimal solution was confirmed by exhaustively searching through the hundreds of thousands of options. Where the metaheuristic approach barely required a few seconds at most to complete each run, the exhaustive search required roughly 15 minutes to complete. The average run-time to completion of the 100 trials in Figure 4.3 is 0.85 seconds, whereas the average time to finding an optimal solution in Figure 4.4 is 7.25 milliseconds.

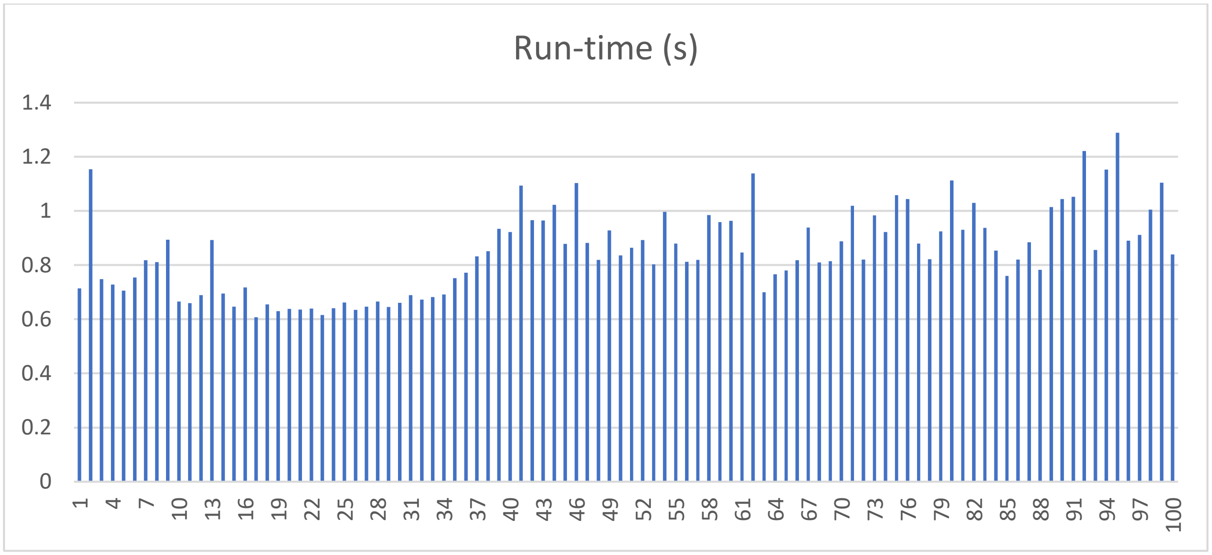


Figure 4.3 - Run-time across 100 trials

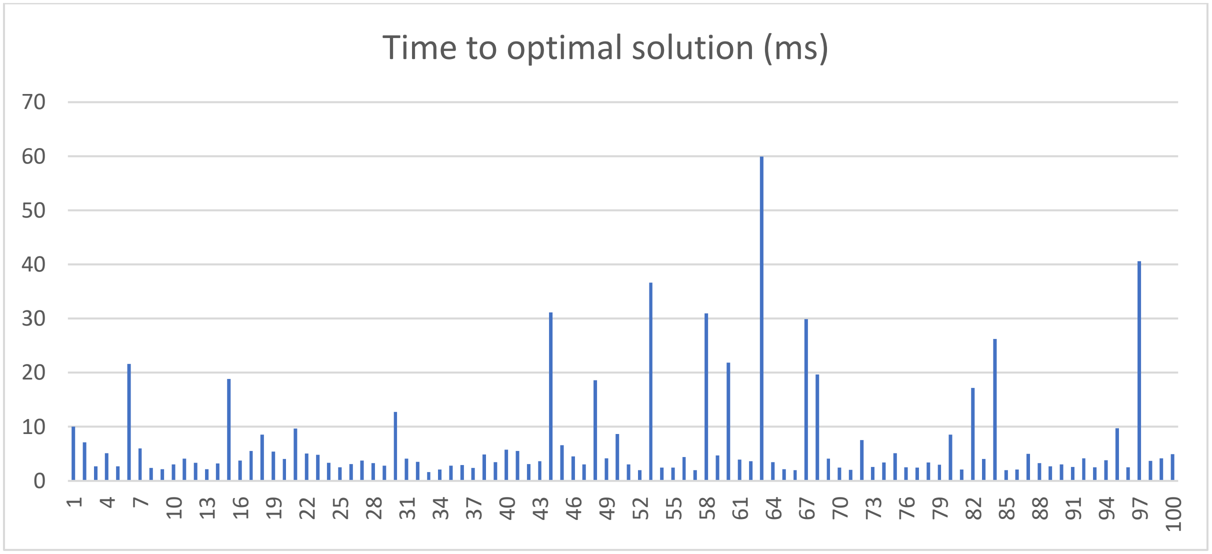


Figure 4.4 - Time to Optimal Solution across 100 trials

While the exhaustive search has proven sufficient to confirm the results, an explanation of the rationale behind the resulting allocation should further support its correctness. First, given the structure of the cost model used, it should be noted that the sum of the cost of a DAL A and DAL C allocation, which is 70, is lower than that of two DAL B allocations, which is 80. This observation explains the absence of level B DALs in the resulting allocation, as due to the two modes used for fault tolerance, the FFSs produced contain up to two members only, with a choice between choosing variations of A and C assignments or B and B assignments.

Next, in the case of the single-member FFS (first three in the list), these Members can only be assigned the DAL of the parent FC i.e. DAL A. Since they do not participate in other FFS, these Members can be assigned immediately in the non-optional allocation stage and not be considered further.

Moving on, it is significant that the rest of the FFS have common Members between them: there are 4 groups containing PedalPosition, BluePump, BlueValve and CMD/ASMeterValveB. Except for the PedalPosition group, the other 3 groups also share their counterpart Members as well. Therefore, an optimal allocation would revolve around assigning the highest of available DALs, which in this example is A, to as few of the common Members possible and the lowest, in this case C, to as many as possible. For instance, in the group of PedalPosition, it only makes sense to assign it DAL A and the rest of the Members C since they do not affect any other Members. For the other groups, finding the correct permutation manually is harder due to the interconnectivity and the answer given in the example's result is one of many possible.

Finally, the resulting allocation also illustrates the effect of the fault tolerant dual-mode architecture employed: while components common to both modes (e.g. BSCU, SelectorValve etc.) have been assigned the DAL of the parent FC, which is DAL A, components in one mode (GreenValve, GreenPump, CMD/ASMeterValveG) have been chosen to have a reduced DAL, i.e. DAL C.

3.4. Comparison to Automatic ASIL Allocation

The ASIL allocation approach from (Azevedo, 2015) using SDMA Tabu Search was also executed on the WBS system for comparison. There are 5 ASILs which range from QM, then A to D, from lowest to highest integrity respectively. One of the optimal allocations found can be seen in Figure 4.5. The cost function used for the DAL allocation was used for this case as well, seen in Table 4.6. The alternative rules governing ASILs requires the sum of the members of each FFS to equal the hazard's ASIL. This is reflected in the result if one notes the split in ASILs across the two hydraulic systems e.g. GreenValve and BlueValve. The reason that an equal split of ASILs was chosen instead of alternative options can be found in the cost function. Based on the cost function, one can see that the option chosen results in the lower sum compared to other options. The same logic applies with regards to assigning PedalPosition the highest ASIL and the rest of the associated components the lowest.

It should be noted that confirming the optimal for ASILs is much harder than for DALs. Three days of continuous exhaustive search were required to identify all of the solutions and confirm the minimum ASIL allocation cost. This should be indicative of the less constrained ASIL allocation rules, compared to the stricter DAL rules.

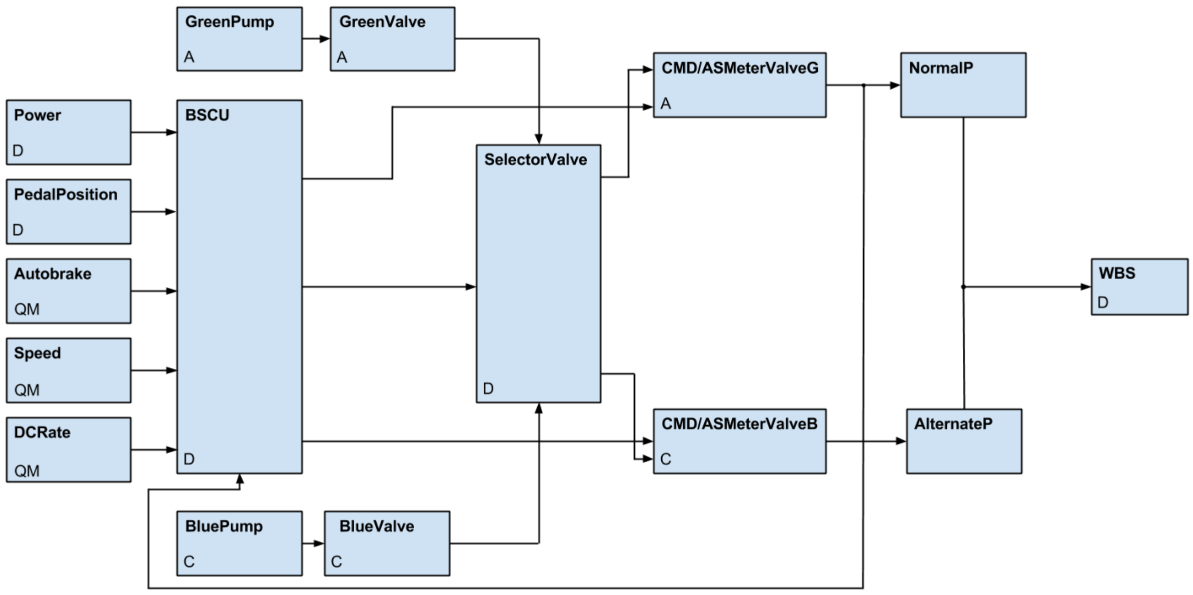


Figure 4.5 - ASIL Allocation on WBS

4. Evaluation

The primary motivation for investigating optimal DAL allocation was to determine whether earlier work on optimal ASIL allocation using metaheuristics is also applicable to DALs. Therefore, establishing whether the metaheuristic approach is **feasible** for DALs is the primary criterion of evaluation. Effectively, this involves ensuring that the solutions found via the method, vis-à-vis the DAL allocations, are correct. In other words, the resulting DAL allocations should be confirmed to meet the ARP4754-A's guidelines.

With regards to the results presented in the case study in Figure 4.2, the produced DAL allocations can be confirmed to be standard-compliant directly. Based on the WBS FFS analysis, the 3 single-member FFS have their members assigned the hazard DAL i.e. DAL A. The remaining FFS have DALs assigned to their members based on the 1st of the two options from Section 2 earlier in Chapter 3.

Given the small number of FFS for the instance, it is relatively easy to confirm they all comply with the standard rules. In the general case, where multiple FFS of potentially hundreds or more allocations might be involved, a scalable way of checking they are standard-compliant would be needed. There are two options for addressing this issue, applicable to the method discussed here. The first option is to confirm that the options for allocation generated for each FFS before optimization are all standard-compliant and that the algorithm does not produce allocations that violate the standard. With regards to the former, this involves testing the implementation to ensure only standard-compliant options are generated. With regards to the latter, we need to evaluate the new option generation step within the algorithm presented in Section 2.2 earlier in this chapter. Once again, the algorithm only chooses from standard-compliant options generated before the optimization stage. Thus, provided the implementation is correct, the algorithm should never produce allocations that are not compliant.

The second option is to adopt a more general approach by automatically evaluating whether the allocations found are compliant. In other optimization techniques, the optimization search could deviate to evaluate allocations which are not compliant in hope of finding a quicker route to undiscovered optimal and compliant allocations through them. In such cases, it should be important to ensure that the final solution presented is confirmed to be standard-compliant. This can be done by iterating through each FFS and confirms that its members have been allocated according to one of the two compliant options.

The second aspect we require of the resulting allocation is **optimality**. Given the nature of metaheuristic techniques, due to the inherent bounds on execution they feature, guarantees of solution optimality are typically absent. Instead, their performance can be evaluated statistically. To evaluate how the method performs in this sense, the frequency with which it identifies optimal DAL allocations is needed. In cases where none of the allocations found happen to be optimal, the proximity to an optimal solution can instead serve as a substitute. The proximity can be measured by the difference in the expected development cost impact between a given allocation and the cost of the optimal allocation. If the optimal is unknown for a particular problem instance, the best solution found for it can serve as the target instead.

Besides the case study shown earlier, 3 other FFSs from other systems were evaluated. The number and the size of the FFSs involved are summarized in Table 4.7. HBSOne and HBSBig are Hybrid Braking Systems where two system functions in each case being allocated DALs. This means that for each of these systems, two sets of FFSs are found, their individual sizes listed separately. The FCS is the system used as a case study in Chapter 6; it is referenced here with regards to the method's performance in allocating DALs to it. The collection of FFSs across these systems should provide an adequate variation in both scale – as the number of FFS increases – and complexity – as the members per each FFS increase.

Name	Total FFS	FFS size 1	FFS size 2	FFS size 3	FFS size 4	FFS size 5
WBS	15	3	12	-	-	-
HBSOne	31	1+10	18+2	-	-	-
HBSBig	94	28+1	2+63	-	-	-
FCS	100	11	50	24	10	5

Table 4.7 - Summary of Systems FFS

The scale in growth is depicted in Figure 4.6. All the examined systems feature some FFS of size 1, while their majority is of size 2. Only the FCS case study, seen in detail in Chapter 6, features more complex FFS of larger sizes.

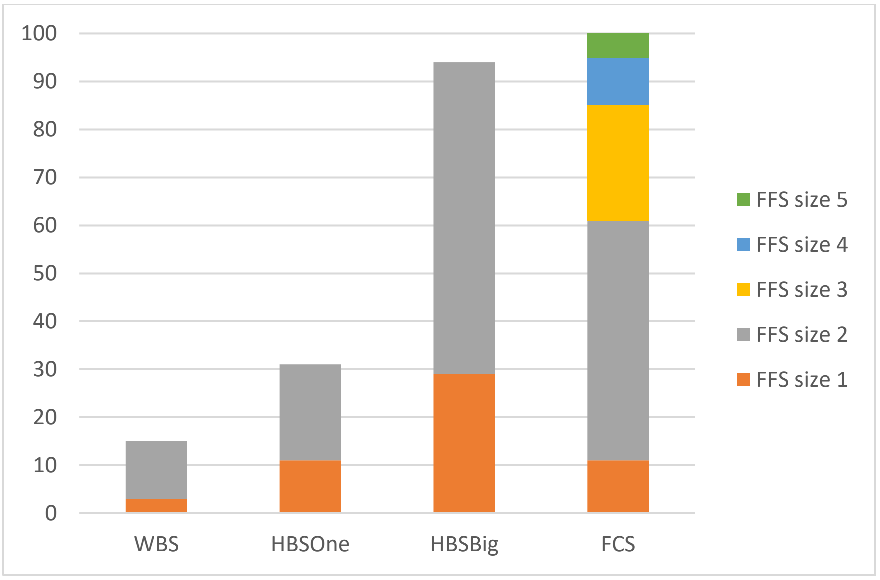


Figure 4.6 - FFS Distribution per System

The results across 1000 runs for each system are summarized in Table 4.8.

Name	Best Cost Found	Average Cost Found	Average % of Time Best Cost Found	Standard Deviation
WBS	470	470	100 %	0
HBSOne	570	570	100 %	0
HBSBig	1380	1381.3	93.7 %	5.05
FCS	1470	1513.1	0.2%	12.8

Table 4.8 - Summary of Results

The results from Table 4.8 are shown in Figure 4.7. For the two systems, there is no deviation between the average found solution and the best solution across the 1000 trials. In the case of the HBSBig and FCS however, the best solutions (across all trials) are not always identified, as indicated by the 3rd column of the table and the deviation seen in both the table and graph. The significant increase in problem size causes only minor deviation in the quality of the solutions found but the best solution is very rarely found in the case of the FCS.

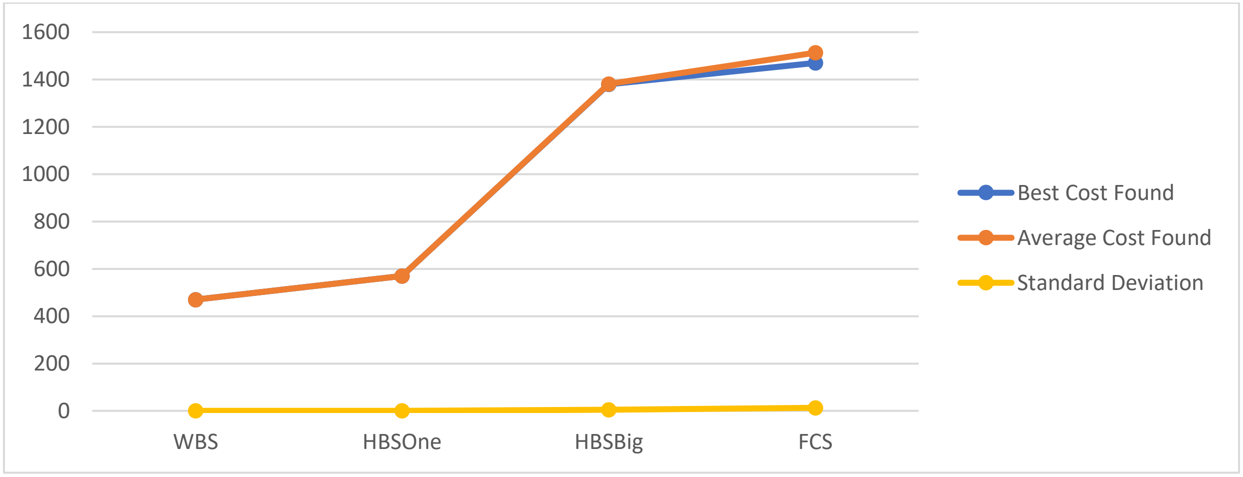


Figure 4.7 - Results across Systems

A more practical view of the method's performance includes the evaluation of its **execution time**. This criterion is particularly relevant when considering the choice between bounded and unbounded optimization techniques, such as metaheuristics versus linear programming. For relatively small problem instances, execution time is likely to remain low for either category for most practical purposes. However, as the size of the problem instance grows, the time required to identify an optimal allocation might quickly deteriorate. In the worst case, this can result in cases where the benefit of knowing the optimal does not justify the time spent searching for it. After all, this time would also contribute to the increase of development cost, which is contrary to the minimization objective. Three quantitative measures are employed to evaluate time. The first is the average time a set of optimization search iterations is required to execute. This measure provides a view of how fast the algorithm performs against problem instances of a specific scale. The second measure is the average time required for a search to locate the optimal or best-found allocation overall, for the particular problem instance. This measure is significant for appreciating how rapidly the search finds the better solutions. This evaluation measure can also provide an estimate on whether less or more iterations are adequate or necessary for finding optima, informing future parameterization choices. The final measure is the average number of iterations required to identify an optimal or best solution. This is similar to the previous measure; however, it is less dependent on software implementation or hardware configuration of the underlying computation system.

The final criterion with which the allocation method is evaluated is **scalability**; the ability to address a problem with comparable efficiency as the problem instances grow in size and complexity. This definition is adapted from the one found in (Bondi, 2000:1) with regards to systems. Given that the execution time for even the largest and most complicated model remained within seconds is arguably an indicator of successfully meeting the criterion.

In Table 4.9 the results of the quantitative performance measures for each problem set are summarized.

Name	Average Time (s)	Average Time to Best (ms)	Average Iteration Count to Best
WBS	0.08	0.65874	8.911
HBSOne	0.168	3.40622	21.385
HBSBig	0.904	285.053	313.745
FCS	4.449	1978.49	450.888

Table 4.9 - Summary of Performance

To better appreciate the results, Figure 4.8 and Figure 4.9 are provided. The measurements in the Figure 4.8 are both expressed in milliseconds for visual consistency. In Figure 4.8, we note the significant jump in the time required to complete the entire set of trials as the problem scale increases. The time required to find the best solution is significantly lower, so there is potential for large gains in execution time if the trial count was reduced.

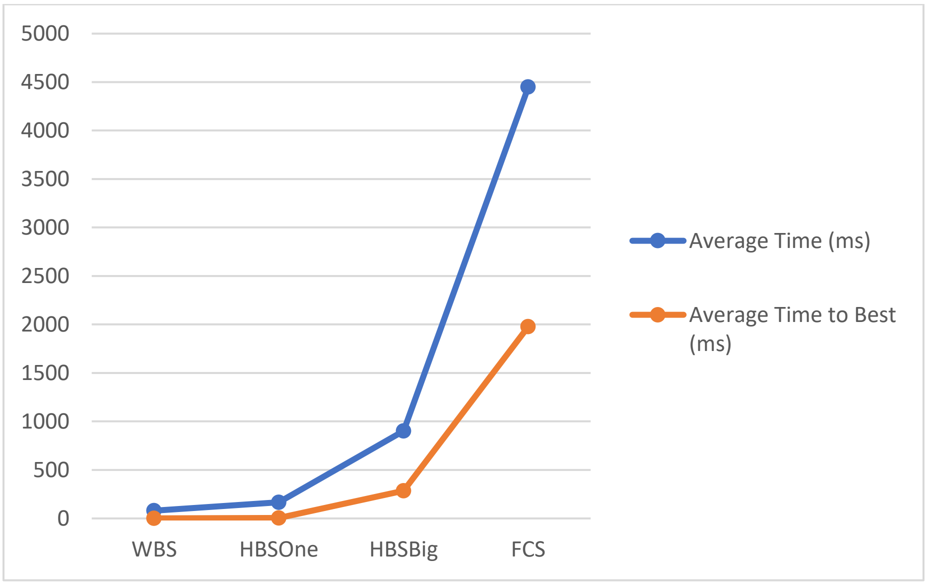


Figure 4.8 - Average Time per System

In Figure 4.9, a similar view is seen, the difference being the larger increase in iteration count to find the best solution across all trials from HBSOne to HBSBig to FCS compared to the corresponding increase in average runtime to find the best solution across all trials. This slope difference can be attributed to the differences in the problem instances i.e. the FFS composition for each system. While having a lower growth in average iteration count should be expected to lead to a similar lower growth in average execution time, increased problem instances can incur additional execution overhead responsible for further delays. Thus, the larger scale and complexity of the FCS system incurs an additional overhead on the execution time required to find the best solution.

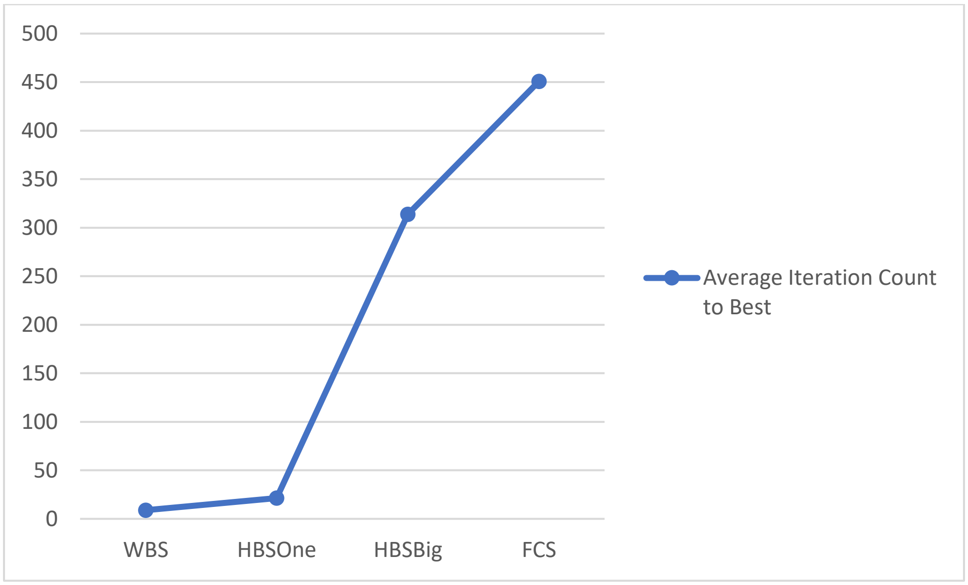


Figure 4.9 - Average Iteration Count per System

5. Summary

In this chapter, a method for automatically allocating DALs across a system's architecture has been presented. The problem of allocating DALs was introduced in-depth and reframed as a cost optimization problem. Earlier work towards optimizing such problems was then presented. Based on this review, Tabu Search was identified at the time as the most promising approach and was chosen to solve the problem. The method developed offered an effective approach for allocating DALs, demonstrated via a case study on a small-scale system.

The method is an important contribution towards addressing DAL allocation on its own given its novel approach for this domain. However, its contribution to the thesis will continue in the following chapters as well, supporting safety argument construction.

Chapter 5: Automatic Generation of Safety Arguments

1. Introduction

In this chapter, we present a method for automatically generating safety argument structures for ARP4754-A. The method uses the previously presented method for automatically allocating DALs, shown in Chapter 4. The chapter begins with a more in-depth analysis of the challenges we aim to address in safety argument construction and maintenance. Next, previously established methods for automatically generating safety arguments are presented. This should help establish what the state-of-the-art research has achieved towards addressing this problem. At this point, with the necessary background for understanding our method established, we move on with an overview of the method itself. Following this, the individual steps of the method are discussed in turn. Finally, a small example of an abstract system has the method applied to it to generate an argument structure. This should assist in clarifying the operation of both the method and the upcoming case study in Chapter 6.

2. Safety Argument Generation Process

At this point, the processes associated with certification for civil aircraft in ARP4754-A have been presented. Given the essential role of assurance in certification, a method which addresses the identified shortcomings is needed. The objective of this method would be to generate an argument structure which supports system safety certification under ARP4754-A compliance. The method should operate with minimal input from the user. Essentially, this process should aim to encapsulate the design criteria, usually applied manually, into an automatically-instantiated argument pattern. The effect of this approach should, ideally, shift the effort from developing all argument structures to mainly developing the pattern. By focusing on the pattern, the cost of constructing and maintaining the safety case should be severely reduced. We will now present an in-depth view of the process we propose.

2.1. Safety Argument Generation Overview

The proposed process is summarized in Figure 5.1 as 7 stages. Each of the stages (depicted as a rectangle) provides an output (depicted as an oval) to the following stage, with the last stage producing the final safety argument structure. The 'Pre-constructed Safety Argument Pattern' is an additional input to the last stage, combined with the output from the previous stage.

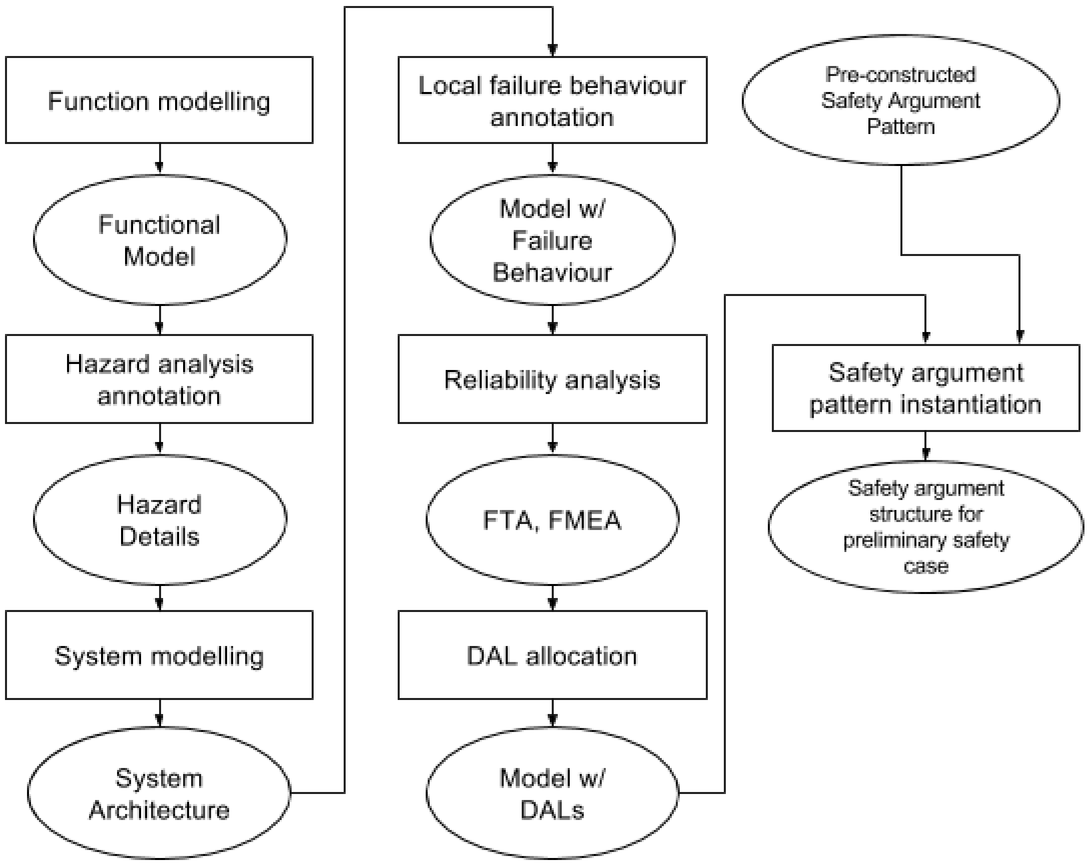

Function modelling
Functional Model
Hazard analysis annotation
Hazard Details
System modelling
System Architecture
Local failure behaviour annotation
Model w/ Failure Behaviour
Reliability analysis
FTA, FMEA
DAL allocation
Model w/ DALs
Pre-constructed Safety Argument Pattern
Safety argument pattern instantiation
Safety argument structure for preliminary safety case

Figure 5.1 - Safety Argument Generation Process

We initiate the process by modelling our system's functions using the Matlab/SIMULINK modelling tool. As described in Section 5.5 of Chapter 2, ARP4754-A advocates performing FHA to identify hazards and classify them. We annotate those hazards for each function using the HiP-HOPS plugin for the modelling tool. We continue the design by moving onto defining the system architecture implementing the functions. Additional hazard analysis for the system can be appended as well. While designing the architecture, we also use the plugin to annotate each (sub-)system with local failure behaviour information, such as what are the possible causes of failure and how do failures propagate. This information is then used by HiP-HOPS, to synthesize and analyse fault trees, as described in (Papadopoulos et al., 2011) . Once the failure analysis is complete, DALs are (near-) optimally assigned to the system, given a user-defined cost mapping per each DAL (see Section 3 of Chapter 3). Finally, the information from both the model and the analysis is used to instantiate a safety argument pattern.

2.2. The Library

The information gathered during the stages preceding the pattern instantiation is compiled into what we refer to as the Library. The Library is a versatile storage utility through which the pattern generation process can access functional, architectural, failure analysis and other model data. We will describe how the information aggregation can be performed in the following sections in more detail. During the pattern instantiation stage, the Library provides access to the heterogenous information through the use of specialised accessor syntax declared within the pattern. For example, the statement 'library: Element Name' instructs the pattern to retrieve from within the Library an element named 'Element Name'. Additional type information can be provided to specify what kind of element is being sought. This feature can also serve as a validation mechanism, ensuring that the correct types of information are used throughout the process. By combining the entity and structural pattern abstraction with the Library semantics, it is possible to embed model-based rules with which to guide the argument structure. In Figure 5.2 we can see the Library represented in Unified Modelling Language (UML) (Rumbaugh et al., 1999). 'Alias' signifies the name the argument pattern will use to refer to the element, 'Type' is used for type-consistency checking and 'Value' is a context-dependent value.

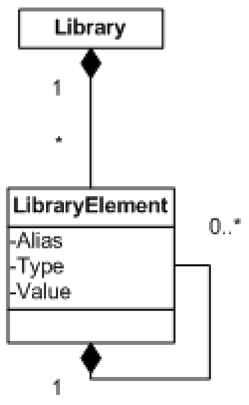


Figure 5.2 - Library UML Description

As elements of the model and the pattern are read in from each stage of the process, new entries are appended. The design of the Library is based on the Composite software design pattern (Gamma et al., 1994:183). Through this structure, the Library can include elements of the model which can act as containers of other elements. For example, this is useful when defining sub-functions of functions or sub-systems of systems. In both of these cases, the containment of one element by another can be viewed as both elements being part of a 'supported_by' relationship. In other words, a function is 'supported_by' its sub-function and so on. Similar logic can be applied with regards to an element's properties; they can be seen to be related through a 'property_of' relationship. For particular elements, which require attributes whose properties might include dynamic elements or operations, sub-classing from the LibraryElement allows them to be incorporated into the Library homogeneously.

2.3. Function Modelling

During this initial stage, the system's high-level functionality – its purpose – is decided upon. For instance, typical functionality for an aircraft would include 'Flight', 'Navigation', 'Landing' and so on. Each of these functions could be further refined upon investigation, leading to more specialized sub-functions. A sub-function can specialise based on the context under which the function is used or the means through which it is delivered. For example, a 'Braking' function could be further specialised into

‘Wheel Braking’ and ‘Speed Braking’; braking using the aircraft’s wheel brakes on the ground and wing spoilers in both the air and ground respectively. Additionally, aircraft operation demands the identification of the flight phases during which the function is applicable. For instance, ‘Wheel braking’ would be applicable during the ‘Taxi’, ‘Take-Off’ and ‘Landing’ phases but not the ‘Flight’ phase. As mentioned in Chapter 3, ARP4754-A views functions as a parent in the system’s architectural hierarchy, with sub-functions and systems providing support.

Functional requirements are also decided upon for each of the identified functions. These can vary greatly depending on the aircraft in question. For example, smaller aircraft are likely to support much fewer passengers than most standard commuter aircraft. This would also likely entail lower limits on maximum weight supported by the aircraft.

There are numerous software tool options for designing system functions and underlying architecture, both commercial and freely available. We prototyped our method based on the widely-employed MATLAB software platform (Higham & Higham, 2005) with the SIMULINK extension library (Dabney & Harman, 2004). As described in Chapter 4, this combination of tools supports HiP-HOPS integration. This allows us to annotate failure behaviour information directly on the system model during design. XML (see below) output from HiP-HOPS is parsed by the extension prototype tool, written in C++ (Stroustrup, 2014). An alternative approach would be to make use of the SimulationX software platform (ESI, 2017), which also supports HiP-HOPS integration.

Naturally, developers could replicate the HiP-HOPS methodology and integrate it with modelling tools of their choice as well; the standard does not restrict the use of tools. However, the adoption of a particular approach should always be supported with appropriate safety arguments. Therefore, a robust safety case should include reasoning regarding the choice of development tools and whether they can impact a system's safety. Additionally, where needed, item independence should be established through the use of alternative software and hardware tools.

Functions can be incorporated as input to the process through the use of a common exchange format. We employ the widely-used Extensible Markup Language (XML) (Bray et al., 1997). This format allows for improved readability and manual modification for small models, at the cost of being verbose and costing more memory space, if no compression technique is applied. The format allows the transfer of information from MATLAB to HiP-HOPS and then to our prototype tool. Naturally, other file formats that are dependent on the choice of design tools can be employed. In this case, developers should ensure compatibility of the exchange formats supported by their tools or produce file conversion tools.

Function data is parsed into the Library by incorporating the following features:

- Function name
- Function description
- List of sub-functions (if any); these are simply references to functions that will be defined later
- List of sub-systems (if any); again, references to systems defined later

The Library would create an entry of type 'Function' for a basic function, with its name as its value and its alias. The sub-functions or sub-systems would then be included as child references when they are parsed. Other data regarding a function (at the design level) can be similarly incorporated. For example, a particular functional requirement can be annotated to be used later by the safety case.

2.4. Hazard Analysis Annotation

Functional Hazard Analysis (FHA), mentioned earlier in Section 5.5 of Chapter 2, is performed. While there are several ways of performing this analysis, there are no known automatic means of doing so. In cases where the systems being evaluated are already known and the context is equivalent, previous results of FHA can be incorporated. Unfortunately, this is not always an option in the general case, where novel or altered systems/context might be involved. Therefore, the only provision of the method towards this end is allowing the user to annotate the function model with FHA information.

To annotate a function with the information produced during the FHA, the following documentation features are included in the data structures, based on ARP4761's advice (SAE, 1996:31–39).

- FHA name; allows tracing an analysis entry for a function to the FHA responsible for it
- Subject function; the name of the function being analysed
- Per each hazard identified as associated with the function
    - Name; a succinct but descriptive name for easily distinguishing the hazard
    - Description; short description of how the hazard could come to pass
    - Effect; description of what the impact of the hazard is
    - Phase; flight phase during which the hazard can occur
    - Classification; a severity classification of the hazard, as per the guidelines seen in Section 2 of Chapter 3
    - DAL; the assigned DAL given the hazard's classification
    - Associated lower-level requirements; these are requirements that lower architectural levels should satisfy as a result of the hazard's implications
    - Supporting material; where applicable, information justifying the choice of classification
    - Verification method; the method chosen to verify the requirements for addressing the hazard

The methods with which the developer should identify each of the above features is, once more, left purposefully open by the guidelines.

2.5. System Modelling

The system modelling stage, involves identifying and designing the architecture supporting the high-level functionality. As per the function modelling stage, we make use of the MATLAB/Simulink toolset for system design. This has the benefit of integrating the functional and system views, which can help clarify the hierarchical links between them. It makes sense to use common modelling tools across functions and systems. To begin with, the graphical representation remains consistent and the relationship between functions and systems is readily available. Additionally, in most cases, functions can be represented as abstract systems anyway so there's no reason to use different tools. This stage outputs a list of systems, the architecture they are configured under and requirements from the earlier stages assigned to each.

2.6. Local Failure Behaviour Annotation

At this stage, the user manually annotates elements of the system with failure behaviour logic and relevant data necessary to the use of HiP-HOPS. In Section 3 of Chapter 4, we have demonstrated how this annotation can be applied on an abstract Wheel Braking System from ARP4761. In summary, the output of each element of the architecture is annotated. For functions and systems, this consists of logically establishing which combinations of component or subsystem failures can trigger a system failure. For components, this consists of describing which types of failure can be propagated forward from their output. This information can include likelihood of occurrence of failure when such information is available. This enables quantitative probabilistic analysis during the automatic FTA through HiP-HOPS.

2.7. Reliability Analysis

During this stage, HiP-HOPS receives the architectural and failure behaviour information from the earlier stages to automatically perform FTA for the model. Much of this stage has already been explained in

Section 7.2.3 of Chapter 2. To briefly recapitulate, ‘mini’ fault trees for each component are created, then merged together into an overall fault tree. This ‘global’ fault tree is then simplified and Minimal Cut Sets (MCS) from it are generated. MCS are essential for the next stage as they allow direct identification of the lower-level failures which are necessary and sufficient to trigger system and function failures.

An additional benefit of employing HiP-HOPS for this stage is that it automatically provides a Failure Modes and Effects Analysis (FMEA) report as well. This report can also be incorporated into the safety case as safety evidence. As noted in Section 5.5 of Chapter 2, the ARP4754-A advises the use of FMEA and FMEA summaries (FMESs). Specifically, an FMES can be employed to more easily verify whether component and system failure modes have been addressed to meet their DAL and other safety requirements. FMESs are also recommended to be provided as part of the documentation for the certification process. Thus, HiP-HOPS’ automatically generated FMEAs can be aggregated as FMES and incorporated into corresponding safety arguments during the verification stage of development.

2.8. DAL Allocation

During this stage, the process described in Chapter 4 is applied. The resulting allocation maps a DAL to each of the architectural elements defined in the earlier stages. Given that function DALs are identified/assigned via FHA, this effectively includes only system and component DALs. In the earlier chapter, a mechanism for justifying the choices of DALs the optimisation algorithm makes was omitted. The mechanism was not relevant at that point and would have introduced unnecessary complexity. As it is relevant for contributing towards the argument generation, this mechanism will now be presented.

To briefly recap, the algorithm shown in Chapter 4 allocates DALs by iteratively searching through the pool of potential assignments generated as options from the MCS for the cheapest overall combination. During the generation of options from the MCS, it is possible to append justification for each option regarding the correctness and motivation behind its potential choice. For instance, let us consider a system with its MCSs seen in Table 5.1.

MCS	O-Comp1 AND O-Comp2	O-Comp2 AND O-Comp4	O-Comp3 AND O-Comp4

Table 5.1 - Example MCS

The potential valid options for allocation can be seen in Table 5.2. We have omitted options which are valid but obviously inferior, such as all components receiving DAL A.

DAL Allocation	Comp1	Comp2	Comp3	Comp4
#1	B	B	B	B
#2	B	B	C	A
#3	A	C	C	A
#4	C	A	B	B
#5	C	A	C	A
#6	C	A	A	C

Table 5.2 - Viable options for DALs on Example MCS

Regardless of the cost function used, each of the valid options can have a justification generated. For example, should DAL Allocation #5 from Table 5.2 be chosen, the reasoning per each component can be seen in Table 5.3.

Component	Reasoning
Comp1	Assigned DAL A through full inheritance from System DAL of A and being member of a 2-member MCS with: Comp2
Comp2	Assigned DAL C through decomposition by 2 levels from System DAL of A and being member of a 2-member MCS with: Comp1 and a 2-member MCS with: Comp4
Comp3	Assigned DAL C through decomposition by 2 levels from System DAL of A and being member of a 2-member MCS with: Comp4
Comp4	Assigned DAL A through full inheritance from System DAL of A and being member of a 2-member MCS with: Comp3 and a 2-member MCS with: Comp2

Table 5.3 - Generated justification for DAL Allocation

These justifications are quite verbose and could be trimmed down for the purpose of being included in a graphical safety argument structure. Alternatively, it could also be decomposed into a small argument structure, establishing a claim supporting the correctness of the allocation. An example of such a structure can be seen in Figure 5.3.

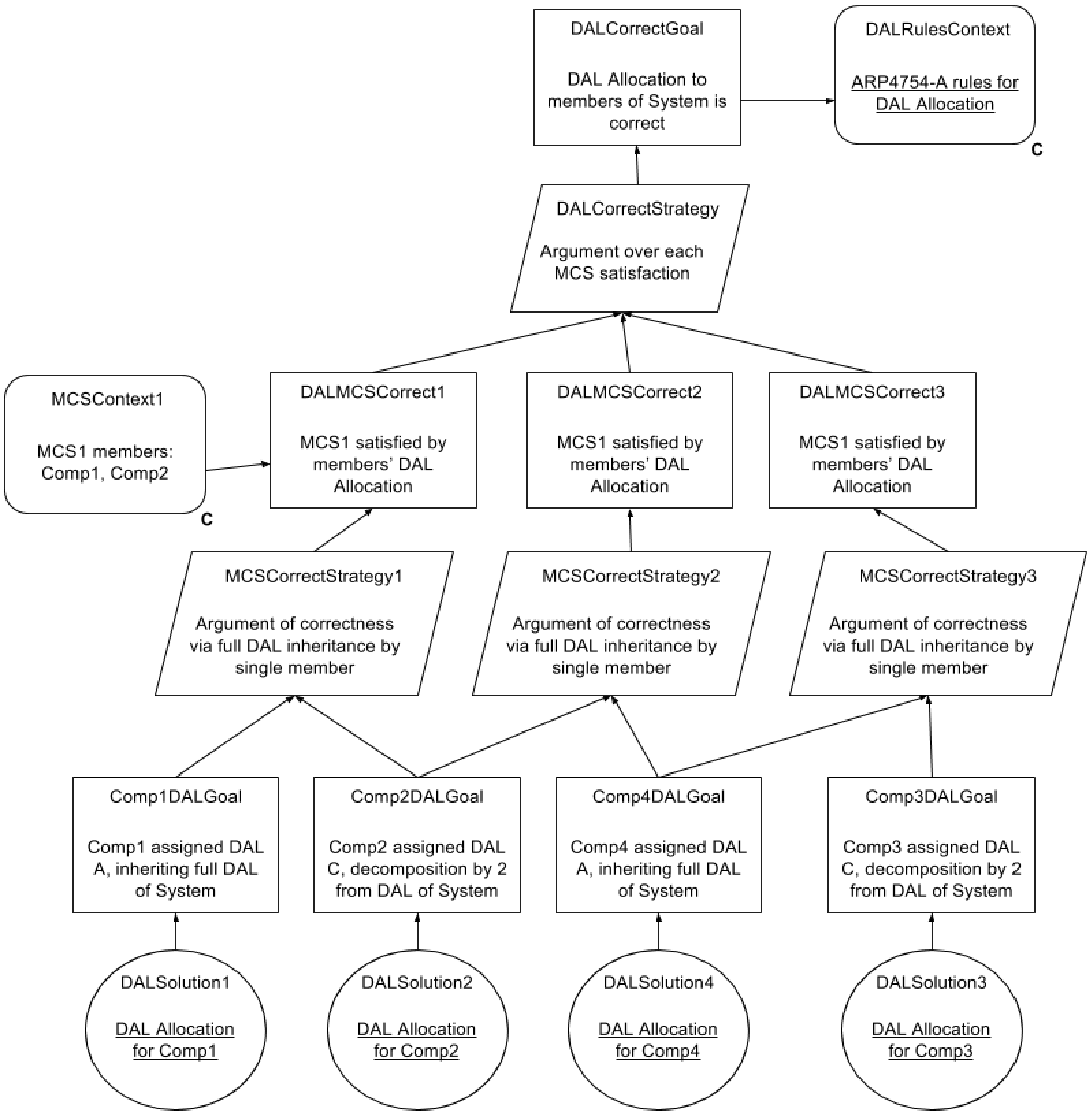

DALCorrectGoal
DAL Allocation to members of System is correct
DALRulesContext
ARP4754-A rules for DAL Allocation
C
DALCorrectStrategy
Argument over each MCS satisfaction
MCSContext1
MCS1 members: Comp1, Comp2
C
DALMCSCorrect1
MCS1 satisfied by members' DAL Allocation
DALMCSCorrect2
MCS1 satisfied by members' DAL Allocation
DALMCSCorrect3
MCS1 satisfied by members' DAL Allocation
MCSCorrectStrategy1
Argument of correctness via full DAL inheritance by single member
MCSCorrectStrategy2
Argument of correctness via full DAL inheritance by single member
MCSCorrectStrategy3
Argument of correctness via full DAL inheritance by single member
Comp1DALGoal
Comp1 assigned DAL A, inheriting full DAL of System
Comp2DALGoal
Comp2 assigned DAL C, decomposition by 2 from DAL of System
Comp4DALGoal
Comp4 assigned DAL A, inheriting full DAL of System
Comp3DALGoal
Comp3 assigned DAL C, decomposition by 2 from DAL of System
DALSolution1
DAL Allocation for Comp1
DALSolution2
DAL Allocation for Comp2
DALSolution4
DAL Allocation for Comp4
DALSolution3
DAL Allocation for Comp3

Figure 5.3 - Example of argument structure supporting correctness of DAL Allocation

Figure 5.3 shows how it is possible to claim (DALCorrectGoal) that a calculated DAL allocation is 'correct' according to the rules described in ARP4754-A. The claim depends on each MCS associated with the hazard being satisfied (DALMCSCorrect1, 2 and 3). In turn, this requires the members of each MCS to have DALs allocated according the standard rules. To support each of the final set of claims, some form of documentation confirming the appropriate DAL allocation can be provided and linked as solution to the claims (DALSolution1, 2, 3 and 4). The allocation rules are also provided as contextual reference for the reader's benefit (DALRulesContext). Finally, a list of the members of each MCS as context should also be provided with each claim supporting the MCS's satisfaction. Due to space constraints, only one, for MCS1 (MCSContext1), was included in Figure 5.3.

Regardless of which way the allocation is justified, we should note that all of the information relayed through each explanation is available during the option generation stage. Thus, generating such justification requires no further user input.

The (near-)optimal choice of DAL allocation can also be incorporated into the safety argument. By doing so, the safety argument can explain why the particular choice of DALs is favourable. Such an argument might not be significant to regulatory or certification authorities; ARP4754-A does not require justification with regards to the cost of the system. However, the regulatory authorities are not the sole stakeholder of the safety case. Other parties, such as the developers and the owners of the system, would presumably have a vested interest in minimizing the cost of safety during development and maintenance. Otherwise, budget restrictions might fail to be met or the system's cost might become unprofitable. This view is consistent with the ALARP principle as well, mentioned in Section 1 of Chapter 2. By selecting lowest in cost but still standard-compliant DAL allocations, both risk and development cost are expected to be minimized.

An argument of this kind will need to incorporate the assumptions made with regards to the development cost of each architectural element for its corresponding DAL. For example, let us assume the DAL Allocation #5 from Table 5.2 was chosen on the basis of the cost function seen in Table 5.4.

DAL	A	B	C	D	E
Cost	50	40	20	10	0

Table 5.4 - Example Cost Function

The total cost of the allocation choices in Table 5.2 can then be evaluated as follows in Table 5.5. Options #3, #5 and #6 are all optimal when comparing them strictly under the above cost criterion.

Allocation	#1	#2	#3	#4	#5	#6
Total Cost	160	150	140	150	140	140

Table 5.5 - Overview of Allocation Costs

Given this choice, it is now possible to generate appropriate justification for DAL Allocation #5 from Table 5.2. An example of how this justification could be incorporated into an argument structure can be seen in Figure 5.4.

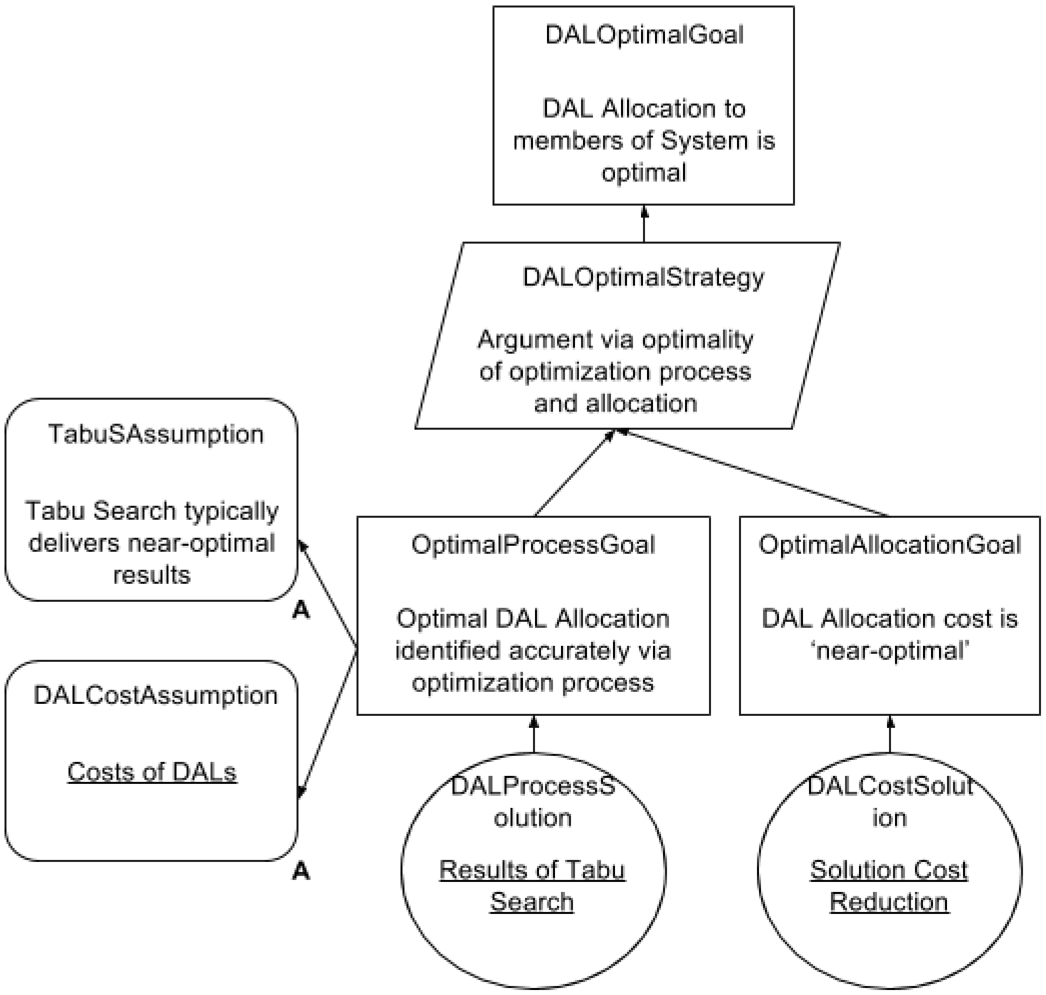
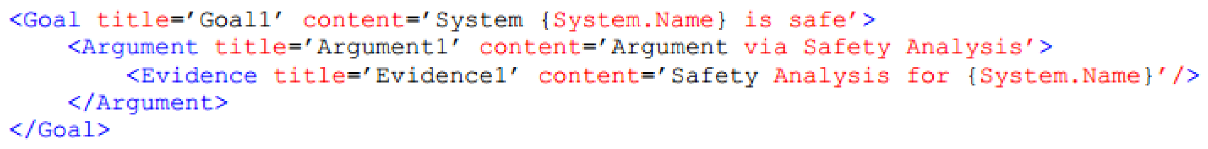

DALOptimalGoal
DAL Allocation to members of System is optimal
DALOptimalStrategy
Argument via optimality of optimization process and allocation
TabuSAssumption
Tabu Search typically delivers near-optimal results
A
DALCostAssumption
Costs of DALs
A
OptimalProcessGoal
Optimal DAL Allocation identified accurately via optimization process
OptimalAllocationGoal
DAL Allocation cost is 'near-optimal'
DALProcessSolution
Results of Tabu Search
DALCostSolution
Solution Cost Reduction

Figure 5.4 - Example of argument structure supporting optimality of DAL Allocation

The argument structure in Figure 5.4 supports a claim that the DAL Allocation to the system's components is 'optimal' (DALOptimalGoal). Due to the nature of the optimization used here, this claim functions similarly to the safety case's overall claim of safety. Specifically, it is intended to be an aspiration and not a guarantee. However, as with the case of the rest of the safety case, the claim goes beyond that of a mere promise. It is supported by a rationally applied optimization process which aims to yield near-optimal results (DALOptimalStrategy). Both the choice of the optimization and the assumptions of cost with which the process searches are included as relevant assumptions (TabuSAssumption and DALCostAssumption). Finally, the results of the Tabu Search and the cost reduction as a result of the choice of DALs are incorporated as support. There are various ways of presenting evidence for the cost reduction. The simplest would be a calculation of the difference in cost between an allocation where all members were trivially assigned the maximum DAL to satisfy the requirements which the optimization process identified.

2.9. Safety Argument Pattern Instantiation

As mentioned in Section 4.1 of Chapter 2, the method does not employ argument patterns in an advisory role. Instead, patterns are used to provide control over the emerging argument through its generation process. The notation chosen for expressing the argument pattern was GSN, although CAE could be supported as well. The argument pattern is stored in the form of an XML document, which facilitates modification and readability. The structure of either notation system can be directly translated to XML, with tags representing individual GSN node elements and embedding of tags representing node relationships such as 'Is_Supported_By' and 'In_Context_Of'. An example of this structure in XML can be seen in Figure 5.5.



```
<Goal title='Goal1' content='System {System.Name} is safe'>
    <Argument title='Argument1' content='Argument via Safety Analysis'>
        <Evidence title='Evidence1' content='Safety Analysis for {System.Name}'/>
    </Argument>
</Goal>
```

Figure 5.5 - Basic Example of Argument Structure in XML viewed in Notepad++ (Ho, 2017)

The example in Figure 5.5 can be represented graphically as the argument structure seen in Figure 5.6. The "System.Name" reference in this case has been instantiated with "Navigation Control". This process will be explained further in the following sections.

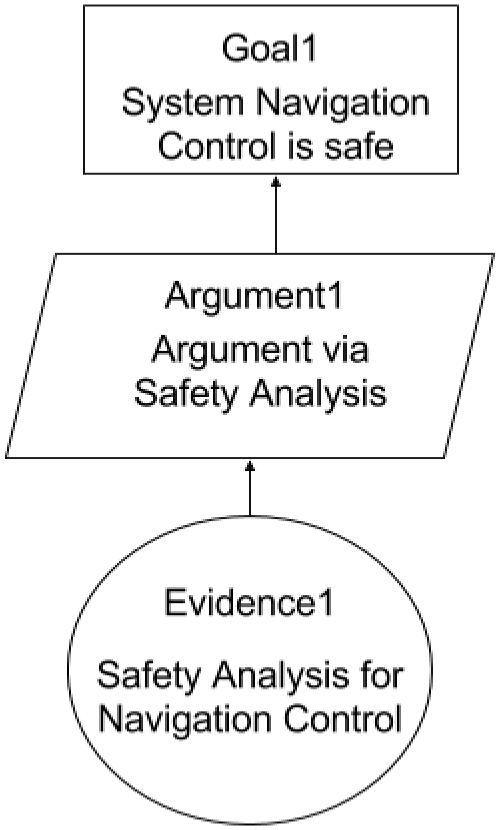

Goal1
System Navigation Control is safe
Argument1
Argument via Safety Analysis
Evidence1
Safety Analysis for Navigation Control

Figure 5.6 - Argument structure from XML example

The instantiation algorithm is relatively straightforward; it simply parses the elements found in the pattern structures (loaded from the XML files). It then appends said elements onto the main argument structure. The algorithm can be summarised as follows:

```
Parse-Pattern(Current-Node, Current-Argument, Current-Library)

    1.  Current-Node = First node of the element list
    2.  While Current-Node is not empty
            a.  If (Current-Node is basic node)
```

```
        i. Generate new argument element of given type, title and content
        ii. Instantiate title and content, if needed
        iii. Update Library if needed
        iv. Insert node into Current-Argument
        v. Update Current-Argument
        vi. For each Child-Node in Current-Node
            1. Parse-Pattern(Child-Node, Current-Argument, Current-Library)
    b. else if (Current-Node is for-each node)
        i. Handle for-each pattern element
    c. else if (Current-Node is module node)
        i. Handle module element
```

Regarding the parameters to the algorithm:

- Current-Node is the node of the pattern currently being parsed
- Current-Argument is the position in the argument graph where nodes are currently appended
- Current-Library is the current state of the Library

When recursively called, the algorithm passes copies of each of the above parameters to the following call. The purpose behind this will become apparent further on, through the explanation of the for-each element.

Sub-step (2.a.ii) of the algorithm expands textual elements based on the Library. For example, in Figure 5.5, the '{System.Name}' element from the generation pattern is replaced by the system's actual name during instantiation. This is possible by parsing segments of the pattern denoted with the open and closed brackets. This means the Library is searched for an entry with an alias of 'System', which in turn contains a property with an alias of 'Name'. The value of the 'Name' is the name of the system referred to, in the case of this example, ‘Navigation Control’. The pattern instantiation parser can recognize the ‘dot’ operator in ‘{System.Name}’ and isolate the ‘Name’ as a property of ‘System’. Similarly, other properties of a system can be referenced in this manner during instantiation. Additionally, iteration through the members of a LibraryElement is possible. For example, if ‘Function’ contains systems in the library, ‘Function.Systems’ provides a reference to those systems that can be iterated through.

To traverse through sets of elements provided by containers, the ‘for-each’ pattern element can be used, as seen in Figure 5.7.

```
<for-each variable='{System}' type='System' source='{Function.Systems}'>
    <Goal title='{System.Name}SafetyGoal'
    content='{System.Name} does not contribute to hazardous conditions of
    {Function.Name}'/>
</for-each>
```

Figure 5.7 - Example of pattern iteration in XML

Figure 5.7 demonstrates the definition of the for-each element. In the example’s case, it is used to traverse the library container ‘Systems’ of ‘Function’. For each ‘System’ found within, it generates a goal tailored to that system. The for-each element establishes a context within it; in the case of the example, the variable ‘System’ means, per each iteration, the corresponding system contained within the function. This meaning is maintained until the </for-each> tag of the pattern is reached. At that point, any previous definition of ‘System’ has that meaning restored. The behaviour of the for-each pattern can be expressed, in abstract, in Figure 5.8.

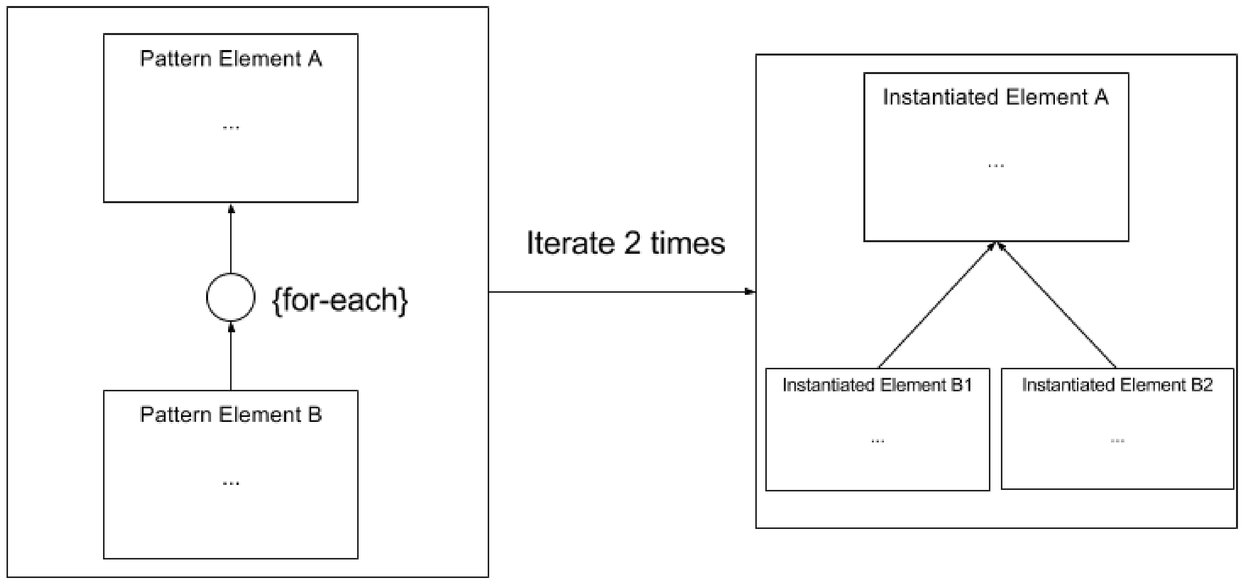


Figure 5.8 - For-Each Pattern Instantiation Example

As is shown in Figure 5.8, when the iteration algorithm encounters the for-each pattern element, it generates copies of the elements that follow within it. While the subgraph under the for-each element is parsed, a copy of the Library is maintained. This copy is exclusive to the subgraph being generated and is removed when returning to the remaining argument. The implication of this fact is that when patterns include for-each elements embedded in for-each elements, the copy of the Library maintained by each is expanded upon. This property applies to every recursive call of the Parse-Pattern algorithm as well; context is local for every subgraph parsed.

Sub step (2.b.i) of the Parse-Pattern can now be presented fully:

```
1.  Instantiate for-each parameters (Variable, Type, Target)
2.  Variable = First element of given Type found in Target
3.  While (Variable is not empty)
    3.1. Parse-Pattern(Current-Node, Current-Argument, Current-Library)
    3.2. Variable = Next element of given Type found in Target
```

The for-each element mirrors the multiplicity form of structural abstraction defined for GSN. It improves upon the concept by requiring a more accurate definition of the iteration range. The type of the iterated-upon elements is now also strictly defined. The other form of structural abstraction in GSN, optionality, has been reinterpreted as the ‘if-then’ element, seen in Figure 5.9.

```
<if-then condition='{Function.Systems.Count}>2'>
    <then>
        <Goal title='RedundancyPresent'
        content='Function safety supported by fault tolerance through redundancy'/>
    </then>
    <else-if condition='{Function.Systems[0].DAL} >= DAL(B)'>
        <Goal title='HighIntegrity'
        content='Function safety supported by high DAL of sole system'/>
    </else-if>
    <else>
        <Goal-undeveloped title='FunctionUndeveloped'
        content='Function safety unsupported'/>
    </else>
</if-then>
```

Figure 5.9 - If-then pattern example in XML

The if-then element allows sub-graphs of the argument pattern structure to be conditionally included or excluded from the instantiated argument. In the example, the pattern evaluates during instantiation whether the function is supported by at least 2 systems. If so, a claim arguing the function is fault tolerant as there’s at least two systems, one main and at least one redundant, supporting it. If not, the pattern evaluates whether the single system supporting the function has been allocated a relatively high DAL of B or A. It does so by comparing the numerical value of the assigned DAL (see Section 2 of Chapter 3) with that of the helper function “DAL”, which will be explained in the next paragraph. If the system’s DAL exceeds or is equal to B, an equivalent claim is included into the instantiated argument. If not, the function is unsupported, a fact reflected through an appropriate yet undeveloped claim.

The sub step (2.c.i) of the algorithm instantiates modules in a manner essentially identical to the for-each element, so further details will be omitted.

The “DAL” helper function returns the equivalent numerical value of the alphanumeric input from the pattern’s XML. For example, “DAL(B)” is evaluated to be 3, therefore, if the system has a DAL of 3 (equivalent to DAL B) or higher, the condition is truthful. These type of “helper functions” can be incorporated into the pattern-parsing process to improve ease of use.

We can annotate argument patterns with the documentation elements identified in Section 4.1 of Chapter 2 directly within the XML files defining the argument patterns. Although XML comments can be employed, more explicit annotation via dedicated tags can prove more precise. For instance, the <intent> and <related_patterns> tags could be used. Using explicit tags allows such tags to be parsed into a graphical interface representation. In turn, this enables more effective editing and representation of comments directly within the user interface, alongside the pattern’s representation in GSN.

3. Basic Example

We will now demonstrate how the process is applied via an example of a basic system. Let us consider a simple system with a single function, similar to the one seen in the examples in Chapter 4. The system can be seen in Figure 5.10. In the figure, the arrow indicates the direction of the system’s output.

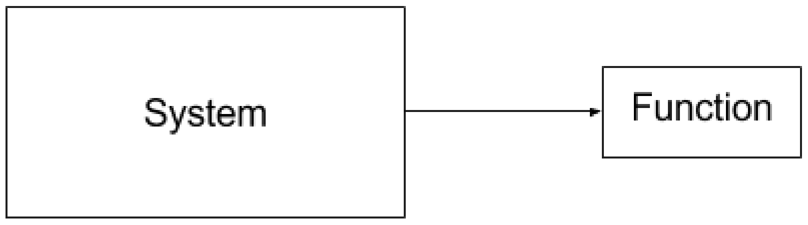


Figure 5.10 - Overview of the System & Function

Let us now assume that the FHA of the system’s function can be summarised as follows:

- FHA Name: Function FHA
- Function Name: Function

- Hazard Name: Hazard
- Hazard Description: Loss of Function
- Hazard Effect: Dangerous consequences due to loss of Function
- Phase: Flight
- Hazard Classification: Catastrophic
- DAL: A
- System requirements: Likelihood of occurrence at most 10E-9 per flight hour
- Supporting material: N/A
- Verification method: Quantitative and Qualitative Analyses

To clarify, the system requirements set an upper limit on the (estimated) likelihood that this kind of hazard might be triggered. Simply put, this hazard should not be expected to occur more than once every billion flight hours, on average. Supporting material was not necessary to be included, as it is usually employed only for cases where the hazards are not well understood. Finally, the choice of the verification method comes directly from ARP4761 (SAE, 1996:23) for Catastrophic hazards. This means that the method(s) used to verify the requirements for this hazard will need to assess the system both quantitatively and qualitatively. The combination of FTA and FMES can cover our generic example in this respect, although in practice more particular cases might require additional methods to be applied.

At this point, the functional modelling and the FHA have been completed, so the next step is to model the underlying system. Figure 5.11 shows the system architecture, consisting of 3 components.

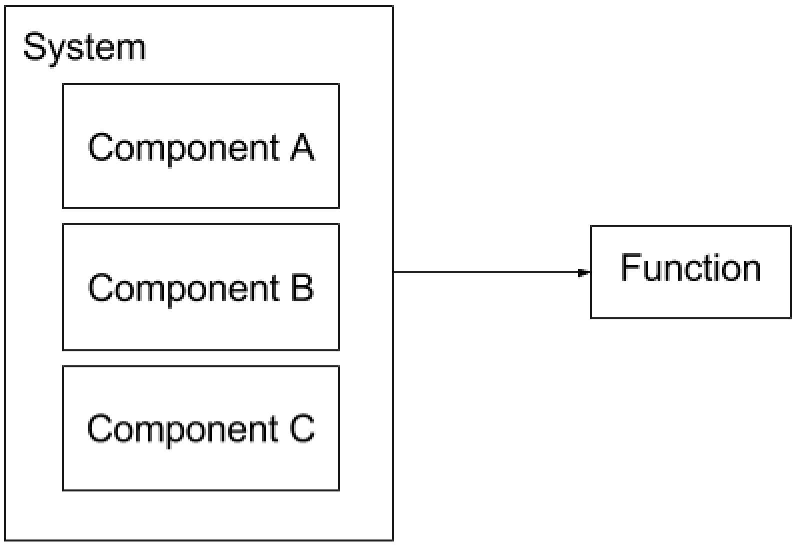


Figure 5.11 - Overview of the System architecture

Following this stage, failure behaviour annotation can be applied to the model. Due to the nature of the function's hazard, the applicable failure mode for the system would be an 'omission'. This means the failure analysis for the system will investigate the causes that can lead to an omission of output when one is demanded. In actual systems, many more possibilities would have to be considered as potential failure modes.

The potential cause of failure for the function – its omission – is clear in this case; it is an omission of the system's output. For our system, we can determine that only a particular combination of component omission failures can cause the system's omission failure. This combination can be expressed via Boolean logic in a Disjunctive Normal Form (DNF):

Omission-of-Function = Omission-of-System-Output = (Omission-of-Component-A AND Omission-of-Component-B) OR Omission-of-Component-C OR System-Failure-TypeA

Each of the Omission-of variables constitutes a Boolean variable representing an omission of its corresponding element's output. System-Failure-TypeA is also a Boolean variable, representing a particular way the System could fail and trigger an omission of output on its own. DNF representation is useful as it allows direct identification of the MCS. Each conjunction within the disjunction constitutes an individual MCS.

The annotation is now complete; HiP-HOPS can be applied to automatically produce the fault tree for the function (trivial) and the system. Figure 5.12 shows the System's fault tree.

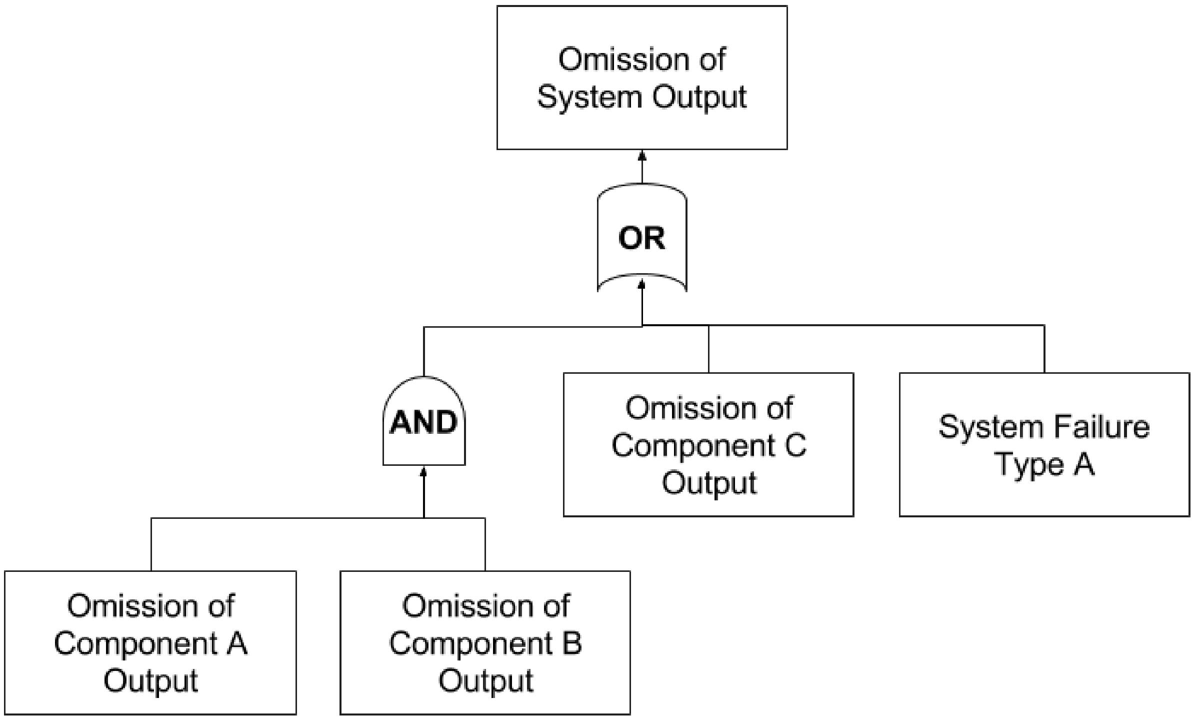

Omission of System Output
OR
AND
Omission of Component C Output
System Failure Type A
Omission of Component A Output
Omission of Component B Output

Figure 5.12 - Example System Fault Tree

Given the above fault tree, the MCS for the System become straightforward to identify:

- Omission of Component A Output AND Omission of Component B Output
- Omission of Component C Output
- System Failure Type A

The members of the MCS must be assigned DALs collectively contributing to a DAL A hazard. The following options are available, seen in Table 5.6.

Allocation #	Component A	Component B	Component C	System
#1	A	C	A	A
#2	C	A	A	A
#3	B	B	A	A

Table 5.6 - Options for DAL Allocation on Example System

Given the cost function in Table 5.4 from before, the option that emerges as the optimal is either #1 or #2. Let us assume that #1 was chosen arbitrarily, as there are no additional criteria defined for the selection.

All that is needed now is for a user-defined argument pattern to be applied. For the purposes of this example, the pattern will be kept relatively simple to illustrate the process. Figure 5.13 demonstrates the pattern.

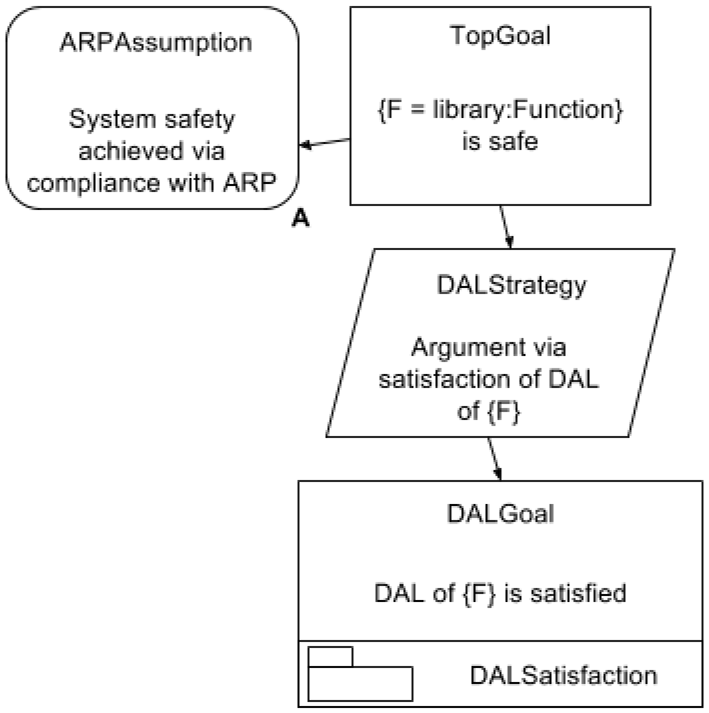

ARPAssumption
System safety achieved via compliance with ARP
A
TopGoal
{F = library:Function} is safe
DALStrategy
Argument via satisfaction of DAL of {F}
DALGoal
DAL of {F} is satisfied
DALSatisfaction

Figure 5.13 - Basic System Example Pattern - Overview

ARPAssumption refers to the implicit assumption in the ARP4754-A (and supporting documents') assumption. That is the assumption that by adhering to the guidelines within the standard, developed systems are acceptably safe. Based on this assumption, the TopGoal claim is established supporting the system's safety. The 'F' variable is designated as the input to the pattern. The pattern will initialize it to represent the system Function and incorporate it into the Library. DALStrategy is a simplification of the line of argument seen in the standard. By meeting the system requirements – simplified as the DALs for the example – the TopGoal is supported. Finally, DALGoal is an external goal referenced in the DALSatisfaction module. The incorporation of the module enables the pattern to process the entirety of the system architecture. This should become apparent by inspecting the module, seen in Figure 5.14.

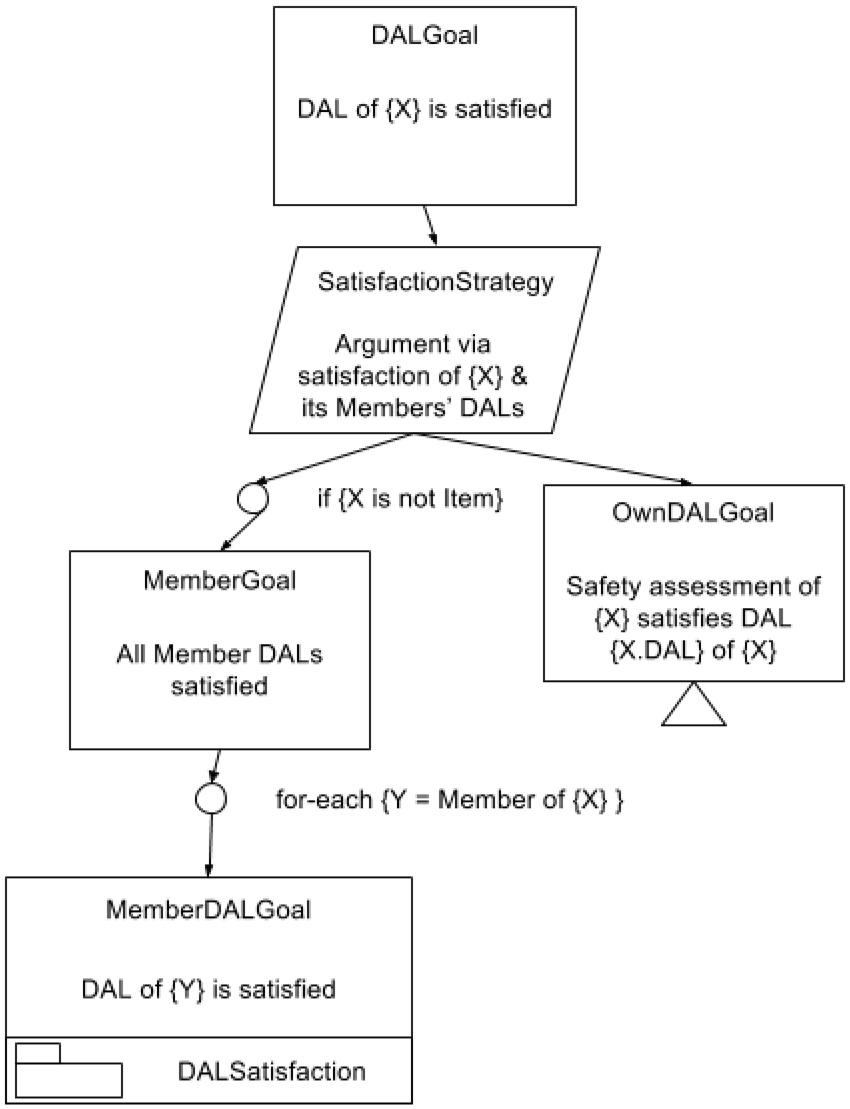

DALGoal
DAL of {X} is satisfied
SatisfactionStrategy
Argument via satisfaction of {X} & its Members' DALs
if {X is not Item}
OwnDALGoal
Safety assessment of {X} satisfies DAL {X.DAL} of {X}
MemberGoal
All Member DALs satisfied
for-each {Y = Member of {X} }
MemberDALGoal
DAL of {Y} is satisfied
DALSatisfaction

Figure 5.14 - Basic System Example - DAL Module

DALGoal is the claim referenced in Figure 5.14, supporting that the DAL of the subject X (whatever that may be) has been satisfied. Satisfaction depends on meeting the safety assessment procedures for the subject and its Members' DALs, expressed in SatisfactionStrategy. Two supportive claims are then established; MemberGoal and OwnDALGoal. The former establishes DAL satisfaction per each Member and the latter of the subject itself. The OwnDALGoal is purposefully undeveloped and simplistic; in practice, much more detail would be expected to argue and support this kind of claim. The inclusion of the MemberGoal is dependent on whether the subject is an Item or not; the if-then element above

MemberGoal evaluates the type of X. Finally, a for-each element iterates through each Member of the subject and instantiates another module claim. Each Member of X is referenced through the Y variable defined in the for-each element, which also feeds as input into the subsequent module instantiations. It should be noted that the typical strategy element linking goals is absent between MemberGoal and MemberDALGoal. Their relationship is trivial and the pattern is more concise as a result.

By instantiating the module within itself, this effectively produces a recursive effect, not unlike a recursive algorithm. The recursion terminates when a Member which is an Item is reached. Thus, in the case of the example, the pattern will instantiate for the Function, then for the System and finally for each of the components of the System.

The instantiation follows the algorithm described earlier; the resulting argument structure can be seen in Figure 5.15. Each of the argument structure elements have been instantiated, producing an argument which encompasses the entire system architecture. Although the argument presented is trivial to produce in this case, the cost of doing so is limited to the crafting of the pattern. From that point onwards, reproducing the argument for changes that do not affect the pattern – such as architectural changes – will only require another application of the instantiation process.

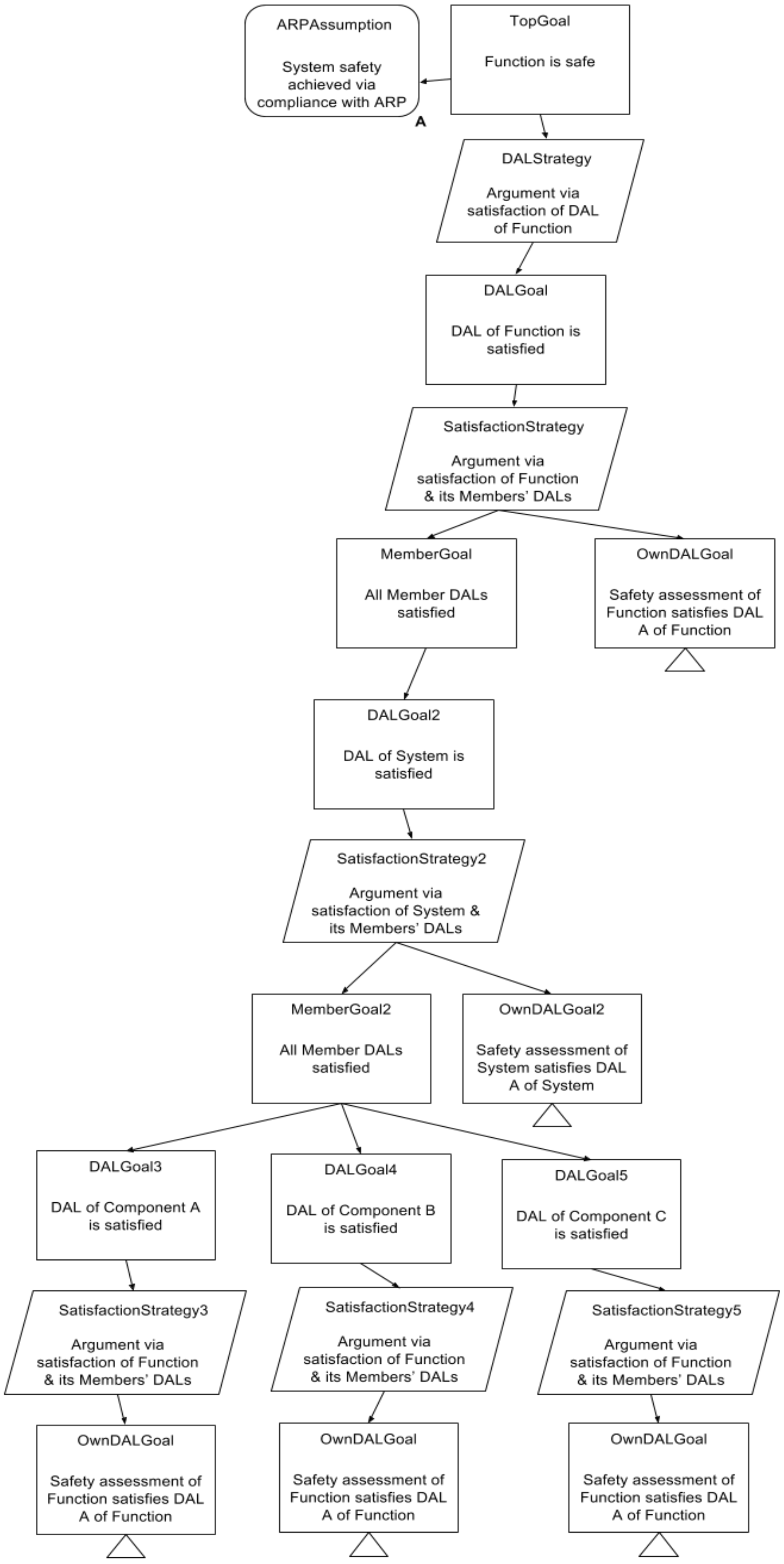

ARPAssumption
System safety achieved via compliance with ARP
A
TopGoal
Function is safe
DALStrategy
Argument via satisfaction of DAL of Function
DALGoal
DAL of Function is satisfied
SatisfactionStrategy
Argument via satisfaction of Function & its Members' DALs
MemberGoal
All Member DALs satisfied
OwnDALGoal
Safety assessment of Function satisfies DAL A of Function
DALGoal2
DAL of System is satisfied
SatisfactionStrategy2
Argument via satisfaction of System & its Members' DALs
MemberGoal2
All Member DALs satisfied
OwnDALGoal2
Safety assessment of System satisfies DAL A of System
DALGoal3
DAL of Component A is satisfied
DALGoal4
DAL of Component B is satisfied
DALGoal5
DAL of Component C is satisfied
SatisfactionStrategy3
Argument via satisfaction of Function & its Members' DALs
SatisfactionStrategy4
Argument via satisfaction of Function & its Members' DALs
SatisfactionStrategy5
Argument via satisfaction of Function & its Members' DALs
OwnDALGoal
Safety assessment of Function satisfies DAL A of Function
OwnDALGoal
Safety assessment of Function satisfies DAL A of Function
OwnDALGoal
Safety assessment of Function satisfies DAL A of Function

Figure 5.15 - Basic System Example - Instantiated Argument

4. Comparison to Previous Work

At this point it would be useful to summarize the differences and similarities with earlier approaches mentioned in Chapter 3.

With regards to FLAR2SAF, FPTC is employed versus our approach which uses FTA. As mentioned previously in Chapter 3, FPTC is employed locally per each component, consistent with a bottom-up approach. Although HiP-HOPS also requires local annotation of failure behaviour per each component, the method is applied top-down, from systems to components. This follows more closely the approach advised in ARP4754-A. Partly due to the use of FPTC, FLAR2SAF is focused on deriving safety arguments for components, specifically commercial off-the-shelf (COTS) software. Deciding whether to employ COTS software solutions is a decision that typically occurs later in the design stage. Our approach is instead applicable from earlier stages of system development. Finally, FLAR2SAF addresses particular component requirements derived from the component FPTC specification, whereas our approach addresses Development Assurance Levels (see Section 2 of Chapter 3). Further, our approach assigns these levels over the underlying system architecture as an intermediate step; this step is assumed to be performed in advance as the subject is mainly developed COTS.

With regards to THRUST, although the associated methods focus on process-based safety arguments and a unique, interactive construction approach, parallels can be drawn to the work presented in this thesis. In both cases, safety arguments are constructed based on a model of the system (in the case of THRUST, its development process). Further, both produced argument structures conform to the software integrity levels appropriate for the system (or process). Finally, argument patterns are employed in both cases to transform system/process models (and relevant information) to instantiated safety argument structures.

We shall now compare this thesis' method with the verification-based approaches in Section 8.1.1.3 of Chapter 3. The first approach is comparable to the one in this thesis in that it employs FTA in combination with argument patterns to produce safety arguments, however that is effectively the extent of the similarity. The main difference with the work presented here is that our method can be applied from the early stages of development, in a top-down fashion. Instead, the referenced method can only be applied following verification, which requires an implementation to be available.

The second approach constructs an extensive safety case, incorporating a plethora of information and detail across both its higher and lower levels. The divergence with this thesis lies once again in the automatic construction mechanism, as this thesis exploits local failure behaviour annotated on the system model. The referenced method instead uses PROLOG annotation and links tool-constructed higher and lower-level safety arguments. While this approach seems very useful when applied in the later stages of development, it requires significant detail to be provided in the specification of the model and its verification artefacts. The method proposed in this thesis does not require such in-depth specification detail of the system model to be applied, meaning it is suitable across all stages of development.

Finally, a comparison with the method presented in Section 8.1.1.4 of Chapter 3 will now be presented. Both methods use HiP-HOPS and the FTA and MCS safety assessment artefacts it produces as part of their argument generation process. An argument pattern is employed in both methods, which acts as a basis for the generated argument. The pattern automatically integrates tool-generated information from the system model and its safety assessment models as well. The main difference between the two approaches is the SPL context. An extensive metamodel is required by the referenced method to incorporate SPL information, requiring significant more detail than our generic approach.

5. Summary

In this chapter, we introduced a novel method for producing argument structures automatically, based on a user-defined argument pattern. First, some of the challenges associated with safety case construction and maintenance were highlighted. Then, some of the earlier work aiming to address some of these challenges was presented. A simple data model which enables model-based integration of safety assessment and assurance processes was introduced. The data model allows the model of the system to be further extended with DAL allocation information. The DAL allocation was produced using the HiP-HOPS extension introduced in the previous chapter. An instantiation algorithm which uses the data model in conjunction with a user-defined argument pattern was then presented. Differences with the relevant methodology reviewed in Chapter 3 were also discussed. Finally, a small example illustrating how the process can be applied concluded the chapter.

Chapter 6: Case Study on Abstract Flight Control System

1. Introduction

This chapter presents how the method introduced in the previous chapter can be applied towards a system of scale larger than the example shown in Chapter 5. The system in case is an abstract one, although heavily based on the Flight Control System (FCS) found in the Airbus A320 civilian commercial aircraft. The A320 is an aircraft with a considerable flight record, dating back to its first flight in 1987. In its roughly 30-year operational lifecycle, the aircraft has seen relatively few incidents involving fatalities considering the millions of flights it has been in service for. Unfortunately, in some of these incidents the loss of life was severe e.g. the 2007 TAM Linhas Aereas crash with about 200 passengers, crew and bystanders deceased at the crash location (CENIPA, 2007). However, software failures have not been the cause of the vast majority of such incidents. In (Holloway, 2015:1), the author argues that "no fatal commercial aircraft accident [has] been attributed to a software failure". On the contrary, "many of the technological improvements that have been credited with significantly reducing the accident rate have relied heavily on software".

From a technical standpoint, the A320 introduced the novelty of the digital 'fly-by-wire' flight control to a commercial aircraft. Fly-by-wire installs an electronic interface between the pilot and the aircraft controls. Thus, when a pilot issues a command, it undergoes processing through digitisation and computational control. The end result is an electronic command that issues the final order to steer the aircraft. A number of high-level control 'laws' define modes of flight control that vary the degree of direct control the pilot retains. In most cases, emergency mechanical control is provided through a corresponding law for situations where major electronic failure has occurred. Until the launch of the A320, fly-by-wire had only been employed in military aircraft.

The remainder of the chapter is structured as follows; first, a more detailed presentation of the A320's FCS follows to provide the necessary context for understanding its operation. A more abstract version of the A320 FCS is used for the case study. The method presented in Chapter 5 is applied to produce a safety argument structure for the abstract FCS.

2. Background

Controlling an aircraft is achieved through the appropriate orientation, deployment and withdrawal of its 'control' surfaces. Such surfaces are movable parts of the aircraft chassis, usually situated on the extremities of its wings and tail. On the wings, they include the ailerons, spoilers, flats and slats. On the tail, they include the rudder, elevators and horizontal stabilizer. Figure 6.1 is an informal representation of the A320 control surfaces for the wings and Figure 6.2 for its tail.

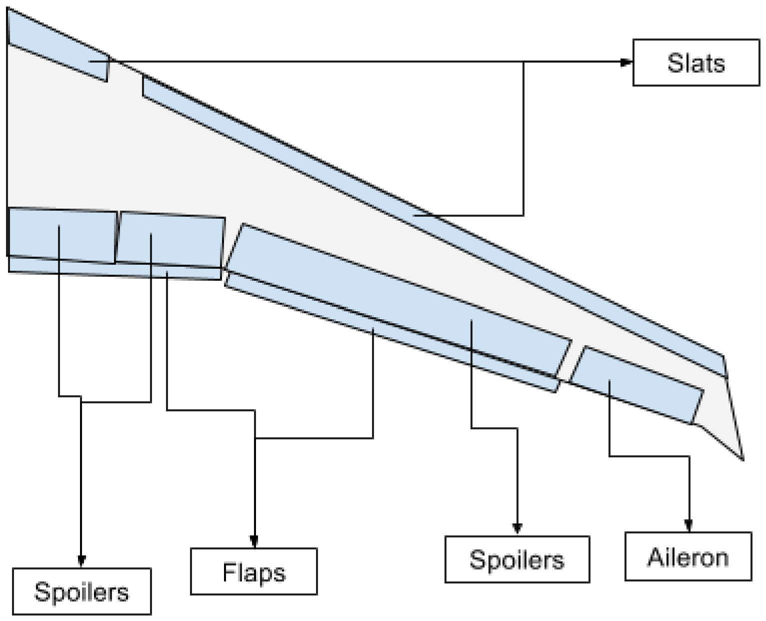


Figure 6.1 - A320 Flight Control Surfaces on Right Wing

Figure 6.1 shows the right wing of an A320 aircraft seen vertically from above, with all the control surfaces annotated. For clarification, the flaps are located beneath the spoilers, occupying parts of the mid-section of the wing. Spoilers are located above the flaps, on the upper-side of the wing. Finally, ailerons are located near the outer edge of the wings and slats are on the upper-front side of the wings.

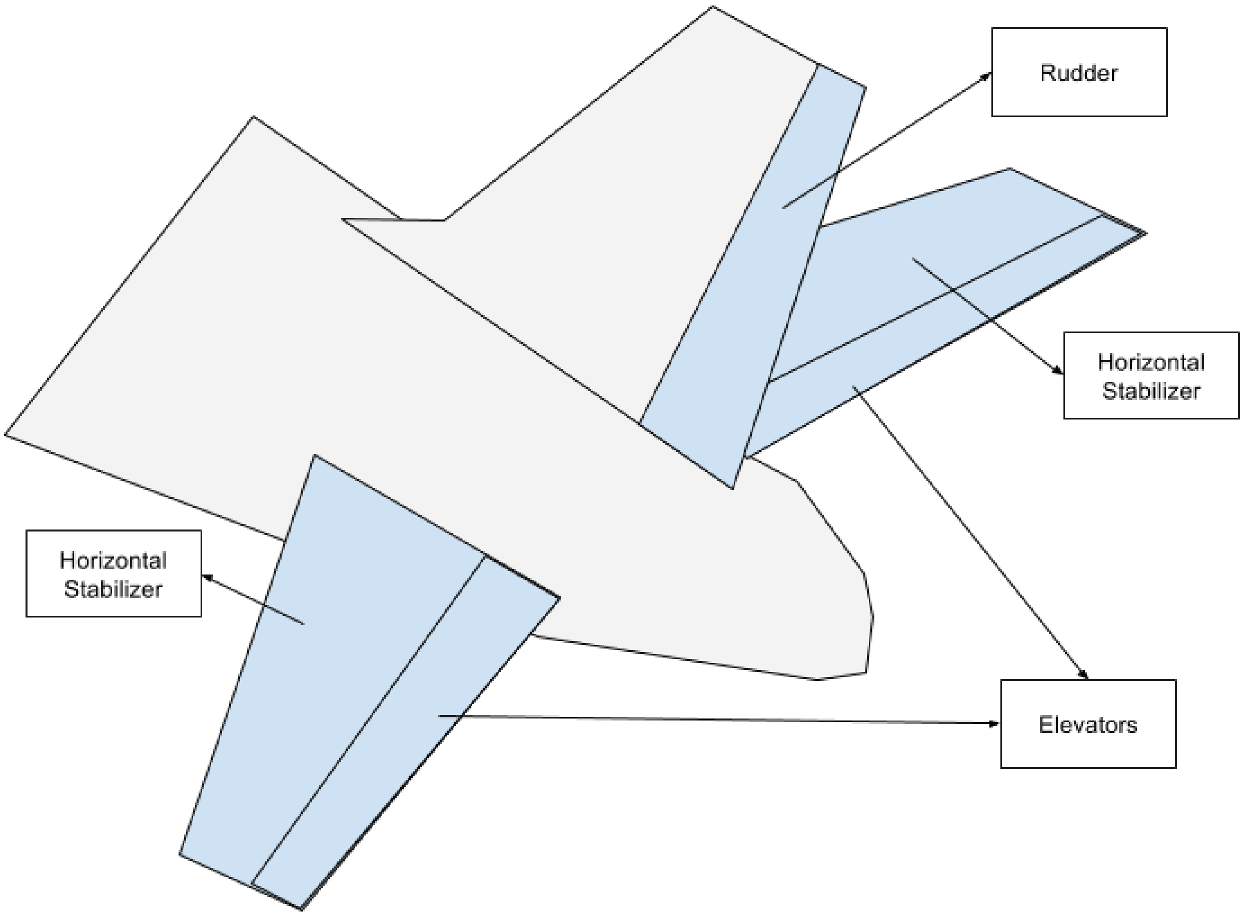


Figure 6.2 - A320 Flight Control Surfaces on Tail

Figure 6.2 represents the tail end of the A320 chassis, seen at an angle from the left, slightly above and behind the aircraft. Although the horizontal stabilizer is annotated twice, as it includes one part on each side of the chassis, it operates as a single unit. On each stabilizer's backside lies an elevator. Finally, the rudder is situated on the 'fin's' backside.

Each flight control surface is positioned appropriately through an electromechanical actuator. Each actuator receives a combination of signals from the aircraft's FCS, which result in an appropriate response. By combining the independent positioning of the control surfaces, the aircraft can achieve its required orientation during flight. Each actuator is powered hydraulically, by one or two out of three available hydraulic subsystems.

An aircraft's orientation can be defined as its rotation around each axis of three-dimensional space. For aircraft, these axes are aligned relative to the aircraft itself, as well as the Earth's curvature beneath it. They are known as:

- Roll, rotation about an axis aligned with the aircraft's forward direction.
- Pitch, rotation about an axis aligned across the aircraft's wings.
- Yaw, rotation about an axis vertically aligned between the centre of the aircraft and the centre of Earth.

For the A320, roll is provided through combined movement of the ailerons and spoilers, yaw is controlled via the rudder and pitch via the elevators and horizontal stabilizer.

The FCS intermediates between the pilots in the cockpit and the control surfaces. For controlling flight, the pilots employ sidesticks for providing input to the spoilers and ailerons. Each sidestick sends independent signals to the FCS. For controlling the rudder, two pairs of interconnected pedals are available for manual control. Finally, a wheel is located between the two pilots, allowing each to turn it via interconnected handles on each side. The wheel controls the angle of the horizontal stabilizer.

Input is propagated from the cockpit devices to the onboard computers that are part of the FCS. The incoming digital signal is processed by the computers, which take into consideration the current mode of flight, also known as a flight control law. Based on the input and the current law, the FCS computers output appropriate signals towards the control surface actuators.

A total of seven computers comprise the FCS, organized in three groups:

- the Flight Augmentation Computers (FACs). There are two FACs, they control the rudder actuators.
- the Elevator Aileron Computers (ELACs). There are 2 ELACs, they control the ailerons and elevators.
- the Spoiler Elevator Computers (SECs). There are 3 SECs, controlling the spoilers and horizontal stabilizer.

Each group of computers supports redundancy via ‘reconfiguration’. This implies that when the computer responsible for actively controlling a surface has been detected to have failed, another one of the group takes over its active role. As an example, if the primary ELAC is detected to have failed during its active control of the ailerons, ELAC 2 takes over. The computers not performing an active role are sometimes delegated to applying ‘damping’ commands to the surfaces. When in damping mode, a surface follows the movement of the active surface. By mirroring the movement of the active surface, the damping surface helps stabilize the aircraft against oscillation due to aerodynamic forces. Moreover, damping surfaces are immediately available to switch to an active role in case of failure of the currently active ones.

Each computer in the FCS features a ‘dual-channel’ architecture. The control channel is responsible with providing control over the corresponding surface or surfaces the computer is associated with. The monitoring channel is responsible with detecting whether the control channel is functioning correctly. Each channel can be considered an independent computational unit, as it includes its own processing unit, memory, input/output to actuators and sensors, power supply and software. Software in particular differs not only across each pair of channels per computer, but is also different for each group of ELACs and SECs. FACs are different from the other groups in that they are not redundant; if both FACs fail, pilot input will be required to maintain smooth rudder control. Malfunctions are detected mainly when the evaluation of the monitoring channel deviates significantly from the control. If the difference between the output of the two channels exceed a predetermined threshold for a predetermined minimum time period, then a failure is detected. In the case of such an event, the computer in question is isolated from further operation.

Software for the FCS computers was developed following the guidelines of DO-178A, an earlier version of the current DO-178C. Details regarding the exact process are limited, however, in (Brière & Traverse, 1993:2) the author explains that validation of the software was focused on the functional specification of the system.

The functional specification itself was developed via a Computer-Aided-Design (CAD) tool. All of the functional elements that comprise the system, e.g. control laws, data flow, control surface logic, reconfiguration logic and so on, were modelled this way. Each functional element included a formal definition and, logical rules which defined its interface interactions. This allowed digital configuration management support as well as partial syntax-checking of the specification (Brière & Traverse, 1993:2).

Returning to the subject of the FCS validation, once the functional specification was defined and validated, the next important step was to establish the software specification. The software specification was derived as a “copy” (Brière & Traverse, 1993:3) from the functional with an approach that aimed to maintain the software behaviour as close to the functional specification as possible. This allowed the remaining validation of the software specification to be minimized, requiring only evaluation of the interface between software and hardware.

Verification of the FCS relied on testing whether the software behaviour met its specification. Functional testing, aka black box testing was initially applied. This form of testing evaluates the software taking only into account its input-output behaviour. Following this set of tests, structural, aka white box testing was applied. White box testing inspects the actual code within a particular code module for errors (Ammann

& Offutt, 2008:21). Details on what precisely were the testing criteria could not be found; in (Brière & Traverse, 1993:3) the authors mention that “adequate coverage must be obtained for the internal structure and input range”. Presumably, each input to a particular code module would have its possible input range analysed. Based on this analysis, sub-ranges which yielded equivalent results would then be organized into equivalence classes. Thus, testing would require selecting appropriate test input data from each of the equivalence classes to achieve complete coverage. Acceptance of these criteria and any additional that might be required was subject to the approval of the aircraft stakeholders. Testing was performed via input simulation per each code module, as well as actual use with the system fully configured and installed on the aircraft in testing, flight simulator and live-flight environments. A standard set of tests as well as ad-hoc sets developed on-demand by engineers were regularly used both during development and operation.

Functional independence (see Section 2.3 of Chapter 4) is supported by distributing support for yaw, roll and pitch control over more than 2 groups of computers. Furthermore, within each computer group, one computer for the FACs and ELACs and 2 for the SECs are available to take over in the case of failure. The backup computers are maintained operational at all times so that they are readily available in case of failure. Item independence (see Section 2.3 of Chapter 4) is supported by differentiating the software of the monitoring and control channel of each computer. Additionally, as mentioned earlier, the SECs and ELACs featured independently developed software and hardware. All of the above contribute towards minimizing the risk of common errors for either the software or hardware of the FCS. Information regarding the independence status of the FACs was limited at the time of writing.

3. Flight Control

To initiate the process from Chapter 5, functional design of the aircraft is performed first. The flight control of the FCS is identified as the subject of the process. Naturally, more functions would be included in an actual aircraft, however the case study will be restricted due to space limitations. Figure 6.3 depicts how the function can be represented in a functional hierarchy. Braking, Navigation and Communication are also included as examples of other high-level functions. Pilot Feedback refers to the information provided from the FCS back to the pilots via the cockpit displays. Although in the actual A320 feedback extends to other subsystems as well, for the purposes of the case study, analysis will be limited to just the FCS.

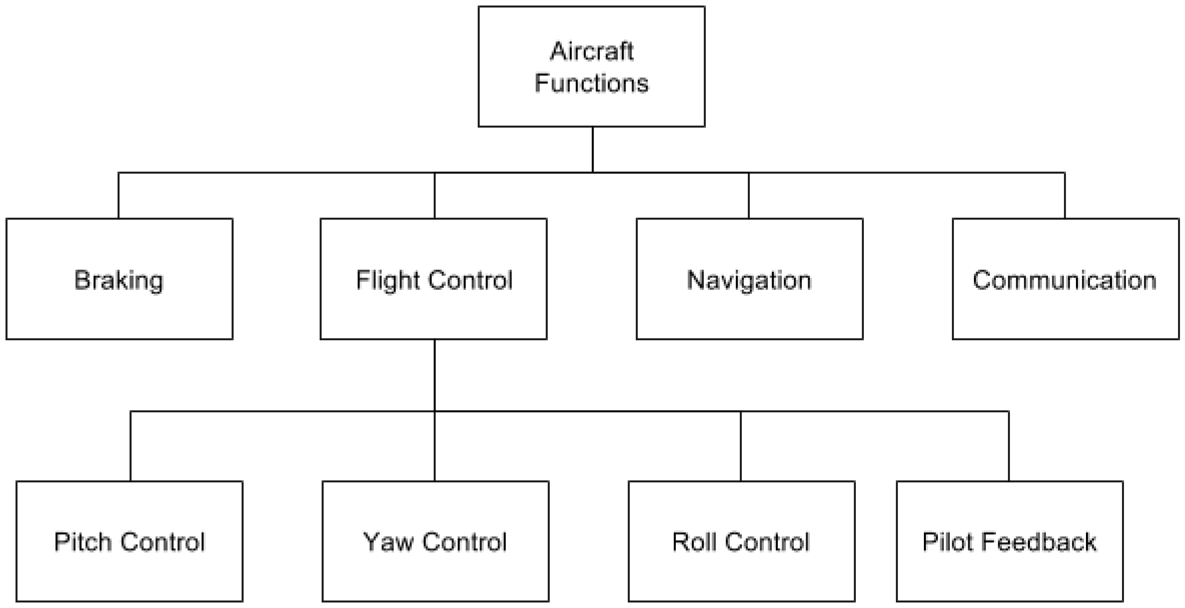


Figure 6.3 - FCS Functional Hierarchy

4. Functional Hazard Analysis

Functional Hazard Analysis (FHA) is performed for the Flight Control function as per Section 2 of Chapter 3. The results of the FHA are as follows, in Table 6.1.

Function	Failure Condition	Phase	Effect	Class	Verification
Flight Control	Undetected Loss	Take-off; Flight; Landing	Collision; Aircraft destruction; Passenger and crew fatalities / severe injuries	Catastrophic	Peer Review; Formal Analysis; Software & Hardware Testing
Flight Control	Detected Loss	Take-off; Flight; Landing	Increased crew workload; Passenger or crew injuries	Major Hazardous	

Table 6.1 - FHA for Flight Control Function

The case study will focus on two types of loss of Flight Control for simplicity. The first type results in a scenario where the crew is unaware of the loss of control. This has led to accidents with fatal results in the past (Holloway & Johnson, 2008) and is classified as ‘Catastrophic’ as a result. The second type is the case where the crew are aware of the loss of control. In this case, the pilots can apply measures to maintain manual flight control and minimize the risk of injuries or fatalities. Due to the potentially challenging situation the flight might be experiencing at the moment of failure, the pilots might be required to tackle multiple problems simultaneously. This would result in an increased workload for the crew. However, if the crew becomes aware of the failures, mechanical backup can provide sufficient control to land safely. In such situations, fatalities might be avoidable although the danger of injuries remains. Thus, the hazard is classified as ‘Major Hazardous’.

5. Flight Control System Specification

Following the identification of the Flight Control function and its attributes, design would be expected to proceed towards defining the system architecture that would deliver the function. Flight Control is provided exclusively through one system; the FCS. Subsystems such as power, hydraulics, pilot controls and sensor input provide support to the FCS. Figure 6.4 depicts an overview of the FCS, including its major subsystems, input subsystems and the functions it provides. The arrows indicate the flow of input, including data, commands and energy, from one system to another.

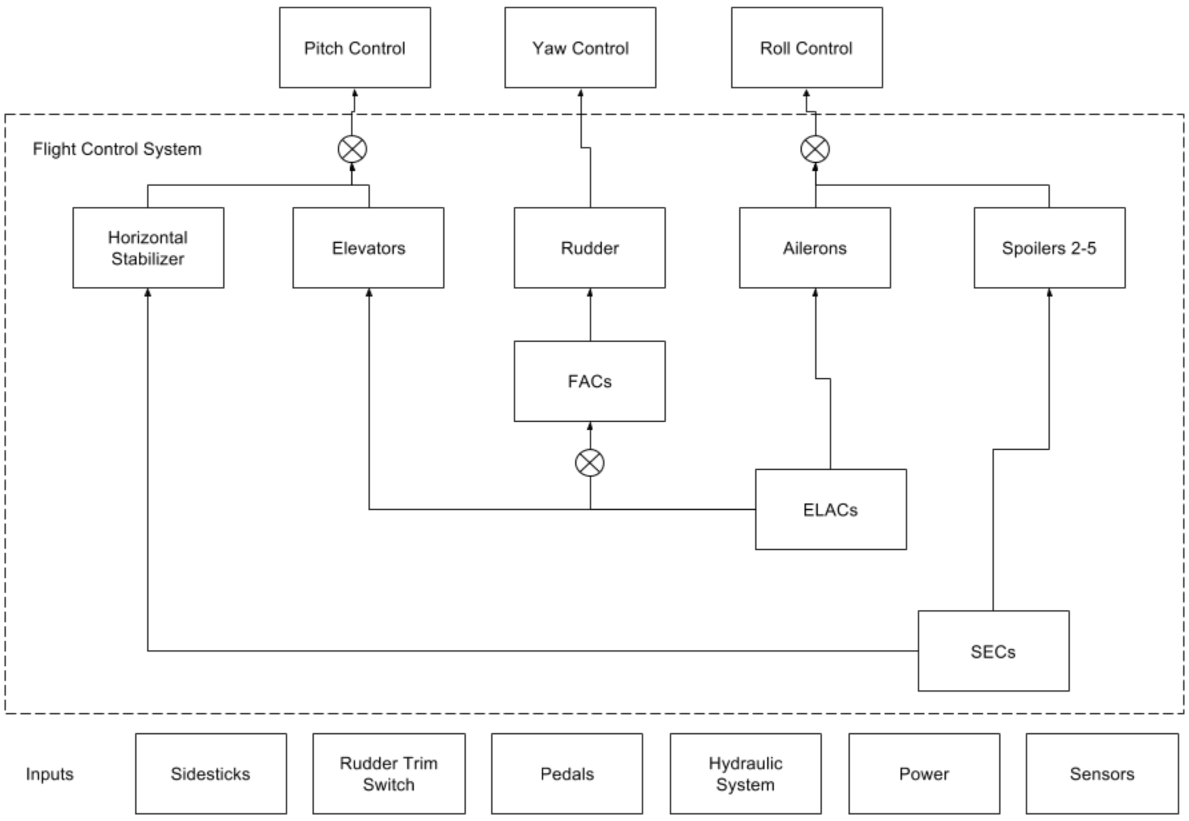


Figure 6.4 - Flight Control System Overview

The boundaries of the FCS are represented via the dashed lines. The subsystems providing input are simply listed near the bottom of Figure 6.4 to avoid overburdening the figure and to maintain clarity.

The relationship between the input subsystems and the FCS' subsystems will be discussed further in the upcoming sections.

Figure 6.5 shows a closer view of the Yaw Control function of the FCS and the subsystems involved in its delivery.

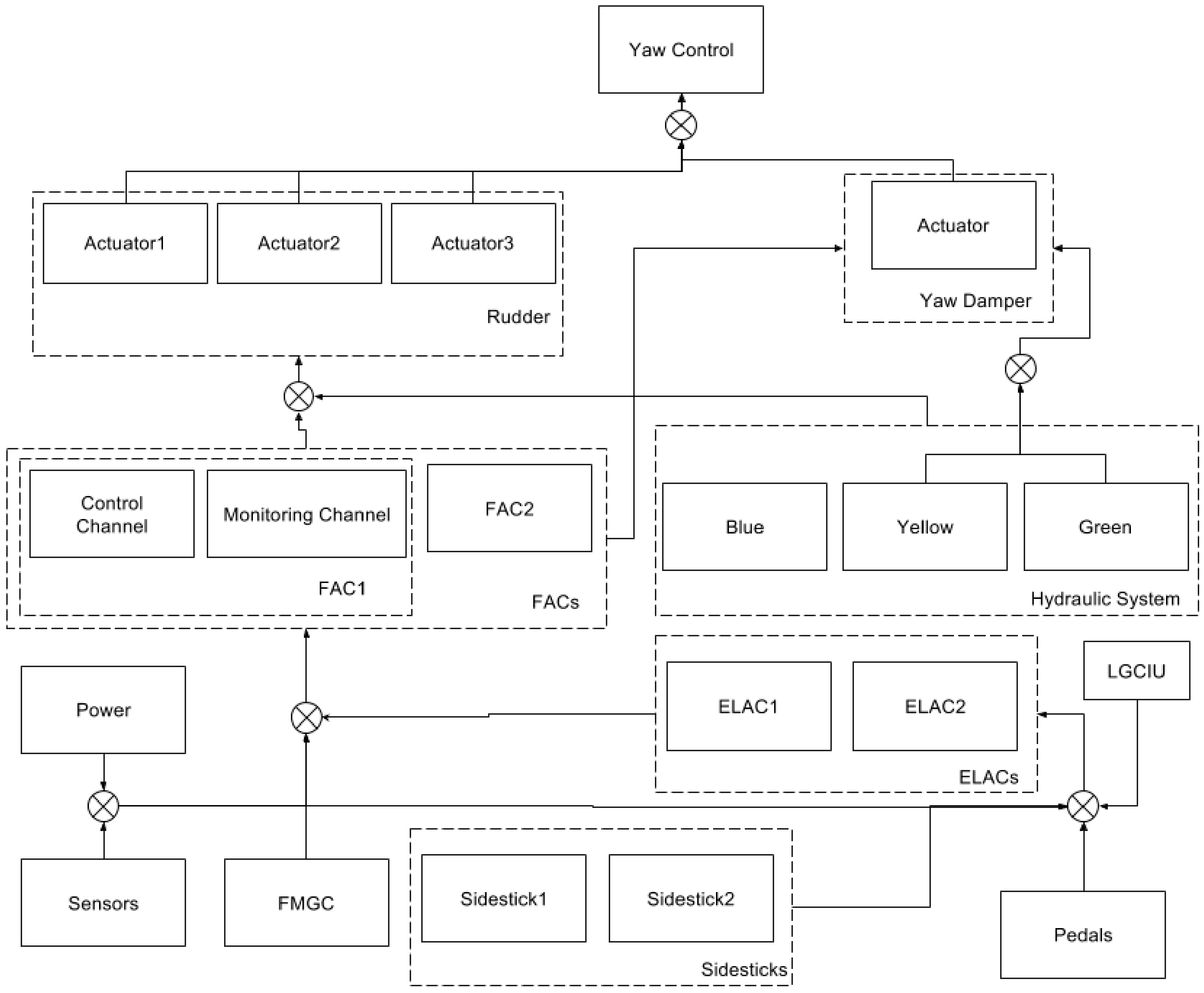


Figure 6.5 - Flight Control System - Yaw Control

The rudder provides primary control over the yaw axis rotation of the aircraft, alongside a yaw damper. The yaw damper provides automatic support to smooth the rudder motion against oscillation and similar phenomena. Yaw orders are computed by the ELACs, which receive pilot input. The orders are then transmitted to the FACs, which interpret the orders into direct commands to the rudder and damper actuators. The aircraft’s hydraulic system powers, via its Yellow and Green subsystems, the damper and each subsystem also powers directly an individual rudder actuator. The contribution from the hydraulic subsystems to each actuator is depicted merely as a link from the entire hydraulic system to the rudder for simplicity. Each sidestick sends commands through individual channels to the computers, whereas the pedals send commands via the same channel. The Flight Management Guidance Computer (FMGC) and Landing Gear Control Interface Unit (LGCIU) provide information to the ELACs regarding the flight path and landing gear status. The FACs also receive FMGC input. The FACs do not receive input directly but are provided yaw orders from the ELACs. The ELACs compute the desired yaw based on pilot input and the current flight control law. Finally, sensors and power are also required for the computers’ operation and will be shown in greater detail in the following figures. In Figure 6.6 the Pitch Control function of the FCS is illustrated.

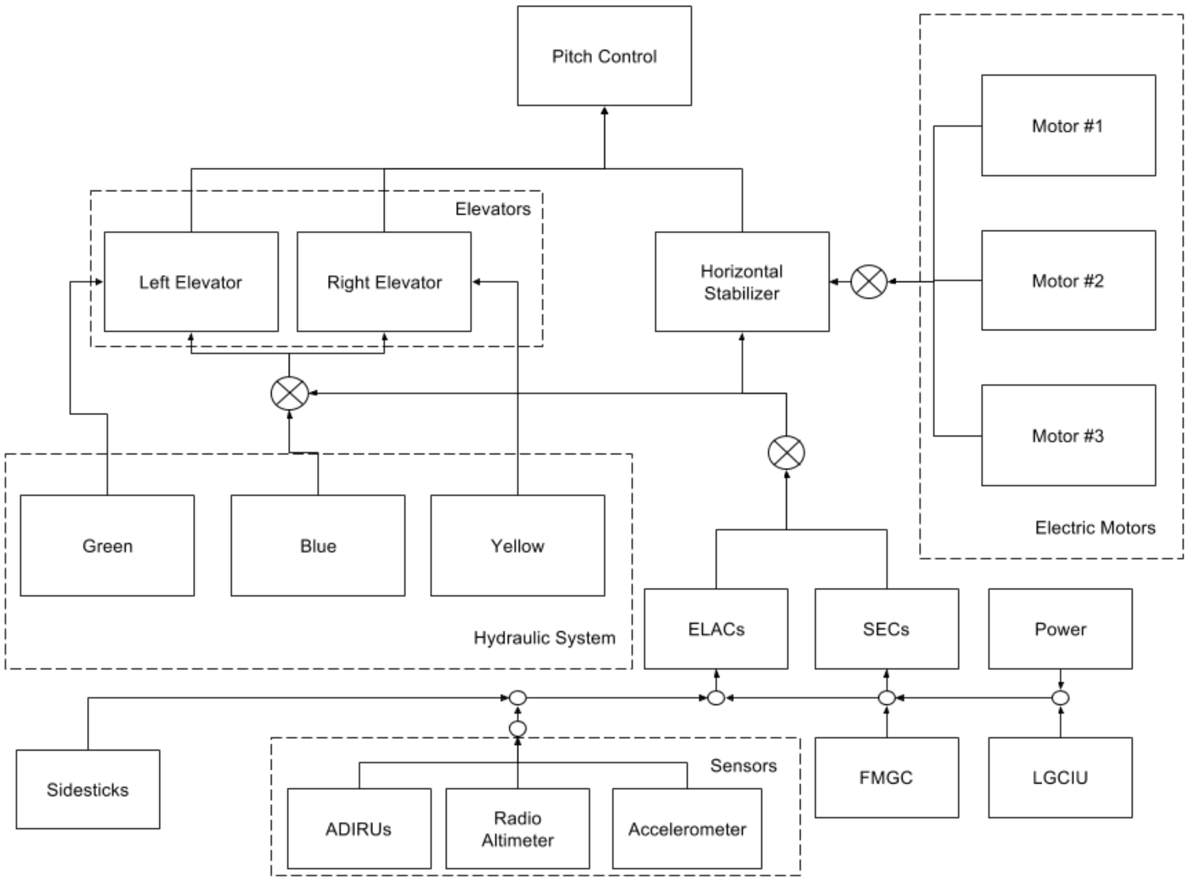


Figure 6.6 – Flight Control System – Pitch Control

Pitch is provided through the elevators and horizontal stabilizer, seen as the "Elevators" and Horizontal Stabilizer elements in Figure 6.6. The stabilizer actuator is powered via one of three electric motors. The elevator actuators are powered by the common hydraulic system; the Blue subsystem serves as backup, whereas the Green and Yellow power each actuator normally. Commands to the actuators are produced by the ELACs and SECs computers. In this figure, the Sensors common subsystem is presented in greater detail. Sensors includes Air Data Inertia Reference Units (ADIRUs), which provide information regarding the aircraft's speed and orientation. There are 3 ADIRUs supplying independently information to the FCS computers for redundancy. Next, the Radio Altimeter provides the current altitude of the aircraft. Finally, the Accelerometer calculates the various acceleration forces e.g. gravity that affect the aircraft.

The FMGC and LGCIU from the earlier Figure 6.5 are also featured. Figure 6.7 illustrates the Roll Control function of the FCS.

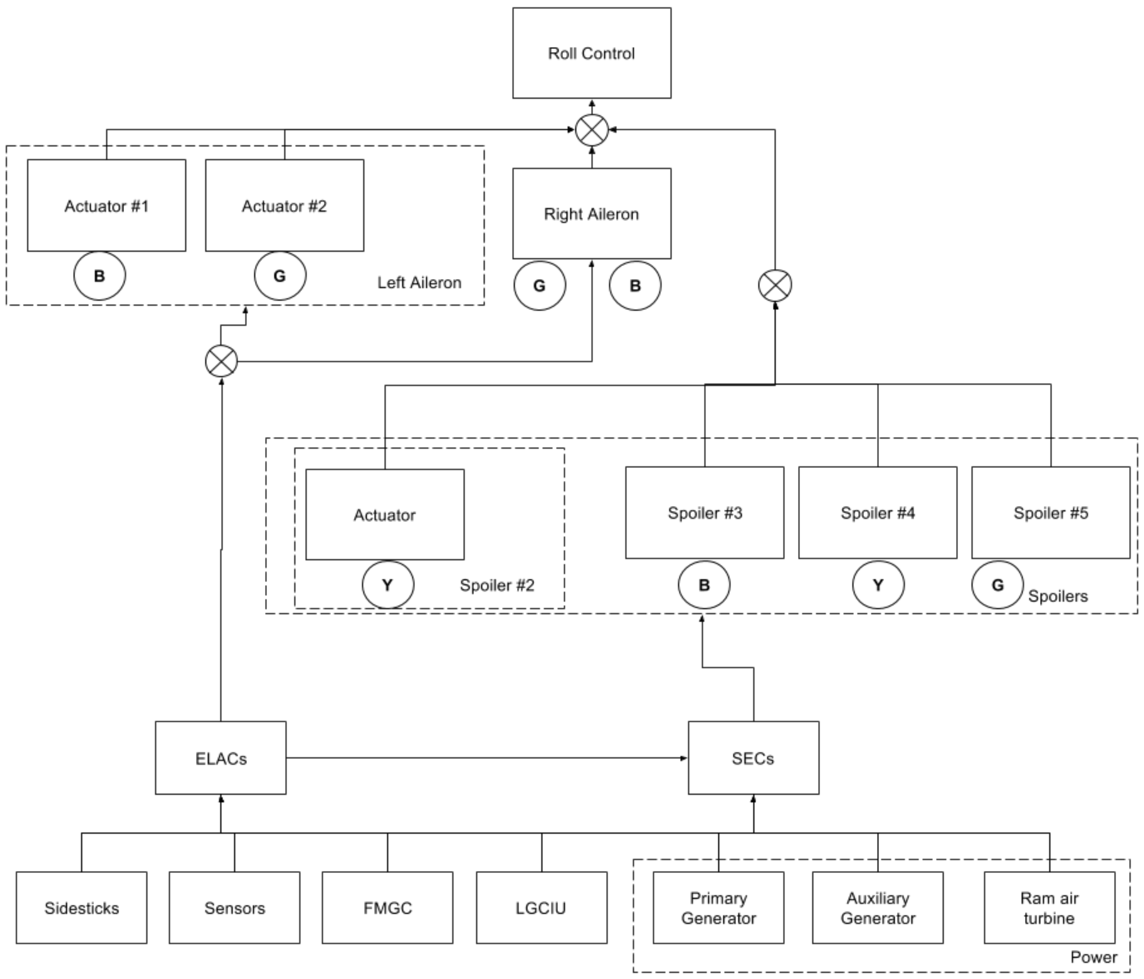


Figure 6.7 - Flight Control System - Roll Control

In Figure 6.7 Roll Control is provided via the Left and Right Ailerons as well as Spoilers #2 to #5. Spoiler #1 is used to provide braking during flight aka ‘speed braking’ and on the ground and will not be considered further. As per the previous figures, the common hydraulic system powers the control surface actuators and the ELACs and SECs provide commands. The hydraulic subsystems Green, Blue and Yellow are represented below each control surface they control respectively, represented by their initial in a circle. In this figure in particular, the Power subsystem is presented in greater detail. Two diesel generators power the computer systems, one primary and one auxiliary. In extreme cases where both generators are rendered unavailable, a Ram Air Turbine (RAT) is deployed. The RAT extends on the outside of the aircraft chassis and consists of a propeller and a wind generator. As the aircraft travels through the air, the wind current rotates the propeller. The motion generated by the wind is converted via the generator into electricity for the aircraft systems.

Finally, to provide the Pilot Feedback function from Figure 6.3, the Electronic Instrument System (EIS) is used. The EIS is shown in Figure 6.8 below. The EIS provides output into 6 Display Units (DUs) located in the cockpit. Information regarding the status of the aircraft is made available to the pilots through the DUs. There are two sets of System Data Acquisition Computers (SDACs), Flight Warning Computers (FWCs) and Display Management Computers (DMCs). The SDACs centralize information collection from the various electronic systems of the aircraft. The DMCs manage the information from the SDACs to be shown on the DUs as necessary. The FWCs monitor the SDAC data for anomalies, i.e. whether the data lies within safe thresholds. An example of this would be evaluating whether the aircraft's orientation is dangerously towards or away from the ground, either of which can lead to a hazardous situation. In case of danger being detected, the FWCs output to speakers and monitors in the cockpit in order to draw the pilots' attention to the danger. For simplicity, only the DUs have been included in Figure 6.8. The SDACs are the endpoint of data feeding in from various aircraft systems. For the FCS, the Flight Control Data Concentrators (FCDCs) are responsible for gathering the data and transmitting it to the EIS. Two FCDCs are included for redundancy.

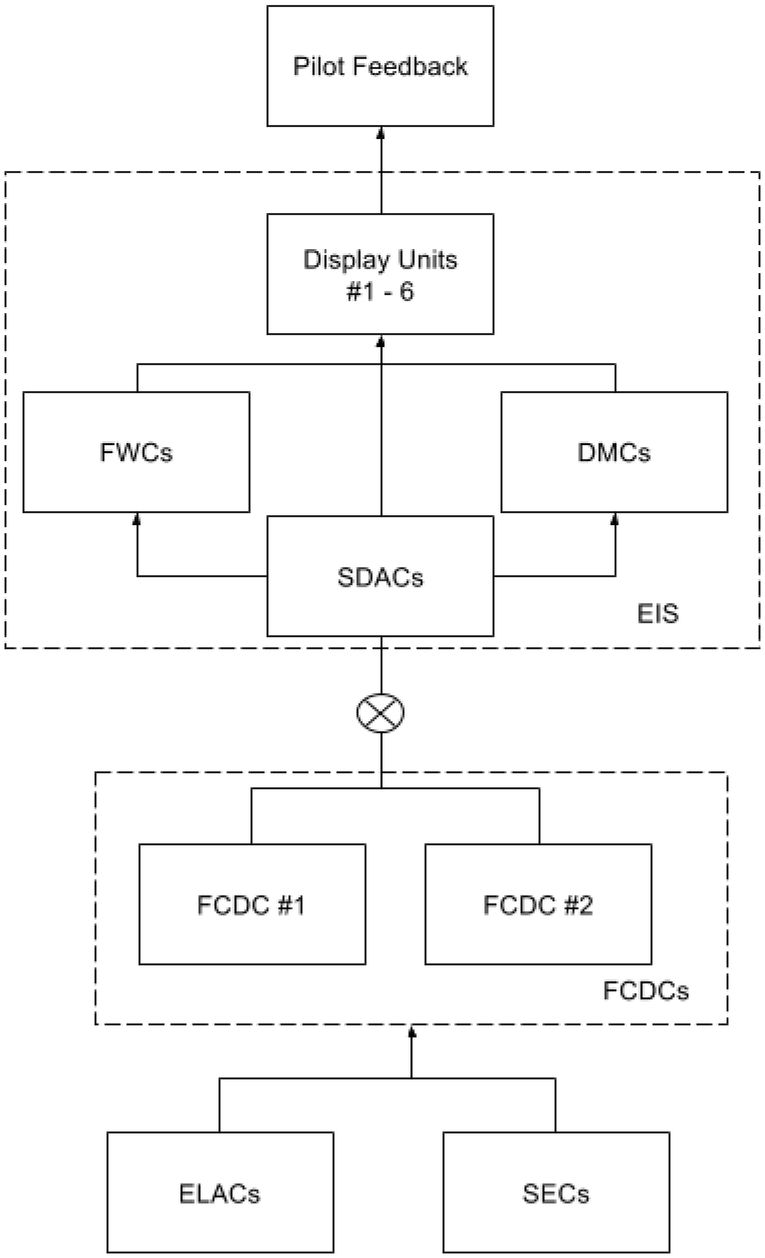


Figure 6.8 - Flight Control System – EIS

6. Functional & System Failure Analysis

The FCS was modelled in MATLAB using the Simulink extension library. The system modelling followed the description and figures in Section 5 of the current Chapter. As each system element was designed, its failure behaviour was also annotated via the HiP-HOPS plugin for MATLAB. The failure behaviour also follows the description provided in the earlier Section. Once the system modelling and failure behaviour annotation was complete, a hazard i.e. system failure needs to be defined. This allows automatic FTA and FMEA to be conducted, setting the hazard as the 'top event' of the analysis. In the case study, the two hazards from the FHA in Section 4 of the current Chapter are used.

Some instances of the annotation process will now be shown to demonstrate it. Figure 6.9 shows how the Undetected Loss of the Flight Control function from Table 6.1 is annotated. The annotation is explained in the form of a fault tree; however, this is merely for presentation purposes. In practice, each system element would have failure logic attached to it locally instead. The summary of Figure 6.9 is that an undetected loss of yaw, pitch or roll can lead to a hazardous situation for the aircraft, crew and passengers.

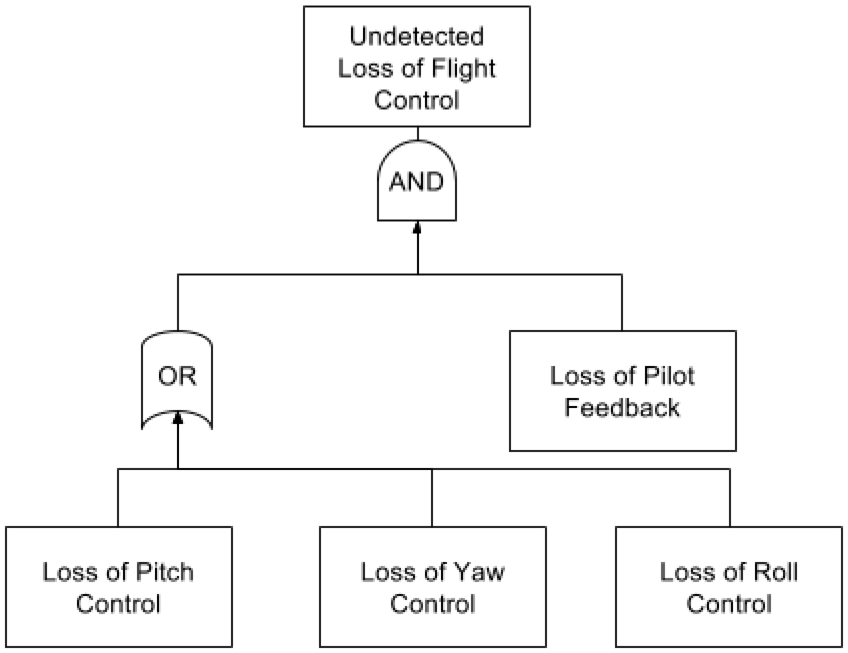


Figure 6.9 - Flight Control System - Functional Failure Analysis

Figure 6.10 shows how the loss of pitch control is annotated in the FCS. Each of the elements of the FCS contributing to the Pitch Control function are evaluated in turn to construct the fault tree.

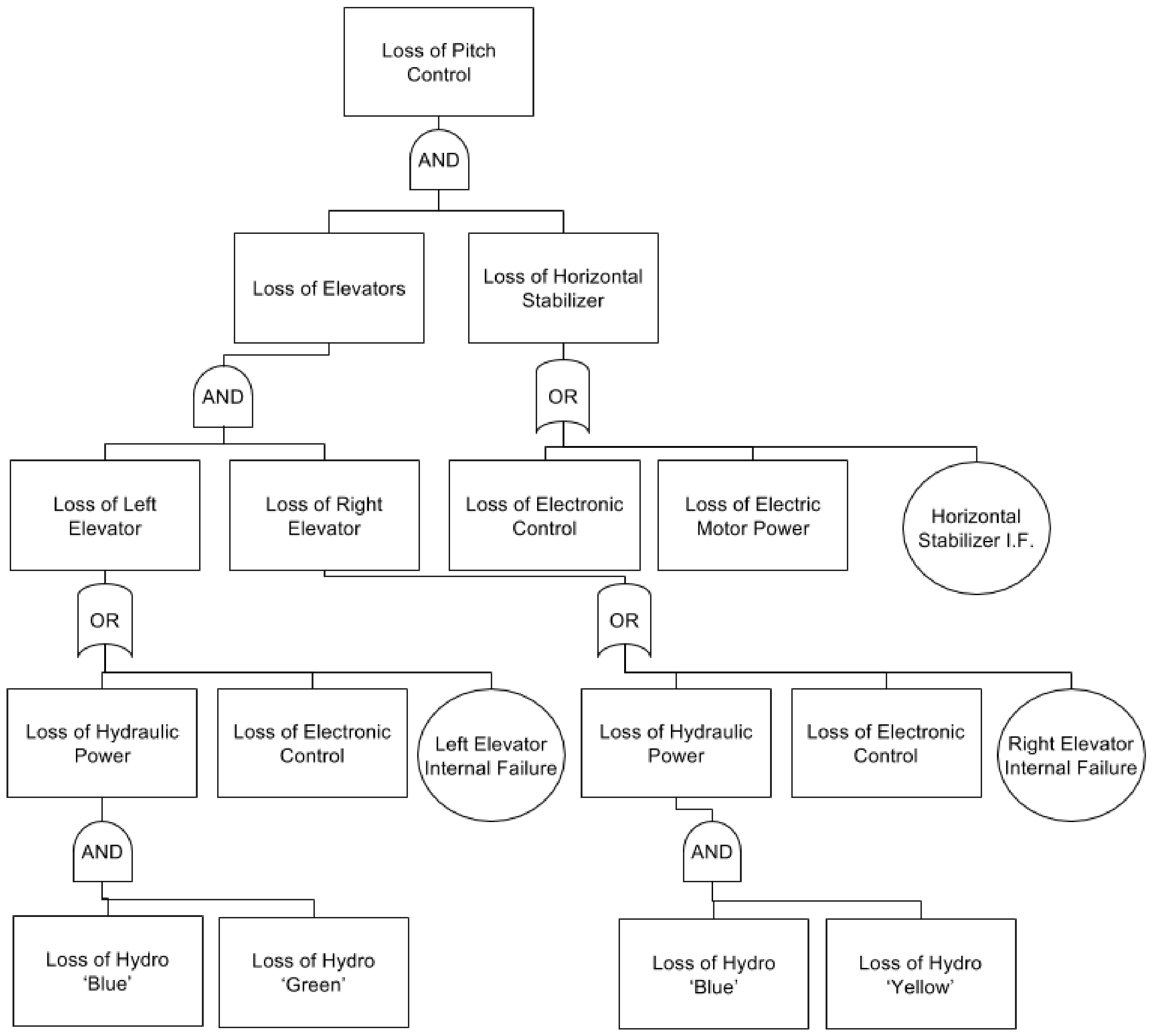


Figure 6.10 - FCS - System Failure Analysis - Loss of Pitch Control

As is shown in Figure 6.10, loss of pitch control can be caused when both elevators and the horizontal stabilizer fail. For an elevator to fail, there are three possibilities:

- Loss of both of the hydraulic systems powering the elevator
- Loss of electronic control, i.e. input from the computers
- Internal failure of the elevator. Essentially, this would imply there is some failure of the elevator actuator

Each elevator requires a different combination of hydraulic systems to fail, hence the different causes of failure are assigned to each loss of hydraulic power in the figure. Similarly, loss of electric motor power for the horizontal stabilizer would involve failure of all three of the motor generators powering it. This has not been included in the figure due to space limitations. Finally, an internal failure ("Horizontal Stabilizer I.F") can cause the horizontal stabilizer to fail independently of other causes.

Figure 6.11 shows how the Loss of Electronic Control for the Left Elevator can be annotated.

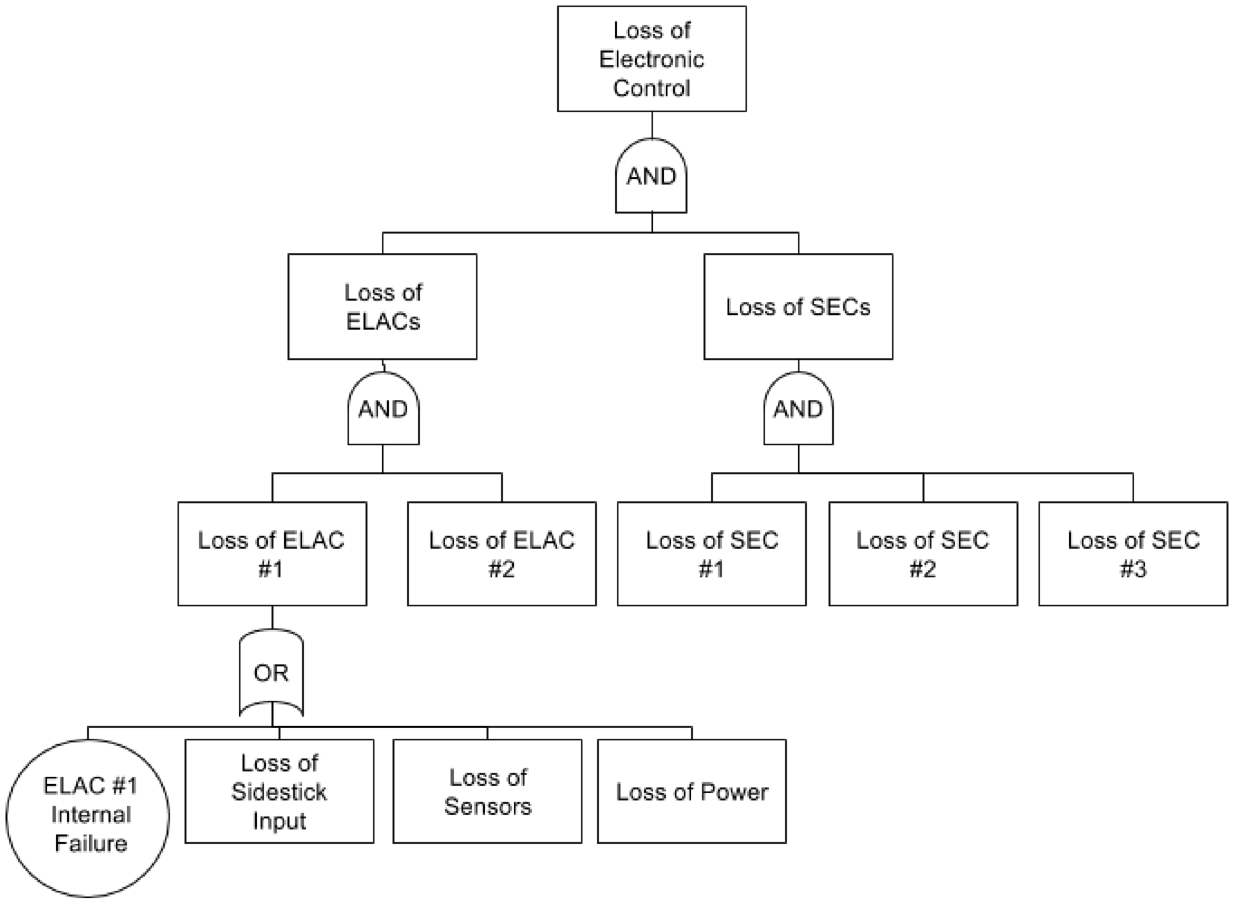


Figure 6.11 - Loss of Electronic Control of Left Elevator Fault Tree

The figure shows only part of the annotation due to space limitations, as there is a great deal of repetition across the annotated failure behaviour. Loss of electronic control implies that both computer groups controlling the elevators and horizontal stabilizer have failed. This means that all of the ELACs and SECs have failed. As mentioned before, in case of failure of a particular ELAC or SEC, another takes over following a reconfiguration sequence. However, because this particular analysis is evaluating a scenario where all electronic control has been lost, the order of failures does not matter in this case. Simply put, if all computers have failed, loss of control is inevitable regardless of the order they failed in. Each ELAC or SEC is similar regarding the causes it might fail to function:

- An internal failure of the computer
- Failure of both pilot sidesticks
- Failure of sensors (all ADIRUs, the accelerometer or the radio altimeter)
- Loss of power (primary generator, auxiliary and the ram air turbine generator)

Internal failure of each computer implies that its control channel has failed; the analysis is simplistic so the scenario where the monitoring channel has also failed will not be evaluated. Once the annotation is complete, the next step in the process can be initiated.

7. Automatic DAL Allocation

Given a complete failure behaviour annotation, as per step 5 in Section 2.1 of Chapter 5, reliability analysis via HiP-HOPS is applied. The result of the analysis is a complete fault tree for each of the hazards from step 1 of the process. Additionally, FMEA (see Section 6.5 of Chapter 2) results are available. Following the analysis, the automatic DAL allocation from Chapter 4 will be applied. The cost function for the allocation will be the same used in the example from Table 5.4. Some of the allocations produced will be presented in this Section. Figure 6.12 shows how the allocations are applied to the pitch control elements of the FCS. It should be noted that the Pilot Feedback function was assigned DAL C by the user during the allocation process. The user deemed the Flight Control sub-functions inherently more critical than the Pilot Feedback.

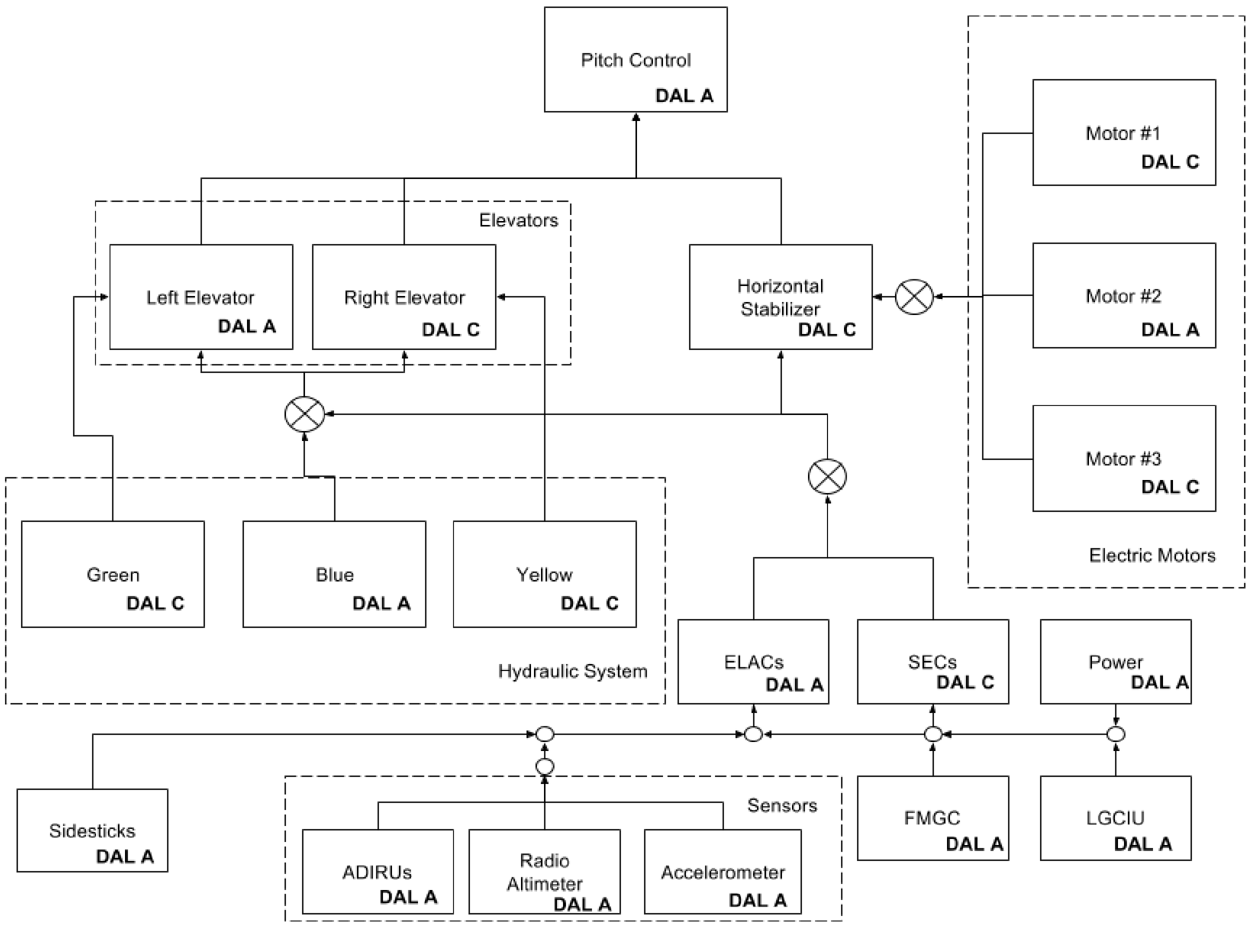


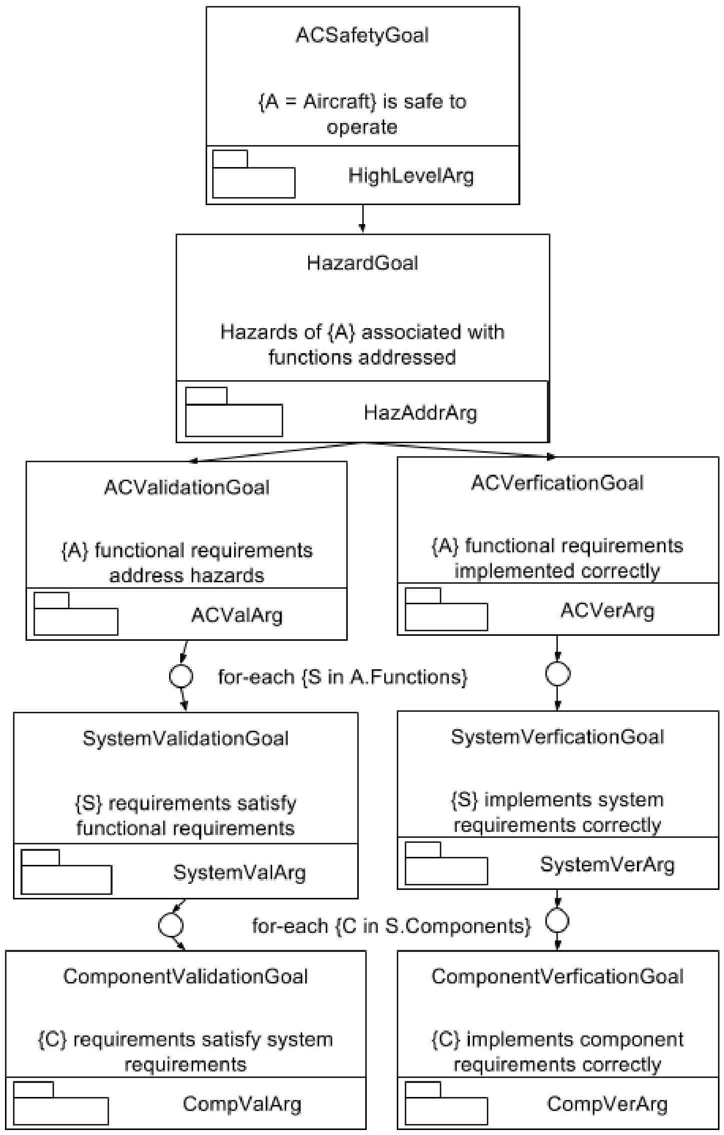
Figure 6.12 - Flight Control System - Pitch Control DAL Allocation

Although the DALs were assigned on the basis of the two hazards being analysed, the final allocation is common. The highest DAL across all hazards analysed is assigned to each architectural element. For the purposes of the case study, we will assume all of the architectural elements included feature some electronic functionality. This simplifies the model by allowing DALs to be applicable to those elements as well.

Once the DAL allocation stage is complete, the model is now ready to be used to instantiate the user-defined argument pattern and produce the desired argument structure.

8. Argument Pattern

Major parts of the argument pattern used in the case study will now be presented.



ACSafetyGoal
{A = Aircraft} is safe to operate
HighLevelArg
HazardGoal
Hazards of {A} associated with functions addressed
HazAddrArg
ACValidationGoal
{A} functional requirements address hazards
ACValArg
ACVerficationGoal
{A} functional requirements implemented correctly
ACVerArg
for-each {S in A.Functions}
SystemValidationGoal
{S} requirements satisfy functional requirements
SystemValArg
SystemVerficationGoal
{S} implements system requirements correctly
SystemVerArg
for-each {C in S.Components}
ComponentValidationGoal
{C} requirements satisfy system requirements
CompValArg
ComponentVerficationGoal
{C} implements component requirements correctly
CompVerArg

Figure 6.13 - Argument Pattern - Module Overview

Figure 6.13 shows an overview of the modules that compose the argument pattern used. The top claim ACSafetyGoal, initially establishes a claim supporting that the aircraft is safe to operate because it was developed following the safety guidelines recommended by regulatory authorities. In turn, the subsequent claim relies on the HazardGoal claim, which argues that all of the hazards of the aircraft have been addressed. HazardGoal is supported by two subgraphs, each establishing a claim that mirrors the two sides of the ‘v-model’ of system development (see Section 5.5 of Chapter 2). The left subgraph of the pattern, via claim ACValidationGoal, intends to construct an argument defending the correctness of the aircraft’s functional requirements. In other words, ACValidationGoal aims to convince that the functional requirements of the aircraft indeed safeguard acceptably against the occurrence of hazards. The right subgraph, in claim ACVerificationGoal intends to construct an argument defending the correctness of the implementation of the system’s requirements. Both the ACValidationGoal and ACVerificationGoal claims are further supported by other modules internally, which progress deeper down to lower levels of the aircraft architecture, from systems to components. Similar claims to the above are established regarding each system’s, subsystem’s and component’s requirements. Figure 6.14 presents the ACSafetyGoal module in greater detail.

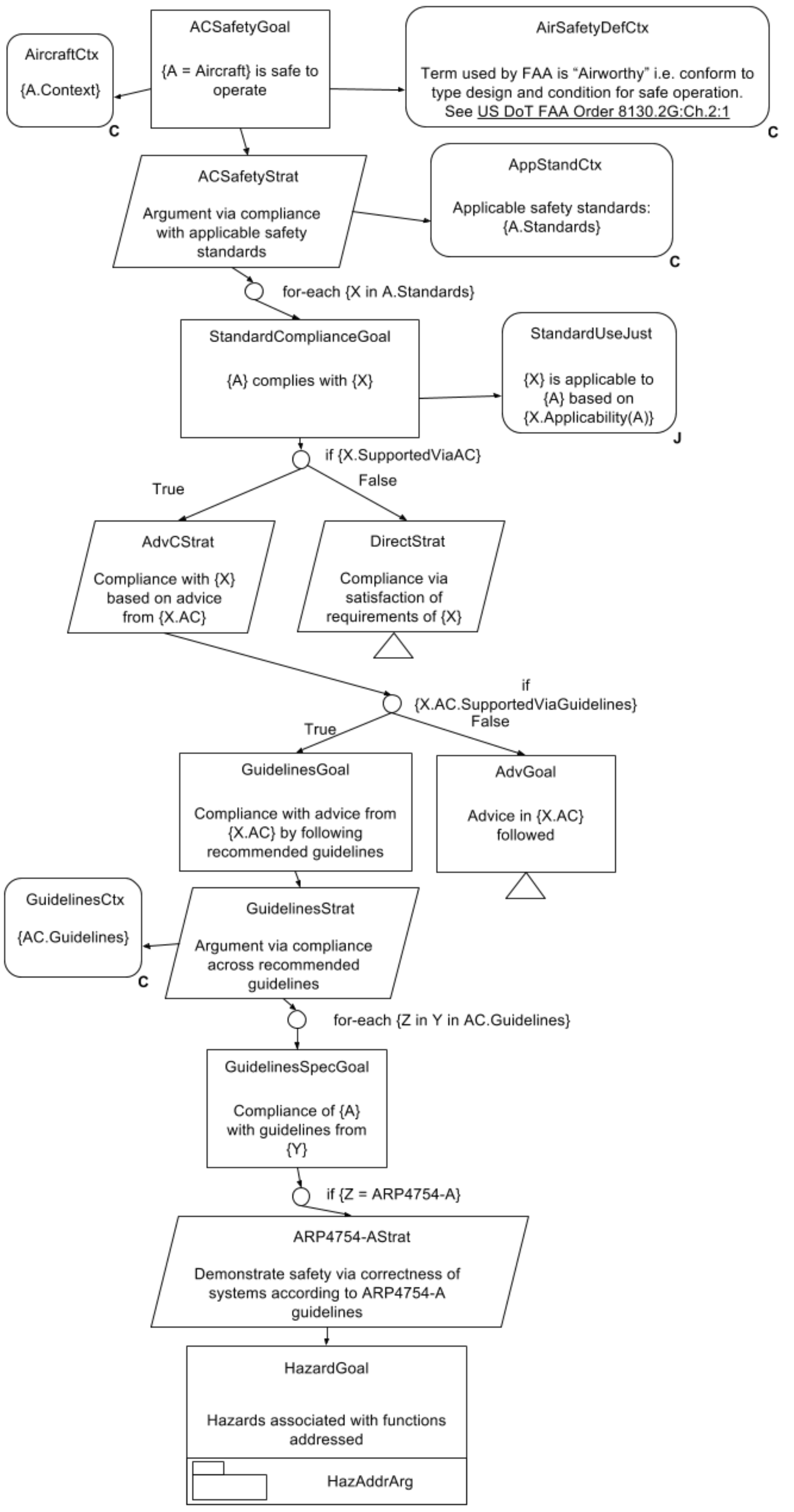

ACSafetyGoal
{A = Aircraft} is safe to operate
AircraftCtx
{A.Context}
C
AirSafetyDefCtx
Term used by FAA is "Airworthy" i.e. conform to type design and condition for safe operation. See US DoT FAA Order 8130.2G:Ch.2:1
C
ACSafetyStrat
Argument via compliance with applicable safety standards
AppStandCtx
Applicable safety standards: {A.Standards}
C
for-each {X in A.Standards}
StandardComplianceGoal
{A} complies with {X}
StandardUseJust
{X} is applicable to {A} based on {X.Applicability(A)}
J
if {X.SupportedViaAC}
True
False
AdvCStrat
Compliance with {X} based on advice from {X.AC}
DirectStrat
Compliance via satisfaction of requirements of {X}
if {X.AC.SupportedViaGuidelines}
True
False
GuidelinesGoal
Compliance with advice from {X.AC} by following recommended guidelines
AdvGoal
Advice in {X.AC} followed
GuidelinesCtx
{AC.Guidelines}
C
GuidelinesStrat
Argument via compliance across recommended guidelines
for-each {Z in Y in AC.Guidelines}
GuidelinesSpecGoal
Compliance of {A} with guidelines from {Y}
if {Z = ARP4754-A}
ARP4754-AStrat
Demonstrate safety via correctness of systems according to ARP4754-A guidelines
HazardGoal
Hazards associated with functions addressed
HazAddrArg

Figure 6.14 - Argument Pattern - ACSafetyGoal Module

In Figure 6.14 the pattern aims to construct an argument that supports the top claim (ACSafetyGoal) that the aircraft is safe to operate. AircraftCtx provides contextual information regarding the intended operational environment and role of the aircraft. The official definition of safety relevant to the regulatory authorities (assumed to be the FAA in the case study) is also provided as context in AirSafetyDefCtx. The ACSafetyGoal claim is supported by demonstrating compliance with all of the safety regulations applicable to the aircraft. The set of relevant standards might differ across different types of aircraft e.g. manned/unmanned, civil/military etc. For this reason, the pattern makes use of information embedded in the aircraft model that register which standards are applicable. AppStandCtx lists the applicable regulations for reference. The use of the for-each element following ACSafetyStrat iterates over the applicable standards and argues the compliance of the aircraft with the standard. Obviously, if the aircraft where to fail to comply with any one of the applicable standards, the top claim would be ultimately unsupported in that regard. With each applicable standard, a justification of why it is applicable to the aircraft is also provided in the StandardUseJust element of the pattern. Due to the particular nature of the 14CFR-Part 23 regulations, which are relevant to the type of aircraft developed in the case study, compliance can be reached by following the relevant Advisory Circular (AC) (see Section 5.3 of Chapter 2). If compliance with the standard were to be demonstrated without involving the AC, the DirectStrat subgraph would be followed. For the case study's purposes, the AdvCStrat will be the one chosen during instantiation, as the AC advises using the guidelines of ARP4754-A to comply with the regulations. AdvGoal would be instantiated for cases where the advice in the AC did not involve particular guidelines documents and could be supported differently. In the case study, GuidelinesGoal will be instantiated, establishing the claim that in order to comply with the regulations, the recommended guidelines are sufficient. GuidelinesCtx lists the applicable guideline documents for reference. GuidelinesStrat supports the GuidelinesGoal claim by iterating over each applicable guideline document. For the case study, this would be the ARP4754-A document. For each of the guidelines documents that are applicable, the corresponding GuidelinesSpecGoal claim would then explain how the aircraft development followed the guidelines. A conditional specialization is then used to implement support for the claim in the case of ARP4754-A. In the general case, other guidelines documents that are applicable would also have to have a conditional check at the same position in the pattern and an appropriate argument strategy supporting them. Finally, the HazardGoal is a reference to a claim in another module establishing that all of the hazards associated with the aircraft's functions have been addressed. Figure 6.15 shows how the HazAddrArg supporting the former claim is structured.

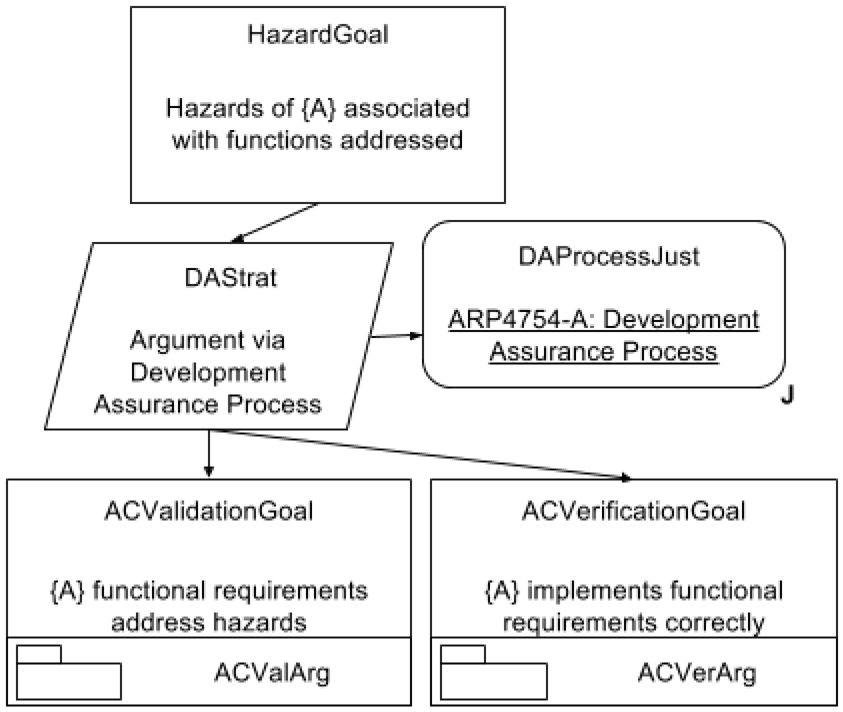

HazardGoal
Hazards of {A} associated with functions addressed
DAStrat
Argument via Development Assurance Process
DAProcessJust
ARP4754-A: Development Assurance Process
J
ACValidationGoal
{A} functional requirements address hazards
ACValArg
ACVerificationGoal
{A} implements functional requirements correctly
ACVerArg

Figure 6.15 - Argument Pattern - HazAddrArg

HazAddrArg is relatively simple; its purpose is to support the HazardGoal claim that all functional hazards have been addressed. To do so, the central process of ARP4754-A, the Development Assurance Process is employed. The two primary sides of the process, the validation (ACValidationGoal) and verification (ACVerificationGoal) of requirements are used to support the claim. Each claim is further supported in its respective module, ACValArg and ACVerArg. Figure 6.16 presents the ACValArg module in detail.

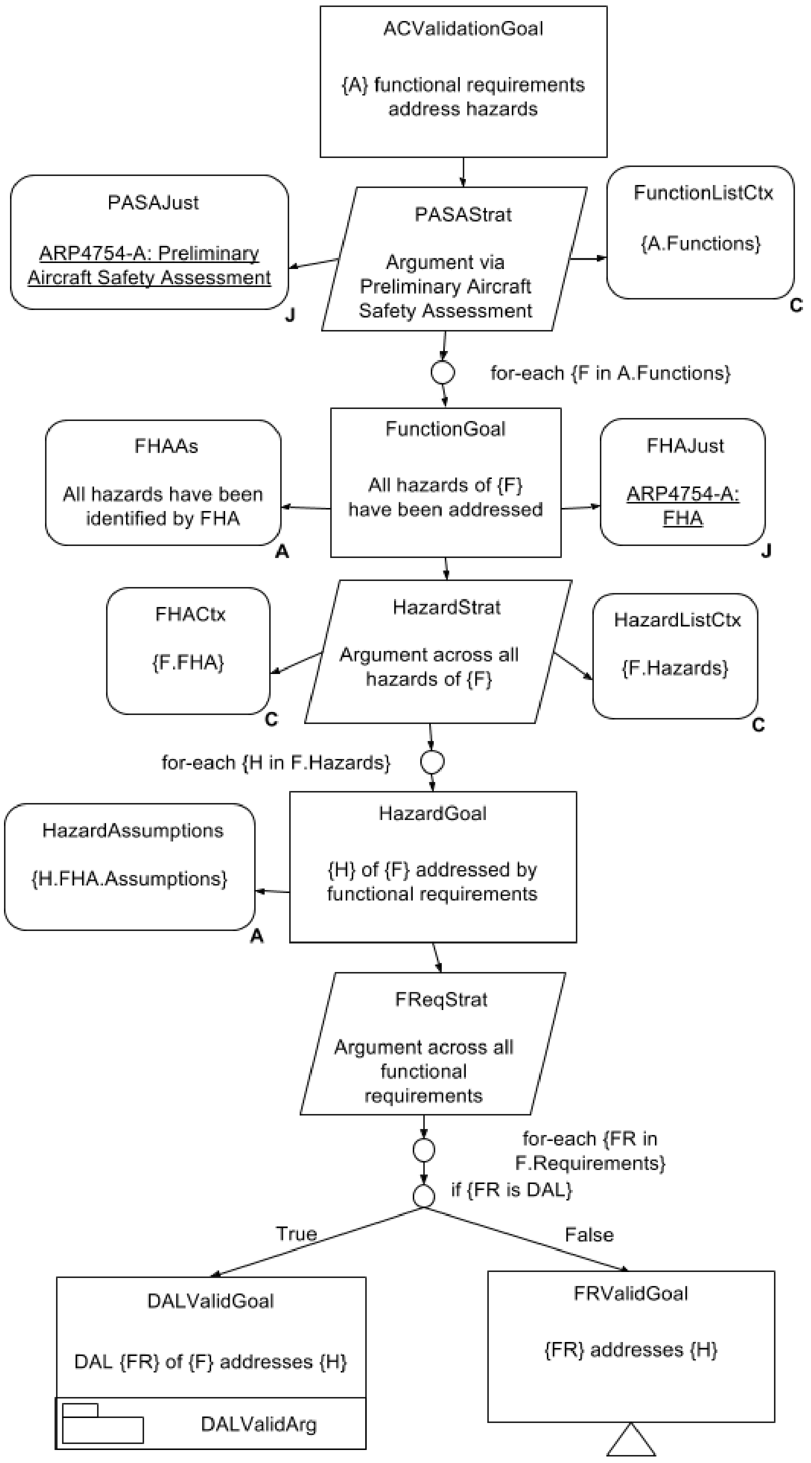

ACValidationGoal
{A} functional requirements address hazards
PASAJust
ARP4754-A: Preliminary Aircraft Safety Assessment
J
PASAStrat
Argument via Preliminary Aircraft Safety Assessment
FunctionListCtx
{A.Functions}
C
for-each {F in A.Functions}
FHAAs
All hazards have been identified by FHA
A
FunctionGoal
All hazards of {F} have been addressed
FHAJust
ARP4754-A: FHA
J
FHACtx
{F.FHA}
C
HazardStrat
Argument across all hazards of {F}
HazardListCtx
{F.Hazards}
C
for-each {H in F.Hazards}
HazardAssumptions
{H.FHA.Assumptions}
A
HazardGoal
{H} of {F} addressed by functional requirements
FReqStrat
Argument across all functional requirements
for-each {FR in F.Requirements}
if {FR is DAL}
True
False
DALValidGoal
DAL {FR} of {F} addresses {H}
DALValidArg
FRValidGoal
{FR} addresses {H}

Figure 6.16 - Argument Pattern - ACValArg

ACValArg essentially addresses the analysis of functions for hazards via the FHA (see Section 5.5 of Chapter 2). ACValidationGoal is the top claim, arguing that the functional requirements identified are sufficient to address all hazards. An underlying assumption is that all hazards were identified by the FHA, as seen in the FHAAs assumption. To support this claim, subclaims are made across each hazard of each function. This is seen in FunctionStrat, which iterates across all functions, and HazardStrat, which iterates across each hazard. Given that the standard approach to analysing for hazards based on the ARP4754-A is the FHA, the relevant guidance is provided via the FHAJust justification. Additionally, the results of the function's corresponding FHA are also provided in FHACtx. Multiple FHAs can be conducted over the course of development so it is important to maintain the relationship of each function with its latest FHA. This information is accessed via the {F.FHA} instantiation in FHACtx. Each hazard denotes the assumptions associated with its identification in HazardAssumptions. A claim that the functional requirements defined for the aircraft address the hazard is established in HazardGoal. The claim in HazardGoal is supported by FReqStrat, by iterating over the functional requirements associated with the function and hazard. A distinction is made at this point; for DALs, the DALValidGoal claim is made, referencing the DALValidArg module. For other kinds of requirements, the undeveloped FRValidGoal would support their validity. Figure 6.17 presents the DALValidArg module in detail.

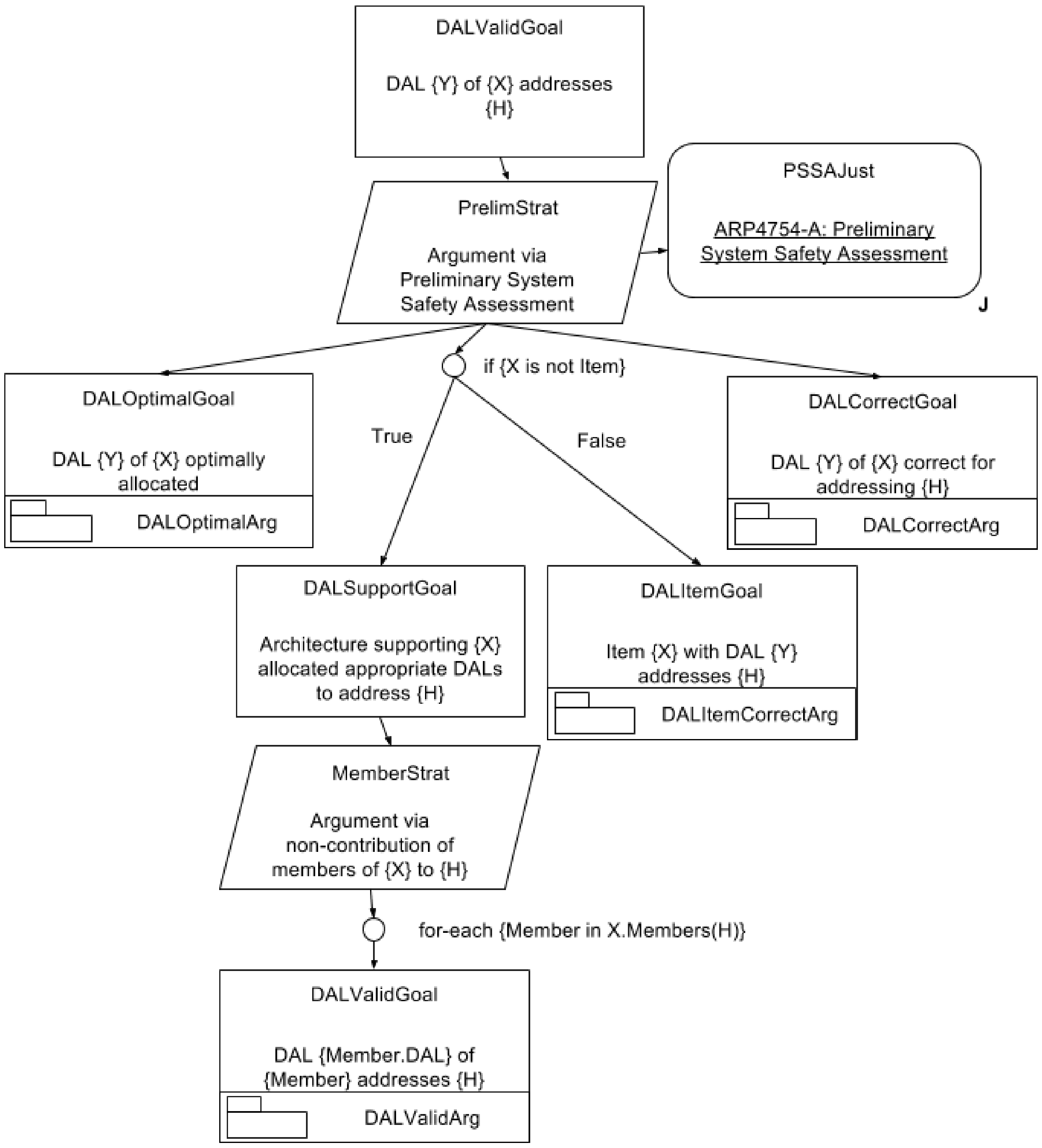

DALValidGoal
DAL {Y} of {X} addresses {H}
PrelimStrat
Argument via Preliminary System Safety Assessment
PSSAJust
ARP4754-A: Preliminary System Safety Assessment
J
if {X is not Item}
True
False
DALOptimalGoal
DAL {Y} of {X} optimally allocated
DALOptimalArg
DALCorrectGoal
DAL {Y} of {X} correct for addressing {H}
DALCorrectArg
DALSupportGoal
Architecture supporting {X} allocated appropriate DALs to address {H}
DALItemGoal
Item {X} with DAL {Y} addresses {H}
DALItemCorrectArg
MemberStrat
Argument via non-contribution of members of {X} to {H}
for-each {Member in X.Members(H)}
DALValidGoal
DAL {Member.DAL} of {Member} addresses {H}
DALValidArg

Figure 6.17 - Argument Pattern - DALValidArg

DALValidArg begins by establishing a claim in DALValidGoal that the DAL of an architectural element is appropriate for a given hazard. The DAL, the element and the hazard are essentially provided as inputs to the module. Based on the architectural level the element belongs to (function, system or component), the corresponding safety assessment process strategy is employed. The Preliminary System Safety Assessment (PSSA) analyses the potential causes of failure that can potentially lead to hazards. This is reflected in the PrelimStrat strategy, coupled with a justification that references the respective ARP4754-A guidance for that process. The pattern argues why the chosen allocation of DAL for the given element is correct, via the DALCorrectGoal, and optimal, via the DALOptimalGoal. For components, the claim DALItemGoal argues that their development meets the standard for their allocated DAL. For functions and systems, the underlying architecture is shown to support the allocated DAL towards addressing the hazard, via the DALSupportGoal. The DALOptimalGoal claim is essentially identical to the argument regarding the optimality of the DAL allocation presented in Section 2.8 of Chapter 5. Finally, the DALSupportGoal iterates through MemberStrat over each subsystem or component of the current element and establishes another DALValidGoal. This makes the pattern recursive, applicable over progressively lower levels of the system's architecture hierarchy. When components are encountered, the recursion halts, thanks to the relevant if condition preceding the DALSupportGoal. The DALItemGoal referenced in the DALItemCorrectArg module covers issues that extend beyond the purview of ARP4754-A. Depending on the component in question, one of the supporting documents to the standard should provide appropriate guidance for component validation and verification activities. The DO-178C addresses software components and efforts to develop explicit argument structures based on it have been described in (Holloway, 2015).

The DALOptimalArg module can be seen in Figure 6.18. Given a DAL allocation and the corresponding architectural element as input, it constructs an argument defending the optimality of the allocation as per Section 2.8 of Chapter 5. If alternative methods of determining the optimal allocation where employed, the pattern can easily be modified to reflect such changes.

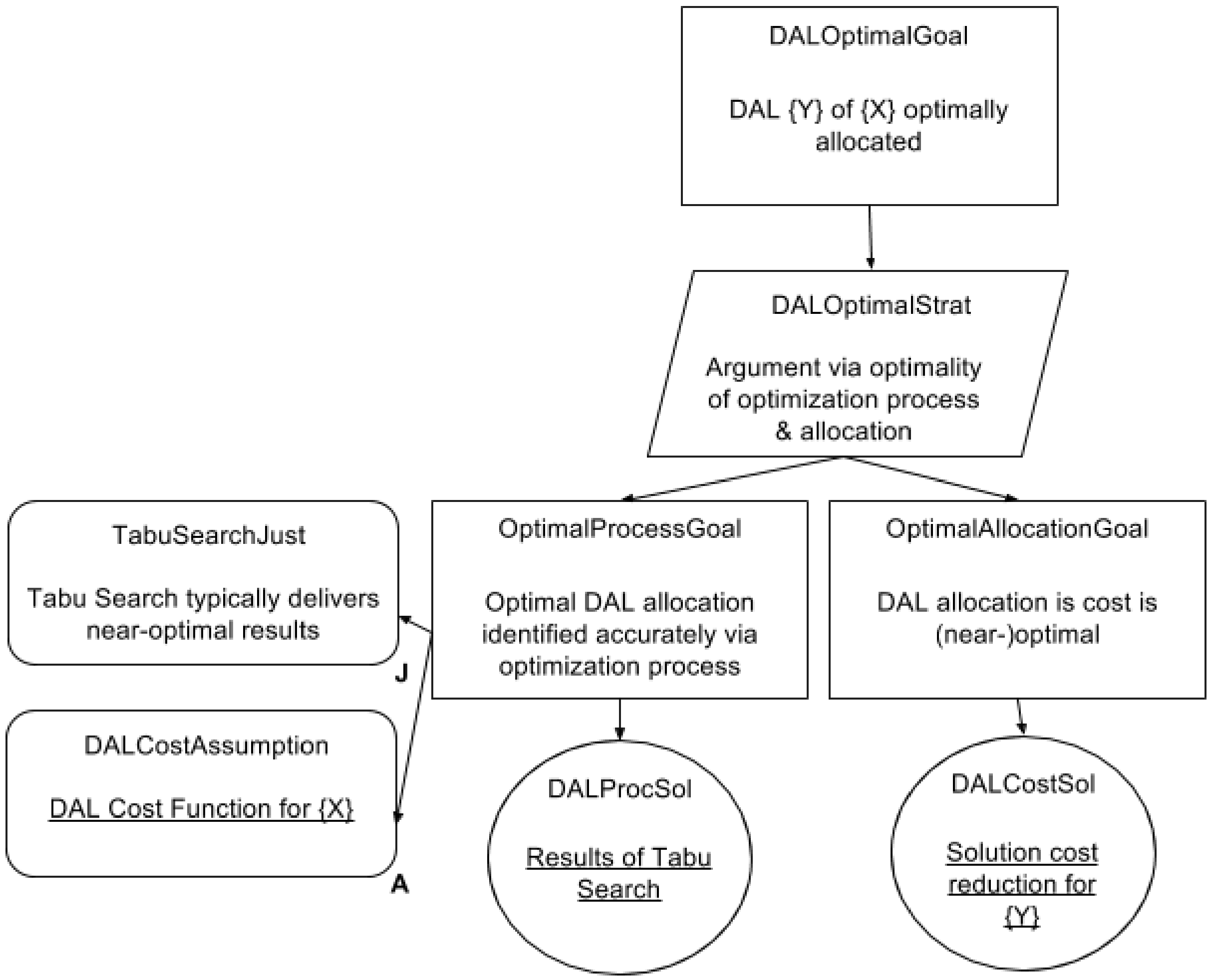

DALOptimalGoal
DAL {Y} of {X} optimally allocated
DALOptimalStrat
Argument via optimality of optimization process & allocation
TabuSearchJust
Tabu Search typically delivers near-optimal results
J
OptimalProcessGoal
Optimal DAL allocation identified accurately via optimization process
OptimalAllocationGoal
DAL allocation is cost is (near-)optimal
DALCostAssumption
DAL Cost Function for {X}
A
DALProcSol
Results of Tabu Search
DALCostSol
Solution cost reduction for {Y}

Figure 6.18 - Argument Pattern – DALOptimalArg

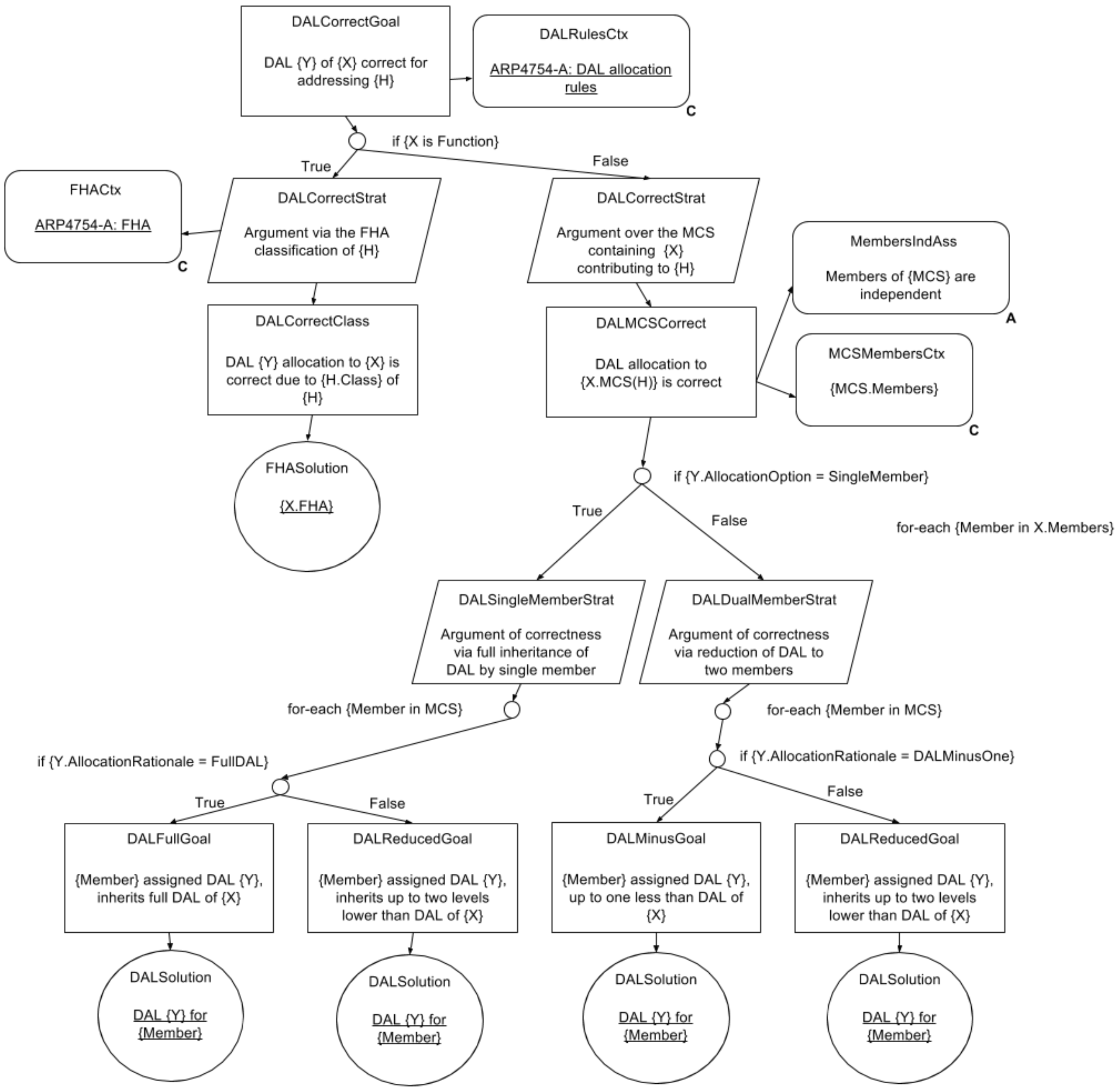

DALCorrectGoal
DAL {Y} of {X} correct for addressing {H}
DALRulesCtx
ARP4754-A: DAL allocation rules
C
if {X is Function}
True
False
FHACtx
ARP4754-A: FHA
C
DALCorrectStrat
Argument via the FHA classification of {H}
DALCorrectStrat
Argument over the MCS containing {X} contributing to {H}
MembersIndAss
Members of {MCS} are independent
A
DALCorrectClass
DAL {Y} allocation to {X} is correct due to {H.Class} of {H}
DALMCSCorrect
DAL allocation to {X.MCS(H)} is correct
MCSMembersCtx
{MCS.Members}
C
FHASolution
{X.FHA}
if {Y.AllocationOption = SingleMember}
True
False
for-each {Member in X.Members}
DALSingleMemberStrat
Argument of correctness via full inheritance of DAL by single member
DALDualMemberStrat
Argument of correctness via reduction of DAL to two members
for-each {Member in MCS}
for-each {Member in MCS}
if {Y.AllocationRationale = FullDAL}
if {Y.AllocationRationale = DALMinusOne}
True
False
True
False
DALFullGoal
{Member} assigned DAL {Y}, inherits full DAL of {X}
DALReducedGoal
{Member} assigned DAL {Y}, inherits up to two levels lower than DAL of {X}
DALMinusGoal
{Member} assigned DAL {Y}, up to one less than DAL of {X}
DALReducedGoal
{Member} assigned DAL {Y}, inherits up to two levels lower than DAL of {X}
DALSolution
DAL {Y} for {Member}
DALSolution
DAL {Y} for {Member}
DALSolution
DAL {Y} for {Member}
DALSolution
DAL {Y} for {Member}

Figure 6.19 - Argument Pattern - DALCorrectArg

On the other hand, the DALCorrectArg in Figure 6.19 is a more detailed argument pattern and will be discussed further. To explain why assigning the given DAL for the architectural element is correct in DALCorrectGoal, the DAL allocation rules of ARP4754-A are provided as reference in DALRulesCtx. To support the DALCorrectGoal, the MCS responsible for allocating the DAL of the element is referenced (see Section 2 of Chapter 3) in DALCorrectStrat. To locate the corresponding MCS, a relationship property of element X is used in the DALMCSCorrect claim. Specifically, {X.MCS(H)} locates the MCS, of which X is a member, which contributes to hazard H. It then can be included in the DALMCSCorrect claim, which argues that the DAL allocation with regards to the MCS is correct. The members of the MCS are listed as context in MCSMembersCtx. The assumption that the members are independent between them is reiterated in MembersIndAss. Next, depending on which allocation strategy was selected during the optimization stage for the MCS members, one of two strategies are selected to support the DALMCSCorrect claim. The DALSingleMemberStrat is selected when one member of the MCS inherits (up to) the full DAL of the parent element. Otherwise, when two members inherit (up to) one less than the DAL of the parent element, the DALDualMemberStrat is selected. Each strategy then iterates over the members of the MCS. For each member, the appropriate claim is established depending on whether it inherits (up to) the full, one less or two less than the parent element DAL.

Figure 6.20 shows details of the ACVerArg module. In ACVerificationGoal, the pattern uses the aircraft model as input to argue that all of the functional requirements are met by the implementation. Similar to ACValArg, the corresponding safety verification assessment is used as the argument strategy ASAStrat to support the ACVerificationGoal claim. Verification via the Aircraft Safety Assessment process involves confirming that all of the components, systems and functions implement correctly the requirements identified for them earlier. Thus, ASAStrat iterates over each aircraft function and then FReqStrat over each function’s requirements. For DAL requirements, the DALVerifiedGoal claim explains how the DALs of functions are met. For other kinds of requirements, FRVerifiedGoal would expand upon their implementation.

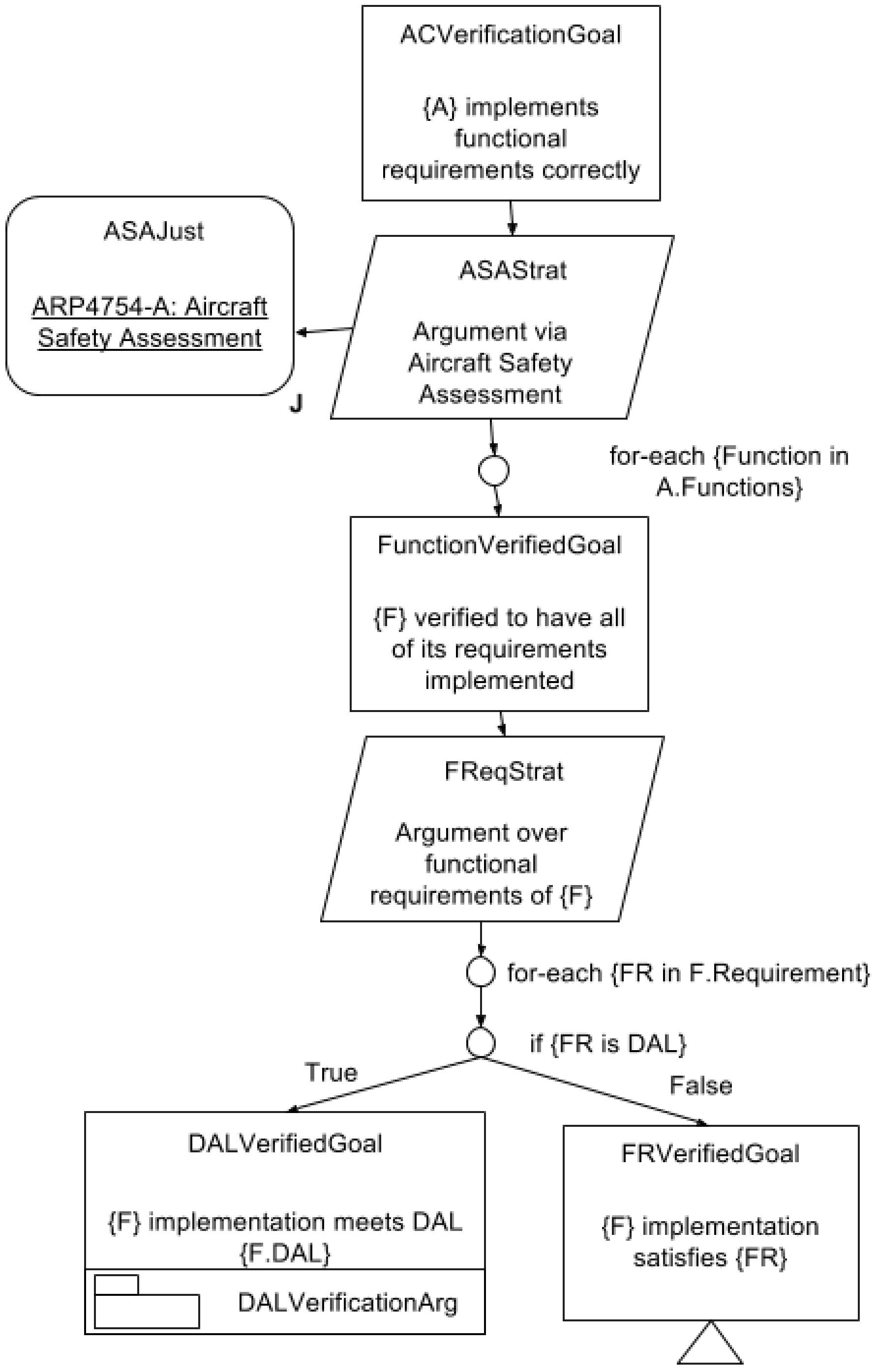
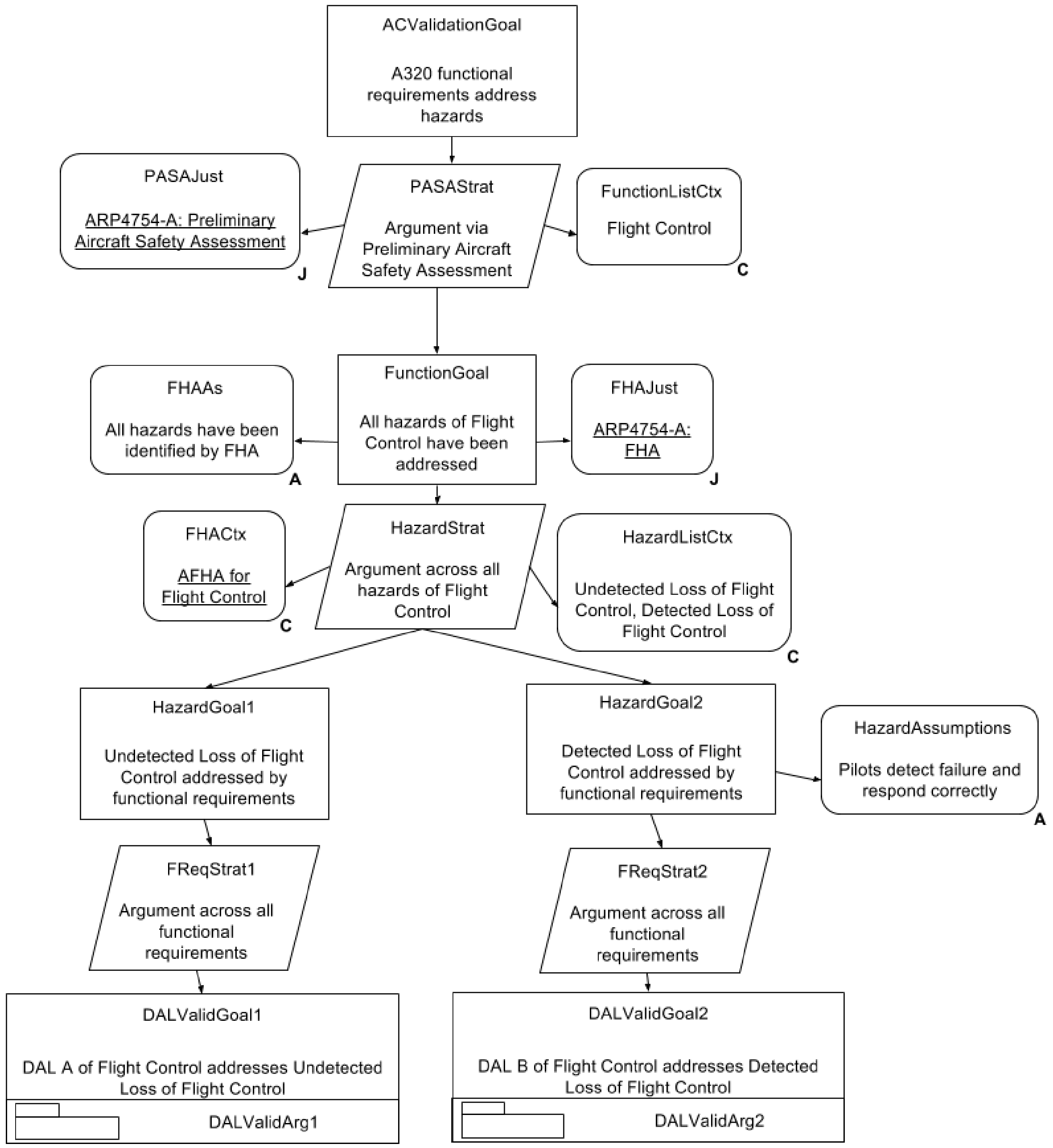

ACVerificationGoal
{A} implements functional requirements correctly
ASAJust
ARP4754-A: Aircraft Safety Assessment
J
ASAStrat
Argument via Aircraft Safety Assessment
for-each {Function in A.Functions}
FunctionVerifiedGoal
{F} verified to have all of its requirements implemented
FReqStrat
Argument over functional requirements of {F}
for-each {FR in F.Requirement}
if {FR is DAL}
True
False
DALVerifiedGoal
{F} implementation meets DAL {F.DAL}
DALVerificationArg
FRVerifiedGoal
{F} implementation satisfies {FR}

Figure 6.20 - Argument Pattern – ACVerArg

With the argument pattern developed, the final stage of the argument generation process can be applied. This involves providing the system architecture, fault tree analysis, DAL allocation and argument pattern information to the instantiation algorithm.

9. Produced Argument Structure

Some excerpts of the argument structure generated by the instantiation stage will now be presented and discussed. Figure 6.21 shows the resulting claim validating the (hypothetical) A320's functional requirements.



ACValidationGoal
A320 functional requirements address hazards
PASAJust
ARP4754-A: Preliminary Aircraft Safety Assessment
J
PASAStrat
Argument via Preliminary Aircraft Safety Assessment
FunctionListCtx
Flight Control
C
FHAAs
All hazards have been identified by FHA
A
FunctionGoal
All hazards of Flight Control have been addressed
FHAJust
ARP4754-A: FHA
J
FHACtx
AFHA for Flight Control
C
HazardStrat
Argument across all hazards of Flight Control
HazardListCtx
Undetected Loss of Flight Control, Detected Loss of Flight Control
C
HazardGoal1
Undetected Loss of Flight Control addressed by functional requirements
HazardGoal2
Detected Loss of Flight Control addressed by functional requirements
HazardAssumptions
Pilots detect failure and respond correctly
A
FReqStrat1
Argument across all functional requirements
FReqStrat2
Argument across all functional requirements
DALValidGoal1
DAL A of Flight Control addresses Undetected Loss of Flight Control
DALValidArg1
DALValidGoal2
DAL B of Flight Control addresses Detected Loss of Flight Control
DALValidArg2

Figure 6.21 - Argument Structure – A320 ACValidArg

The validity of SEC #1's DAL allocation claim is shown in Figure 6.22.

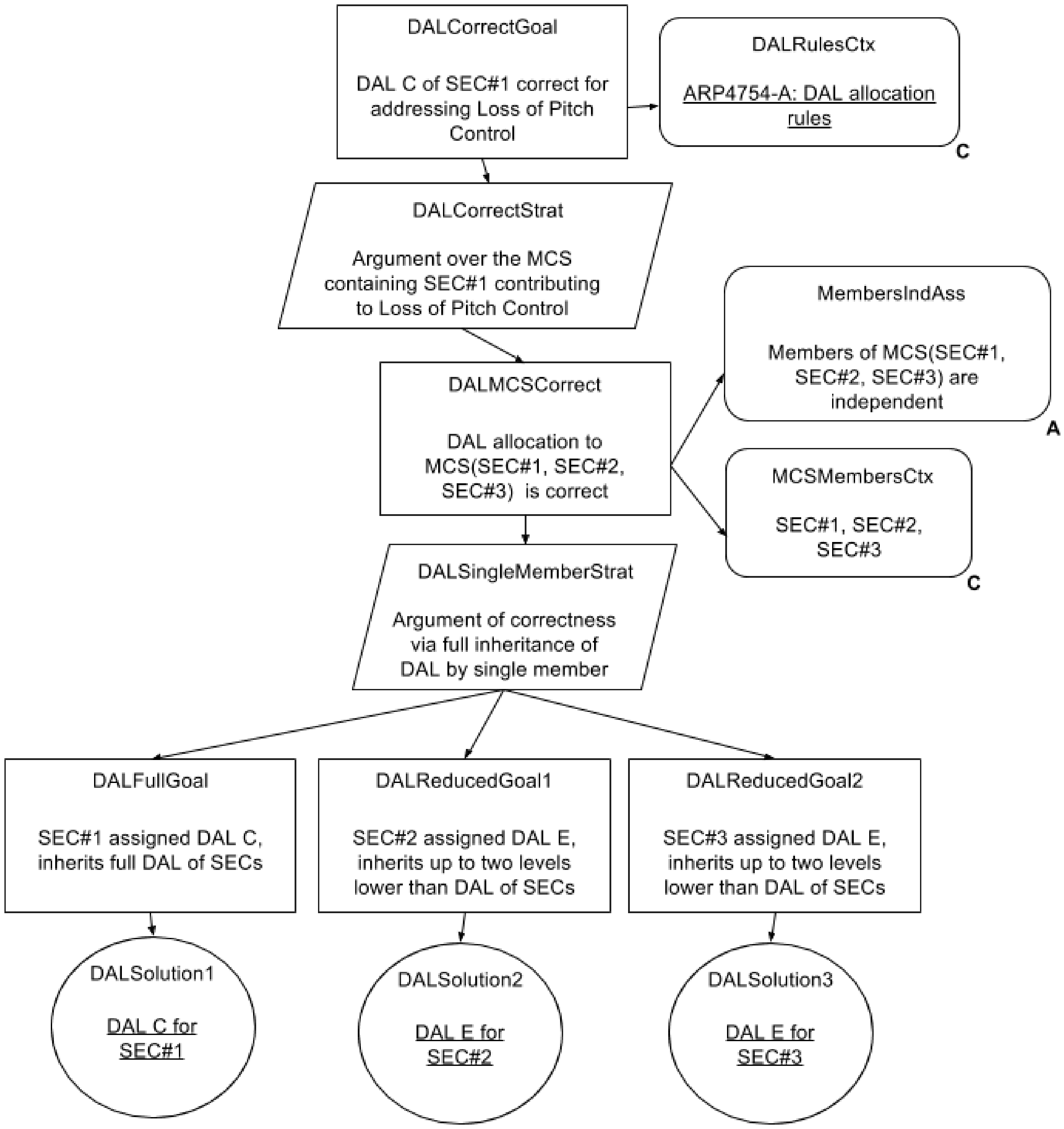

DALCorrectGoal
DAL C of SEC#1 correct for addressing Loss of Pitch Control
DALRulesCtx
ARP4754-A: DAL allocation rules
C
DALCorrectStrat
Argument over the MCS containing SEC#1 contributing to Loss of Pitch Control
MembersIndAss
Members of MCS(SEC#1, SEC#2, SEC#3) are independent
A
DALMCSCorrect
DAL allocation to MCS(SEC#1, SEC#2, SEC#3) is correct
MCSMembersCtx
SEC#1, SEC#2, SEC#3
C
DALSingleMemberStrat
Argument of correctness via full inheritance of DAL by single member
DALFullGoal
SEC#1 assigned DAL C, inherits full DAL of SECs
DALReducedGoal1
SEC#2 assigned DAL E, inherits up to two levels lower than DAL of SECs
DALReducedGoal2
SEC#3 assigned DAL E, inherits up to two levels lower than DAL of SECs
DALSolution1
DAL C for SEC#1
DALSolution2
DAL E for SEC#2
DALSolution3
DAL E for SEC#3

Figure 6.22 - Argument Structure - SEC#1 DALCorrectArg

10. Incorporating Change

As explained earlier in Section 5.5 of Chapter 2, the development lifecycle in ARP4754-A presumes many stages of iterative design, implementation and verification. Safety assessment processes are also meant to be repeated as new information and change is applied to the design. Derived requirements can precipitate further changes originating from lower levels of the architecture. Given all of these sources of dynamism, it is in the interest of system developers to anticipate such change and manage the safety assessment and assurance process accordingly. In accordance to this mentality, it is important to evaluate how well the safety argument generation method handles changes over the course of development. We will focus on two particular types of change, architectural and knowledge change.

10.1. Architectural Change

By architectural change, we refer to modification of the system architecture and its elements. This can include the introduction of new elements, removal of previously existing ones, rerouting of functionality, altered interfaces and similar events that change the system. For the purposes of the case study, an architectural change will be introduced to the model. The effects of the change will be evaluated by recreating the safety argument structure of the FCS and comparing the differences with the previous argument.

Let us assume that over the course of development, it is determined that an additional SEC is needed to bear the load of managing spoilers and control surfaces. The presence of an additional SEC can act as a source of risk mitigation via redundancy. However, it can also act as an additional source of risk on its own. In either case, the change precipitates a reappraisal of the system's safety. For a system developed traditionally, this would mean reapplying the DAL allocation process manually and then incorporating the results into the safety case. By applying the proposed method instead, all that is required is for the updated model and failure behaviour to be provided by the user. Figure 6.23 shows how the change is reflected in the updated argument structure regarding the validity of the SEC#4's DAL allocation.

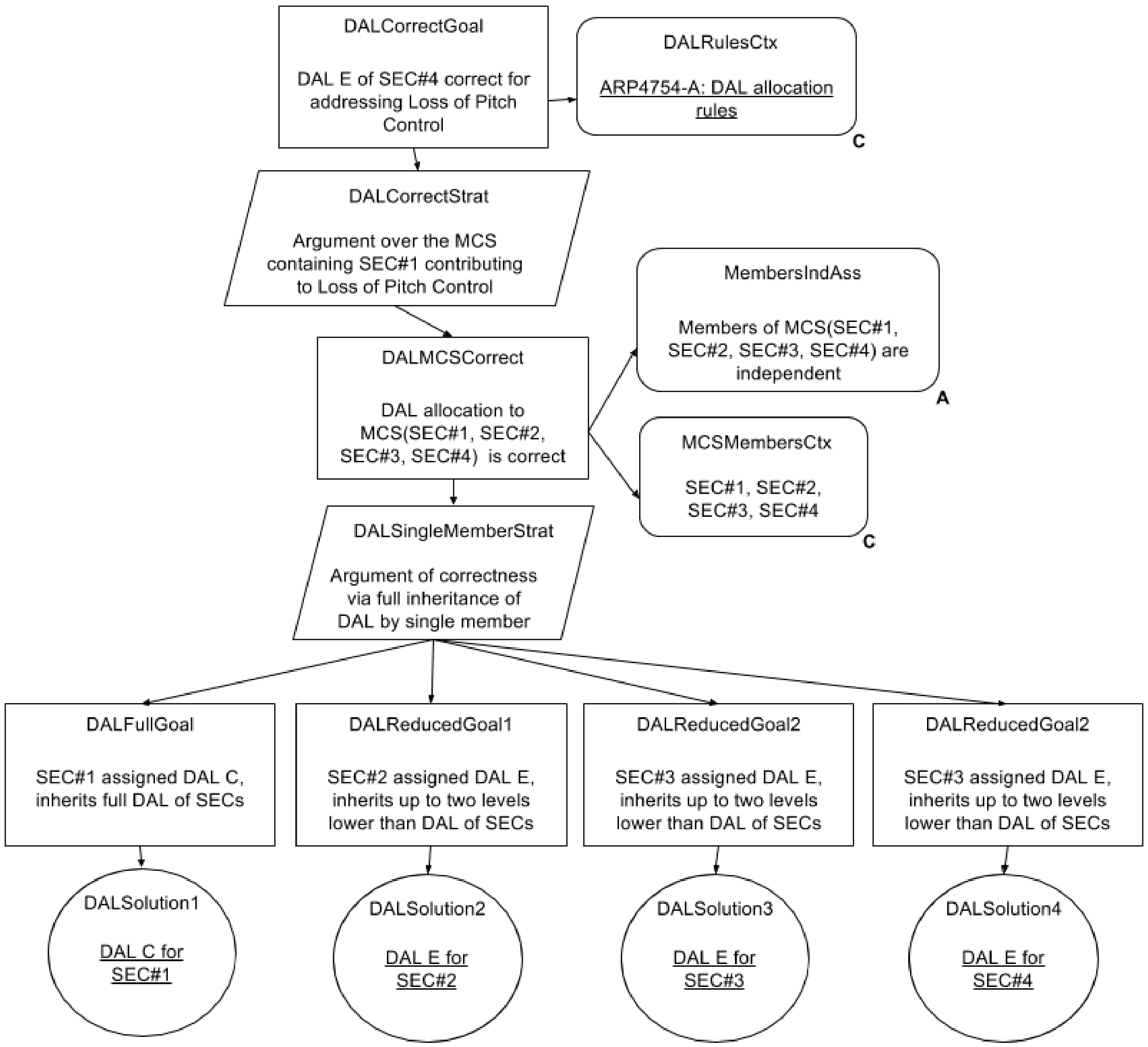

DALCorrectGoal
DAL E of SEC#4 correct for addressing Loss of Pitch Control
DALRulesCtx
ARP4754-A: DAL allocation rules
C
DALCorrectStrat
Argument over the MCS containing SEC#1 contributing to Loss of Pitch Control
MembersIndAss
Members of MCS(SEC#1, SEC#2, SEC#3, SEC#4) are independent
A
DALMCSCorrect
DAL allocation to MCS(SEC#1, SEC#2, SEC#3, SEC#4) is correct
MCSMembersCtx
SEC#1, SEC#2, SEC#3, SEC#4
C
DALSingleMemberStrat
Argument of correctness via full inheritance of DAL by single member
DALFullGoal
SEC#1 assigned DAL C, inherits full DAL of SECs
DALReducedGoal1
SEC#2 assigned DAL E, inherits up to two levels lower than DAL of SECs
DALReducedGoal2
SEC#3 assigned DAL E, inherits up to two levels lower than DAL of SECs
DALReducedGoal2
SEC#3 assigned DAL E, inherits up to two levels lower than DAL of SECs
DALSolution1
DAL C for SEC#1
DALSolution2
DAL E for SEC#2
DALSolution3
DAL E for SEC#3
DALSolution4
DAL E for SEC#4

Figure 6.23 - Modified Argument Structure for SEC#4

10.2. Knowledge Change

By knowledge change, we refer to new information becoming available to the safety assurance process. For instance, an assumption upon which a claim was based early in development, might eventually be invalidated due to testing. The information that the assumption is now invalid should trigger changes to the assurance strategy employed leading to new claims replacing the currently unsupported ones. It is important to ensure that errors are not introduced as the new information is disseminated across the various processes. The impact this kind of change can have on the safety case depends on the extent to which the argument structure construction relied on the modified information.

For example, during the verification stages, the potential of a common error across SECs is identified. Specifically, all SECs are found to include an identical memory module. Should a software or hardware error be present in this module, it can potentially trigger a failure across all SECs simultaneously. This discovery would presumably present sufficient cause to alter the safety case assumption regarding the independence between the SECs. In a system developed using traditional methods, this would require re-evaluating all arguments involving the SECs independence assumption. Following the reappraisal of assumption, the developers decide to alter the failure behaviour model to incorporate the potential of failure due to the common memory module. Normally, this would require repeating the DAL allocation process for the affected parts of the architecture and reflecting the changes in the safety case. New requirements might be identified to mitigate risk, requiring additional validation and verification. Finally, all of this change should be reflected in the safety case. All of these activities, when applied manually, are potential sources of risk. Instead, if the process described in Chapter 5 is applied, only the failure behaviour changes need to be provided by the user to produce the required argument structure.

In Figure 6.24, the updated failure behaviour regarding the potential failure of the SECs is shown. Earlier in Figure 6.11, Loss of Electronic Control could be triggered by loss of all SECs, among other reasons. Now, the failure of the memory module across all SECs is an additional possibility.

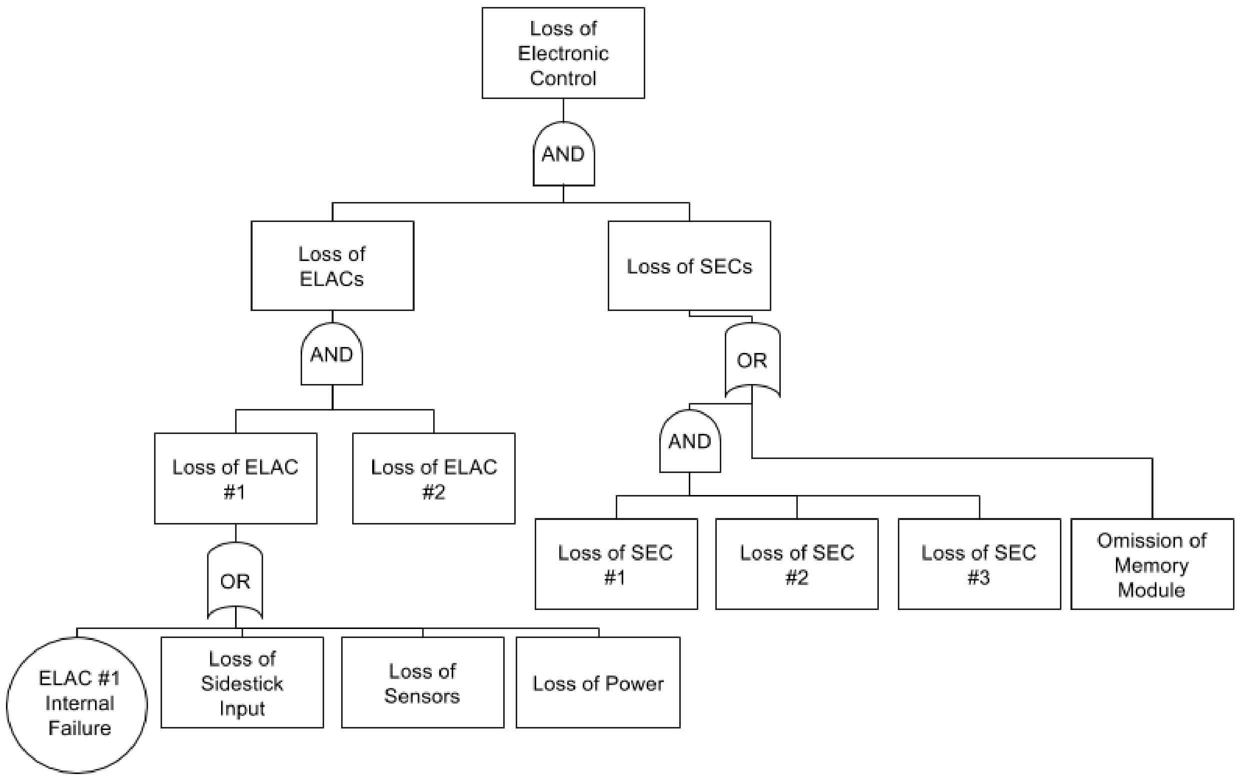


Figure 6.24 - Modified Loss of Electronic Control Failure Behaviour

By reapplying the process, the memory module within each SEC can trigger a failure. Thus, each of the SECs individually will inherit the full DAL of the SECs group, DAL C. Finally, the independence assumption and MCS groups referenced in the argument structure before would not be present. All of this can be seen in Figure 6.25, showing SEC#1's DALCorrectArg. Naturally, similar changes are applied to the argument structures of the other SECs.

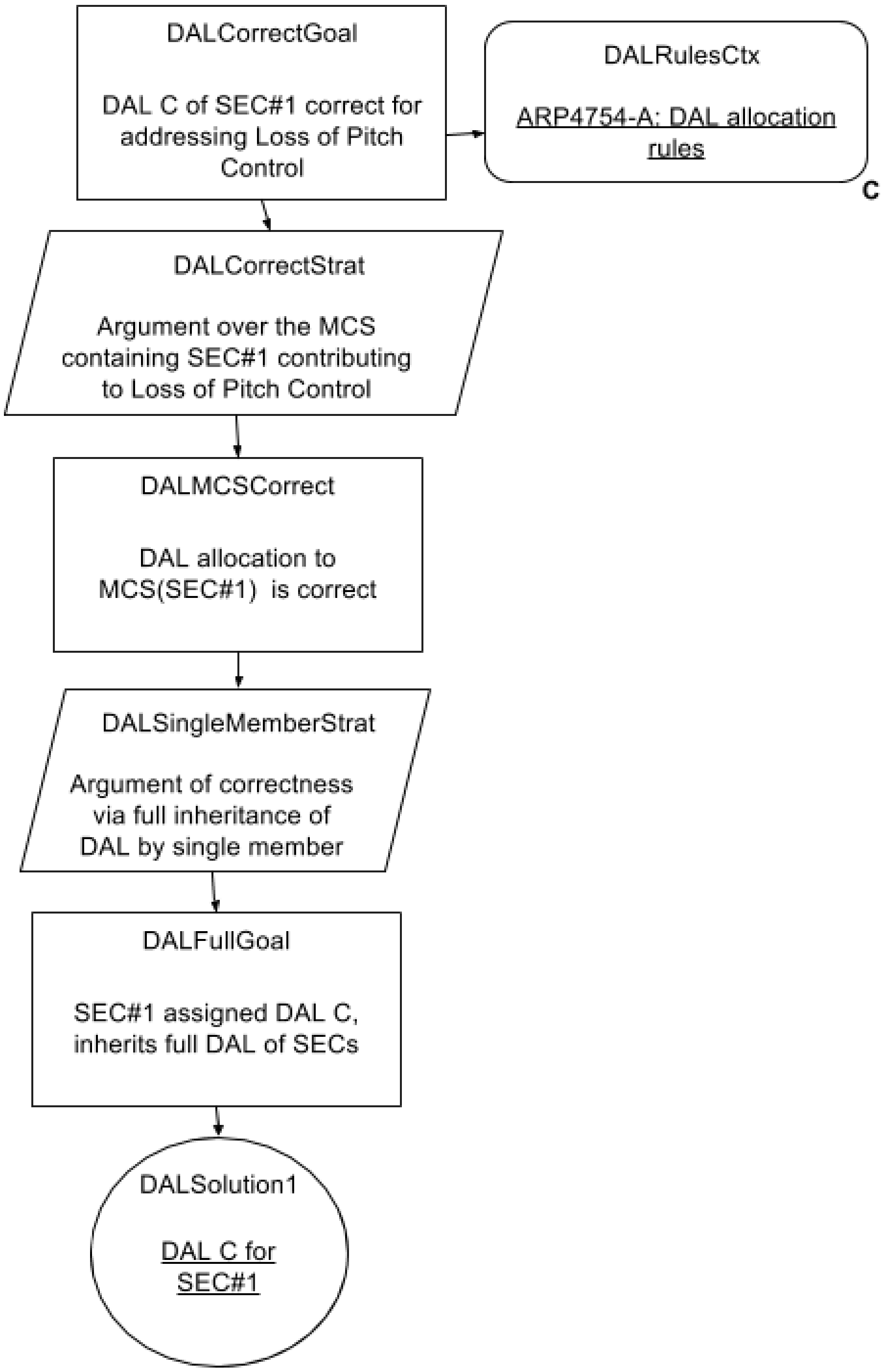

DALCorrectGoal
DAL C of SEC#1 correct for addressing Loss of Pitch Control
DALRulesCtx
ARP4754-A: DAL allocation rules
C
DALCorrectStrat
Argument over the MCS containing SEC#1 contributing to Loss of Pitch Control
DALMCSCorrect
DAL allocation to MCS(SEC#1) is correct
DALSingleMemberStrat
Argument of correctness via full inheritance of DAL by single member
DALFullGoal
SEC#1 assigned DAL C, inherits full DAL of SECs
DALSolution1
DAL C for SEC#1

Figure 6.25 - Modified Flight Control System - SEC#1 DALCorrectArg

11. Evaluation

Safety cases are evaluated manually, as developers, safety experts, regulatory authorities and other stakeholders review them. Systematically reviewing a safety case is certainly feasible, with previous work based on notions such as mathematical uncertainty and counter-argumentation, see (Mayo, 2006) and (Cyra & Gorski, 2011). However, in all of these cases, extensive automatic support has yet to be provided and remains a significant challenge as it always fundamentally relies on detailed engineering judgement. Therefore, no avenue under which the results of the case study could be evaluated automatically is currently available and to pursue this endeavour would be beyond the scope of the thesis.

Instead, evaluation of the case study focuses on two overall aspects; evaluating the correctness of the resulting safety argument manually and evaluating the benefits of the approach towards the development lifecycle. With regards to the former, the produced argument structure was reviewed top-down, applying an evaluation at each GSN element which questioned whether:

- The element is **relevant** to the overall safety argument and its parent element. The aim of this criterion is to dismiss unrelated or redundant elements from the argument.
- The element links **directly** to its parent element. This criterion aims to minimize the risk of ambiguity or confusion being introduced while transitioning to lower-level arguments.
- The **coverage** a parent element receives from its supporting elements is sufficient. This criterion aims to ensure that the parent can be justifiably inferred, or less commonly deduced, from its supporting elements.

The safety argument patterns and resulting arguments from the example in Chapter 5 and the case study in Chapter 6 were informally reviewed according to the above criteria. Following a number of iterations, the end result of which was presented in the preceding sections, the pattern and produced argument were found acceptable for illustration purposes. That being said, there are certainly limitations regarding the quality of the produced arguments that were not overcome. The primary limitation is that both the FCS model and the argument pattern were constructed informally, using information gathered from academic publications, aircraft manuals and unofficial sources. Thus, despite best effort to maintain a degree of realism, there is uncertainty regarding particular technical details of the model and pattern. An industrial case study would provide an answer towards this issue. Unfortunately, such an opportunity did not present itself during the research of the thesis. Further limitations with regards to the thesis are presented in Chapter 7.

The second aspect of evaluation focuses on the criteria with which the method was constructed. The first criterion is **scalability**, explained previously in Section 4 of Chapter 4. Scalability is directly supported via the inclusion of the relevant parameterizable elements associated with the method's use of argument patterns. These include the foreach and if-then elements, the use of argument modules and the instantiation algorithm. To support this claim quantitatively, the relationship between the elements across the argument pattern, the architectural model and the resulting argument can be used. Table 6.2 summarizes this relationship for the example from Chapter 5 and the FCS from Chapter 6.

Name	**# of Architectural Elements**	**# of Argument Pattern Elements**	**# of Argument Elements**	**Average Execution Time (s)**
Example	5	10	20	<1s
FCS	67	112	< 65.000	<15s

Table 6.2 - Scalability Metrics

As can be seen in Table 6.2, despite the significant increase in the size of the system model, argument pattern and the produced argument, the method completes its processing in seconds in both cases.

Given that the DAL allocation itself requires on average approximately 4.5 seconds to complete for the FCS (see Section 4 of Chapter 4), the additional time required to generate the argument is negligible.

The next criterion with which the method is evaluated is **traceability**. This property assesses the method's ability to maintain links between import safety assurance and assessment artefacts. For example, this concept is important in ARP4754-A, as the standard advises that low-level requirements should remain traceable to the high-level counterparts they were designed to satisfy (SAE, 2010: 13, 26, 30). Ensuring such connections are kept visible to the developers throughout the development lifecycle facilitates both safety assessment and assurance of relevant requirements. To evaluate traceability, we need to consider the capability the argument pattern provides the user with. Specifically, let us suppose a link is required to be maintained between a property *P* of a system *S* and a property *P'* of a component supporting the former's function. Then, the pattern should provide the ability to declare a supporting link to $S.P$ by invoking $S.S'.P^{(n)'}$, then $S.S'.S''.P^{(n-1)'}$ and so on until $S\ldots C.P'$, where *S', S''* are successive subsystems of S, eventually containing *C* and $P^{(n)'}$, $P^{(n-1)'}$ … P' are corresponding properties supporting property *P* at each architectural level. Based on the Library concept, explained in Section 2.2 of Chapter 5, it is possible to maintain such links and represent them within an argument pattern. Thus, a user can define the pattern to meet traceability according to a safety standard's guidelines. An example of this capability can be found in Figure 5.3, where the DALCorrectGoal element maintains the link between the DAL of SEC#4 and the high-level hazard it contributes to.

The final criterion is **maintainability**, addressing how the method supports and integrates within an iterative design lifecycle, such as the one described in Section 5.5 of Chapter 2. In other words, as changes occur over the course of development, a maintainable approach to safety assessment and assurance should incorporate such changes. At the same time, the method should reuse information with regards to safety or the system architecture that has remained unchanged. The examples shown in

Section 10 of this chapter should be sufficient to illustrate how the criterion is met. In the first example, architectural change incurred during development is incorporated automatically to reproduce a new argument. The updated argument reflects the change, without need for further user interaction. In the second example, the user is required to incorporate a change in the argument pattern due to an invalidated assumption detected within it. Following this modification, the argument is restructured, once more without the need for further interaction.

12. Summary

In this chapter, the method presented earlier in Chapter 5 has been applied to illustrate its benefits on an abstract, large-scale system. The resulting argument structure is produced (semi-)automatically, with user input being limited to providing an appropriate argument pattern.

The potential of the method towards addressing forms of change was also demonstrated, with two examples. In the following chapter, the method will be evaluated as a whole, summarising its overall benefits and limitations towards when considering its application within the development lifecycle. Extensions to the method and further work will also be discussed.

Chapter 7: Evaluation, Conclusions and Further Work

1. Introduction

In this chapter, the methods introduced in Chapters 3 and 4 and the case study demonstrating the second method in Chapter 6 are evaluated. The evaluation of this work is performed with respect to the overarching hypothesis and objectives outlined in the Introduction. Achievements, contributions and limitations are highlighted in the light of the theoretical and experimental results of the thesis. Additional comments are provided regarding further work.

2. Evaluation and contributions

Chapter 2 established the concept of safety cases, in the context of the development of safety-critical systems. Important techniques involved in the construction and management of safety cases were additionally introduced in Chapter 3. Another concept introduced in Chapter 2 was that of the model-based safety assessment (MBSA) paradigm. MBSA has proven to be useful in addressing traceability and synchronization between model and safety analyses. Thus, the effectiveness of such activities during development can be improved. Finally, the regulatory framework surrounding the certification of civil aircraft was presented. By understanding the processes involved, the argument rationale developers are required to follow to defend their systems' safety should be clarified.

The end of the Introduction defined a set of objectives for the thesis. The first objective, involved an investigation into the relevant literature of subjects mentioned above. The material presented in Chapter 3 is the end-product of the literature review. Although certainly not exhaustive, the investigation was arguably fruitful and illuminated much of the background necessary for fulfilling the remaining objectives. The next objective involved the development of a method that addressed the issue of scale in the construction of safety cases for civil aircraft. Safety assurance by means of the safety case is supported by a safety assessment framework. For certification purposes, the framework is adopted following safety standards. It is through this framework that the evidence needed to substantiate the claims of the safety case is provided. By analysing the safety assessment processes required by the relevant civil aircraft safety standards, it was determined that the central process for performing most of the required assessment activities is the allocation of Development Assurance Levels (DALs). Thus, scalability of this method needed to be addressed first to make any further progress. This is achieved via the method introduced in Chapter 4.

The method in Chapter 4 allows DALs to be allocated semi-automatically across a system architecture annotated with failure behaviour information. The details of the mechanism which identified the correct and cost-optimal DAL allocations were also provided. Finally, a small-scale system was examined as a case study, presented to support the method's effectiveness. By exploiting the method in Chapter 4, it is now possible to construct large safety argument structures.

Towards this end, the concept of safety argument patterns is employed in a novel way. Traditionally, argument patterns are used as guides for human designers to construct safety arguments. Instead, patterns are used by the method in Chapter 5 to algorithmically control the generation of argument structures. When combined with the capability of automatically allocating DALs, their potential for generating major parts of a civil aircraft safety case is realized. This capability is exemplified through a small-scale example of the method, as well as a more in-depth case study of a hypothetical flight control system in Chapter 6.

With regards to the case study, a large argument structure was generated using a collection of comparatively smaller patterns. Argument patterns used manually can, of course, assist in similar ways and reduce the need for repetitive construction of similar arguments. However, the method enhances this feature by applying patterns algorithmically. The enhancement is manifold:

- The effect of the algorithmic pattern application is to effectively eliminate the cost of constructing arguments that can be generated by the pattern.
- The provision of the model post-DAL-allocation serves to address a large part of the argumentation needed for ARP4754-A.
- Automatic application of argument patterns means that room for human error is severely limited. Specifically, any potential errors are much more likely to be found within the particular argument pattern or model design. Thus, the potential for errors in the resulting argument is significantly reduced. As such, it is easier to identify by testing and correcting the pattern, which is typically much smaller than the resulting argument.
- Top-down development is fully supported, consistent with the guidelines of ARP4754-A. Development iteration is also better supported as changes affecting the model can immediately be reapplied through the process. Thus, affected requirements are reallocated and appropriate assurance is regenerated, available for another design iteration if needed.
- The method is applicable from the earlier stages of design. As shown in Section 3 of Chapter 5, the process can be applied following a preliminary Functional Hazard Analysis, the very first safety assessment process to be performed according to the ARP4754-A guidelines. Furthermore, the process provides gains in time and effort spent on previously manual procedures with each application. Thus, the earlier the process is adopted and the further the development progresses while making use of it, the larger the benefits in cost reduction.
- Changes to the model, which are very typical of an iterative design approach, are managed without further manual input than the bare minimum, which is effectively data entry. This benefit applies to the pattern as well. This aspect, combined with the low execution time of the algorithms involved, establishes the potential for interactive Model-Based Safety Assessment and Assurance.

- Although DAL allocation is performed automatically, the reasoning governing the choice of allocation is incorporated into the argument generation. This feature can enable the construction of arguments which not only support the correctness of the allocation but its optimality with regards to development cost as well. This type of assurance covers safety assurance as viewed through the ALARP principle, explained in Section 1 of Chapter 2. This type of argument can be more effective in convincing stakeholders interested in achieving safety certification while also addressing economic constraints.

3. Limitations

In his address at the Nelson Mandela Foundation on 23rd November 2004, Desmond Tutu advised “Don’t raise your voice, improve your argument" (Nelson Mandela Foundation, 2004).

A reflection on the limitations of this work thus follows.

- The ‘Library’ is straightforward to use but can become overwhelmingly verbose. For example, a basic data structure describing a system and a component, each with descriptions can be simply defined as per the left side of Figure 7.1. However, within the Library, this relationship would instead be described as per the right side of Figure 7.1. As can be seen, the Library requires much more information to be maintained, a trade-off for its generic, flexible design.

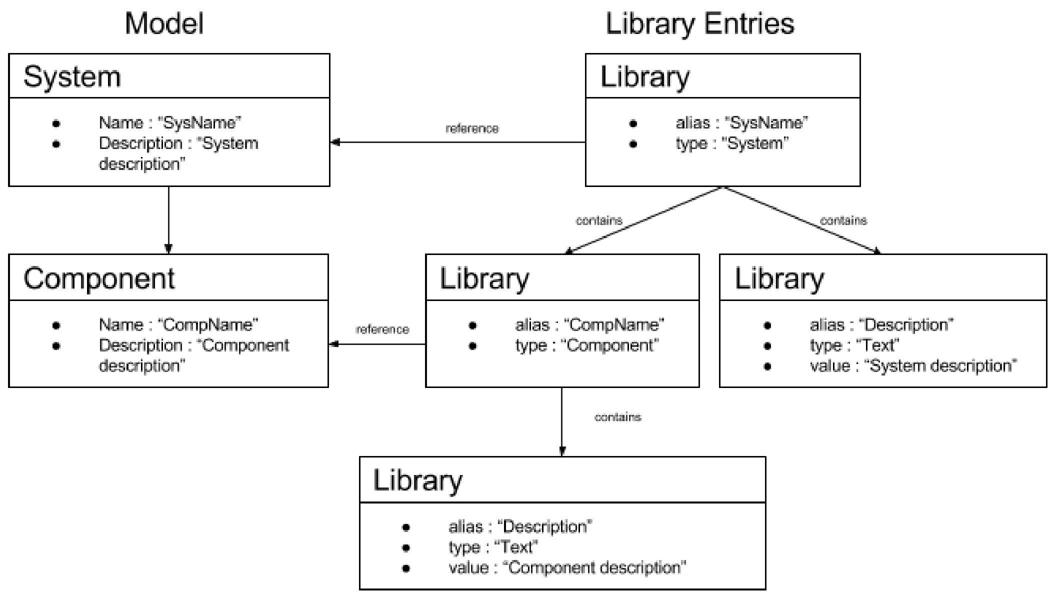


Figure 7.1 - Example of Model and corresponding Library

- The extent of the benefits in cost reduction depends on the pattern used. The more iteration and recursion which scales off the model is employed, the greater the cost reduction. Thus, for applications which do not make full use of patterns in this sense, manual construction of arguments could yield comparable results.
- Custom changes to the argument can be applied following its production, however they would need to be incorporated into the pattern to be retained upon reapplication. The more unique the argumentation needed, the more extensive such customization is likely to be applied. That being said, provided the changes are stable, their development cost should be expected to be incurred only once, when they are initially incorporated into the pattern.
- Pattern construction becomes more complicated than before, requiring the use of elements similar to those found in programming. We can envision users being required to test multiple iterations of patterns to arrive at a satisfactory argument. In view of the benefits that can be reaped, the development-time cost of pattern construction is arguably justified.

However, depending on the particular requirements of a development project, the application benefits could vary considerably.

- There are currently ongoing attempts in defining certification-friendly argument patterns relevant to civil aircraft development, for instance in (Holloway, 2015), (Weaver, 2003), (Sljivo et al., 2014), and (Oliveira et al., 2015). Unfortunately, there is no wide community nor official consensus on acceptable argument patterns for the ARP4754-A. If and when such support is provided, the incorporation of high-quality, widely-accepted patterns would benefit tremendously towards certification.
- Investigation regarding the issue of uncertainty or confidence in safety cases has been investigated by various authors, for example in (Weaver et al., 2006) and (Denney et al., 2011). Extending the method proposed to support annotation of probabilistic elements used to quantitatively analyse the generated arguments in this regard is a promising avenue of investigation.
- More investigation of the proposed methodology in addressing industrial case studies is needed. The examples and case studies presented, although sufficient for the purposes of the thesis, cannot provide an accurate evaluation of the method. Instead, applying the method in practice would allow the hypotheses and experimental results to be criticized. The lack of industrial application also limited the option for exploring the usability of the method. If that option was available, user feedback could be incorporated towards improving the method’s features and usability.

4. Further Work

There are numerous options for pursuing further research to extend the method proposed in this thesis. First of all, the DAL allocation process should be investigated further, with respect to identifying optimization techniques which would improve its performance. Plausible alternatives for initial investigation include linear programming and other metaheuristic techniques, outlined in Section 5 of Chapter 3. Preliminary evaluation of Ant Colony Optimization (ACO) has indicated that Tabu Search's performance is superior. However, in the author's opinion, further investigation is required to yield a more definitive view in both cases with regards to performance.

The next option for further research relates to previous work in (Denney & Pai, 2013). The authors formalize standard and abstract GSN elements and introduce their own instantiation algorithm. Both the definitions and instantiation algorithm introduced there could be adopted by the method to construct formally sound arguments. The method the authors propose uses a requirements/hazards table to instantiate a pattern of their variant. Such tables can be initially generated by the method from Chapter 4 and filled in manually with other requirements. The potential of generating formally-sound arguments is very attractive for supporting safety or more generally assurance cases whose structure can be tested logically with tool support.

The 'Library' concept proposed in Section 2.2 of Chapter 5 is a straightforward approach to generically structuring system architectures and their properties. The aim of the concept was to support argument patterns referencing elements or properties of a given generic model. The concept was adequate to address the examples and case studies presented earlier in the thesis. However, considering application of the method in industry, the concept's limitations might prove to be restrictive. An alternative approach would be to pursue a metamodel-based approach, such as the 'weaving model' (Hawkins et al., 2015). This option would ideally maintain the flexibility of application across a wide variety of models, while also being able to efficiently specialize for particular needs of a given model.

A number of other methods for automatically constructing safety arguments can extend the proposed method:

- In (Sljivo et al., 2014), bottom-up automatically constructed safety arguments can provide better argumentation coverage for components, especially COTS.
- In (Basir et al., 2010) and (Denney et al., 2012), automatically constructed safety arguments for verification via formal methods of automatically constructed software components are presented.
- In (Gallina et al., 2014b), process-based arguments are constructed automatically from process line specifications. The argumentation derived from the method proposed in the thesis emphasizes product-based arguments. Thus, the two approaches can complement each other in constructing a more complete safety case.

References

Alexander, C., Ishikawa, S. & Silverstein, M. (1977) *A Pattern Language: Towns, Buildings, Construction*. New York, NY, USA: Oxford University Press USA.

Ammann, P. & Offutt, J. (2008) *Introduction to Software Testing*. New York, NY, USA: Cambridge University Press.

Arnold, A., Point, G., Griffault, A. & Rauzy, A. (1999) The Altarica formalism for describing concurrent systems. *Fundamenta informaticae*, 40(2,3), 109–124.

Azevedo, L.S. (2015) *Scalable allocation of safety integrity levels in automotive systems*. PhD Thesis. Hull, UK: University of Hull. Available online: https://hydra.hull.ac.uk/resources/hull:13618 [Accessed 20/4/2017].

Azevedo, L.S., Parker, D., Walker, M., Papadopoulos, Y. & Araújo, R.E. (2014) Assisted Assignment of Automotive Safety Requirements. *IEEE Software*, 31(1), 62–68. Available online: http://dx.doi.org/10.1109/MS.2013.118 [Accessed 16/5/2017].

Basir, N., Denney, E. & Fischer, B. (2010) Deriving Safety Cases for Hierarchical Structure in Model-Based Development. *International Conference on Computer Safety, Reliability, and Security (SAFECOMP)*. Vienna, Austria, 14 September 2010. Springer, Berlin, Heidelberg, 68–81. Available online: http://dx.doi.org/10.1007/978-3-642-15651-9_6 [Accessed 21/4/2017].

Bendraou, R., Combemale, B., Cregut, X. & Gervais, M.P. (2007) Definition of an Executable SPEM 2.0. *14th Asia-Pacific Software Engineering Conference (APSEC'07)*. Aichi, Japan, December 2007. IEEE, 390–397. Available online: http://dx.doi.org/10.1109/ASPEC.2007.60 [Accessed 16/5/2017].

Bieber, P., Bougnol, C., Castel, C., Kehren, J.-P.H.C., Metge, S. & Seguin, C. (2004) Safety assessment with altarica : Lessons learnt based on two aircraft system studies. *Building the Information Society*. Toulouse, France, 2004. Springer, Boston, MA, 505–510. Available online: http://dx.doi.org/10.1007/978-1-4020-8157-6_45 [Accessed 21/4/2017].

Bieber, P., Delmas, R. & Seguin, C. (2011) DALculus–theory and tool for development assurance level allocation. *International Conference on Computer Safety, Reliability, and Security*. Naples, Italy, 2011. Springer, Berlin, Heidelberg, 43–56. Available online: http://dx.doi.org/10.1007/978-3-642-24270-0_4 [Accessed 20/4/2017].

Bishop, P.G., Bloomfield, R. & Guerra, S. (2004) The future of goal-based assurance cases. *2004 International Conference on Dependable Systems and Networks*. Florence, Italy, 2004.390–395. Available online: http://dx.doi.org/10.1.1.64.1568 [Accessed 16/5/2017].

Bishop, P.G. & Bloomfield, R.E. (1995) The SHIP Safety Case Approach. *Safe Comp 95*. London, UK, 1995. Springer, London, 437–451. Available online: http://dx.doi.org/10.1007/978-1-4471-3054-3_30 [Accessed 18/4/2017].

Bloomfield, R. & Netkachova, K. (2014) Building Blocks for Assurance Cases. *2014 IEEE International Symposium on Software Reliability Engineering Workshops*. Naples, Italy, November 2014. IEEE, 186–191. Available online: http://dx.doi.org/10.1109/ISSREW.2014.72 [Accessed 16/5/2017].

Blum, C. & Roli, A. (2003) Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison. *ACM Computing Surveys*, 35(3), September, 268–308. Available online: http://dx.doi.org/10.1145/937503.937505 [Accessed 20/4/2017].

Bondi, A.B. (2000) Characteristics of Scalability and Their Impact on Performance. *2nd International Workshop on Software and Performance*. Ottawa, Ontario, Canada, 2000. ACM, 195–203. Available online: http://dx.doi.org/10.1145/350391.350432 [Accessed 20/4/2017].

Bowen, J. & Stavridou, V. (1993) Safety-critical systems, formal methods and standards. *Software Engineering Journal*, 8(4), July, 189–209. Available online: http://dx.doi.org/10.1049/sej.1993.0025 [Accessed 16/5/2017].

Bozzano, M. & Villafiorita, A. (2003) Improving system reliability via model checking: The FSAP/NuSMV-SA safety analysis platform. *International Conference on Computer Safety, Reliability, and Security*. Edinburgh, UK, 2003. Springer, 49–62. Available online: http://dx.doi.org/10.1007/978-3-540-39878-3_5 [Accessed 21/4/2017].

Bray, T., Paoli, J., Sperberg-McQueen, C.M., Maler, E., Yergeau, F. & Cowan, J. (1997) *Extensible Markup Language (XML) 1.1*, 2nd edition. W3C.

Brière, D. & Traverse, P. (1993) AIRBUS A320/A330/A340 electrical flight controls-a family of fault-tolerant systems. *The Twenty-Third International Symposium on Fault-Tolerant Computing, 1993. Digest of Papers.* Toulouse, France, 22 June 1993. IEEE, 616–623. Available online: http://dx.doi.org/10.1109/FTCS.1993.627364 [Accessed 21/4/2017].

Capelle, V. & Houtermans, M. (2006) Functional safety: a practical approach for end-users and system integrators. *WSEAS Transactions on Systems*, 5(8), 1946–1956. Available online: http://dx.doi.org/10.2118/102516-PA [Accessed 19/4/2017].

CENELEC (1999) *EN 50126-1: Railway applications – The specification and demonstration of Reliability, Availability, Maintainability and Safety (RAMS)*. European Committee for Electrotechnical Standardization (CENELEC).

CENIPA (2007) *Final report A-N 67/CENIPA/2009*. Aeronautical Accident Investigation and Prevention Center (CENIPA). Available online: http://www.cenipa.aer.mil.br/cenipa/paginas/relatorios/pdf/3054ing.pdf [Accessed 21/4/2017].

Černý, V. (1985) Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *Journal of Optimization Theory and Applications*, 45(1), 1 January, 41–51. Available online: http://dx.doi.org/10.1007/BF00940812 [Accessed 20/4/2017].

Chang, G.W., Aganagic, M., Waight, J.G., Medina, J., Burton, T., Reeves, S. & Christoforidis, M. (2001) Experiences with mixed integer linear programming based approaches on short-term hydro scheduling. *IEEE Transactions on power systems*, 16(4), 743–749. Available online: http://dx.doi.org/10.1109/59.962421 [Accessed 20/4/2017].

Cimatti, A., Pistore, M., Roveri, M. & Traverso, P. (2003) Weak, strong, and strong cyclic planning via symbolic model checking. *Artificial Intelligence*, 147(1–2), 35–84. Available online: http://dx.doi.org/10.1016/S0004-3702(02)00374-0 [Accessed 21/4/2017].

Clarke, E.M. & Emerson, E.A. (1981) Design and synthesis of synchronization skeletons using branching time temporal logic. *Workshop on Logic of Programs*. New York, NY, USA, 4 May 1981. Springer, Berlin, Heidelberg, 52–71. Available online: http://dx.doi.org/10.1007/BFb0025774 [Accessed 21/4/2017].

Cleland-Huang, J., Zemont, G. & Lukasik, W. (2004) A heterogeneous solution for improving the return on investment of requirements traceability. *Proceedings of the 12th IEEE International Requirements Engineering Conference*. Kyoto, Japan, September 2004. IEEE, 230–239. Available online: http://dx.doi.org/10.1109/ICRE.2004.1335680 [Accessed 16/5/2017].

Clocksin, W. & Mellish, C.S. (2003) *Programming in PROLOG : Using the ISO Standard*, 4th edition. Springer, Berlin, Heidelberg.

Cullen, R.J. (1996) Safety as a Design Tool. *Managing Risk in a Changing Organisation Climate-Proceedings of the Safety and Reliability Symposium, Swindon, UK*. Swindon, UK, 1996.

Cyra, L. & Gorski, J. (2011) Support for argument structures review and assessment. *Reliability Engineering & System Safety*, 96(1), 26–37. Available online: http://dx.doi.org/10.1016/j.ress.2010.06.027 [Accessed 21/4/2017].

Dabney, J.B. & Harman, T.L. (2004) *Mastering simulink*. Upper Saddle River, NJ, USA: Pearson/Prentice Hall.

De Moura, L. & Bjørner, N. (2008) Z3: An efficient SMT solver. *International conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*. Budapest, Hungary, 2008. Springer, Berlin, Heidelberg, 337–340. Available online: http://dx.doi.org/10.1007/978-3-540-78800-3_24 [Accessed 20/4/2017].

Delange, J. & Feiler, P. (2014) Architecture fault modeling with the AADL error-model annex. *40th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA), 2014*. Verona, Italy, 2014. IEEE, 361–368. Available online: http://dx.doi.org/10.1109/SEAA.2014.20 [Accessed 21/4/2017].

Denney, E. & Pai, G. (2013) A formal basis for safety case patterns. 2013. Springer, 21–32. Available online: http://link.springer.com/chapter/10.1007/978-3-642-40793-2_3 [Accessed 21/4/2017].

Denney, E. & Pai, G. (2014) Automating the Assembly of Aviation Safety Cases. *IEEE Transactions on Reliability*, 63(4), December, 830–849. Available online: http://dx.doi.org/10.1109/TR.2014.2335995.

Denney, E., Pai, G. & Habli, I. (2011) Towards Measurement of Confidence in Safety Cases. *2011 International Symposium on Empirical Software Engineering and Measurement*. Banff, AB, Canada, September 2011. IEEE, 380–383. Available online: http://dx.doi.org/10.1109/ESEM.2011.53 [Accessed 16/5/2017].

Denney, E., Pai, G. & Pohl, J. (2012) Heterogeneous Aviation Safety Cases: Integrating the Formal and the Non-formal. *2012 IEEE 17th International Conference on Engineering of Complex Computer Systems*.

Paris, France, July 2012. IEEE, 199–208. Available online: http://ieeexplore.ieee.org/document/6299215/ [Accessed 16/5/2017].

Dhouibi, M.S., Saintis, L., Barreau, M. & Perquis, J.-M. (2014) Automatic decomposition and allocation of safety integrity level using system of linear equations. *PESARO 2014, The Fourth International Conference on Performance, Safety and Robustness in Complex Systems and Applications*. Nice, France, 2014. IARIA, Available online: http://dx.doi.org/10.13140/2.1.2856.0321 [Accessed 20/4/2017].

Dorigo, M., Birattari, M. & Stutzle, T. (2006) Ant colony optimization. *IEEE Computational Intelligence Magazine*, 1(4), November, 28–39. Available online: http://dx.doi.org/10.1109/MCI.2006.329691.

Dorigo, M., Maniezzo, V. & Colorni, A. (1991) The ant system: An autocatalytic optimizing process, Available online: http://dx.doi.org/10.1.1.51.4214 [Accessed 20/4/2017].

Doyle, S.A. & Dugan, J.B. (1995) Dependability assessment using binary decision diagrams (BDDs). *Twenty-Fifth International Symposium on Fault-Tolerant Computing, 1995. FTCS-25. Digest of Papers.* Pasadena, CA, USA, 1995. IEEE, 249–258. Available online: http://dx.doi.org/10.1109/FTCS.1995.466973 [Accessed 21/4/2017].

DPIE (1991) *Report on the Consultative Committee on Safety in the Offshore Petroleum Industry*. Australia: Department of Primary Industries and Energy (DPIE). Available online: http://www.mrt.tas.gov.au/mrtdoc/petxplor/download/OR_0935/OR_0935.pdf.

Dugan, J.B., Bavuso, S.J. & Boyd, M.A. (1992) Dynamic fault-tree models for fault-tolerant computer systems. *IEEE Transactions on reliability*, 41(3), 363–377. Available online: http://dx.doi.org/10.1109/24.159800 [Accessed 21/4/2017].

Dugan, J.B., Venkataraman, B. & Gulati, R. (1997) DIFTree: a software package for the analysis of dynamic fault tree models. *1997 Annual Proceedings of the Reliability and Maintainability Symposium*. Philadelphia, PA, USA, 1997. IEEE, 64–70. Available online: http://dx.doi.org/10.1109/RAMS.1997.571666 [Accessed 21/4/2017].

EASA (2011) *CS-25/Amendment 11: AMC 25.1309 System Design and Analysis*. European Aviation Safety Agency (EASA). Available online: https://www.easa.europa.eu/document-library/certification-specifications/cs-25-amendment-11 [Accessed 20/4/2017].

Eberhart, R. & Kennedy, J. (1995) A new optimizer using particle swarm theory. *Sixth International Symposium on Micro Machine and Human Science, 1995. MHS '95*. Nagoya, Japan, October 1995. IEEE, 39–43. Available online: http://dx.doi.org/10.1109/MHS.1995.494215 [Accessed 16/5/2017].

Ericson, C.A. (2005) *Hazard Analysis Techniques for System Safety*. John Wiley & Sons.

ESI (2017) *SimulationX for Universities*. Available online: https://www.simulationx.com/simulation-software/universities.html [Accessed 21/4/2017].

EUROCAE (2012) *ED-12C Software considerations in airborne systems and equipment certification*. Saint-Denis, France: European Organisation for Civil Aviation Equipment (EUROCAE). Available online: https://eshop.eurocae.net/eurocae-documents-and-reports/ed-12c/ [Accessed 15/5/2017].

EUROCAE (2010) *EUROCAE ED 79 - GUIDELINES FOR DEVELOPMENT OF CIVIL AIRCRAFT AND SYSTEMS*. Saint-Denis, France: European Organisation for Civil Aviation Equipment (EUROCAE). Available online: http://standards.globalspec.com/std/1299977/eurocae-ed-79 [Accessed 15/5/2017].

EUROCONTROL (2011) *Preliminary safety case for ADS-B airport surface surveillance application*. Brussels, Belgium: European Organisation for the Safety of Air Navigation (EUROCONTROL). Available online: https://www.eurocontrol.int/sites/default/files/publication/files/surveillance-cascade-preliminary-safety-case-for-airports-surface-surveillance-applications-201111.pdf [Accessed 20/4/2017].

FAA (1988) *AC 25.1309-1A - System Design and Analysis – Document Information*. Washington, DC, USA: Federal Aviation Administration (FAA). Available online: https://www.faa.gov/regulations_policies/advisory_circulars/index.cfm/go/document.information/documentID/22680 [Accessed 15/5/2017].

Fenelon, P. & McDermid, J.A. (1993) An integrated tool set for software safety analysis. *Journal of Systems and Software*, 21(3), 279–290. Available online: http://dx.doi.org/10.1016/0164-1212(93)90029-W [Accessed 15/5/2017].

Fogel, L.J., Owens, A.J. & Walsh, M.J. (1966) *Artificial Intelligence Through Simulated Evolution*. John Wiley & Sons.

Frana, R.B., Bodeveix, J.-P., Filali, M. & Rolland, J.-F. (2007) The AADL behaviour annex–experiments and roadmap. *Engineering Complex Computer Systems, 2007. 12th IEEE International Conference on*. Auckland, New Zealand, 2007. IEEE, 377–382. Available online: http://dx.doi.org/10.1109/ICECCS.2007.41 [Accessed 21/4/2017].

Gallina, B. (2014) A model-driven safety certification method for process compliance. *2014 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*. Naples, Italy, 2014. IEEE, 204–209. Available online: http://dx.doi.org/10.1109/ISSREW.2014.30 [Accessed 20/4/2017].

Gallina, B., Javed, M.A., Muram, F.U. & Punnekkat, S. (2012) A Model-Driven Dependability Analysis Method for Component-Based Architectures. *2012 38th Euromicro Conference on Software Engineering and Advanced Applications*. Cesme, Izmir, Turkey, September 2012. IEEE, 233–240. Available online: http://dx.doi.org/10.1109/SEAA.2012.35.

Gallina, B., Kashiyarandi, S., Zugsbratl, K. & Geven, A. (2014a) Enabling cross-domain reuse of tool qualification certification artefacts. *International Conference on Computer Safety, Reliability, and Security*. Cham, Germany, 2014. Springer, 255–266. Available online: http://dx.doi.org/10.1007/978-3-319-10557-4_28 [Accessed 20/4/2017].

Gallina, B., Lundqvist, K. & Forsberg, K. (2014b) THRUST: a method for speeding up the creation of process-related deliverables. *Digital Avionics Systems Conference (DASC), 2014 IEEE/AIAA 33rd*. Colorado Springs, CO, USA, 2014. IEEE, 5D4–1. Available online: http://dx.doi.org/10.1109/DASC.2014.6979489 [Accessed 20/4/2017].

Gallina, B., Pitchai, K.R. & Lundqvist, K. (2014c) S-TunExSPEM: Towards an extension of SPEM 2.0 to model and exchange tunable safety-oriented processes. *Software Engineering Research, Management and Applications*. Kitakyushu, Japan, 2014. Springer International Switzerland, 215–230. Available online: http://dx.doi.org/10.1007/978-3-319-00948-3_14 [Accessed 20/4/2017].

Gallina, B. & Provenzano, L. (2015) Deriving reusable process-based arguments from process models in the context of railway safety standards. *Ada User Journal*, 36(4), 237–241. Available online: http://www.ada-europe.org/archive/auj/auj-36-4.pdf#page=39 [Accessed 20/4/2017].

Gamma, E., Helm, R., Johnson, R. & Vlissides, J. (1994) *Design Patterns: Elements of Reusable Object-Oriented Software*. Pearson Education.

Ge, X., Paige, R.F. & McDermid, J.A. (2009) Probabilistic Failure Propagation and Transformation Analysis. *International Conference on Computer Safety, Reliability, and Security*. Hamburg, Germany, 15 September 2009. Springer, Berlin, Heidelberg, 215–228. Available online: http://dx.doi.org/10.1007/978-3-642-04468-7_18 [Accessed 21/4/2017].

Gérard, S., Dumoulin, C., Tessier, P. & Selic, B. (2010) Papyrus: A UML2 Tool for Domain-Specific Language Modeling. In Giese, H., Karsai, G., Lee, E., Rumpe, B. & Schätz, B. (eds) *Model-Based Engineering of Embedded Real-Time Systems*. Lecture Notes in Computer Science 6100. Springer Berlin Heidelberg, 361–368.

Gheraibia, Y. & Moussaoui, A. (2013) Penguins search optimization algorithm (PeSOA). *International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems*. Amsterdam, Netherlands, 2013. Springer, 222–231. Available online: http://link.springer.com/10.1007/978-3-642-38577-3_23 [Accessed 20/4/2017].

Gheraibia, Y., Moussaoui, A., Azevedo, L.S., Parker, D., Papadopoulos, Y. & Walker, M. (2015) Can aquatic flightless birds allocate automotive safety requirements? *2015 IEEE Seventh International Conference on Intelligent Computing and Information Systems (ICICIS)*. Cairo, Egypt, 2015. IEEE, 1–6. Available online: http://ieeexplore.ieee.org/abstract/document/7397214/ [Accessed 20/4/2017].

Glover, F. (1986) Future Paths for Integer Programming and Links to Artificial Intelligence. *Computers and Operation Research*, 13(5), May, 533–549. Available online: http://dx.doi.org/10.1016/0305-0548(86)90048-1 [Accessed 20/4/2017].

GSN Working Group (2011) *Goal Structuring Notation*. GSN Working Group. Available online: http://www.goalstructuringnotation.info/ [Accessed 19/4/2017].

Haddon-Cave, C. (2009) *The Nimrod Review: an independent review into the broader issues surrounding the loss of the RAF Nimrod MR2 aircraft XV230 in Afghanistan in 2006*. London, UK: HMSO.

Hansen, P. & Jaumard, B. (1990) Algorithms for the maximum satisfiability problem. *Computing*, 44(4), 1 December, 279–303. Available online: http://dx.doi.org/10.1007/BF02241270 [Accessed 20/4/2017].

Hawkins, R., Clegg, K., Alexander, R. & Kelly, T. (2011) Using a Software Safety Argument Pattern Catalogue: Two Case Studies. *Proceedings of the 30th International Conference on Computer Safety, Reliability, and Security*. Naples, Italy, 2011. Springer-Verlag, 185–198. Available online: http://dl.acm.org/citation.cfm?id=2041619.2041640 [Accessed 19/4/2017].

Hawkins, R., Habli, I., Kolovos, D., Paige, R. & Kelly, T.P. (2015) Weaving an assurance case from design: a model-based approach. *IEEE 16th International Symposium on High Assurance Systems Engineering (HASE), 2015*. Daytona Beach, FL, USA, 2015. IEEE, 110–117. Available online: http://ieeexplore.ieee.org/abstract/document/7027421/ [Accessed 20/4/2017].

Hawkins, R. & Kelly, T.P. (2013) *A Software Safety Argument Pattern Catalogue*. York, UK: Department of Computer Science, University of York. Available online: https://www.cs.york.ac.uk/ftpdir/reports/2013/YCS/482/YCS-2013-482.pdf [Accessed 19/4/2017].

Hertz, A. & Werra, D. de (1987) Using tabu search techniques for graph coloring. *Computing*, 39(4), 1 December, 345–351. Available online: http://dx.doi.org/10.1007/BF02239976 [Accessed 20/4/2017].

Higham, D.J. & Higham, N.J. (2005) *MATLAB guide*. SIAM.

Ho, D. (2017) *Notepad++ Home*. Available online: https://notepad-plus-plus.org/ [Accessed 21/4/2017].

Holland, J.H. (1992) *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. Cambridge, MA, USA: MIT Press.

Holloway, C.M. (2015) Explicate '78: Uncovering the Implicit Assurance Case in DO-178C. *Safety-Critical Systems Club Annual Symposium*. United Kingdom, 3 February 2015. Available online: https://ntrs.nasa.gov/search.jsp?R=20150009473 [Accessed 27/12/2016].

Holloway, C.M. & Johnson, C.W. (2008) How past loss of control accidents may inform safety cases for advanced control systems on commercial aircraft. *2008 3rd IET International Conference on System Safety*. Birmingham, UK, 2008. IET, 1–6. Available online: http://dx.doi.org/10.1049/cp:20080732 [Accessed 21/4/2017].

Houck, O.A. (2010) Worst Case and the DEEPWATER HORIZON Blowout: There Ought to Be a Law. *Tulane Environmental Law Journal*, 24(1). Available online: http://heinonline.org/HOL/Page?handle=hein.journals/tulev24&id=3&div=&collection=.

HSW (1974) *Health and Safety at Work*, Chapter 47. London, UK: HMSO.

IEC (2017) *IEC - TC 56 Dependability Definitions*. International Electrotechnical Commission (IEC). Available online: http://tc56.iec.ch/about/definitions.htm#Dependability [Accessed 18/4/2017].

IEC (2010) *IEC 61508: Functional safety of electrical/electronic/programmable electronic safety-related systems*. International Electrotechnical Commission (IEC). Available online: https://webstore.iec.ch/publication/22273 [Accessed 18/4/2017].

ISA (1997) *ISA 84.01 : Application Of Safety Instrumented Systems For The Process Industries*. International Society of Automation (ISA). Available online: https://infostore.saiglobal.com/store/details.aspx?ProductID=702949 [Accessed 21/4/2017].

ISO (2011) *ISO 26262: Road vehicles -- Functional safety*. International Organization for Standardization (ISO). Available online: https://www.iso.org/standard/43464.html [Accessed 18/4/2017].

ISO (2009) *ISO 31000: Risk management*. International Organization for Standardization (ISO). Available online: https://www.iso.org/standard/43170.html [Accessed 18/4/2017].

Joshi, A., Heimdahl, M.P., Miller, S.P. & Whalen, M.W. (2006) *Model-based safety analysis*. Washington, DC, USA: National Aeronautics and Space Administration (NASA). Available online: https://ntrs.nasa.gov/search.jsp?R=20060006673 [Accessed 21/4/2017].

Kaiser, B. & Gramlich, C. (2004) State-event-fault-trees–a safety analysis model for software controlled systems. *International Conference on Computer Safety, Reliability, and Security*. Potsdam, Germany, 2004. Springer, Berlin, Heidelberg, 195–209. Available online: http://dx.doi.org/10.1007/978-3-540-30138-7_17 [Accessed 21/4/2017].

Kaiser, B., Liggesmeyer, P. & Mäckel, O. (2003) A new component concept for fault trees. *Proceedings of the 8th Australian workshop on Safety critical systems and software*. Canberra, Australia, 2003. Australian Computer Society, Inc., Darlinghurst, Australia, 37–46. Available online: http://dl.acm.org/citation.cfm?id=1082054 [Accessed 21/4/2017].

Kelly, T.P. (1998) *A systematic approach to managing safety cases*. York, UK: University of York. Available online: https://www-users.cs.york.ac.uk/tpk/tpkthesis.pd [Accessed 16/5/2017].

Kirkpatrick, S., Gelatt, C.D. & Vecchi, M.P. (1983) Optimization by simulated annealing. *Science*, 220(4598), 671–680. Available online: http://dx.doi.org/10.1126/science.220.4598.671 [Accessed 20/4/2017].

Knox, J.E. (1989) *The Application of Tabu Search to the Symmetric Traveling Salesman Problem*. PhD Thesis. Boulder, CO, USA: University of Colorado at Boulder. Available online: http://dl.acm.org/citation.cfm?id=916090 [Accessed 16/5/2017].

Laguna, M., Barnes, J.W. & Glover, F.W. (1991) Tabu search methods for a single machine scheduling problem. *Journal of Intelligent Manufacturing*, 2(2), 1 April, 63–73. Available online: http://dx.doi.org/10.1007/BF01471219 [Accessed 20/4/2017].

Le Berre, D. & Parrain, A. (2010) The SAT4J library, Release 2.2, System Description. *Journal on Satisfiability, Boolean Modeling and Computation*, 7, 59–64. Available online: https://hal.archives-ouvertes.fr/hal-00868136 [Accessed 20/4/2017].

Lee, W.S., Grosh, D.L., Tillman, F.A. & Lie, C.H. (1985) Fault Tree Analysis, Methods, and Applications: A Review. *IEEE Transactions on Reliability*, R-34(3), 194–203. Available online: http://dx.doi.org/10.1109/TR.1985.5222114.

Leveson, N.G. (2011a) *Engineering a Safer World*. Cambridge, MA, USA: The MIT Press.

Leveson, N.G. (2011b) The Use of Safety Cases in Certification and Regulation. *Journal of System Safety*, 47(6), December. Available online: http://www.system-safety.org/ejss/past/novdec2011ejss/spotlight1_p1.php [Accessed 18/4/2017].

Li, Y.-F., HuAng, H.-Z., Liu, Y. & Li, H. (2012) A new fault tree analysis method: fuzzy dynamic fault tree analysis. *Maintenance and Reliability*, 14(3), 17–56. Available online: http://www.relialab.org/Upload/files/01_05_Fuzzy%20Dynamic%20FTA_MR_2012.pdf [Accessed 15/5/2017].

Linling, S., Wenjin, Z. & Kelly, T. (2011) Do safety cases have a role in aircraft certification? *Procedia Engineering*, 17, 358–368. Available online: http://dx.doi.org/10.1016/j.proeng.2011.10.041 [Accessed 21/4/2017].

Linnosmaa, J. (2016) *Structured safety case tools for nuclear facility automation*. MSc Thesis. Tampere, Finland: Tampere University of Technology. Available online: https://dspace.cc.tut.fi/dpub/bitstream/handle/123456789/23931/linnosmaa.pdf?sequence=3 [Accessed 19/4/2017].

Mader, R., Armengaud, E., Leitner, A. & Steger, C. (2012) Automatic and optimal allocation of safety integrity levels. *2012 Proceedings Annual Reliability and Maintainability Symposium*. Reno, NV, USA, January 2012. IEEE, 1–6. Available online: http://dx.doi.org/10.1109/RAMS.2012.6175431 [Accessed 16/5/2017].

Mahmud, N. (2012) *Dynamic Model-based Safety Analysis: From State Machines to Temporal Fault Trees By*. PhD Thesis. Hull, UK: University of Hull. Available online: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.404.1163 [Accessed 27/4/2017].

Manquinho, V., Martins, R. & Lynce, I. (2010) Improving unsatisfiability-based algorithms for boolean optimization. *International conference on theory and applications of satisfiability testing*. Edinburgh, UK, 2010. Springer, Berlin, Heidelberg, 181–193. Available online: http://dx.doi.org/10.1007/978-3-642-14186-7_16 [Accessed 20/4/2017].

Marsh, W. (1999) *SafEty and Risk Evaluation using bayesian NEts: SERENE*. SERENE Partners. Available online: http://www.eecs.qmul.ac.uk/~norman/papers/serene.pdf [Accessed 18/4/2017].

Martinez-Ruiz, T., Garcia, F., Piattini, M. & Munch, J. (2011) Modelling software process variability: an empirical study. *IET Software*, 5(2), April, 172–187. Available online: http://dx.doi.org/10.1049/iet-sen.2010.0020.

Matsuno, Y. (2011) *D-case editor: A typed assurance case editor*. Tokyo, Japan: University of Tokyo. Available online: https://static.lwn.net/images/conf/rtlws-2011/proc/Matsuno.pdf [Accessed 19/4/2017].

Mayo, P.R. (2006) Structured safety case evaluation: A systematic approach to safety case review. *The First Institution of Engineering and Technology International Conference on System Safety, 2006*. London, UK, 2006. IET, Available online: http://dx.doi.org/10.1049/cp:20060214 [Accessed 21/4/2017].

Mazzini, S., Favaro, J., Puri, S. & Baracchi, L. (2016) CHESS: an open source methodology and toolset for the development of critical systems. *3rd International Workshop on Open Source Software for Model Driven Engineering (OSS4MDE 2016)*. Saint Malo, France, 2016. Available online: http://mase.cs.queensu.ca/oss4mde/data/uploads/papers/chess_an_open_source.pdf [Accessed 20/4/2017].

McDermid, J.A. (1994) Support for safety cases and safety arguments using SAM. *Reliability Engineering & System Safety*, 43(2), 111–127. Available online: http://dx.doi.org/https://doi.org/10.1016/0951-8320(94)90057-4 [Accessed 15/5/2017].

Metropolis, N., Rosenbluth, A.W., Rosenbluth, M.N., Teller, A.H. & Teller, E. (1953) Equation of state calculations by fast computing machines. *The journal of chemical physics*, 21(6), 1087–1092. Available online: http://dx.doi.org/10.1063/1.1699114 [Accessed 21/4/2017].

MoD (1997) *DS 00-55: Requirements for Safety Related Software in Defence Systems*. Defence Standard 00-55. London, UK: HMSO.

MoD (1996) *DS 00-56: Safety Management Requirements for Defence Systems*. Defence Standard 00-56. London, UK: HMSO.

MoD (2016) *DSA02-DMR - MOD Shipping Regulations for Safety and Environmental Protection*. Defence Standard A02–DMR. London, UK: HMSO.

Modarres, M., Kaminskiy, M. & Krivtsov, V. (1999) *Reliability Engineering and Risk Analysis: A Practical Guide*. CRC Press.

Murashkin, A., Azevedo, L.S., Guo, J., Zulkoski, E., Liang, J.H., Czarnecki, K. & Parker, D. (2015) Automated decomposition and allocation of automotive safety integrity levels using exact solvers. *SAE International Journal of Passenger Cars-Electronic and Electrical Systems*, 8(1), 70–78. Available online: http://dx.doi.org/10.4271/2015-01-0156 [Accessed 20/4/2017].

Nelson Mandela Foundation (2004) *The Second Nelson Mandela Annual Lecture Address*. Available online: https://www.nelsonmandela.org/news/entry/the-second-nelson-mandela-annual-lecture-address [Accessed 18/5/2017].

Nordhoff, S. (2012) *DO-178C/ED-12C: The new software standard for the avionic industry: goals, changes and challenges*. Cologne, Germany: SQS Software Quality System AG. Available online: https://www.academia.edu/13635180/The_new_software_standard_for_the_avionic_industry_goals_changes_and_challenges [Accessed 20/4/2017].

Norris, J.R. (1998) *Markov chains*. Cambridge University Press.

NTSB (2014) *Crash of Asiana Flight 214 Accident Report Summary*. USA: National Transportation Safety Board. Available online: https://www.ntsb.gov/news/events/Pages/2014_Asiana_BMG-Abstract.aspx [Accessed 19/4/2017].

Oliveira, A.L. de, Braga, R.T.V., Masiero, P.C., Papadopoulos, Y., Habli, I. & Kelly, T. (2015) Supporting the automated generation of modular product line safety cases. *Theory and Engineering of Complex Systems and Dependability: Proceedings of the Tenth International Conference on Dependability and Complex Systems*. Brunow, Poland, 2015.319–330. Available online: http://dx.doi.org/10.1007/978-3-319-19216-1_30 [Accessed 21/4/2017].

OMG (2016) *Structured Assurance Case Metamodel*. Object Management Group (OMG). Available online: http://www.omg.org/spec/SACM/ [Accessed 19/4/2017].

Papadopoulos, Y., Walker, M., Parker, D., Rüde, E., Hamann, R., Uhlig, A., Grätz, U. & Lien, R. (2011) Engineering failure analysis and design optimisation with HiP-HOPS. *Engineering Failure Analysis*, 18(2), 590–608. Available online: http://dx.doi.org/10.1016/j.engfailanal.2010.09.025 [Accessed 20/4/2017].

Parker, D.J. (2010) *Multi-objective optimisation of safety-critical hierarchical systems*. Hull, UK: The University of Hull. Available online: http://ethos.bl.uk/OrderDetails.do?uin=uk.bl.ethos.526329 [Accessed 21/4/2017].

Pfitzer, T., DeLong, T., Saralyn, D., Frost, J. & West, D. (2013) Safety Case Workshop. *APT Research Safety Case Workshop*. Huntsville, AL, USA, 2013. NASA, Available online: https://sma.nasa.gov/docs/default-source/News-Documents/newsfeed-safetycaseworkshop.pdf?sfvrsn=2.

Rechenberg, I. (1989) Evolution Strategy: Nature's Way of Optimization. *Optimization: Methods and Applications, Possibilities and Limitations*. Bonn, Germany, 1989. Springer, Berlin, Heidelberg, 106–126. Available online: http://dx.doi.org/10.1007/978-3-642-83814-9_6 [Accessed 20/4/2017].

Redmill, F. (2010) *ALARP Explored*. Newcastle upon Tyne, UK: School of Computing Science, University of Newcastle upon Tyne. Available online: http://eprint.ncl.ac.uk/pub_details2.aspx?pub_id=161155 [Accessed 18/4/2017].

Redmill, F. (2000) Understanding the use, misuse and abuse of safety integrity levels. *Eighth Safety-critical Systems Symposium*. Southampton, UK, 2000. Springer, 8–10. Available online: http://dx.doi.org/10.1.1.95.7797 [Accessed 21/4/2017].

Robens, Lord, Beeby, G.H., Robinson, S.A., Shaw, A., Windeyer, B.W., Wood, J.C. & Wake, M. (1972) *Safety and Health at Work: Report of the Committee, 1970-72*. London, UK: HMSO.

RTCA (2011) *DO-178C Software considerations in airborne systems and equipment certification*. Radio Technical Commission for Aeronautics (RTCA). Available online: http://www.rtca.org/store_product.asp?prodid=803 [Accessed 19/4/2017].

RTCA (2000) *DO-254 - Design Assurance Guidance for Airborne Electronic Hardware*. Radio Technical Commission for Aeronautics (RTCA). Available online: https://my.rtca.org/NC__Product?id=a1B36000001IcjTEAS [Accessed 15/5/2017].

Rumbaugh, J., Jacobson, I. & Booch, G. (1999) *Unified Modeling Language Reference Manual*. Reading, MA, USA: Addison Wesley Longman, Inc.

Rushby, J. (2015) *The interpretation and evaluation of assurance cases*. Menlo Park, CA, USA: Stanford Research Institute (SRI) International. Available online: http://www.csl.sri.com/users/rushby/papers/sri-csl-15-1-assurance-cases.pdf [Accessed 18/4/2017].

SAE (2010) *ARP4754-A: Guidelines for Development of Civil Aircraft and Systems*. Society of Automotive Engineers (SAE). Available online: http://standards.sae.org/arp4754a/ [Accessed 18/4/2017].

SAE (1996) *ARP4761: Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment*. Society of Automotive Engineers (SAE). Available online: http://standards.sae.org/arp4761/ [Accessed 21/4/2017].

Savage, I. (2013) Comparing the fatality risks in United States transportation across modes and over time. *Research in Transportation Economics*, 43(1), July, 9–22. Available online: http://dx.doi.org/10.1016/j.retrec.2012.12.011 [Accessed 19/4/2017].

Schwaber, K. (2004) *Agile Project Management with Scrum*. Microsoft Press.

Sharvia, S. & Papadopoulos, Y. (2011a) IACoB-SA: An approach towards integrated safety assessment. *IEEE Conference on Automation Science and Engineering (CASE), 2011*. Trieste, Italy, 2011. IEEE, 220–225. Available online: http://ieeexplore.ieee.org/abstract/document/6042514/ [Accessed 21/4/2017].

Sharvia, S. & Papadopoulos, Y. (2011b) Integrated Application of Compositional and Behavioural Safety Analysis. In *Dependable Computer Systems*. Springer, Berlin, Heidelberg, 179–192.

Sierksma, G. (2001) *Linear and Integer Programming: Theory and Practice, Second Edition*. CRC Press.

Sljivo, I., Gallina, B., Carlson, J., Hansson, H. & Puri, S. (2014) A method to generate reusable safety case fragments from compositional safety analysis. *Software Reuse for Dynamic Systems in the Cloud and Beyond (ICSR 2015)*. 2014.253–268. Available online: http://dx.doi.org/10.1007/978-3-319-14130-5_18 [Accessed 20/4/2017].

Stamatelatos, M., Vesely, W., Dugan, J., Fragola, J., Minarick, J. & Railsback, J. (2002) *Fault tree handbook with aerospace applications*. Washington, DC, USA: National Aeronautics and Space Adminstration (NASA).

Stroustrup, B. (2014) *Programming: principles and practice using C++*. Pearson Education.

Sullivan, K.J., Dugan, J.B. & Coppit, D. (1999) The Galileo fault tree analysis tool. *Twenty-Ninth Annual International Symposium on Fault-Tolerant Computing, 1999. Digest of Papers.* Madison, WI, USA, 1999. IEEE, 232–235. Available online: http://dx.doi.org/10.1109/FTCS.1999.781056 [Accessed 21/4/2017].

The SEI AADL Team (2005) *An extensible open source AADL tool environment (OSATE)*. Software Engineering Institute (SEI). Available online: http://www.aadl.info/aadl/downloads/AADLToolUserGuide1.0.pdf [Accessed 16/5/2017].

Toulmin, S.E. (2003) *The Uses of Argument*. Cambridge University Press.

TRR (1994) *The Railways (Safety Case) Regulations 1994*. London, UK: HMSO.

USA Government (1964) *e-CFR: Title 14: Aeronautics and Space*. *Electronic Code of Federal Regulations*. [Online]. Title 14: Aeronautics and Space.Available online: https://www.ecfr.gov/cgi-bin/text-idx?SID=e5bbd444e72b1a2660cd996906c68ad0&mc=true&node=pt14.1.25&rgn=div5#sp14.1.25.f [Accessed 15/5/2017].

Walker, M.D. (2009) *Pandora: a logic for the qualitative analysis of temporal fault trees*. Hull, UK: The University of Hull. Available online: http://ethos.bl.uk/OrderDetails.do?uin=uk.bl.ethos.518653 [Accessed 21/4/2017].

Wallace, M. (2005) Modular architectural representation and analysis of fault propagation and transformation. *Electronic Notes in Theoretical Computer Science*, 141(3), 53–71. Available online: http://dx.doi.org/https://doi.org/10.1016/j.entcs.2005.02.051 [Accessed 15/5/2017].

Weaver, R. (2003) *The safety of software: constructing and assuring arguments*. PhD Thesis. York, UK: University of York, Department of Computer Science. Available online: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.64.3627&rep=rep1&type=pdf.

Weaver, R., Kelly, T. & Mayo, P. (2006) Gaining Confidence in Goal-based Safety Cases. *Developments in Risk-based Approaches to Safety*. Bristol, UK, 2006. Springer, London, UK, 277–290. Available online: http://dx.doi.org/10.1007/1-84628-447-3_16 [Accessed 18/4/2017].

Wilkinson, P.J. & Kelly, T.P. (1998) Functional hazard analysis for highly integrated aerospace systems. *IEEE Certification of Ground/Air Systems Seminar (Ref. No. 1998/255)*. London, UK, February 1998. IET, 4/1-4/6. Available online: http://dx.doi.org/10.1049/ic:19980312 [Accessed 16/5/2017].

Wilson, S.P., Kelly, T.P. & McDermid, J.A. (1997) Safety case development: Current practice, future prospects. *Safety and Reliability of Software Based Systems*. Bruges, Belgium, 1997. Springer, London, UK, 135–156. Available online: http://dx.doi.org/10.1007/978-1-4471-0921-1_6 [Accessed 15/5/2017].

Wilson, S.P. & McDermid, J.A. (1995) Integrated analysis of complex safety critical systems. *The Computer Journal*, 38(10), 765–776. Available online: http://dx.doi.org/10.1093/comjnl/38.10.765 [Accessed 15/5/2017].