The Journal of Systems and Software 220 (2025) 112256

Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss

Andromeda: A model-connected framework for safety assessment and assurance ☆

Athanasios Retouniotis [a,*], Yiannis Papadopoulos [a], Ioannis Sorokos [b]

[a] Dependable Intelligent Systems Group, School of Computer Science, Cottingham Rd, Hull, HU67RX, East Riding of Yorkshire, UK
[b] Safety Engineering Department, Fraunhofer IESE, Fraunhofer-Platz 1, Kaiserslautern, 67663, Rheinland-Pfalz, Germany

ARTICLE INFO

Keywords:
Safety cases
Model-based assurance
Automation
Argument patterns

ABSTRACT

Safety is a key factor in the development of critical systems, encompassing both conventional types, such as aircraft, and modern technologies, such as autonomous vehicles. Failures during their operation can be potentially far-reaching and impact people and the environment. To certify these systems and enable their employment, regulatory bodies require, among others, a safety case. However, the growing complexity of modern systems and iterative nature of development pose significant challenges to the traditional approaches for creating safety cases that are still used in practice. Furthermore, safety cases are often generated in an ad-hoc manner and remain disconnected from system models and related artefacts. Without these connections it is difficult to construct the proper infrastructure for producing and maintaining safety cases in a structured manner throughout the system lifecycle. This paper presents our innovative method, Andromeda, and its underpinning metamodel, which establish connections between safety cases, system models, safety assessment activities aligned with international safety standards, and argument patterns. Automation is applied across various stages of the production of argument structures that support safety assurance and certification activities. Andromeda is complemented by tool-support designed to facilitate its application, and we demonstrate our work through a case study from the aviation industry.

1. Introduction

Technological systems are integral to the function of modern societies, shaping humanity through its ongoing evolution. Whether in the domain of energy production, where machines oversee the generation and dissemination of power, or in the transportation industries, such as the railway, aviation and automotive, with sensors and mechanisms that help control the behaviour of the vehicle, our reliance on this infrastructure is clear.

This dependence has not gone unrecognised by governments and regulatory bodies, which maintain regulations and standards to guide the development and use of safety-critical systems in order to avert potential societal hazards, acknowledging that failure of these systems could lead to catastrophic consequences. Safety standards do not have just an advisory role but they effectively constitute the basis of certification, which in turn ensures that a system meets the appropriate compliance levels and is acceptably safe to operate under a given context.

In the past, standards have been more prescriptive and assurance was governed by a rules-based approach (Leveson, 2011a). However, this rendered the introduction of novel technologies and software engineering techniques difficult since the respective authorities were not able to formulate universally applicable rules for these innovations and their fast-paced changes. In addition, due to the check-list mentality, system developers would invest considerable effort into meeting the specific requirements set by the standards and often overlook valuable contextual information, potentially diminishing the quality of the assurance artifacts submitted to the authorities (Haddon-Cave, 2009).

Consequently, the mentality shifted to a goal-based approach, where regulatory bodies set the certification goals and developers must provide explicit safety claims, a clear and comprehensive argument along with the supporting evidence that collectively help to meet these goals. This process, along with work products from other tasks such as requirement specification, architectural designs, testing and simulations results, are encapsulated in a series of documents known as the safety case.

The concept of the safety case has been widely adopted in the safety-critical sector, with industries such as the aviation, automotive

☆ Editor: Prof. Raffaela Mirandola.

* Corresponding author.
E-mail addresses: AthanasiosRetouniotis@outlook.com (A. Retouniotis), y.i.papadopoulos@hull.ac.uk (Y. Papadopoulos), ioannis.sorokos@iese.fraunhofer.de (I. Sorokos).

https://doi.org/10.1016/j.jss.2024.112256
Received 15 February 2024; Received in revised form 4 October 2024; Accepted 8 October 2024
Available online 15 October 2024

and nuclear leveraging their potential for over two decades. There are variations of the definition of safety cases but the most prevalent is found in Kelly et al. (1999)[p. 22], which describes it as a means to "communicate a clear, comprehensive and defensible argument that a system is acceptably safe to operate in a particular context".

This argument of safety can be presented in a variety of ways but practitioners tend to use graphical notations, such as the Goal Structuring Notation (GSN) (Assurance Case Working Group, 2021), to compact the information and aid in the construction and review processes. These notations typically include a variety of elements that represent all the essential entities necessary for an argument structure. For example, GSN uses rectangular shapes to depict goals (i.e. claims) or ellipsoids for assumptions and offers a set of different arrows to connect shapes and highlight their relationship. The practitioners are able to use these elements and produce an argument structure, in the form of a tree diagram, that explains how a goal is broken down into sub-goals and ultimately provide a reference of the supporting evidence. Note that notations typically provide rules to prevent illogical structures.

Moreover, to further facilitate the creation of argument structures, inspired by the concept of software design patterns, safety engineers have also incorporated the use of patterns. Specifically, they abstract the details from previous successful arguments and re-employ their structure in appropriate scenarios, with the intent to facilitate the production of safety cases and improve argument consistency and robustness.

Regulatory authorities advise towards a proactive approach, in which safety activities are initiated at the earliest stages of the development lifecycle, and heavily emphasise the continuous improvement of the related processes. This helps identify and resolve any safety issues and establish a good safety culture (Ministry of Defence, 1996, p. 10) (International Organization for Standardization, 2018, p. 58). To maintain these principles, safety cases must follow the evolutionary nature of system development, requiring continuous updates that align with ongoing system changes.

Considering that the majority of evidence artifacts are generated through safety analysis techniques and testing, coupled with the repetitive development process, it becomes evident that this is an arduous task, particularly when a substantial portion of the activities relies on manual processes. Even in the instances where safety assessment is handled by supporting-tools semi-automatically, there is considerable manual work towards the assurance and evaluation of the safety case (Nair et al., 2015). For example, the safety argument needs to be adjusted as changes occur, re-evaluated to ensure that its reasoning remains consistent, re-assessed to validate and verify its supporting safety artifacts, as well as analysed for valuable feedback from the rest of the development teams (Hawkins et al., 2015). In addition, the volume of a safety case scales accordingly as systems become more complex and their architecture increases in size.

Thus, the construction and maintenance of safety cases is far from trivial, and to avoid costs and effort, safety can sometimes be neglected during certain stages of development (Leveson, 2011b). However, this is inappropriate not only because safety standards recommend that safety assurance should remain up-to-date at all times, but also because as the system model evolves, our considerations and understanding of its behaviour can potentially change. For example, with each design iteration, our understanding of undesired events, their causes, and relevance could change drastically. This suggests that particular evidence artifacts might not be relevant anymore, invalidating the safety claims they support. After all, part of the safety case depends on assumptions, initial requirements and predictions, necessitating the reconstruction of the safety argument if any such changes occur.

With the emergence of contemporary intelligent systems or autonomous vehicles the maintenance of safety cases requires even more attention, not only during the production stages, but also during operation (Denney et al., 2015). These systems often employ advanced software and machine learning techniques that allow them to evolve either to improve their performance or to respond to changes happening in their environment. As a result, if the safety case remains static it is more likely that there will be a mismatch between the initial assumptions and the evolving behaviour of the system. That would effectively undermine the initial safety argument constructed during the development lifecycle.

As already mentioned earlier, certification is governed by international standards. For instance, ISO26262 is the major guideline for achieving functional safety in the automotive domain. It provides a general framework involving a series of tasks developers should follow including the definition of safety goals, specification and allocation of requirements, verification and validation processes, and specific safety analysis techniques, such as Fault-Tree Analysis (FTA), they could employ to assess their system.

Analysis techniques play a significant role across different stages of the safety lifecycle. In the concept phase, engineers employ Hazard Analysis and Risk Assessment (HARA) to identify and assess possible hazards at the earliest opportunity. In the phase of product development at system level, techniques like FTA and Failure Mode and Effects Analysis (FMEA) help evaluate the effects of system-level failures. Furthermore, the ISO 26262 requires the creation of a safety case that gradually captures any outcomes generated during the safety lifecycle (International Organization for Standardization, 2018, p. 51). Given that safety analysis and assurance activities unfold iteratively throughout the different stages, implementing systematic processes and automation becomes essential for ensuring that safety tasks keep pace with the development process, effectively addressing the aforementioned challenges, and improving the safety framework.

The introduction of the Model-Based Safety Analysis (MBSA) paradigm empowered engineers to link system architectures with assessment-related activities and their outcomes (Joshi et al., 2006). Typically, methods embracing the MBSA create an unambiguous model enriched with elements from the system architecture, failure information and formalised properties. This is achieved with advanced metamodeling that captures relationships between the nominal and other models. Furthermore, MBSA, by harnessing the power of software-tools and combining it with (semi-)formal models, facilitated the adoption of automation. Thus, alleviating inconsistencies between generated safety products, induced by traditional or other manual analysis methods, whilst staying in sync with the fast pace of modern development. Due to these benefits, various approaches, including Andromeda, have utilised and further expanded the concept of MBSA towards safety assessment and assurance. In Section 5: Related Work, we will explore notable examples of MBSA applications and highlight how Andromeda relates to them highlighting areas where it advances the state-of-the-art.

As systems become increasingly complex to improve functionality, addressing the associated challenges stemming from this complexity remains a persistent issue. This paper elaborates on our approach to support the assessment and assurance process, with the goal of integrating safety as a foundational element that drives system design. Andromeda tackles the aforementioned challenges by identifying common elements in safety standards and utilising them to guide safety assurance, while incorporating automation into multiple stages of the safety framework. Key contributions of the method entail:

1. **Assurance Patterns:** We elicited patterns that reflect commonalities between emerging standards and use them for the construction of safety arguments.
2. **Automated Synthesis:** We automated the synthesis of safety arguments from system models, safety analysis artifacts and assurance patterns via algorithms.
3. **Metamodel:** We designed and implemented an abstract structure that links system models with safety assessment and assurance processes.

4. **Cost-Optimal Requirement Allocation:** We showcased how the concept of abstract requirements, known within the standards as Safety Integrity Levels (SILs), can be utilised as a means to form argument structures. We also provide a way to cost-optimally allocate them to subsystems and components using metaheuristics.

Andromeda adheres to established safety standards, such as ARP4754-A for the aviation and ISO 26262 for the automotive industries. It employs automated processes to generate safety artifacts, which are then used for the automatic generation of argument structures via argument pattern instantiation. This effectively incorporates safety considerations into the system development process and produce essential outputs required for system certification. The metamodel developed to underpin our approach ensures that analysis results, requirements and system architecture are interconnected.

1.1. Structure of the paper

Following this introduction, the rest of this paper unfolds as follows. In Section 2, we present a brief summary of the essential elements related to the guidelines used for our case study. In Section 3 we outline an overview of Andromeda, its underpinning metamodel, and the necessary inputs and generated outputs. These sections help the reader familiarise themselves with our method and grasp the basic aspects before delving into the case study in Section 4. In Section 5, we provide information on related work and explain how our approach sets itself apart. Finally, in Section 6 we conclude and discuss Andromeda's limitations, evaluations, and potential avenues for future work.

2. Background

Andromeda is versatile and requires only moderate adjustments to adapt to various safety-critical domains. However, the case study presented later originates from the civil aviation industry. This section briefly introduces the safety framework recommended by ARP4754-A, a standard widely adopted in the sector.

Certification in civil aviation focuses on the safety of aircraft and their embedded systems, primarily overseen by the International Aviation Organisation (ICAO), the Federal Aviation Administration (FAA) in the United States, and the European Aviation Safety Agency (EASA) in Europe. The differences between FAA and EASA standards are minor, making the method applicable to either. The ARP comprises mainly two documents, the ARP4754-A, which forms the basis for the development of aircraft systems and their functions, and the ARP4761, which serves as a process handbook explaining how these advised processes can be performed. Additionally, there are other supporting documents, such as the Document Objects (DO), with the most notable being the DO-178C and DO-254 for the development of software and hardware components, respectively. In this section, we discuss only the key elements from the safety framework found in ARP; however, similar concepts are integrated into guidelines in other industries.

The systems development lifecycle, in many safety-critical sectors, typically follows the V-model approach to help navigate development milestones. As shown on the left side of Fig. 1, the lifecycle has three main phases: (a) system design, (b) implementation and (c) integration and testing, before the system proceeds to production and deployment. Development initiates with defining the concept, functionality, and specifying requirements. These requirements are gradually decomposed into lower levels before an informal architecture is designed. Following numerous iterations of requirement refinement and decomposition, the process advances to implementation, component integration, and testing, before the system finally reaches the production and deployment stages. As previously mentioned, it is crucial for the safety process to align with development from the early stages. Therefore, the guidelines provide a framework to integrate safety activities within each phase of development. On the right side of Fig. 1, the distribution of safety-related tasks across each phase is presented, in accordance with the ARP guidelines.

Table 1
DAL allocation based on severity classification (Radio Technical Commission for Aeronautics, 2011, p. 13).

DAL	Severity	Description of Effects
A	Catastrophic	Airplane Loss / High-Mortality Accident
B	Hazardous	Serious Injuries / Low-Mortality Serious function/safety margins restriction
C	Major	Major physical discomfort Significant function/safety margins restriction
D	Minor	Minor physical discomfort Slight function/safety margins restriction
E	No Safety Effect	No Effects

The validation processes determine whether all the identified requirements are appropriate and suitable against the higher-level requirements, ensuring that the system fulfils its intended purpose. In contrast, verification tasks assist in understanding whether the system implementation conforms to the specified requirements, confirming that the system is built according to the design and specifications. These tasks help build the confidence required that the system is reliable and safe before moving into production.

The first key element from standards is the safety lifecycle, outlining the sequence in which safety activities are performed for different stages, often represented graphically using the V-model.

The second key point involves the concept of Safety Integrity Levels (SILs), an abstract requirement that dictates the level of rigour required for assessment activities. While these requirements share a similar foundation as a concept, differences exist among standards regarding the criteria that influence the decision-making of the allocation process. For instance, in the civil aviation, SILs allocation is based on the severity of the identified hazard and its probability of occurrence. In the automotive sector, allocation is also influenced by exposure and controllability traits. The former reflects the frequency of the vehicle's exposure to the hazard, whereas the latter focuses on system-user's ability to gain control of the hazardous situation.

Since our case study is drawn from the aviation industry, it is important to note that the concept of SILs is known as Development Assurance Levels (DALs). Even though SILs is the generic form of abstract requirements which we still mention later, when appropriate, we will refer to the concept of abstract requirements as DALs. The standard defines five classes of DALs, ranging from E to A, with E representing the least stringent requirements and A the most stringent. For each identified hazard, the appropriate DAL should be appointed based on the severity classification and the description of the effects, as shown in Table 1:

The process of DAL allocation begins once the high-level functional model has been constructed and analysed with Functional Hazard Analysis (FHA). The results of this analysis include all hazards, also referred to as the Failure Conditions (FCs) within the ARP, associated with each function defined on that specific design cycle, and then the appropriate DAL is assigned on each of the corresponding hazards for every defined function.

Once the system architecture starts forming, developers can allocate DALs to lower parts of the architecture in a top-down manner. For this purpose, the standard utilises the Functional Failure Sets (FFSs), more commonly known as minimal cut sets. These represent the minimal combination of basic events that result in the occurrence of the top event, in this case, the hazard. The guidelines suggest to use FTA or Markov Analysis to examine the system and identify all the FFSs that can cause the top-level hazards.

To assist with the allocation of SILs, the guidelines provide a set of rules, which also differ across industries. For instance, under the

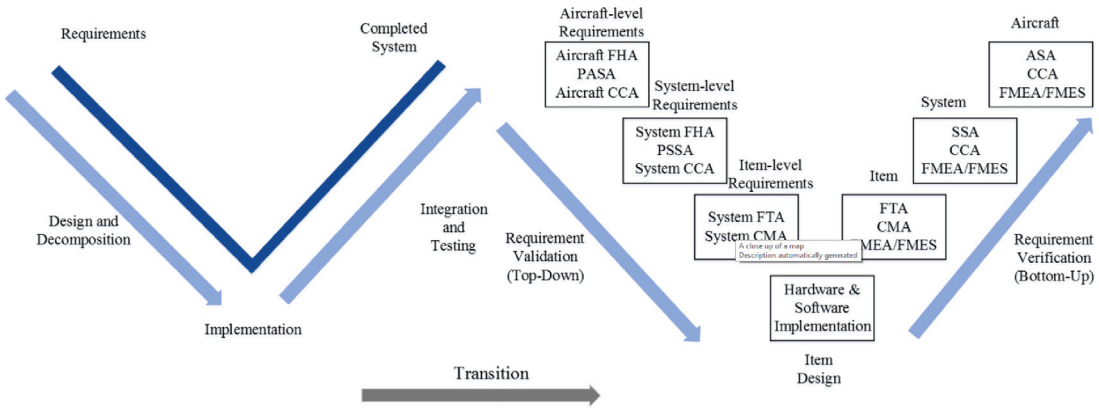

Fig. 1. The development lifecycle under the V-Model (SAE International, 2010).

ARP4754-A, if an FFS contains only a single member, then the DAL assigned to that element is identical to the DAL allocated for the hazard. In cases where the FFS contains two or more members, then the developer can follow the rules prescribed by the guidelines (SAE International, 2010, p. 44).

- One of the members of the FFS that contribute to the top-level hazard is assigned the same DAL with the parent system, whereas the rest of the members are assigned a DAL equal or up to two levels lower than the parent-system.
- Two of the members of the FFS are assigned a DAL one level lower than the parent system, whereas the remaining members are assigned a DAL equal or up to two levels lower than the parent system.

Both of these options plausible, and the decision is up to the developers and specific implementation details. However, a challenge arises as the system size increases, leading to numerous possible combinations that often result in combinatorial explosion. Additionally, the repetitiveness of this process renders any manual approaches impractical. Consequently, substantial research, such as in Bieber et al. (2011), Azevedo et al. (2013) (Sorokos et al., 2015), has been focused on automating the SILs allocation process at the system architecture level, whereas in Frigerio et al. (2019), the authors automated the SILs decomposition and utilised it to modify the system architecture, adjusting the redundancy of elements during system implementation.

As already mentioned, SILs represent the level of rigour assigned to safety assessment activities, leading to increased effort and developmental costs. Therefore, engineers must examine the different configurations of allocated SILs and, based on the cost of a system-wide allocation, choose the most cost-efficient option.

Achieving this involves defining a basic cost value and assigning each DAL a unique multiplier to calculate the overall cost. For example, DAL A can have a multiplier of 10, DAL B a multiplier of 9, and DAL C a multiplier of 5. This scaling helps understand that the effort required between the different activities does not progress linearly across different DAL classifications and that is reflected on the final cost once the multiplier is applied.

Thus, considering the vast variety of configurations and the size of the architecture, it becomes an intractable problem for manual approaches. Therefore, developers turned to optimisation methods where the SILs' allocation is viewed as a constrained optimisation problem. The objective is to minimise the costs, the constraints are the decomposition rules and the decision variables are the SIL values throughout the architecture. Due to the strict timeframe of development, engineers often opt for approximate methods like metaheuristics, known for delivering relatively good results within a reasonable timeframe.

Overall, optimisation techniques provide an additional layer of automation that benefits system development.

3. The approach

3.1. Overview of andromeda

This section presents Andromeda, our method for the safety assessment and assurance of safety-critical systems. In formulating our method, it was imperative to incorporate processes outlined by industry guidelines. This not only demonstrates compliance but also instils confidence, as guidelines comprise best practices that have evolved over the years and are crucial for certification.

Additionally, recognising the significance of leveraging automation was a key factor to alleviate the overall process and address some of the challenges associated with traditional techniques and the iterative development. Our investigation into functional standards, such as the ARP4754-A, and various approaches that enable automation, including the MBSA paradigm and argument patterns, guided the development of Andromeda.

Fig. 2 provides in an illustrative way how Andromeda as a method was conceptualised and eventually implemented from safety standards. The rectangles are abstract representations of information. The coloured rectangles, represent the information encapsulated in different guidelines (e.g. ISO).

Functional safety standards typically specify individual safety frameworks; however, they share common elements and processes, which if abstracted from specific details it can contribute to a generic framework for safety. The grey rectangle, represents the framework that Andromeda proposes for safety assessment and assurance which was elicited from the common elements among these standards.

Expanding upon this concept, we enhanced it by incorporating features from MBSA, argument structure notations (i.e. GSN) and argument patterns. The rest of the shapes represent the transition from that framework to a metamodel (high-level design), which captures the links between the assessment activities (of the framework), analyses techniques (suggested by standards), argument pattern (that can be created by the practitioner in Andromeda), system architecture, allocated requirements, and the concrete argument itself.

The "Software Design" shape represents the implementation of the metamodel in a software tool, describing the software architecture and its corresponding data structures that capture system model and safety-related artifacts. Note that, the software design block has three coloured blocks attached that match the respective standards. This is to signify that implementation details vary to account for standard differences such as the requirement decomposition rules or the different criteria during the initial SIL allocation.

Finally, the "System architecture" rectangle, is the potential system architecture and the annotated information the user can create in the software tool such as failure data, contextual information and evidence artifacts. Lastly, there is an additional lower-level abstraction layer,

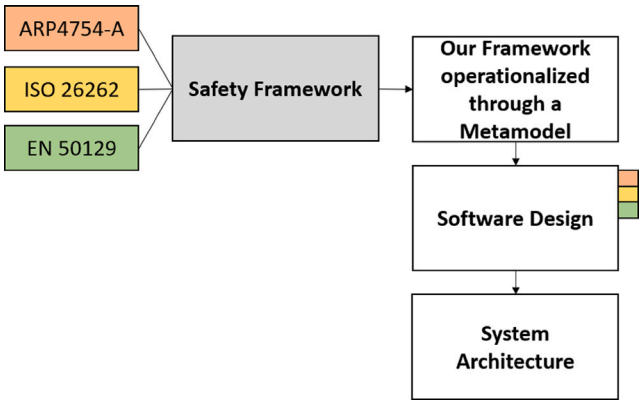

Fig. 2. Transition from standards to the metamodel and tool.

not shown here, that represents potential raw data from manual analyses, tests or stakeholder requirements. These can also be incorporated within the system model and presented, upon demand, in the argument structure.

3.2. The pattern

Safety standards consider safety as a system property that should be controlled as early as possible and maintained throughout development. Once the initial FHA is conducted, and the preliminary system design is formulated, integrity requirements (i.e. DALs) suitable to fulfil the higher-level requirements can be determined.

To further assist on that front, the guidelines have provided a scheme for allocating requirements. In this context, by scheme, we mean a systematic plan for allocating requirements. For example, guidelines suggest a top-down approach for the allocation and they provide a set of rules for the decomposition of those requirements from high-level to lower-levels. Specifically, high-level DALs can then be decomposed and allocated accordingly based on architectural dependencies and failure propagation as shown on the left side of Fig. 3.

As the design matures, the whole process of analysis and requirement allocation are revisited. The cycle concludes upon completion of development, with the establishment of an acceptably safe system substantiated by sufficient evidence, confirming the systematic mitigation of all identified hazards via the appropriate allocation of subsystem and component safety integrity requirements.

Andromeda utilises this scheme as a mechanism for providing structure to the safety argument following the logic mentioned in the right side of the figure. Note that this process is applicable for both validation and verification activities, aligning with the V-model, as recommended by the safety guidelines for the development of systems (SAE International, 2010, p. 24).

3.3. Method integration

Andromeda is an integrated approach which incorporates methods that our research group has been developing over the years. One of the important facets of this integration is the automation capabilities, which can be used to significantly facilitate the construction and maintenance of safety arguments. While the early phases still require manual effort and input, such as functional design and high-level requirements specification, the safety analysis of the architecture can be automated using the embedded analysis techniques and algorithms. Andromeda further extended on the assessment capabilities by adopting the different activities, typically conducted in the aviation sector, such as the Preliminary Aircraft Safety Assessment (PASA). The latter is useful for establishing a list of failure conditions on the aircraft-level functions and the appropriate requirements for addressing them.

Following the scheme explained earlier, we used that logic behind the argument structure as a basis and combined it with the concept of argument patterns. On this front, collaborative groups such as the Assurance Case Working Group (ACWG) have extended graphical notations to support argument patterns with concepts like element and structural abstractions.

For example, GSN's structural abstraction is achieved via the multiplicity and optionality extensions. The former generalises n-ary relationships between argument elements by iterating similar arguments over them. The optionality extension captures alternative relationships between argument elements, effectively allowing conditional connections and the creation of different paths based on varying circumstances.

Andromeda primarily employs GSN as the main graphical notation to both build the pattern and present the concrete argument. GSN was our first choice due to its extensions of modularity and structural abstraction fitting the purpose, as well as its widespread acceptance in the domain.

It is worth noting, however, that comparable results could be attained using alternative notations such as the Claims Arguments Evidence (CAE) notation or the Structured Assurance Case Metamodel (SACM) (C.A.E. Framework, 2020; Object Management Group, 2018).

However, since GSN's model-based support is limited, Andromeda adjusted these entities in the metamodel with the necessary structures to capture relationships between the system model, evidence artefacts and argument patterns. This link between the pattern and the model information along with our implemented instantiation algorithm, allowed us to automatically generate concrete argument structures.

3.4. The metamodel

Since Andromeda involves various steps, progressing from functional design to safety argument structures, it is more suitable to witness it in practice in the following section instead of explaining all the steps here. However, before delving into practical applications, it is useful to explore the metamodel, which makes all this possible.

The HiP-HOPS approach already comes with an extensive metamodel that connects elements from system architectures, such as subsystems or components, ports acting as interfaces for information propagation, and safety evidence artefacts such as minimal cut sets and integrity requirements. Hence, we utilised it as a foundation for storing system-related information in order to exploit the HiP-HOPS capabilities.

As mentioned earlier, the metamodel is also enriched with elements from the GSN standard as a means to incorporate argument and pattern elements, structure rules and semantics. Moreover, the addition of an extensive data structure, referred to as Model Connecting Storage Unit (MCSU), helps store relevant information and interconnect all these components. Fig. 4 depicts a high-level view of what the metamodel entails.

Since standards provide the basic scheme for the argument structure, the requirement decomposition rules, and a guide for the appropriate safety analysis activities, they are also connected with both the MCSU structure and the system architecture. The connection with the architecture is important to notify the analysis and decomposition engine about specific details based on the corresponding system. For example, if the system is from the automotive industry, the decomposition rules will differ compared to a system from the aviation industry. In addition, the connection with the MCSU is important because, in some industries, the outcomes of certain processes might be directly usable as evidence artifacts in the final argument. For instance, under aviation guidelines, an FHA table could be used as evidence to support a claim arguing the correctness of DALs allocated to specific high-level functions. The implemented instantiation algorithm draws all the necessary information from the MCSU, and based on that and the pattern, it generates the concrete structure.

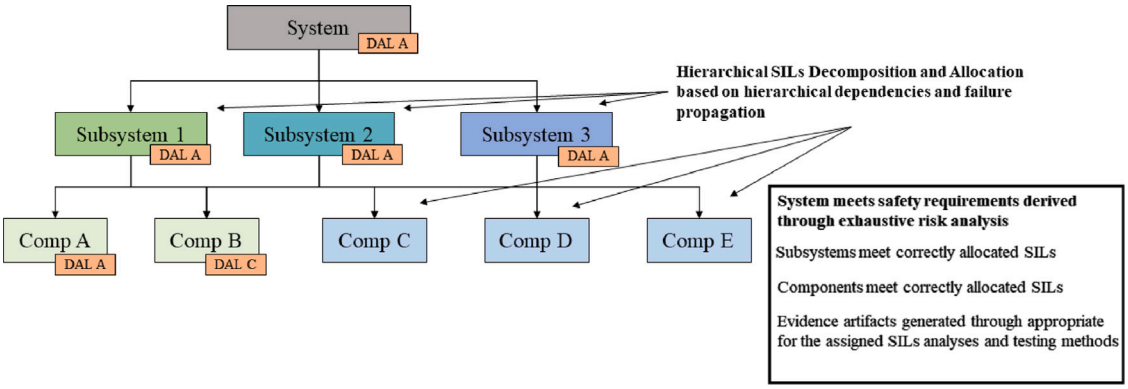

Fig. 3. DALs Allocation on a simple architecture that forms the basis for the argument structure.

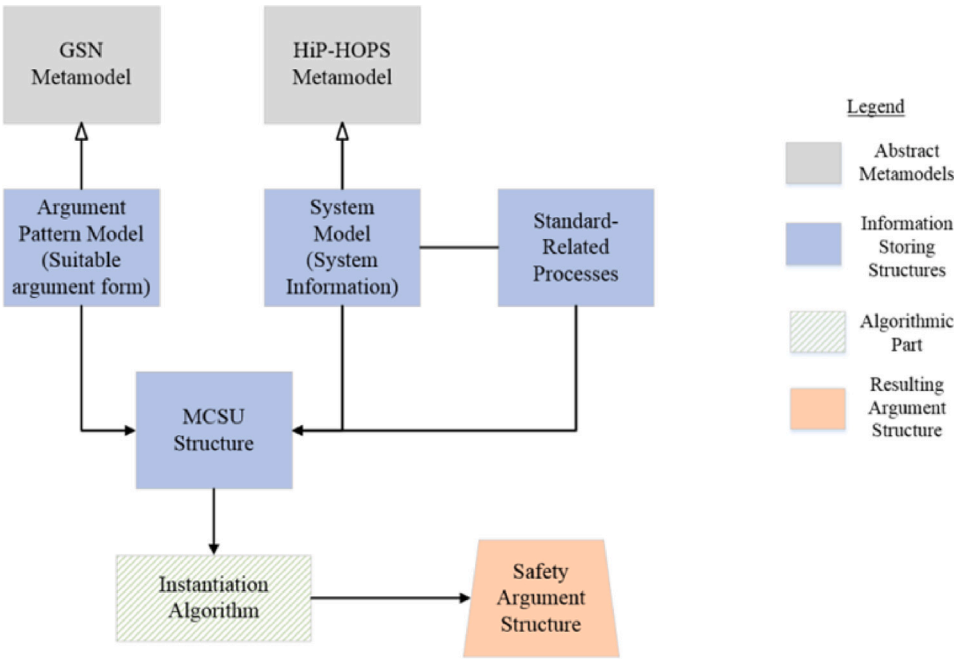

Fig. 4. Overview of the Metamodel.

Below, Figs. 5 and 6 present a more detailed view of the metamodel in Unified Modeling Language (UML) format. The first figure depicts how details of the pattern, model and argument elements entities are interconnected. The nodes are high-level nodes. For example, the "MCSU Element" node would be the equivalent of the SACM's "ArtifactElement". This is the base class for elements in our metamodel. Essentially, all elements which extend the MCSUElement, are MCSU entities. It is also a recursive container, meaning it can contain one or more sub-containers.

The second figure showcases the adapted version of the HiP-HOPS metamodel, which we utilise to store information about the system during the modelling phase. This adapted metamodel serves as input to the integrated HiP-HOPS algorithms, facilitating the initiation of the analysis and requirement allocation phases. Together, these figures offer a deeper understanding of the structural foundations of our approach and the interconnections between Andromeda's embedded elements.

The "Coordinator" entity serves as a manager class that oversees a significant portion of the tool's functionalities and is tasked with creating and maintaining the MCSU structure during a project. The MCSU encapsulates essential information about the argument, pattern and model structures, which inherit basic attributes and access to visitor classes from the "MCSUElement" entity.

Note, that the "MCSUElement" is developed using an adaptation of the composite pattern found in Erich et al. (1994)[p. 183], allowing elements to contain other elements; thus, forming a hierarchical structure where each element provides access to the nodes one level lower in the architecture.

The "Visitor" class represents our basic implementation of the software engineering pattern visitor, and supports each of the element types with various capabilities without them including any additional methods or members. Any required changes to the functionality can be achieved through an update to the visitor class.

While the argument and pattern elements share similarities, they also differ in the types of elements. For example, pattern elements have special types such as "IfThen" or "ForEach" as well as expressions and parameters to help follow the system architecture based on an abstract structure.

The "ModelElement" class holds all the information about the system architecture, the failure information and SIL requirements for each component through the "SystemContextData" attribute. This data is indispensable for the instantiation algorithm in order to properly build the argument structure.

Note that all the elements discussed so far (and the ones that follow) are represented within the tool as either object-oriented classes/objects

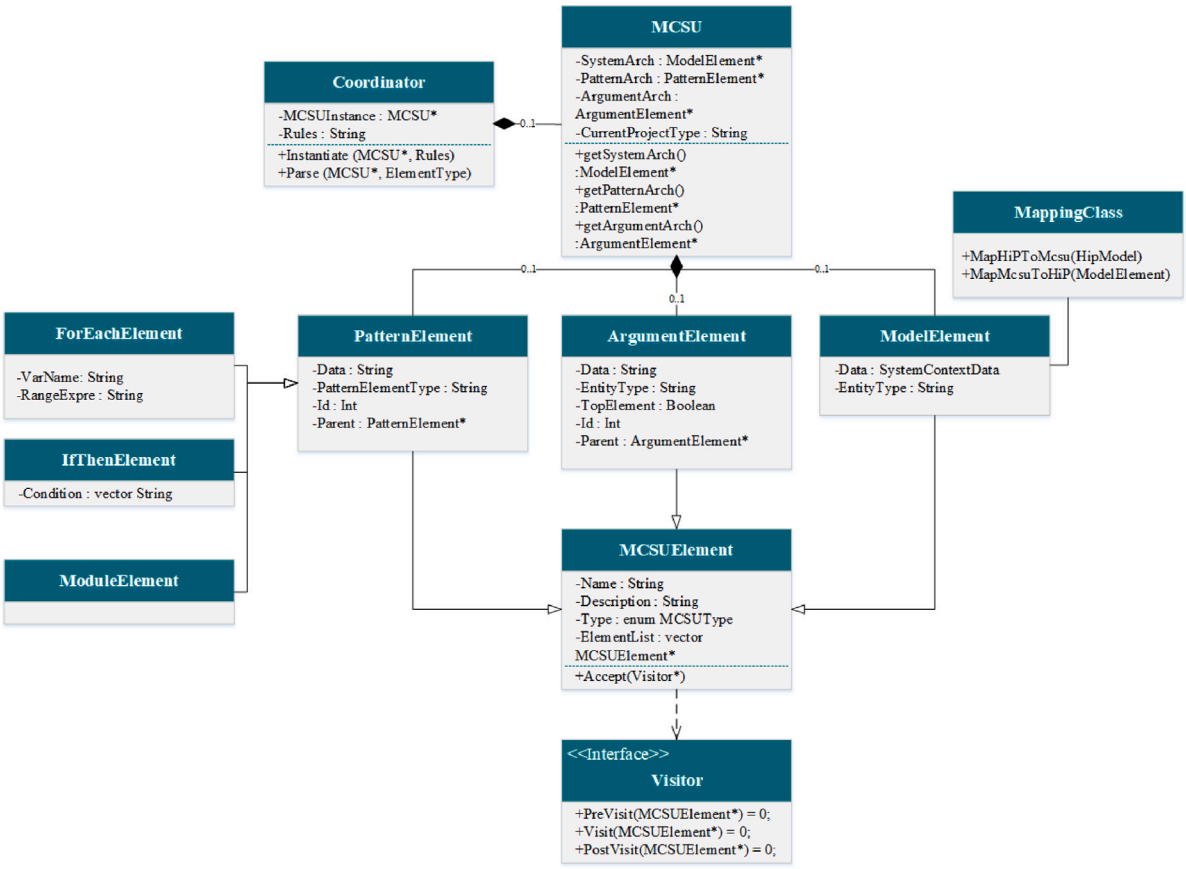

Fig. 5. Model, Argument and Pattern Elements under the MCSU structure.

or C-style structures. The related information for these elements, such as lists of comprising subsystems, properties, and descriptions, as well as mechanisms to retrieve data, are included as member variables, functions and properties. Base elements typically provide general characteristics that are inherited by their derived classes. For example, the MCSUElement in Fig. 5 is the base class for ModelElements and PatternElements. It establishes a foundational set of information, including an element name, description, and the accept method, which is commonly included in the base class.

During the modelling phase, we store the model element information in a separate structure through a model mapping process. This structure serves as input for gaining direct access to the HiP-HOPS engine, allowing us to generate minimal cut sets, FMEA tables and SILs. The "MappingClass" entity is responsible for this mapping process, facilitating the transfer of system information between the MCSU and the HiP-HOPS metamodel. Furthermore, we have implemented a reverse mapping (i.e. from MCSU to HiP-HOPS) for potential future work that could allow engineers to adjust the model information by manually applying changes to the argument structure.

In Fig. 6, the "HiPModel" class serves as the top-level node of the system, functioning as a container for the architecture. The "Perspective" helps distinguish the software from hardware aspects of the architecture as well as between different implementations. The "HiPSystem" are higher-level system entities that have components and lines to guide the connections between the various elements. These components have ports that help them connect through lines with other elements and enable behaviour propagation (both nominal and failure). In addition, the components can have potentially more than one implementation to account for components from different manufacturers. Finally, the "FailureData" class stores all safety related information such as basic events, output deviations and common cause failures for each of the components. As previously mentioned, this aspect of the metamodel is mainly utilised for the direct communication with the HiP-HOPS engine and cannot be accessed by the instantiation algorithm.

3.5. Inputs and outputs of andromeda

In this subsection, the inputs and outputs of Andromeda are outlined below:

The process can initiate during the functional design, effectively just before the FHA stage in the V-model as shown in Fig. 1 earlier. The method requires substantial input before it proceeds with the analysis of the system. Fig. 7 shows an annotation window dialog from the Andromeda tool, which is used for both inputting relevant data, including component failure rates and output deviations, as well as accessing additional dialog windows. The output deviations are the failures that can propagate throughout the architecture, such as Omission, annotated as O in the dialogue parameter. As mentioned earlier, all this input and output are stored within Andromeda in their respective data structures.

1. Selection of industry (Tool-based parameter for the software to be adjusted accordingly)
2. Function Identification and System Definition (Essentially the functional model and system architecture)
3. Failure Behaviour Information (Hazards, risk, high-level DALs)

The automated analysis process generates the following outputs:

1. fault trees
2. FMEA tables
3. minimal cut sets

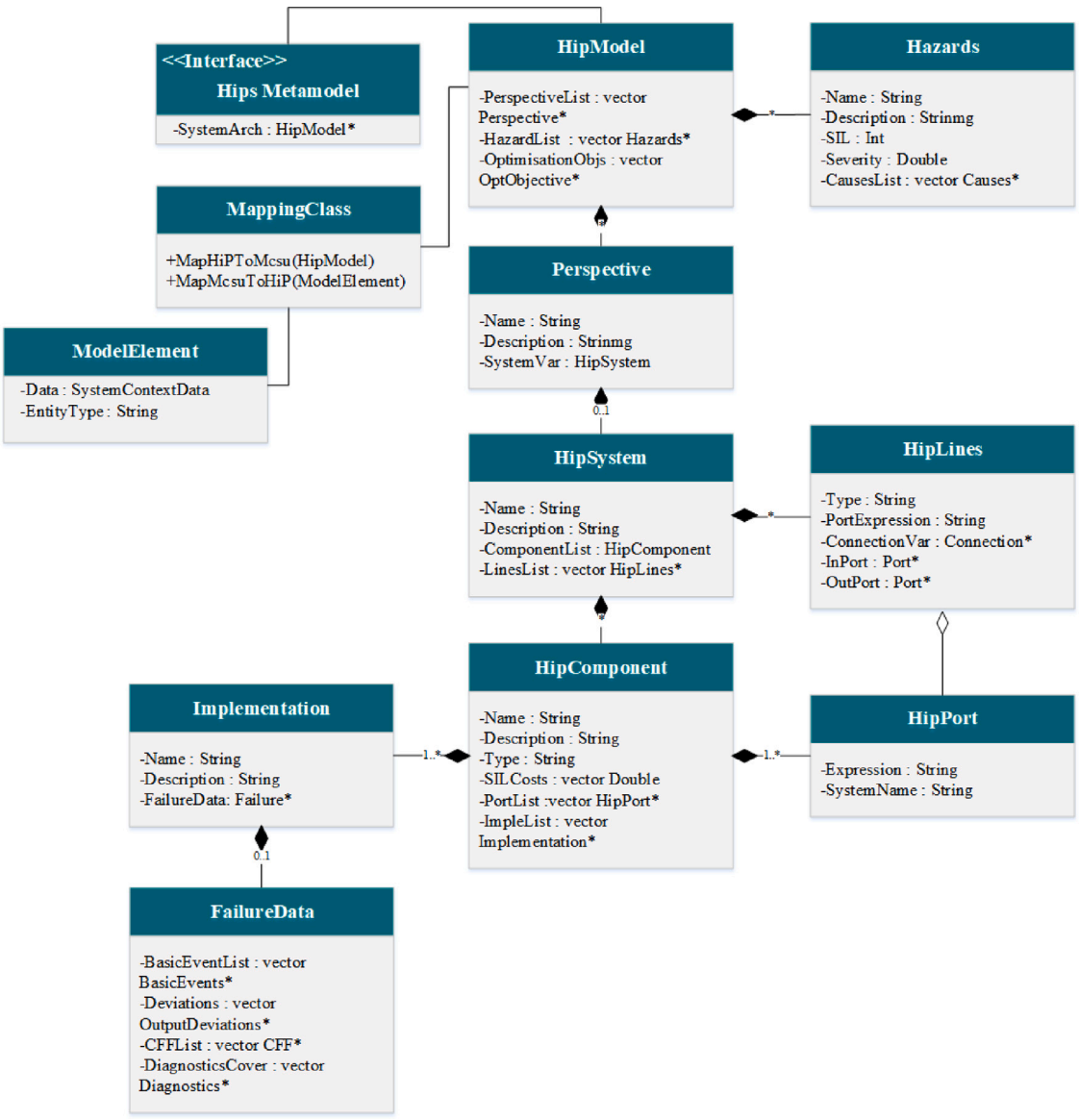

Fig. 6. The integration of the HiP-HOPS metamodel.

Based on this information and the initial high-level requirements converted into DALs, the tool can allocate the remaining requirements hierarchically and cost-optimally. At this point, the user can create the appropriate argument pattern for the industry in question within the tool. Then, by utilising the interconnected models, the pattern, and the instantiation algorithm, Andromeda generates the concrete argument structure. Fig. 8 illustrates how an arbitrary argument structure appears in the tool.

4. Case study

4.1. System details

This section explores the practical application of Andromeda through a case study drawn from the aviation industry, focusing on a brake system designed according to the principles employed in the Boeing 787 Dreamliner. The choice of this example is deliberate, given its contemporary technology and the widespread adoption of similar mechanisms across various aircraft.

In particular, the brake system exemplifies a shift from traditional hydraulic components to electronic counterparts and other characteristics. For example, it uses a carbon disc sensor unit that can assess the wear of the discs in real-time and also features its own data transmission unit to the cockpit. More importantly, this shift significantly reduces random failures which typically occur due to material degradation and require statistical techniques to predict. Instead, the most common faults of electric brakes are systematic, from either hardware or software components, which can be addressed with process or design measures via requirements (DALs) according to the guidelines (SAE International, 2010).

Before applying the method, it is useful to highlight details related to the system and its architecture. The brake unit is responsible for decelerating the aircraft and preventing any motion during extended periods on the ground. In some instances, it also contributes to steering through differential braking. Despite its seemingly straightforward function, commercial aircraft incorporate additional mechanisms such as spoilers, reverse thrust and other brake technologies like antiskid to enhance braking efficiency. Normally, engineers would assess these components both individually and collectively to ensure the safety of the braking function. However, for the sake of simplicity in this

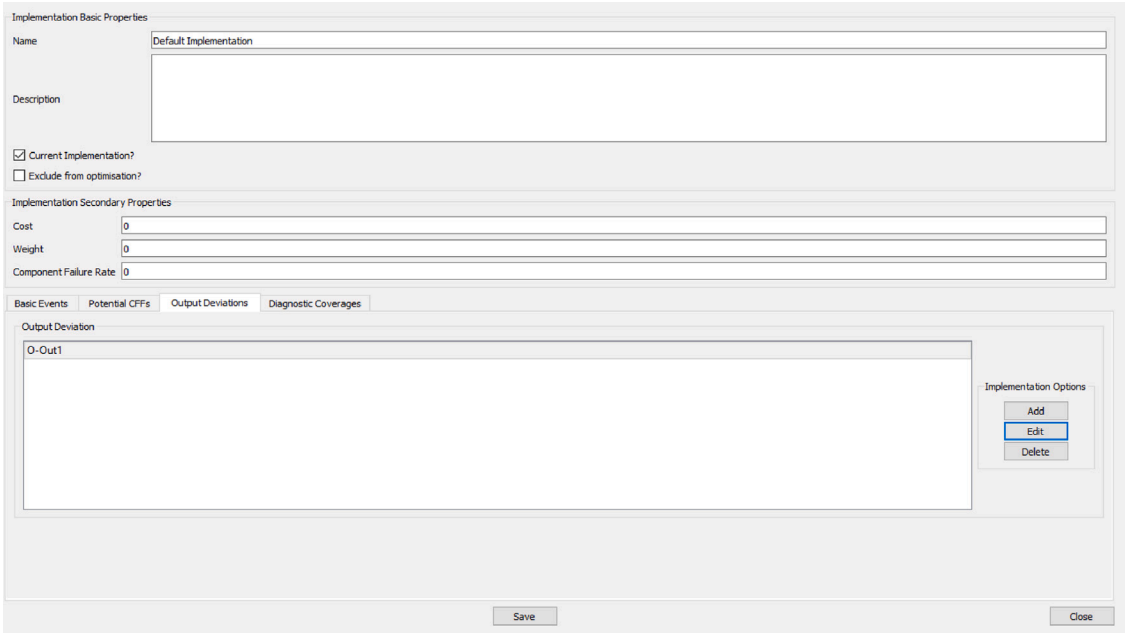

Fig. 7. Example of Failure Annotation Dialogue in the tool.

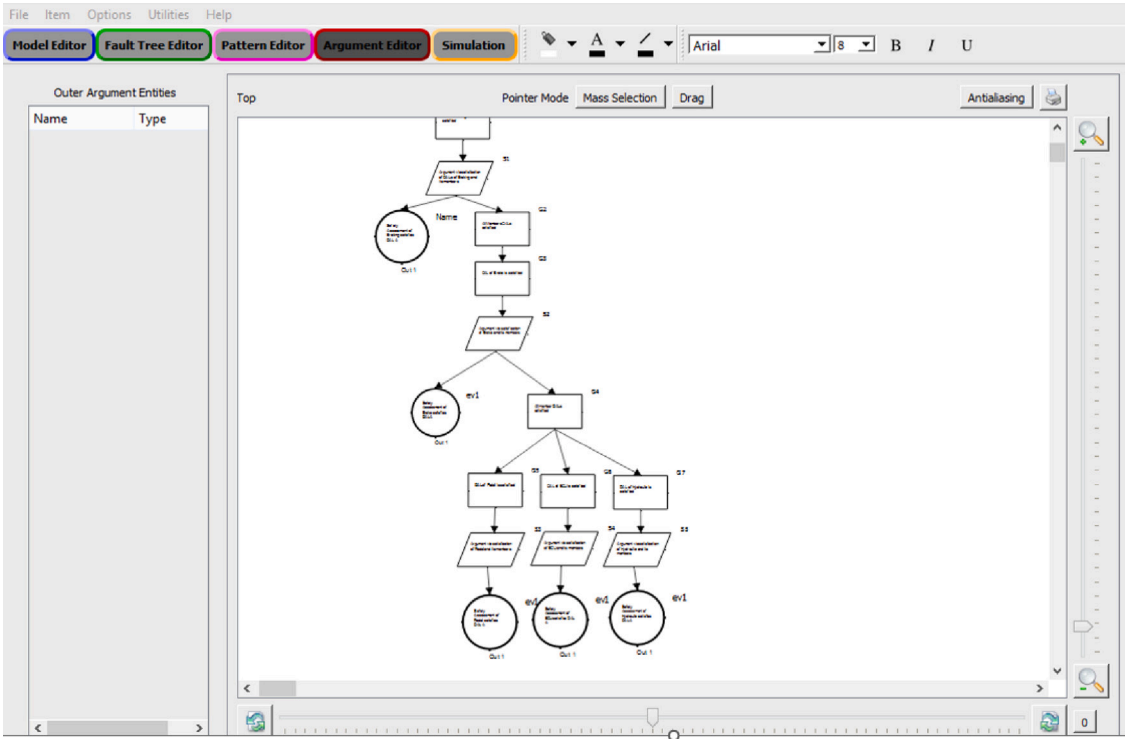

Fig. 8. Example of an argument structure within tool.

example, our focus is narrowed down to the wheel-brake system and relevant actuators and controllers.

The Boeing 787 Dreamliner was released in the early 2000s and comes in different models which vary in size and capabilities (Boeing, 2023). Despite following a similar architecture, the components and safety requirements might differ. For example, the difference in weight between the models suggests many variations in the architecture including potentially an alternative brake setup. In general, the aircraft includes 8 separate braking assemblies, one per main wheel, and they are a contemporary version of the multi-disc brake, known as the rotor-disc brake. Each assembly has a stack of 5 rotors and stators. The rotors are similar to discs, but they have small openings to allow water and heat to escape whereas stators are immobile flat plates covered in isolated blocks that have brake linings attached. When the brakes are activated, the pistons in the assembly exert pressure through a major

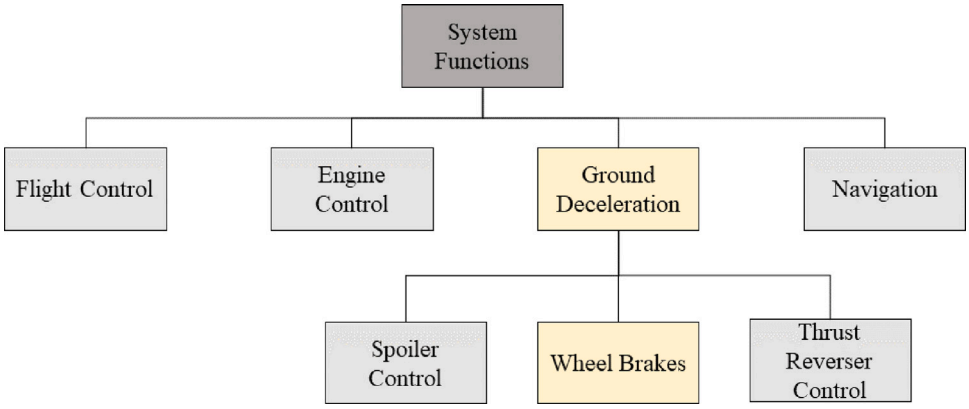

Fig. 9. Overview of High-level Functions.

plate which correspondingly imposes tension to the stack of rotors and stators and creates the friction needed for heavy-duty braking.

With the transition from hydraulic into electromechanical brakes, 787 Dreamliner also introduced a modern actuation mechanism that uses electricity to power the braking mechanism instead of fluids. Manufacturers can choose system components from different vendors, which changes the system drastically. Each configuration needs to be carefully evaluated, as it can alter safety requirements substantially.

4.2. Andromeda core phases

4.2.1. Functional modelling

Development typically commences with the function modelling stage, where a set of high-level functions is defined. As development advances, this process is reiterated to define functions that support the high-level functions and build a hierarchical functional model. This activity is integral for both system development and safety assessment.

On one hand, the work products of this stage are utilised by the engineers to design the initial system architecture. On the other hand, knowledge of the system functionality enables the identification of potential hazards and through the FHA it is possible to evaluate the risk involved and assign integrity requirements.

For these reasons, Andromeda initiates with the functional modelling stage. Naturally, the functional model would include several low-level functions, but to avoid an unnecessary and extensive diagram we only highlight an overview for demonstrative purposes in Fig. 9.

The first layer of this functional model includes only some of the major high-level functions typically found in commercial aircraft. In particular, the "ground deceleration" function involves all sub-functions and systems related to the deceleration capabilities for this aircraft.

The spoiler control and reverse thrust are linked with the ground deceleration function since they are mechanisms that assist that function; however, in practice they are mostly used during the landing phase. Thus, even though they are implied as sub-functions in this example to highlight this connection, their implementation is part of the navigation and engine systems respectively.

Normally, they would still be assessed for safety concerns when examining the ground deceleration, but they are excluded from this example as the focus is solely on the wheel brake system.

At this point, guided by a more mature functional model and accumulated experience, an initial system architecture can be established. The model for our example is shown in Fig. 10, followed by a brief explanation.

As we see in the figure, each pilot has access to two brake pedals, left and right, that send signals to the corresponding Brake System Control Unit (BSCU). This aircraft uses two separate processing units (BSCUs), one for each group of wheels, for the left and right sides respectively. Their responsibilities are to process input from pedals, send the commands to the Electric Brake Actuator Controllers (EBACs) and receive feedback from the brake assembly sensors in order to notify the cockpit or other automatic systems (e.g. autobrake).

The feedback is instrumental for the proper operation of brakes and correct control since it provides information such lining wear, wheel rotation rate and temperature. The EBACs main role is to interpret the signals from BSCUs, control the brake rate and send essential data for health monitoring. Each one of them comes with its unique power supply, shown in the image above as Electric Brake Power Supply Unit (EBPSU).

On each wheel there is a brake assembly that has four Electric Brake Actuators (EBAs), which are engaged through the actuator controllers (EBACs), and apply pressure on the stack of rotors and stators to decelerate the wheel.

The software parts of the processing units are validated and verified directly from the suppliers (e.g. Safran) as well as third-party companies. The methodologies used for the validation and verification activities were not specified, however it was clear that software development was in line with functional specifications through tests that evaluated the outcome based on specific inputs (black-box testing) (Nidhra and Dondeti, 2012).

For the purposes of this example, we consider the software and hardware parts as one unit and assess their interactions as a system. Finally, Boeing has to extensively test all vital parts of the system under both normal and irregular conditions. For example, part of the testing phase examines the aircraft's braking capabilities with overused brake linings or with several brake assemblies disabled to ensure performance is always within the required threshold.

Redundancy and autonomy are incorporated in many parts of the system architecture and with the existence of auxiliary systems for deceleration, such as the engine reverse thrust, supported separately from other sub-systems decreases the likelihood of common errors which indicates that functional independence is supported to an extent.

4.2.2. Risk assessment

The subsequent phase entails identifying hazards for each function, estimating risk, assigning a classification, and a DAL requirement. The results of the FHA activity for two specific functional failures are outlined in Tables 2 and 3.

In Table 2, we examine the effects of losing control over the ability of braking as a whole which could lead to a high-speed overrun with potentially high number of casualties. As a result, the hazard is classified as catastrophic, the acceptable likelihood of occurrence is set to a very low probability and the DAL assigned is A, which is the most stringent. Table 2 explores the possibility of losing the autobrake

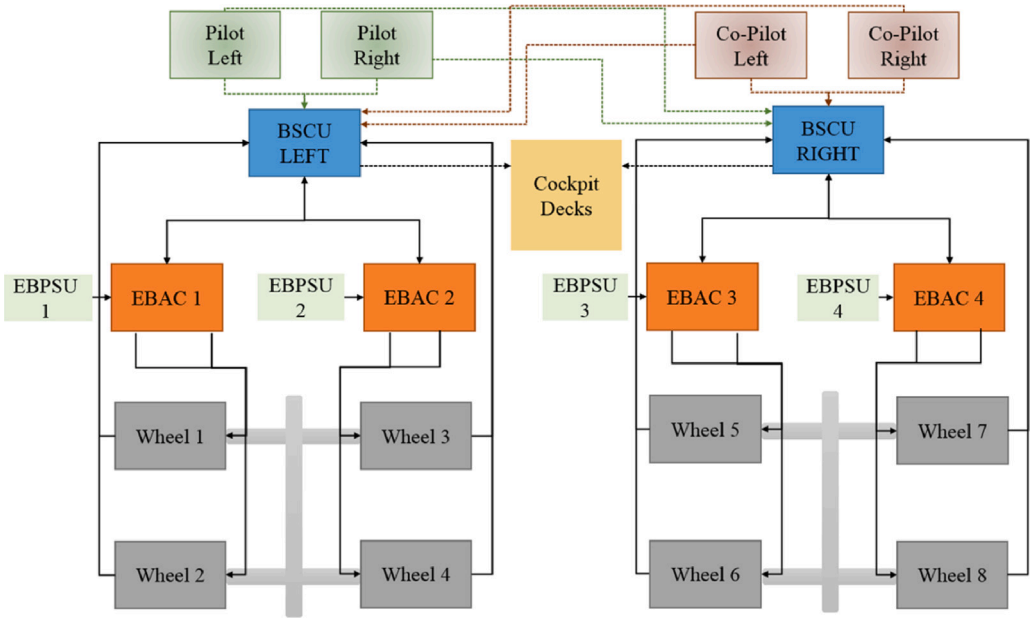

Fig. 10. Example Architecture based on Boeing 787 Dreamliner (Mare, 2017).

Table 2
Example Hazard Analysis and Risk Assessment Summary for Loss of Autobrake.

FHA Name	Function FHA Boeing Dreamliner 787	**Aircraft Phase**	Landing & Rejected Take-off
Function Name	Ground Deceleration	**Hazard Classification**	Catastrophic
Hazard Name	Loss of Deceleration Capability	**DAL**	A
Hazard Description	Unanticipated loss of deceleration capability	**System Reqs**	Likelihood occurrence less than 1E-9 per flight hour
Hazard Effect	Pilots unable to stop the aircraft	**Verification Method**	Aircraft Fault Tree

Table 3
Example Hazard Analysis and Risk Assessment Summary for Loss of Autobrake.

FHA Name	Function FHA Boeing Dreamliner 787	**Aircraft Phase**	Landing & Rejected Take-off
Function Name	Ground Deceleration	**Hazard Classification**	Major
Hazard Name	Loss of Autobrake	**DAL**	C
Hazard Description	Unanticipated loss of automatic brake	**System Reqs**	-
Hazard Effect	Automatic system fails to operate, and pilots switch to manual mode. Slow reaction time might lead to overrun and minor injuries	**Verification Method**	Software & Hardware Testing

function. In this case, the pilots are expected to take over and operate the braking manually. Even assuming slow reaction time, the estimated outcome of such an incident is only minor injuries and for this reason the hazard is assigned only a DAL C.

4.2.3. Failure behaviour annotation

The next phase in Andromeda is to determine the failure behaviour and annotate the system architecture. This process involves the modelling of the architecture within our software tool and providing the failure logic on each of the sub-systems and components. Hence in this part, we convey that behaviour with the use of a fault tree for clarity as depicted in Fig. 11. Note that in reality the system-level fault tree would demonstrate the behaviour of the system as a whole. For example, the top node could be the "Loss of Aircraft" which would be connected with all the potential lower-level hazards that could lead to that.

Furthermore, each function could potentially have multiple hazards and each of these hazards could have different facets. In our example below, we examine the hazard "Loss of Deceleration Capability"

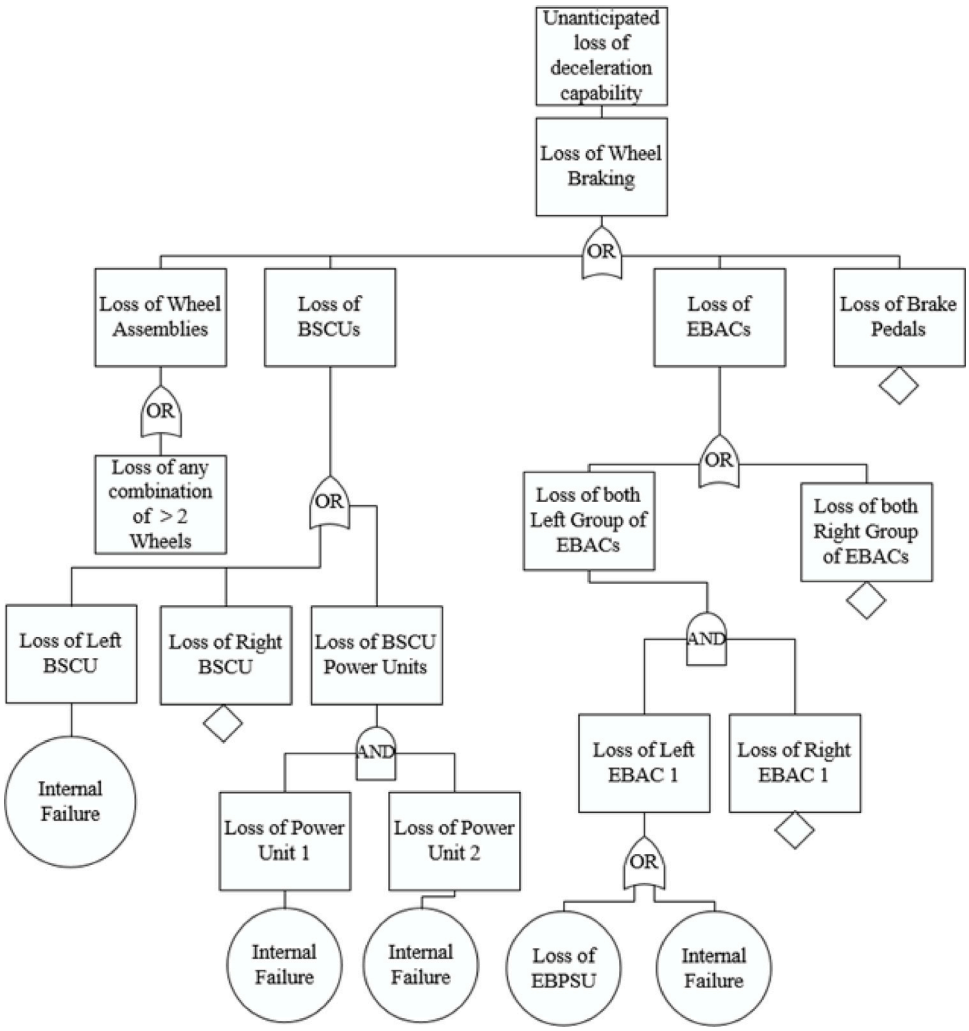

Fig. 11. Fault Tree for the Unanticipated loss of the wheel brake system.

from Table 3, but only for the failure condition "unanticipated loss of deceleration capability". That same hazard could also manifest as "inadvertent deceleration after an attempt for landing" with the same outcome of which is the loss of deceleration.

After careful evaluation, it is calculated that the "loss of wheel braking failure" can occur from any of the following conditions:

- Loss of brake pedals
  - Only possible if both pilots' pedals fail
- Loss of EBACs
  - Loss of both EBACs due to loss of power or internal hardware or software failure
- Loss of BSCUs
  - Failure of BSCU due to internal failure
  - Both BSCU power supplies fail
- Loss of Wheel-Brake Assemblies
  - Any combination of 3, or more, wheel assemblies fail or suffer damage

The conditions in this case are slightly simplified to facilitate the demonstration of the process, but we now have enough information to annotate the system and initiate the automated activities.

4.2.4. System safety analysis

With all this information inside our tool, we can commence the automatic analysis phase, which after a series of processes such as the synthesis of the system-level tree from individual failure fault trees, it produces valuable safety artifacts such as the system-level fault tree, FMEA tables, and Common Cause Analysis (CCA) results. After further qualitative analysis on the fault trees, we generate the minimal cut sets.

4.2.5. DALs decomposition and allocation

During this next phase, the algorithms provide the list of minimal cut sets as input to the optimisation algorithm in order to cost-optimally decompose and allocate the DALs throughout the architecture. The basic idea is that integrity requirements allocation to lower-level systems is influenced by the contribution of sub-systems and components to the system failure. The guidelines suggest FTA or Markov analysis as suitable candidate techniques for determining that contribution and the minimal cut sets is the key artifact exploited for this task. As already discussed, the other significant part of the allocation process is the set of rules provided by the standards when two or more sub-system failures contribute to a higher-level failure. These key elements for the optimal requirement allocation are also graphically depicted in Fig. 12 with the use of a triangle.

In practice, the software tool uses a Tabu search optimisation technique, which is an extension of HiP-HOPS as described in Sorokos et al. (2015)Sorokos et al. (2016), and an abstract cost catalogue. Each candidate solution represents a specific set of allocated DALs and the

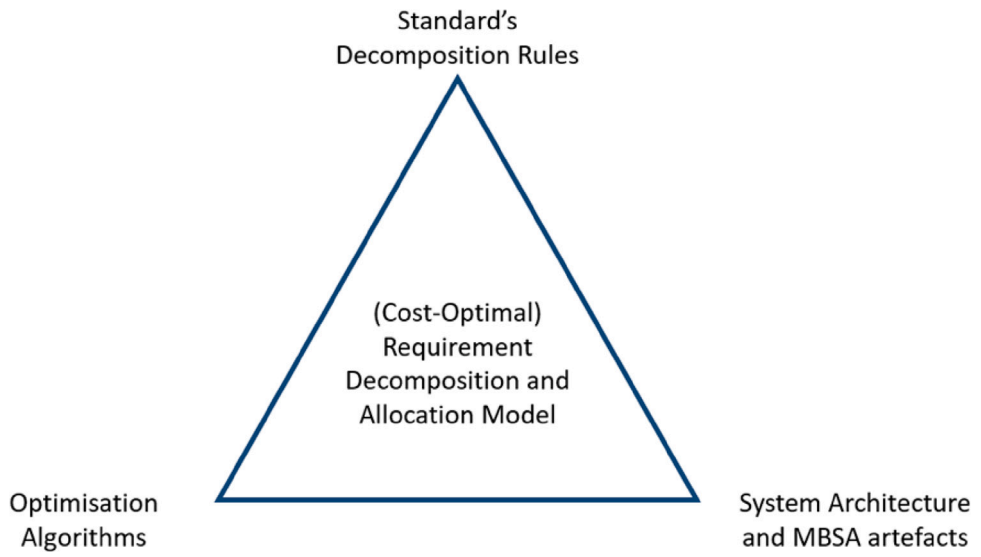

Fig. 12. Integrity requirement allocation model.

1. Generation of random allocation
2. Set the generated allocation as 'current choice' and add it to the memory (Tenure)
3. Repeat until iteration count or time limit reached
   I. Produce alternative allocations randomly from the 'current choice'
   II. Re-order the generated allocations, based on DAL cost, in an ascending order
   III. Pick the allocation with the lowest cost as the 'next choice'
   IV. Repeat until a 'next choice' is selected or until there are no other alternative allocations to examine
      a. If the next choice is not present in the memory, select it as the 'next choice'
      b. If it exists in the memory structure, but fulfils the aspiration criterion, select it as the n'next choice'
      c. Else, examine the next produced choice
4. If the generated allocations are neither Tabu nor aspiring, set the one with the lowest cost as the next choice
5. The next choice becomes the current selection
6. Add the current selection to the Tenure
7. Re-order the allocations in the Tenure, based on DAL cost, in an ascending order

Fig. 13. Optimisation Algorithm.

best candidate is considered the one with the lowest cost. First, the algorithm creates a candidate solution at random which is used in the next cycle to produce a new generation of candidates. After every iteration, the available solutions are compared and the less costly is selected. In addition, the algorithm also checks the short-term memory (Tabu Tenure) for any recent occurrences of the current solution and if an identical exists within the tenure, it gets discarded. However, due to an "aspiration criterion" if the candidate in the tenure has the overall lowest cost in memory, then the algorithm selects that candidate regardless being in the tenure. The algorithm is provided as pseudocode in Fig. 13.

For this particular example, we used the following cost values shown in Table 4.

The generated DALs for the function "Deceleration Capability" have been decomposed and allocated automatically as illustrated in a more detailed system architecture in Fig. 14. Note that the initial DAL assigned is A due to the fact that the hazard in Table 2 requires a more stringent requirement compared to the hazard in Table 3 and as the guidelines propose, the most stringent requirement per element overrules lower values. As can be seen, in the case of the power supply units (BSCU) there is redundancy and therefore the decomposition rules are applicable. In addition, the DALs assigned on the wheel assemblies are to cover for the electronic parts and actuators rather than mechanical parts since these require different methods for determining requirements. Finally, there have been alternative allocations with similar costs but here we present only one of the available options.

Table 4
DAL cost values.

DAL	A	B	C	D	E
Cost	100	80	40	20	0

4.2.6. Pattern creation and instantiation

Once safety assessment is completed for this development cycle, effort must be invested towards system safety assurance by creating

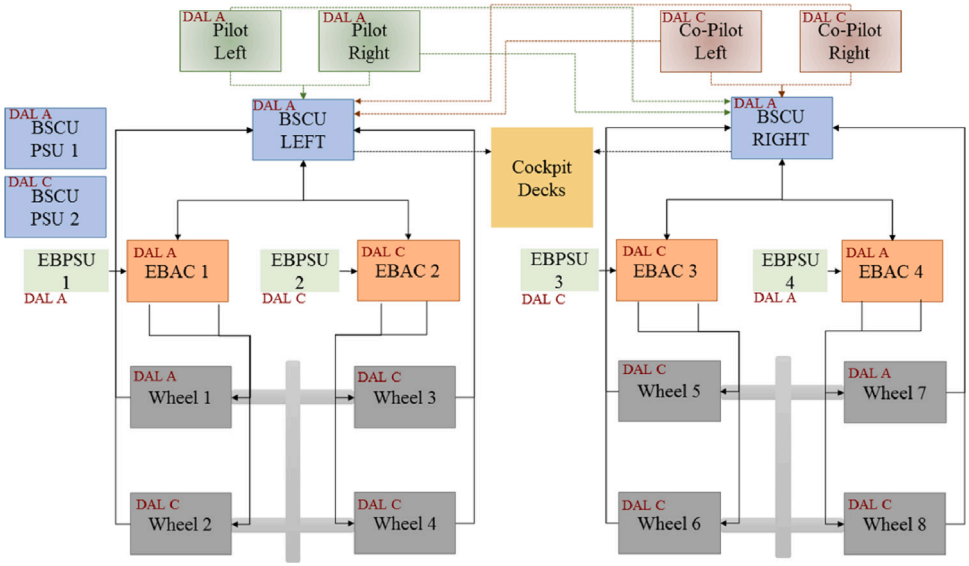

Fig. 14. Cost-Optimal allocation of DALs on the wheel brake.

an argument structure. For this task, we employ the concept of arguments patterns that mandate the argument structure, system model information and an instantiation algorithm to produce the concrete argument.

The general pattern we use for the aviation industry follows closely the safety lifecycle from the ARP4754-A. Despite the pattern being substantially smaller in size than the concrete argument, here we present only certain parts of the pattern due to space limitations. Fig. 15 showcases an overview of the pattern at high-level.

The argument comprises of various subgraphs and their encapsulated modules. The top-level module argues that the system in question (i.e. the aircraft) is safe since the development followed all safety guidelines and addressed all the functional failures identified during the design stage. The following nodes argue that this has been achieved by validation and verification activities as described with the use of the V-model. This is important to ensure that all functional requirements specified during the design stage are appropriate and correct and the implemented architecture is correct according to those requirements. The process is iterative and goes through the entire architecture starting from functions and followed by systems that implement these functions and their sub-systems.

The subgraph in Fig. 16 illustrates the claim that functional requirements address all the underlying hazards identified during the most recent FHA. The justification of assurance is based on the guidelines that suggest Preliminary Aircraft Safety Assessment (PASA), which indicates that the process must continue until all hazards for all aircraft functions have been addressed via the requirements. To notify the algorithm to iterate over all hazards of every function in the system, the pattern uses the GSN's multiplicity extension. On the other hand, we also use the optionality extension to create options based on the current requirement. For example, if the requirement is a DAL, then the claim is supported by the "DAL VAL module" whereas in any other case it is supported with the "RValGoal" node, which is left undeveloped in this case.

Finally, the "DAL Val Module" is expanded in Fig. 17. The top-level node claims that the DAL allocated is suitable for addressing a specific hazard (i.e. parameter "H") of the current function or system element. The strategy utilises the Preliminary System Safety Assessment (PSSA) to examine causes of failure that might lead to the hazard occurring. This is further supported by nodes which claim that the DALs allocated throughout the function's supporting elements (i.e. systems or components) are appropriate and optimal. The appropriateness is specified through the "SupportingDalGoal" node that uses the multiplicity extension to go over all the elements within the architecture that implement the "F" function. The optimality is argued via the "DALOptModule" within the "DALGoal" node and supported via evidence generated through the Tabu search.

Similar to this validation subgraph, there is a part of the pattern that handles arguments related to verification activities (ASA and SSA).

4.2.7. Pattern instantiation

Once the pattern is built within the tool, the engineer can employ the instantiation action, prompting the algorithm to produce the corresponding concrete argument. In this process, the pattern serves as a guiding structure, allowing the algorithm to traverse the system architecture as needed. It replaces all parameters within the nodes with actual information derived directly from the system model.

The algorithm responsible for the instantiation is implemented as three parts within our tool. The initial functions, "Instantiate", initiates the process, checks for the provision of essential user inputs -comprising system model information and structure, assessment artifacts, and the argument pattern- and performs basic model-checking to ensure adherence to specific rules for both the system model and pattern. For instance, it does not accept a pattern where a strategy node is connected directly with an evidence element, as this is invalid based on the GSN standard.

Assuming all the information is suitable, the program creates the top-level element and calls the second function, "ParseandCreate". This function is the core of our algorithm, utilising three inputs: the top-nodes from the system model, the pattern and the concrete argument to gather the information and structure and recursively construct the argument. Specifically, through the parameterised expressions within the pattern nodes, the algorithm mandates the type of information needed from the system.

Our third function, "ParseForEach", is an auxiliary mechanism used when the process meets special pattern elements such as "ForEach"and "IfThen". The latter enables the system to traverse through the model information according to user-defined expressions found in these special nodes. The system model is utilised for both its encapsulated information and structure, allowing the algorithm to identify which parts of the pattern need to be repeated, and ensuring the argument extents throughout the underlying architecture. Finally, the algorithm returns to the "Instantiate" function, which passes through the generated argument structure and calls the tool's editor to create the

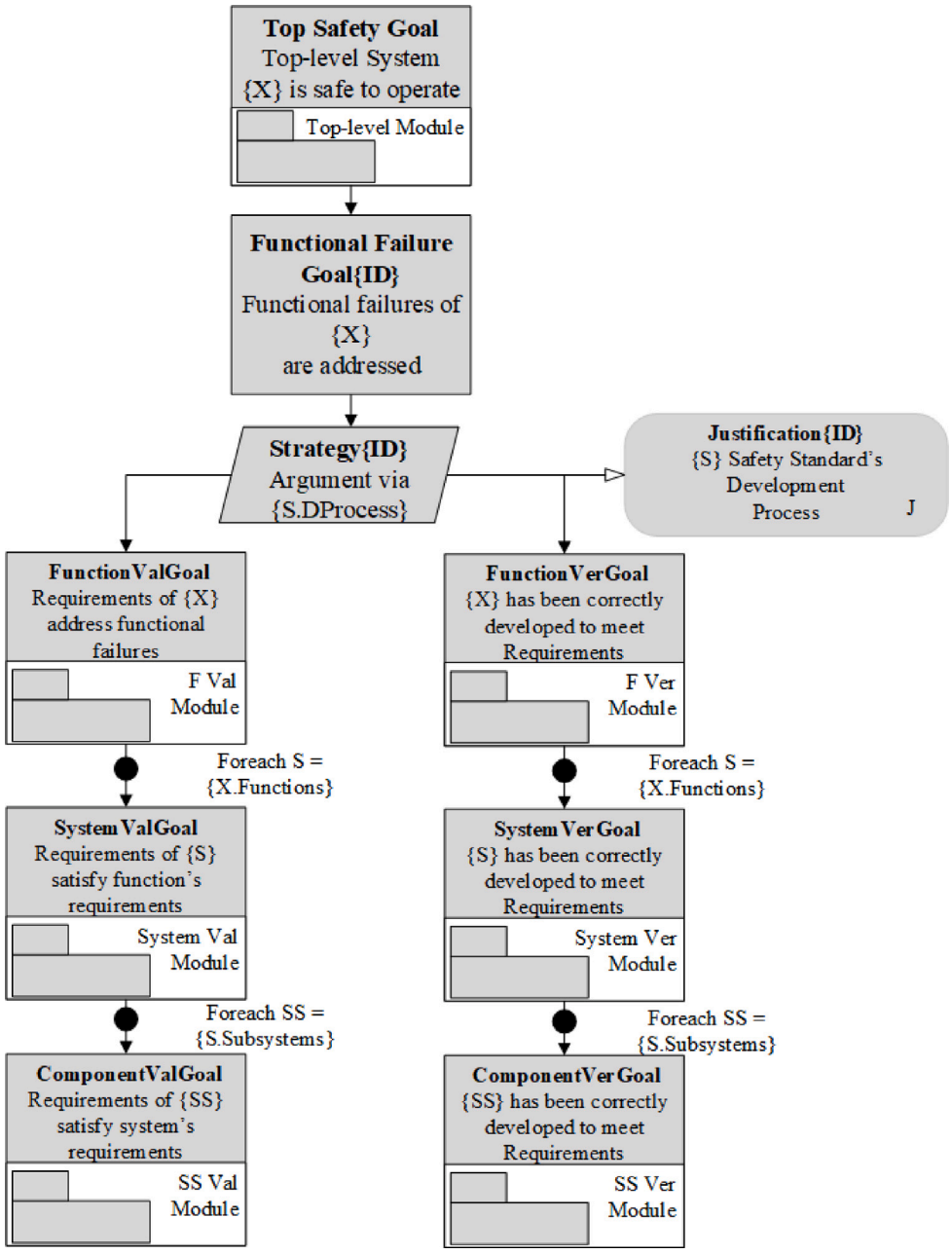

Fig. 15. Overview of the pattern.

graphical representation. The "ParseandCreate" function depicted in Fig. 18 as pseudocode with all the language-related details omitted.

The automatically generated structure is depicted in Figs. 19 and 20. Given the considerable size difference between the argument and the pattern, only two segments are presented here. The first part constitutes the concrete representation of the pattern structure shown earlier in Fig. 15.

Showing compliance with the EASA's guidelines is mandatory since the aircraft operates within the European boundaries and this should be reflected in the argument; however, due to similarities with the FAA, it set as undeveloped in this example.

The second part of the concrete argument, Fig. 20, corresponds to the argument derived from the pattern snippet in Fig. 17. It asserts that the DALs allocated to the left-side BSCU and its underlying systems are adequate to address the hazard examined in this example- namely, the "Unanticipated Loss of Deceleration Capability".

4.2.8. Handling changes

Last but not least, it is essential to discuss how Andromeda accommodates changes to adapt to the continuous evolution of modern development. Arguably, the most common type of change involves alterations to the system model. During development, engineers may determine that certain components are unsuitable for the intended purpose, or issues may arise with suppliers, leading to the adoption of alternative products with distinct architectural differences.

Additionally, authorities may request hardware or software modifications through directives to ensure compliance with the regulations. For instance, in 2019, the FAA issued an Airworthiness Directive (AD) suggesting all Boeing 787-8 and 787-9 aircraft to install a hydraulic tubing and replace their flight control software (Federal Aviation Administration, 2019). These design alterations are easily managed by re-annotating the system model with new failure information, and the remaining tasks are handled automatically by the algorithms. Other types of changes, such as new guidelines or alterations to existing ones,

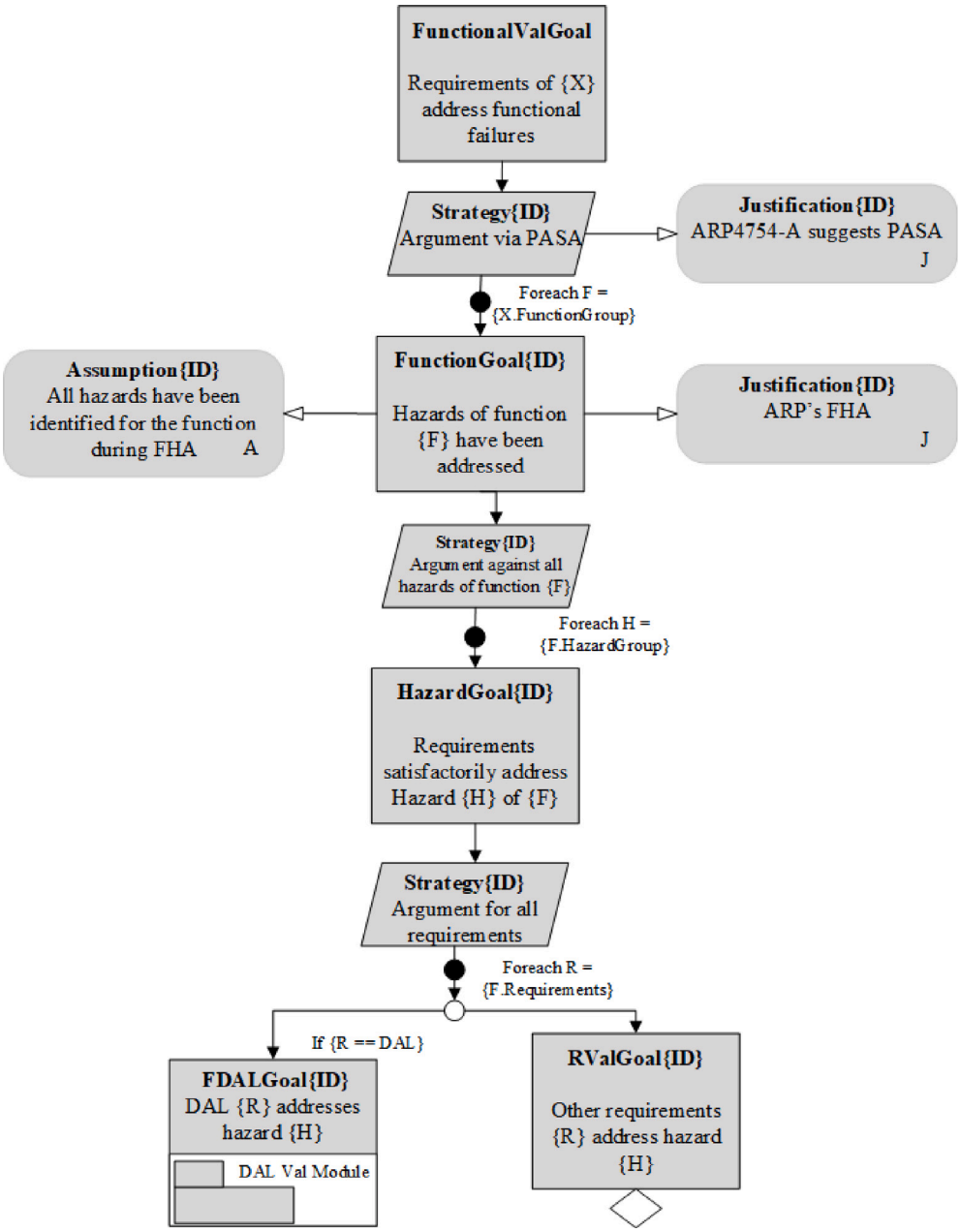

Fig. 16. Function Validation Pattern.

can be accommodated by adjusting the argument pattern to align with the new regime. Finally, changes may also involve the replacement of analysis techniques with different approaches. These can be seamlessly integrated into the tool, either through ad-hoc solutions and minor adjustments to the metamodel and tool or by direct implementation within software's analysis engine.

4.3. Evaluation

Summarising our findings, we begin with a reflective evaluation, which revealed that the integration of all the aforementioned processes enabled a seamless generation of major work products such as integrity requirements and argument structures. In combination with the significant automation introduced in our method, this provides a strong foundation for a scalable safety assurance solution, which is crucial for modern development.

It is important to note that we applied this evaluative case study with an aim to address the paper's research questions and evaluate the tool at its prototypical stage. In addition to our own experience using the tool, we distributed it to industrial partners from EU projects and others with HiP-HOPS licences (the proprietary part of Andromeda). The group of users averaged 5–7 years of experience in reliability engineering. Each user received the files necessary to install the tool on their own machines and used it to run various examples. Once the testing period concluded, we conducted informal interviews remotely through collaborative apps (Microsoft Teams).

From our own experience as the authors, we found that the method handles changes – whether architectural or knowledge-based - reasonably well due to the adjustability of the argument pattern and the automation, as discussed in Section 4. This capability enhances user confidence and facilitates the maintainability of safety arguments. The case study has provided insight into the following key evaluation dimensions:

1. Scalability
2. Validation of Requirements and Traceability

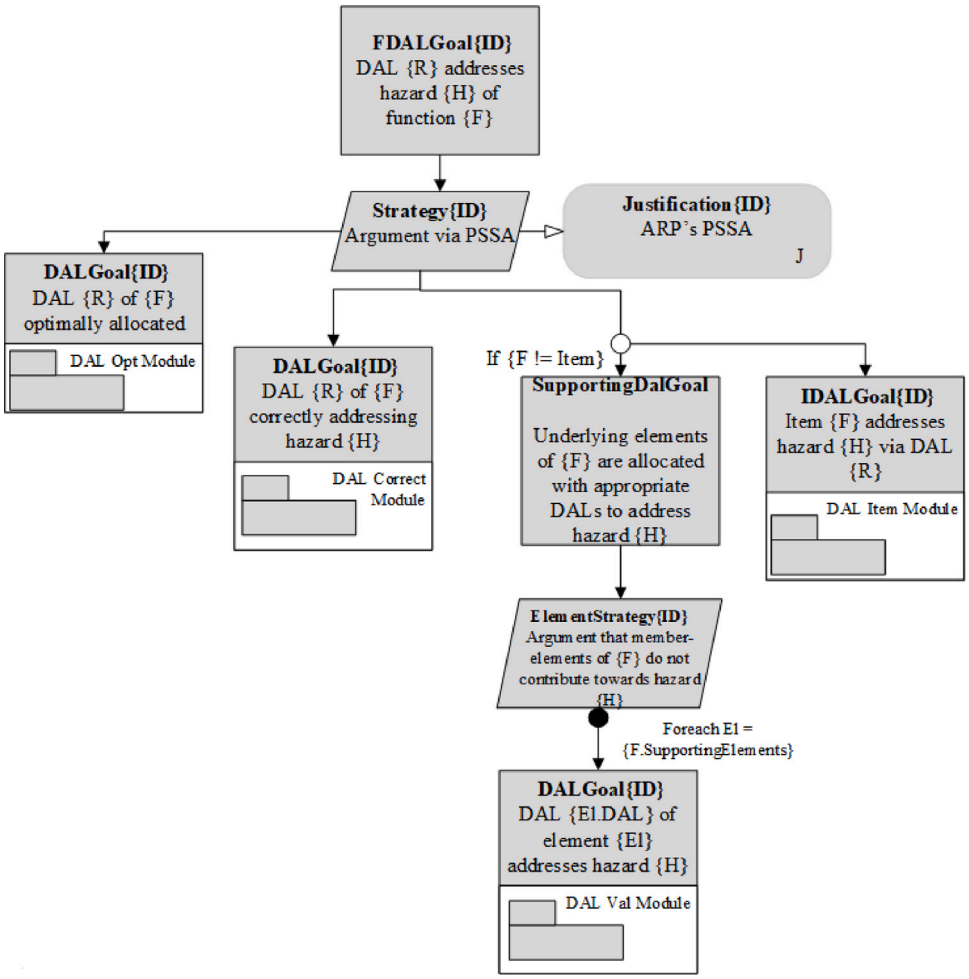

Fig. 17. DAL Validation Pattern.

3. Compliance with Industrial Practice
4. Usability

4.3.1. Scalability

Models and automation naturally simplify labour intensive and error-prone processes. However, it is important to consider the scalability of our algorithms and tool. First, we note that all the key processes that are computationally expensive, such as the analysis and design optimisation, are handled within the Andromeda using the HiP-HOPS engine, a commercial tool used in industrial context, which has been proven to be scalable (Aizpurua et al., 2018; Lampe et al., 2018; da Silva Azevedo et al., 2013; Mian et al., 2019). Regarding the overall application of the method, although the paper presents proof of concept, our case study has been designed on a realistic system architecture, having consulted engineers and a number of published resources. This included not only the "Wheel Brake" system but also other supporting systems of the "Ground Deceleration" function, such as the "Spoiler Control" and "Thrust Reverser Control". Our system is therefore of substantial complexity and size for the purposes of demonstration, comprising over 152 system elements and resulting in thousands of argument nodes. Manually creating such an argument structure would require several person days of modelling, whereas with the automated instantiation in our tool, it took only a few seconds.

The user group reported that, based on extrapolation, the examples they worked with would scale to the typical model sizes they encounter in their applications.

4.3.2. Validation of requirements and traceability

As a model-based method, Andromeda enables traceability from a safety case to various engineering artifacts due to the interconnections between the heterogeneous models established by its underpinning metamodel. For industries like the aviation, traceability is essential for the validation of requirements of sub-systems and items (SAE International, 2010, p. 62). Specifically, each requirement should be traceable to a parent requirement or to the specific design decision that led to its identification. The MCSU structure holds all the information at any time between argument elements and model elements, which can help to algorithmically, using the requirement decomposition rules as a base, examine whether a low-level requirement satisfies the higher-level requirement. In terms of other validation processes, we have implemented analysis methods that are acceptable by the regulatory authorities as means to assist in the validation of safety-related requirements, such as FHA and PSSA. Other validation activities include the demonstration of results from special tests and simulations. To this end, we do not currently offer direct support for such specialised processes, but we have designed generic structures in the pattern to store test results along with the mechanisms necessary to add them externally from other sources.

Regarding this aspect, the users confirmed that the traces relevant to their typical application usage were included.

4.3.3. Compliance with industrial practice

Despite the fact that this method is novel, we have put significant effort into ensuring it is compliant with standards. Andromeda closely follows what ISO26262 and ARP4754-A suggest as suitable processes at

```
ParseAndCreate (pattern_node, model_node, argument_node)
1) If (pattern_node is a basic node)
    a) Instantiate title, content and other relevant information
    b) For each child_node in pattern_node
        i) If (child_node is basic_node)
            (1) Generate an empty argument_node_child
            (2) Set argument_structure's_last = argument_node_child
            (3) ParseAndCreate (child_node, model_node, argument_node_child)
            (4) Update argument_node for the new child
        ii) Else if (child_node is ForEach_node or IfThen_node)
            (1) ParseAndCreate (child_node, model_node, argument_node)
2) Else if (pattern_node is ForEach_node)
    a) ParseForEach (pattern_node, model_node, argument_node)
3) Else if (pattern_node is IfThen_node)
    a) Pick the user-defined condition from the IfThen_node
    b) If (ProcessCondition(condition) is true)
        i) Create an empty argument_node_child based on the child of IfThen_node for true
        ii) ParseAndCreate (child_node, model_node, argument_node_child)
        iii) Update argument_node for the new child
    c) Else
        i) Create an empty argument_node_child based on the child of IfThen_node for false
        ii) ParseAndCreate (child_node, model_node, argument_node_child)
Update argument_node for the new child
```

Fig. 18. Instantiation Algorithm.

each development stage and provides the appropriate safety artifacts as outcomes. For example, hazard identification and mitigation measures, requirements allocated across system architecture, and the generated argument of safety.

Similarly to our own findings, user feedback reassured us that our method adheres to the standard processes. However, we intend to follow up on this qualitative evaluation with more detailed quantitative studies assessing specific standard requirements coverage.

4.3.4. Usability

The selected user group provided substantial unstructured feedback on usability improvements, and overall, reactions were positive for a non-commercial tool. For example, users suggested implementing version control to allow for concurrent development and the ability to revert changes in case of errors. Another recommendation was to transition from a standalone implementation to a web application to enable collaborative work among multiple users.

This preliminary feedback has been prioritised, and some changes were incorporated in the latest version of the tool. After addressing the remaining recommendations, we plan to investigate means of releasing a public version of the tool and distributing it to more industry experts for further review and structured evaluation as part of our future work.

5. Related work

Several methods have built upon and refine the concept of MBSA to either address obstacles associated with the manual nature of classical analysis techniques or to improve the design process through automation and deliverables such as optimised architectural designs.

One approach found in Kaiser et al. (2003) introduced the idea of Component Fault Trees (CFTs), a concept that extends the classical fault trees by incorporating modularisation and elements from the system architecture. Each component represents a system element with its own modular fault tree and many components are connected through input/output ports forming a network structured as directed acyclic graphs, effectively creating a high-level system architecture. The failure propagation applies through the output ports and is computed based on the input received from the connected components and any internal (within that component) events. The modular structure of CFTs

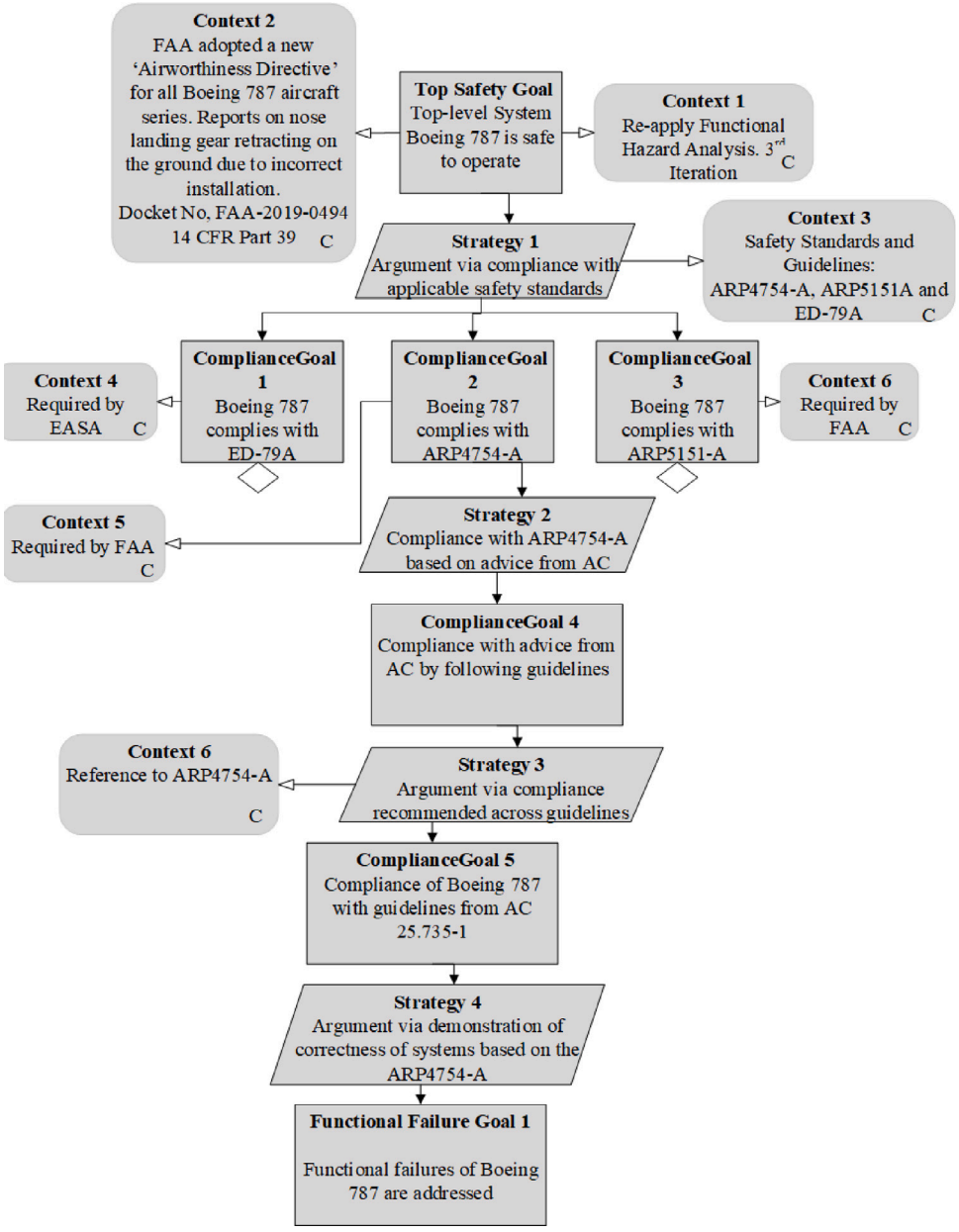

Fig. 19. Top-level Concrete Argument - Aircraft safety via compliance with guidelines.

enhances scalability and traceability, ensuring that all safety-critical aspects can be linked directly to system requirements, which is a significant benefit over classical fault trees.

A novel method that further extends on MBSA and promotes automation for the safety assessment presented in Munk et al. (2018) and referred to as the Semi-Automatic Safety Analysis and Optimisation (SASAO). This is a semi-automatic safety analysis approach that uses a system model capable of encapsulating the cost of architectural elements and error propagation information, represented as CFTs. The outcome of this method includes both a functional and technical architecture that remain consistent with other generated artifacts, such as fault trees for each safety goal and the FMEA results. SASAO also utilises optimisation techniques to aid in system design by selecting the cost-optimal architecture whilst adhering to the safety requirements (i.e., the probability of top events stays within the required threshold).

Another recent approach that incorporates automation for safety analysis of systems is presented in Wei et al. (2022), known as Designing Critical Systems with automated IteratiVe safety analysis (DECISIVE). This methodology facilitates the strategic planning and design of a safety-centric system through a systematic five-step process that is applicable with or without the model-based paradigm. In brief, the engineers define the system and functional requirements and perform hazard analysis and risk assessment. Then, the safety requirements are defined and utilised for designing the architecture and, based on component reliability data, they form the reliability model. After completing these stages, a safety analysis model is created using automated FMEA and identification of safety-related components implemented within the software tool-support. Following a refinement phase, the acceptably safe architectural design is produced. Any safety-related artifacts generated during this recurring process can later serve as supporting evidence for the system's assurance case.

As per DECISIVE, Andromeda also leverages the MBSA paradigm to facilitate semi-automatic safety analysis of systems, producing fault trees, FMEA tables, and minimal cut sets, which are essential evidence artifacts for safety assurance. Additionally, like SASAO, Andromeda features capabilities for architectural optimisation. Through the use of multi-objective genetic algorithms (Parker, 2010), it is capable of

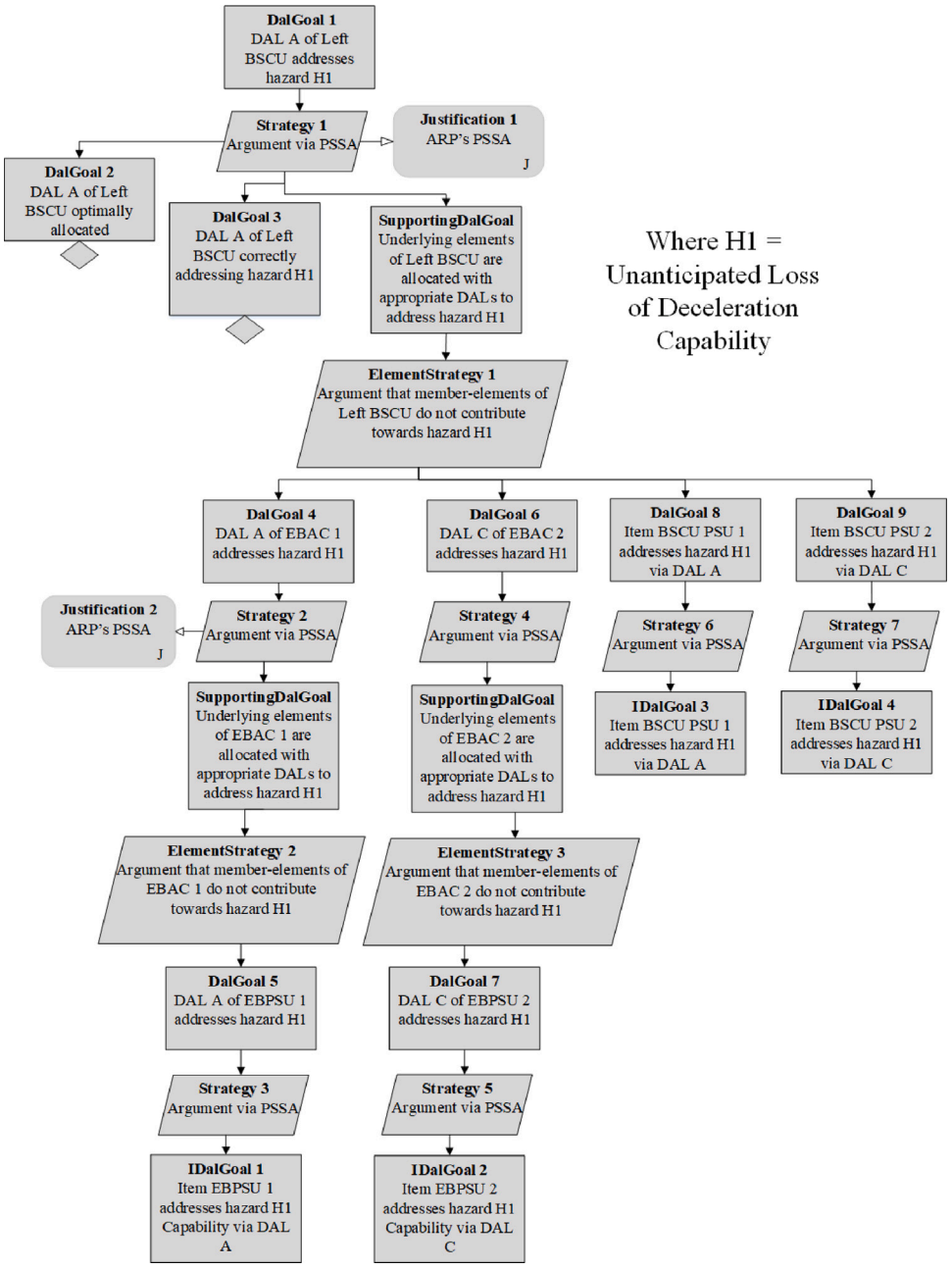

Fig. 20. Argument over Left-side BSCU and supporting systems.

exploring a large design space with the goal of finding cost-efficient and reliable system architectures. However, the key distinction of Andromeda is that it extends beyond the safety assessment. Instead, it utilises these work products to synthesise argument structures that are directly applicable for certification purposes.

There are some approaches for producing assurance artifacts towards a safety-driven architectural design that do not only focus on automation.

For instance, methods such as in Denney et al. (2017) utilise bow-tie diagrams to compose a safety architecture. Broadly, with the use of bow-tie diagrams it is possible to link identified hazards with their causes (using FTA) on one side and the consequences (using Event Tree Analysis) on the other, which also capture the corresponding probabilistic information. The authors strategically adopted this approach to address practical considerations in safety case development of small unmanned aircraft systems (sUAS). This includes maintaining consistency across diverse safety artifacts and ensuring that any changes to the architecture are well-justified allowing them to streamline the decision-making process. This approach is novel, and based on the authors, the safety case produced from the artifacts generated through their method was approved by the corresponding regulatory authority. Andromeda differs in a couple of ways. First, our framework uses abstract requirements, making it more generic and adaptable to various industries with only a few changes whereas their method focuses on the sUAS. Secondly, the argument generated with our approach, which is one of the key parts of a safety case, is generated semi-automatically, whereas in Denney et al. (2017) there is no indication for automation capabilities on that front and no instantiated argument is shown. This means that a more complex system than a sUAS might pose significant challenges during the assurance stage.

Below, we explore various approaches that utilise model-driven development or MBSA to deliver substantial benefits to both the assessment and assurance parts of the safety framework.

In Denney and Pai (2016), the authors presented their work in argument pattern composition. Specifically, they provided a formal foundation for producing complex argument patterns by combining a number of simpler patterns or by abstracting established arguments. Additionally, they implemented a pattern instantiation mechanism in their tool, AdvoCATE, which allowed them to create argument structures and effectively automate a significant part of the safety case synthesis. This method innovatively creates complex argument patterns that can be easily adapted for any industry and utilises tool-support with the appropriate algorithms to instantiate these patterns and generate argument structures. Consequently, the pattern can drastically change depending on the expertise of the safety engineers. In contrast, Andromeda elicited a pattern from commonalities found in known safety standards and uses that as a basis to form argument structures. In addition, our method employs abstract requirements, generated from system models and analysis, to provide the rationale for how sub-goals support higher-level goals, rather than using concrete requirements. As a result, this allows for a high-level framework that can be adapted to various contexts more easily, enabling broader application. Furthermore, Andromeda also features manual creation of argument patterns, allowing developers to adapt the patterns and gain flexibility.

The work proposed in Hawkins et al. (2015), known as the weaving model, helps to establish interconnections between different information models and GSN patterns via appropriate semantics. Adhering to the model-based paradigm, this approach is equipped with automation that facilitates safety case construction and is supported by a software tool developed within the Eclipse framework. This method offers a robust foundation for interconnecting models with GSN pattern structures and instantiating argument structures given the concrete information available. However, while model weaving allows for the automatic extraction of data from design and analysis models, it does not directly generate the appropriate evidence artifacts from the system. Additionally, it lacks specific reasoning for constructing arguments and does not utilise SILs, which are essential in multiple industries. Nevertheless, the weaving model is sufficiently generic to allow for adaptations, and these features could be implemented in principle.

Another notable methodology, the Assurance Case Centric Engineering Safety-critical System (ACCESS), is presented in Wei et al. (2024), along with its tool-support, the Assurance Case Management Environment (ACME), for the development of systems and their assurance cases. The authors explain the importance of assurance cases for certification, discuss the diverse engineering artifacts they comprise, and emphasise the impact of their evaluation on the overall assurance case. They highlight challenges such as the lack of model-based support in traditional graphical notations. Additionally, they stress that with the emergence of robotics and autonomous systems, assurance cases must adapt not only during system development but also during the operational stage. To address the first challenge, ACCESS employs the Structured Assurance Case Metamodel (SACM), which offers substantial benefits over the graphical notations and robust model-based support for creating and managing assurance case models. They have also implemented formal verification capabilities that enhance assurance case evaluation by avoiding argumentation fallacies and increasing confidence. ACCESS comprises five basic steps, starting with system function specification and concluding with system designs, an assurance case, and system verification and validation reports. Additionally, there are two optional steps that support the construction and evaluation of runtime assurance cases, with results recorded for continuous system improvement.

ACCESS is undoubtably a novel method offering significant benefits, such as the integration of formal methods and the creation of runtime assurance cases. Despite both ACCESS and Andromeda providing frameworks for safety assurance, there are substantial differences. For instance, Andromeda is a fully integrated method that internally provides all the elements needed for the semi-automatic generation of safety arguments, including safety analysis artifacts. In contrast, ACCESS focuses mainly on the automated evaluation of the generated assurance structure through the validation and verification of external engineering artifacts. Moreover, Andromeda uses the concept of SILs and analysis techniques, as advised by safety standards, to construct safety cases, while ACCESS primarily utilises formal methods, which are essential in various applications like software development for safety-critical systems. To conclude, these two methods are quite distinct, each with its own benefits and limitations, but could benefit from incorporating features from each other. For example, Andromeda could be adapted to use formal methods for software component verification, whereas ACCESS could implement ASILs to align with guidelines such as the ISO26262.

Finally, in Roback (2023), the author investigates numerous open source and commercial tools for assurance cases and highlights his concerns. The main criteria of this review were ease of use and cybersecurity. In terms of cybersecurity, most of the tools that are developed or heavily utilise the Eclipse Framework found to introduce security risks due to the conflicts between Eclipse IDE's networking protocols and national cybersecurity protocols (per USA standards). Other tools were either web-based, which means they are not suitable for classified projects or found to use platforms that were compromised in the past. The review indicates that the most robust tools found to be these without any dependencies from specific platforms. Based on the second criterion, most tools, with the exception of commercial tools, were either unavailable, had outdated documentation and required quite the expertise to be installed or were outdated. There was a third criterion, about the appropriate use of Model-Based System Engineering in assurance cases, but the author was mostly reviewing tools that are good candidates for USA Department of Defence (DoD) use; therefore, it is not appropriate to focus on it here.

Andromeda's supporting tool was developed as a standalone application using C++ under the Qt framework, making it compatible with a variety of operating systems. Qt offers a wide range of libraries for rendering, user interface design, and peripheral management, and does not involve any online usage once the application is compiled (Qt Group, 2024). This design minimises security risks since the application is usable offline without dependencies on third-party platforms. Regarding ease of use, Andromeda's tool requires no installation and includes straightforward documentation.

Lastly, the idea behind Andromeda was first conceived and proposed in a short conceptual paper in Retouniotis et al. (2017). In that paper, we introduced the concept of building argument structures with a rationale centred on SILs and a structure that follows the system architecture. The reasoning was twofold. First, SILS are flexible abstract requirements extensively used by functional safety standards, making the approach applicable across various industries and relevant to certification authorities. Secondly, the model-based paradigm was ideal as it allows for automation, helping to keep up with the pace and complexity of modern systems development, and providing a top-down approach that enhances traceability, as stressed by regulations.

However, much progress has been achieved since. Our research group has been developing MBSA tools which have become commercial and produce evidence artifacts that meet the requirements of standards. Andromeda introduces a metamodeling infrastructure that enables integration of such sophisticated tools. To connect them with assurance structures, we extended our analysis tool metamodel by adding GSN pattern and argument elements. This allowed us to extract information from system architectural models into GSN pattern elements and instantiate the latter to generate an argument structure. As an integrated method, Andromeda absorbed some of these features, but both the method and the tool have matured substantially since then.

Specifically, the initial pattern of safety was generic and trivial. For example, practitioners would manually set the top-goal and all the sub-goals for the argument structure without any considerations of the specific industry. The instantiated argument lacked validation or verification activities' outcomes because there was no underlying

framework to guide the structure. Andromeda, on the other hand, implements a pattern elicited from commonalities among standards and provides a solid foundation that covers these activities, helping to build major parts of the safety case.

Moreover, Andromeda's metamodel has significantly evolved from earlier work. Initially, the metamodel was externally connecting the GSN argument and pattern elements with the HiP-HOPS elements. It now effectively links all these under a single framework and implements them in the same tool, eliminating the need for artifacts in the form of Extensible Markup Language (XML) files from different tools and linking them in an ad-hoc manner. The primary benefits are that any changes to the system architecture are now reflected in the argument instantly during design iterations, increasing efficiency and traceability. Additionally, we expanded our metamodel by incorporating the different scheme and requirement decomposition rules recommended by the ISO26262. Consequently, our method now directly supports safety assurance activities for both the aviation and automotive industries and can be further adapted for other domains.

6. Conclusions and future work

In this paper, we described our work towards a largely automated safety assessment and assurance framework for safety-critical systems. Our approach, Andromeda, is applicable from the early stages of development and follows the assessment activities in compliance with safety standards to produce and maintain safety argument structures. It integrates elements from various areas. Specifically, it facilitates functional design and analysis, safety activities, requirement allocation and assurance through automation and provides a solid foundation for constructing safety cases that are essential for the certification of modern systems. To achieve this, we designed an innovative metamodel that links together model-based safety analysis, automated and cost-optimal requirement decomposition and allocation via a sophisticated optimisation algorithm, key common elements across various safety standards and the concept of argument patterns accompanied by an instantiation algorithm responsible for automatically constructing the concrete argument. We developed a safety assessment and assurance method that is adaptable, and supported by a software engineering tool, which can be employed directly for practical use in safety assessment and assurance. Furthermore, both the metamodel and the tool are easily customisable and therefore usable as a foundation for future methods that follow the model-based paradigm.

6.1. Limitations

Despite the strengths of Andromeda, it comes with certain limitations. First and foremost, the approach is subject to the "garbage in, garbage out" (GIGO) principle. This means the quality of the work products generated by the method is dependent on the quality of the inputs. For example, if the failure annotation by the safety engineer is not appropriate, then the minimal cut sets and the abstract requirements will not be correct either. As a result, the produced argument will not reflect the true safety state of the system since its evidence is compromised. However, the arguments constructed in Andromeda are explicit and subject to scrutiny. Typically, arguments undergo evaluation, which is possible due to the traceability, and enhancements before being delivered to regulatory bodies. As for the validity of the allocation of requirements, these can be checked during development. The failure model can be examined by experienced practitioners before submission, and the fault trees generated in Andromeda, which form a significant part of the evidence, can be assessed using mathematical models within the tool. Requirement optimisation results can also be manually tested for validity. Overall, it is much harder to scrutinise an argument when it is written on paper rather than in a software tool where most of the results are linked to system models and can be traced back.

Currently, Andromeda focuses mostly on product-based arguments, which address safety issues with functions, systems, and components. However, modern standards also examine other factors that indirectly affect system safety, such as organisational processes or practitioner competency. These are known as process-based arguments, and their implementation could benefit Andromeda by bolstering confidence in evidence artefacts and claims. For instance, an experienced safety engineer is unlikely to introduce many errors when annotating the system model with failure behaviour data compared to an inexperienced one. Our tool allows the user to introduce such information. The argument pattern can easily be extended manually, and there are appropriate structures in place to capture this information. However, we have not utilised it yet on any substantial examples or case studies.

Another limitation is that Andromeda does not incorporate formal methods and therefore lacks the ability to support mathematically rigorous verification and validation techniques, which are useful for various applications, including the development of software for safety-critical systems. Finally, although Andromeda is generic, it still requires modification to be usable in a new industry such as by adjusting the analysis methods used or by modifying the structure of the safety pattern.

Finally, we would like to discuss a practical limitation regarding the argument structure itself. Currently, we provide justifications and assumptions that are fixed within the pattern. While the pattern allows these elements to scale along with the system, thus maintaining consistency between high-level and lower-level arguments elements of the same node type, it lacks dynamic behaviour. For instance, an assumption stating, "Hazards have been addressed during X process" applies uniformly to all similar nodes in the hierarchy. This may be acceptable in some cases, but it could also be unwanted behaviour depending on the scenario. Furthermore, if there is an element lower in the architecture requiring a justification that is not necessary at higher-levels, the practitioner must manually extend the pattern to that level to create the abstract element and supply the needed justifications or evidence, such as simulation results or other specialised tests.

As part of our ongoing work, we have explored this issue in detail and developed a solution. Each system element can be extended with a list of justification and assumption expressions, allowing the safety engineer to provide relevant context in the form of templates. This feature enables the dynamic addition of assumptions and justifications as text integrated with parameterised expressions at any time. During the instantiation stage, if a system element has any assumptions or justifications, these will be automatically included in the argument, even if they are not present in the argument pattern. With the help of parameterised expressions, we can dynamically extract relevant information from the models and provide appropriate context and data. This solution minimises the need to extend the pattern unnecessarily, and it is currently in the implementation phase.

We do not claim that Andromeda produces a complete safety case, meaning the argument is not directly usable for certification purposes. It needs to be scrutinised and undergo enhancement and potentially modification before handled to the authorities. In addition, there are artifacts from simulations and specialised tests that might be necessary for a safety case that are not produced by the tool. There are proper entities in our framework to capture these in the argument, but the artifacts have to be generated by external sources and imported later in Andromeda. Despite these, the work products created are still valuable and would be challenging to produce manually for large scale systems.

6.2. Future work

Currently, our focus has been on the generation of artifacts and argument structures for safety cases. Despite our best efforts to make our approach applicable across various industries, many assumptions remain regarding the structure of the input for the case generator. In that regard, we see great potential to further evolve Andromeda

by introducing a standardised and homogeneous language for specifying inputs across more domains. Therefore, we plan to integrate other methods for systematising the nominal development process, allowing our approach to leverage the yielded relationships between requirements, architecture, V&V planning, testing results, and other argument evidence. This will enable the programmatic construction of the relevant assurance cases. Further benefits include enhanced capability for maintaining and updating the assurance case as the nominal development model is modified, building on the already established benefits of semi-automation, such as effort reduction, consistency, and scalability.

In addition to the previous avenues for future work, we plan to seek external validation from industry experts on the applicability of Andromeda and receive user-experience feedback on the tool.

Data statement

The data used in this study cannot be shared publicly. The source code and software components interact with HiP-HOPS, which is commercial and imposes intellectual property restrictions. Additionally, the questionnaires conducted with engineers, although informal, used examples from personal projects and are therefore restricted due to confidentiality agreements.

CRediT authorship contribution statement

Athanasios Retouniotis: Writing – review & editing, Writing – original draft, Visualization, Validation, Software, Methodology, Investigation, Conceptualization. **Yiannis Papadopoulos:** Writing – review & editing, Conceptualization. **Ioannis Sorokos:** Writing – review & editing, Validation, Supervision, Methodology, Investigation, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

This work was funded by the EU H2020 SESAME project (grant agreement 101017258).

Data availability

No data was used for the research described in the article.

References

Aizpurua, J.I., Papadopoulos, Y., Merle, G., 2018. Explicit modelling and treatment of repair in prediction of dependability. IEEE Trans. Dependable Secure Comput. 17 (6), 1147–1162.

Assurance Case Working Group, 2021. Goal Structuring Notation Community Standard Version 3. Technical Report, Safety-Critical Systems Club, Online at: https://scsc.uk/r141C:1?t=1. (Last accessed on: 15 July 2024).

Azevedo, L.S., Parker, D., Walker, M., Papadopoulos, Y., Araujo, R.E., 2013. Automatic decomposition of safety integrity levels: Optimization by tabu search. In: SAFECOMP 2013-Workshop CARS (2nd Workshop on Critical Automotive Applications: Robustness & Safety) of the 32nd International Conference on Computer Safety, Reliability and Security. p. NA.

Bieber, P., Delmas, R., Seguin, C., 2011. Dalculus–theory and tool for development assurance level allocation. In: Computer Safety, Reliability, and Security: 30th International Conference, SAFECOMP 2011, Naples, Italy, September 19-22, 2011. Proceedings 30. Springer, pp. 43–56.

Boeing, 2023. 787 Airplane Characteristics for Airport Planning. Technical Report, Boeing, Online at: https://www.boeing.com/content/dam/boeing/boeingdotcom/commercial/airports/acaps/787.pdf. (Last Accessed 15 July 2024).

C.A.E. Framework, 2020. CAE Concepts. Claims Arguments Evidence. Technical Report, CAE Framework, Online at: https://claimsargumentsevidence.org/notations/claims-arguments-evidence-cae/. (Last accessed on 15 July 2024).

da Silva Azevedo, L., Parker, D., Walker, M., Papadopoulos, Y., Araújo, R.E., 2013. Assisted assignment of automotive safety requirements. IEEE Software 31 (1), 62–68.

Denney, E., Pai, G., 2016. Composition of safety argument patterns. In: Computer Safety, Reliability, and Security: 35th International Conference, SAFECOMP 2016, Trondheim, Norway, September 21-23, 2016, Proceedings 35. Springer, pp. 51–63.

Denney, E., Pai, G., Habli, I., 2015. Dynamic safety cases for through-life safety assurance. In: 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering, vol. 2, IEEE, pp. 587–590.

Denney, E., Pai, G., Whiteside, I., 2017. Model-driven development of safety architectures. In: 2017 ACM/IEEE 20th International Conference on Model Driven Engineering Languages and Systems. MODELS, IEEE, pp. 156–166.

Erich, G., Richard, H., Ralph, J., Patterns, V.J.M.D., 1994. Elements of reusable object-oriented software.

Federal Aviation Administration, 2019. 14 CFR part 39. Airworthiness directives; the boeing company airplanes. Online at: https://www.federalregister.gov/d/2019-22390. (Last accessed on: 15 July 2024).

Frigerio, A., Vermeulen, B., Goossens, K., 2019. Component-level ASIL decomposition for automotive architectures. In: 2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops. DSN-W, IEEE, pp. 62–69.

Haddon-Cave, Q.C.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.

Hawkins, R., Habli, I., Kolovos, D., Paige, R., Kelly, T., 2015. Weaving an assurance case from design: A model-based approach. In: 2015 IEEE 16th International Symposium on High Assurance Systems Engineering. IEEE, pp. 110–117.

International Organization for Standardization, 2018. ISO 26262: Road vehicles – functional safety. Parts 1-12.

Joshi, A., Heimdahl, M.P., Miller, S.P., Whalen, M.W., 2006. Model-Based Safety Analysis. Technical Report, NASA, Online at: https://ntrs.nasa.gov/citations/20060006673. (Last accessed on: 15 July 2024).

Kaiser, B., Liggesmeyer, P., Mäckel, O., 2003. A new component concept for fault trees. In: Proceedings of the 8th Australian Workshop on Safety Critical Systems and Software-Volume 33. pp. 37–46.

Kelly, T.P., et al., 1999. Arguing Safety: A Systematic Approach to Managing Safety Cases (Ph.D. thesis). Citeseer.

Lampe, J., Rüde, E., Papadopoulos, Y., Kabir, S., 2018. Model-based assessment of energy-efficiency, dependability, and cost-effectiveness of waste heat recovery systems onboard ship. Ocean Eng. 157, 234–250.

Leveson, N.G., 2011a. Engineering a Safer World: Systems Thinking Applied to Safety (Engineering Systems). MIT Press Cambridge.

Leveson, N.G., 2011b. The use of safety cases in certification and regulation. Engineering Systems Division (ESD) Working Paper Series.

Mare, J.-C., 2017. Aerospace Actuators 2 - Signal by Wire and Power by Wire. Wiley-ISTE, http://dx.doi.org/10.1002/9781119332442.

Mian, Z., Bottaci, L., Papadopoulos, Y., Mahmud, N., 2019. Model transformation for analyzing dependability of AADL model by using HiP-HOPS. J. Syst. Softw. 151, http://dx.doi.org/10.1016/j.jss.2019.02.019.

Ministry of Defence, 1996. 00-56 safety management requirements for defence systems. MoD (1996a), UK.

Munk, P., Abele, A., Thaden, E., Nordmann, A., Amarnath, R., Schweizer, M., Burton, S., 2018. Semi-automatic safety analysis and optimization. In: Proceedings of the 55th Annual Design Automation Conference. pp. 1–6.

Nair, S., de la Vara, J.L., Sabetzadeh, M., Falessi, D., 2015. Evidence management for compliance of critical systems with safety standards: A survey on the state of practice. Inf. Softw. Technol. 60, 1–15.

Nidhra, S., Dondeti, J., 2012. Black box and white box testing techniques-A literature review. Int. J. Embedded Syst. Appl. (IJESA) 2 (2), 29–50.

Object Management Group, 2018. Structured Assurance Case Metamodel (SACM). Technical Report, Object Management Group, Online at: https://www.omg.org/spec/SACM/2.0. (Last accessed on: 15 July 2024).

Parker, D.J., 2010. Multi-Objective Optimisation of Safety-Critical Hierarchical Systems University of Hull, (Ph.D. thesis).

Qt Group, 2024. Quick start with qt for Python. URL https://doc.qt.io/qtforpython-6/quickstart.html#quick-start. (Accessed: 2024-08-01).

Radio Technical Commission for Aeronautics, 2011. DO-178c: Software considerations in airborne systems and equipment certification.

Retouniotis, A., Papadopoulos, Y., Sorokos, I., Parker, D., Matragkas, N., Sharvia, S., 2017. Model-connected safety cases. In: Model-Based Safety and Assessment: 5th International Symposium, IMBSA 2017, Trento, Italy, September 11–13, 2017, Proceedings 5. Springer, pp. 50–63.

Roback, K.P., 2023. Review of Potential Assurance Case Tool Options for DoD. Technical Report, Institute for Defense Analyses, Online at: https://apps.dtic.mil/sti/citations/trecms/AD1211550. Last Accessed 15 July 2024.

SAE International, 2010. ARP4754-A: Guidelines for Development of Civil Aircraft and Systems, Revision A SAE International, Warrendale, PA.

Sorokos, I., Papadopoulos, Y., Azevedo, L., Parker, D., Walker, M., 2015. Automating allocation of development assurance levels: An extension to HiP-HOPS. IFAC-PapersOnLine 48 (7), 9–14.

Sorokos, I., Papadopoulos, Y., Walker, M., Azevedo, L., Parker, D., 2016. Driving design refinement: How to optimize allocation of software development assurance or integrity requirements. In: Software Quality Assurance. Elsevier, pp. 237–250.

Wei, R., Foster, S., Mei, H., Yan, F., Yang, R., Habli, I., O'Halloran, C., Tudor, N., Kelly, T., Nemouchi, Y., 2024. ACCESS: Assurance case centric engineering of safety–critical systems. J. Syst. Softw. 213, 112034. http://dx.doi.org/10.1016/j.jss.2024.112034.

Wei, R., Jiang, Z., Guo, X., Mei, H., Zolotas, A., Kelly, T., 2022. Designing critical systems with iterative automated safety analysis. In: Proceedings of the 59th ACM/IEEE Design Automation Conference. pp. 181–186.

Athanasios Retouniotis is a software developer with a strong academic background and research experience in model-based safety assurance. He completed a Ph.D. in Model-Based Systems at the University of Hull, where he also held a contract position as a Research Assistant within the Dependable Intelligent Systems Group. During that time, he participated in the Horizon Europe project DEIS and worked on tool support for the group. In addition to his doctoral studies, he holds a Master's degree in Computer Graphics and has keen interests in game development, web technologies and computer vision.

Professor Yiannis Papadopoulos has pioneered work on model-based dependability assessment and evolutionary optimisation of complex engineering systems known as Hierarchically Performed Hazard Origin and Propagation Studies (HiP-HOPS). He co-authored EAST-ADL, an emerging automotive architecture description language.

These technologies have gained wide academic recognition and have been successfully transferred to the automotive, shipping and other industries, where they have been commercialised and successfully deployed in design and engineering processes. He is currently developing technologies for self-certification of cyberphysical and autonomous systems.

More information about his academic and industrial projects can be found on his personal website @ https://yipapadopoulos.wixsite.com/yiap.

Ioannis Sorokos is a senior researcher and project manager at the Fraunhofer Institute for Experimental SoftwareEngineering (IESE) in the department of Safety Engineering. He received his Ph.D. from the University of Hull in a topic related to model-based safety assurance. He has participated in and managed several industry and international research projects, including the Horizon Europe projects DEIS, BIECO and SESAME. He is a member of the WAISE workshop program committee and an editor in the Frontiers in Robotics and AI journal for AI Safety in safety-critical systems. His research interests include model-based safety assurance, safety-security co-engineering, ML uncertainty and self-adaptive systems.