Model-Connected Safety Cases

Athanasios Retouniotis(✉), Yiannis Papadopoulos, Ioannis Sorokos, David Parker, Nicholas Matragkas, and Septavera Sharvia

Department of Computer Science, University of Hull, Hull HU67RX, UK
A.Retouniotis@2014.hull.ac.uk, I.Sorokos@2012.hull.ac.uk,
{Y.I.Papadopoulos,D.J.Parker,N.Matragkas,S.Sharvia}@hull.ac.uk

Abstract. We propose the concept of a model-connected safety case that could simplify certification of complex systems. System design models support the synthesis of both the structure of the safety case and the evidence that supports this structure. The resultant safety case argues that all hazards are adequately addressed through meeting the system safety requirements. This overarching claim is demonstrated via satisfaction of the integrity requirements that are assigned to subsystems and components of the system through a sound process of model-based allocation that respects the system design and follows industry standards. The safety evidence that substantiates claims is supported by evidence which is also auto-constructed from the system model. As the system model evolves during design, the corresponding model-connected safety case can be auto-updated. The approach is underpinned by a data model that connects safety argumentation and safety analysis artefacts, and is facilitated by a software tool.

Keywords: Safety case · Automation · Safety assessment · ARP4754-A

1 Introduction

Regulatory authorities have established different means to certify safety critical systems. Currently, the most common practice is a 'safety case', a document which aims to provide clear, convincing and comprehensive arguments that a system will operate acceptably safe in a given context supported by appropriate evidence [1]. The structure of the arguments is as critical as the evidence, as it illustrates the relationship between safety evidence and safety requirements as they have been set by regulators and developers [2]. To facilitate this effort, graphical notations have been developed to help improve the representation of arguments and better express compliance claims. These are the Goal Structuring Notation (GSN) [3] and the Claims-Arguments-Evidence (CAE) [4] notation. Moreover, the concept of patterns is now well-established for supporting best practice when constructing and reviewing safety cases [5].

Despite these advances, the production and maintenance of safety cases remains mostly a manual process, and safety cases are unconnected to models of the system which they represent. Additionally, the considerable size of the documents, the heavy usage of cross-referencing and the complexity of evidence required to satisfy modern standards represent a great challenge for safety case developers. As underlying systems become complex, relevant safety cases grow larger and more convoluted. Further, the

M. Bozzano and Y. Papadopoulos (Eds.): IMBSA 2017, LNCS 10437, pp. 50–63, 2017.
DOI: 10.1007/978-3-319-64119-5_4

safety case should be considered a 'living document', requiring maintenance across a system's entire life span. Such maintenance requires significant effort and time reassessing the system for safety as well as validating arguments. The combined challenges mentioned above suggest automation as a potential solution. Prior research has focused on automating only fragments of the safety case, such as safety analysis techniques that provide the necessary evidence or the automatic generation of abstract arguments. Moreover, lack of tool support renders evaluation of newly introduced methodologies difficult.

This paper highlights our effort in supporting the safety case generation process via live connection to system models and automation. This approach integrates the model-based design paradigm with safety analysis methods and integrity levels allocation. The generated model-based safety cases are connected to design models and can be automatically updated in accordance with system changes as these happen during design phase and beyond. The key contributions of the paper are:

- The novel concept of integrating model-based safety analysis with safety integrity levels and user-defined safety argument patterns to automatically construct and maintain safety argument structures which are compliant with contemporary standards.
- The metamodel that underpins the operationalisation of the proposed method.

Alongside these concepts, a supporting software tool is being developed, to enable evaluation of the method through case studies. The paper focuses on the civil aviation industry, under the guidelines of ARP4754-A safety standard, though the approach is generally applicable to other safety-critical industries that centralise safety around the concept of safety integrity levels (SILs). To safeguard against hazards that arise from the development of software and electronic hardware components, which are challenging to address with traditional methods, the ARP4754-A safety standard has introduced the Development Assurance Process. This entails the notion of Development Assurance Levels (DALs), which encapsulate the level of rigour of safety assurance tasks across the system architecture. DALs are derived from the concept of SILs introduced in earlier safety standards, but are specialized for the aerospace industry. A method to optimally allocate DALs was proposed in [6], based on the decomposition rules established by the standard and a cost estimation for implementing an element of the system with a particular DAL. The method was developed as an extension to the reliability analysis method Hierarchically Performed Hazard and Origins Propagation Studies (HiP-HOPS) [7]. Earlier work illustrates how to generate preliminary safety argument structures using GSN's argument patterns [8].

The paper is organised as follows: Sect. 2 introduces the safety argument notations and the safety assessment processes under the guidelines of the ARP4754-A standard. This section also provides the essential background in safety analysis methods and related work. In Sect. 3, we describe our method for automatically constructing safety arguments with an example and provide our metamodel. In Sect. 4 we conclude and discuss benefits and limitations.

2 Background

2.1 Safety Argument Notations

There is a substantial body of work that aims to provide a structured way of constructing and representing safety arguments. The Goal Structuring Notation (GSN) originated from the University of York in the early ’90s [9]. GSN aims to provide a systematic process for construction, maintenance and representation of GSN-structured arguments. Another approach is the Claims Arguments Evidence (CAE) notation, which was developed in the late ’90s by Adelard, an independent specialist consultancy in the UK [4].

To represent safety arguments, GSN uses goal, strategy and solution nodes amongst other elements. CAE uses claims, arguments and evidence as its core elements. Goals or claims represent requirements, objectives, or other properties the system is argued to fulfil or intermediate inferential steps within the argument. Strategies or arguments describe the rationale that links goals or sub-goals with the corresponding evidence. Solutions or evidence are references to information usually deduced from analyses in order to support claims. These elements are rendered in the form of standardized graphical shapes (e.g. goals are represented as rectangles). The popularisation of software design patterns, most notably through [10], were adapted for use within safety cases. Specifically, GSN’s argument patterns were introduced by Tim Kelly in [3] as a means to support the reuse of successful safety arguments between safety cases. Argument patterns capture and promote best practice in safety case design by dissociating the details of an argument from the context of a particular system.

2.2 Safety Assessment in Civil Aircraft

The foundation for safety assessment and the procedures for generating the appropriate evidence are provided by one or more safety standards and differ from industry to industry. Aerospace Recommended Practice (ARP) is a set of standards developed by regulatory bodies and engineers to provide generic guidelines towards the development of civil aircraft and corresponding systems. The ARP4754-A provides general guidance for the development of aircraft functions and systems. ARP4754-A has adopted the concept of Safety Integrity Levels (SILs), known as Development Assurance Levels (DALs) in the aerospace industry. DALs describe the level of rigour necessary for the safety assessment activities applied to corresponding parts of the aircraft architecture. The standard defines 5 DALs: from E (least stringent) to A (most stringent). Regulatory authorities and the standard encourage applying safety assessment in a top-down sequence to better synchronize with system development.

The standard focuses on two major architectural concepts; functions and items. Functions describe an intended high-level behaviour, such as navigation or flight control. Items define the hardware or software components that are responsible for performing said functions. Aircraft functions are typically identified at the early stages of development, during a process known as functional analysis. Failures or errors associated with functions might relate to system hazards and can be identified via a classic analysis technique known as Functional Hazard Analysis (FHA). The standard defines those

associated hazards as Failure Conditions (FCs). Each FC is associated with a severity classification, ranging from No Safety Effect to Catastrophic based on the FC's effect on the aircraft and its occupants. It is important to note that the FHA is revised as soon as new functions or FCs emerge over the course of development. Following the FHA, candidate system architectures supporting functions are evaluated via the Preliminary System Safety Assessment (PSSA) and Common Cause Analysis (CCA) processes. During PSSA, aircraft or system requirements are established and appropriate DALs are assigned based on the FC severity classification from the FHA. Additionally, preliminary evidence that these architectures can meet safety requirements is provided. In CCA, physical and functional requirements are distributed across systems and validate that these have been met. Once a preliminary architecture has been established, a structured failure analysis, such as Fault Tree Analysis (FTA) [11], is conducted to determine if and how failures that can trigger FCs propagate within the architecture. Through FTA, the minimal cut sets are determined, which in the ARP4754-A are stated as Functional Failure Sets (FFSs). The FFSs contain the minimal combinations of basic failure events that are necessary and sufficient to cause a system failure (top event). In general, FFSs highlight vulnerabilities in the system design, such as single points of failure [12]. They are particularly useful for determining the appropriate DALs for systems and items during the PSSA [13]. DAL allocation is applied recursively, from higher to lower levels of the system architecture, and iteratively, following architectural changes. The process completes when the system design is determined and development proceeds with the implementation of components [14].

2.3 DAL Decomposition Rules

During the FHA process, DALs are assigned to top-level aircraft functions based on their highest FC severity classification. The allocation of DALs to systems and items is performed during the PSSA in a top-down approach across the aircraft architecture. The main concern involves cases where a combination of failures of systems, sub-systems or items leads to top-level failure conditions. Thus, to systematically assign DALs to the lower levels of architecture, the ARP4754-A supports the idea of utilizing the FFSs for DAL allocation and is referred to as DAL 'decomposition'. System safety assessment techniques, such as FTA or Markov Analysis [15], are conducted to identify the FFSs for each failure condition and the member of each FFS. Once the FFSs have been identified the ARP4754-A provides the following two rules for allocating DALs to the sub-systems.

- One of the members of the FFS that contributes to the top-level FC is assigned the same DAL with the parent system, whereas the rest of the members are assigned an equal or up to two levels lower than the top system. If the FFS of the system has only one member, then the first rule is obligatory.
- Two of the members of the FFS that contribute to the top-level FC are assigned one lower level DAL, whereas the remaining members are assigned an equal or up to two levels lower than the top system.

Even though only two options are provided for a single combination of members that lead to a system failure, the overall number of alternatives is subject to combinatorial explosion as FFS members increase and members participate across multiple FFSs.

2.4 Related Work on Automatic Construction of Safety Cases

Previous work towards automating the construction of safety cases in [16] focused on generating safety cases for automatically generated code based on formal software safety verification. The basic argument structure is generated via formal analysis of automatically generated code and is adjusted based on the set of formal requirements and assumptions. The contextual elements and other supportive information within the safety argument are derived from other verification activities. In [17], the method presented achieves safety argument construction from 'safety contracts'. These contracts encapsulate arguments of safety properties for commercial off-the-shelf (COTS) software components. The authors generate such contracts from the model of a given COTS component following failure analysis via Fault Propagation and Transformation Calculus (FPTC).

The method proposed in this paper shares some similarities with the methods above; however, there are substantial differences. First, our approach can be applied from the early stages of design and requires less rigorous annotation of the system model compared to formal methods for software components. Second, our approach incorporates the widely-employed concept of SILs (DALs in this case). Last, our approach is a top-down method and our notion of reuse applies on the level of systems and components instead of exclusively to COTS software components.

Another approach that shares similarities with our method is presented in [18]. Specifically, the author integrates compositional safety analysis, allocation of safety integrity requirements, assurance case techniques and variability management into software product line engineering (SPLE) processes. Software Product Line (SPL) is a development method that enables a set of software-intensive systems, which share similar characteristics and fulfil identical purposes, to be developed from a set of core assets in a prescribed way [19]. The approach in [18] focused on providing a systematic way to reuse safety analysis and assurance case safety artefacts for SPL, whereas our method applies to a general range of products from hardware to avionics.

Earlier work in [20] has established a model, known as the 'Weaving model', supporting model-based assurance case development. The Weaving model captures dependencies and reference information across the assurance case and information models, enabling automatic argument instantiations and supporting traceability. In our approach, the model of the system is extended instead to contain all the appropriate information in the form of containers and/or properties, which are exploited for the argument structure generation.

3 Model-Based Safety Argument Construction

The reason that we believe that model-based safety cases can be auto-constructed is that modern standards seem to converge to a common pattern for arguing safety. This pattern can be found in one form or another in the automotive ISO26262, the aerospace APR 4754-A and the generic IEC61508 standard. In all these standards safety is defined as a property that is controlled from the early stages of design and is not left to emerge at the end. At the early stages of design, a process of risk analysis is recommended to establish the system safety requirements by examining the system in its environment. Once an architecture for the system is developed, then designers are asked to determine the integrity requirements of system elements that will fulfill the safety requirements of a system. The allocation should respect dependencies in the model which propagate failures and the overall procedure can be iterated as the system is refined from subsystem to component levels. With a sound process in place, it is possible to argue that a system is adequately safe because all hazards identified in the systematic risk analysis can be shown to be addressed through meeting the appropriately allocated subsystem and component integrity requirements. This means that conceptually the structure of a safety case will always have the same logical form. This above is illustrated in Fig. 1.

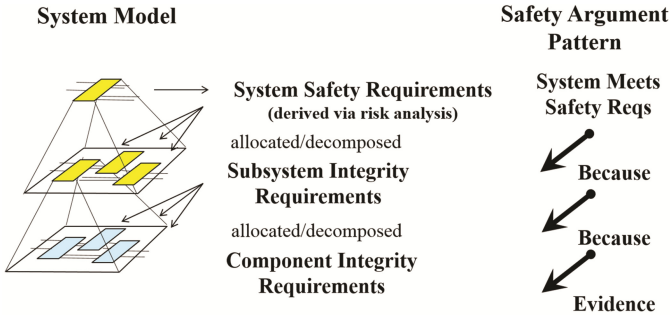

Fig. 1. Diagrammatic overview of the approach

The process illustrated in Fig. 1 is the conceptual basis for the proposed approach. The approach is operationalised by exploiting capabilities of the HiP-HOPS tool. For the aerospace sector, we build on the DAL allocation approach presented in [6].

The results of a risk analysis performed at system level can be inserted as identified hazards and safety requirements in the HiP-HOPS tool. Hazards are linked to logic that connects functional failures at the outputs of a system and requirements for avoidance are specified as DALs. A safety engineer then develops a model of the system architecture in a modeling tool (e.g. MATLAB Simulink or equivalent) and annotates systems and components with local failure behaviour information. The model is parsed by HiP-HOPS, which automatically analyses the model, produces fault trees and then calculates FFSs. This analysis helps to determine the contribution of components to system failures and provides a basis for automatically and cost-optimally allocating DALs across the system architecture. We extend this approach by automatically instantiating a safety

argument pattern that corresponds to the reasoning of the standard. This enables the automatic construction of a safety argument structure.

To improve upon the benefits of HiP-HOPS methodology and its extensions, we are developing an integrated development environment (IDE). Currently, to address model changes, the user is required to introduce them and repeat file parsing across the toolchain involved to obtain the revised safety argument. Considering the vast amount of changes that can occur during a development lifecycle, significant time is spent repeating this arduous task. Additionally, version control of the model and relevant information is currently manual, exacerbating the aforementioned issues. The IDE currently being developed aims to address these inefficiencies and integrates a graphical editor with HiP-HOPS and its various extensions. For example, based on the corresponding FFSs produced by the HiP-HOPS engine, the systems and components will be automatically allocated with the appropriate DALs. With the information about the target system, the engineer is able to manually develop a suitable argument pattern to define the desired argument structure and proceed to the argument generation. As such, if a change to the system occurs, the engineer will only have to update the system model and failure behaviour, assuming the argument pattern remains suitable, the argument will be generated without further effort. On the other hand, if changes occur to the safety assurance process (i.e. an assumption becomes invalidated due to testing) the engineer will have to manually incorporate the changes in the argument pattern. That being said, the latter can require significantly less effort compared to manually altering the argument structure itself, given the potential for changes in the pattern to repeat across generated arguments. This methodology extends the notion of classic safety cases, presently document-based, to a model-based safety case, where system certification procedures are achieved automatically within a software tool.

The metamodel we employ extends the HiP-HOPS metamodel, which combines the system model with elements supporting FTA and failure behaviour annotation. Key structural elements of the HiP-HOPS metamodel are; (a) the "model", the top-level element that is used to contain all other system-related elements, (b) the systems/subsystems, (c) the components and (d) (fault tree) events, including basic events and intermediate nodes. Basic events are base sources of failure, e.g. component's lack of output. Intermediate nodes propagate combined failure from other basic events, usually via Boolean logic gates. For simplicity, we present only a subset of the core elements. We extend the metamodel to support automatic generation and maintenance of safety arguments with elements similar to GSN and OMG's ARM metamodels [21]. Our metamodel is featured in Fig. 2, which illustrates inter-element relationships. On the right side of the figure, we find the system model and related elements. Integrity requirement support is found through the model, system and component elements on the right side of the figure. The center of the figure is populated by elements related to safety assessment artefacts that HiP-HOPS generates such as fault trees. On the left of the figure, safety argumentation elements such as goals and solutions are included. Both FTA and FMEA outputs are usable as safety artefacts, part of the evidence supporting the safety argument in the form of GSN solutions.

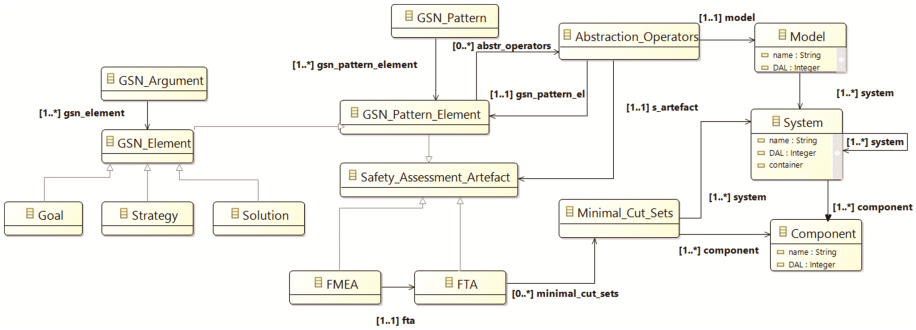

Fig. 2. Prototype of tool's metamodel

Finally, the abstraction operators suggest GSN's structural abstractions of multiplicity and optionality. The former enable sub-graphs of the argument to be applied iteratively across sets of contextual elements, whereas the latter enables the inclusion of optional strands of argumentation. Our metamodel aims to support referencing of reusable safety artefacts generated automatically by HiP-HOPS in safety cases. Towards this aim, argument patterns are linked to the system model and refer abstractly to model and safety assessment elements.

3.1 Automatic DAL Decomposition

The choice of DAL dictates the safety arguments associated with particular functions, systems or items. A brief example is provided to demonstrate how DALs are assigned onto an abstract system. In Fig. 3, the architectural model of an abstract system is presented. The system has a single output and comprises two sub-systems, which in turn include two elements each; A, B, C and D. The element in dotted lines (component E) will be added later when design changes emerge to showcase the maintainability of our approach. Logical gates define how the failure of the elements may lead to the system's functional failure. Assuming the system function has been assigned DAL A from the PSSA, following the ARP4754-A guidelines the following options for the components are available in Table 1. In this example, two FFSs occur; in FFS 1 the system function will fail if components A, C and D fail. In FFS 2, the system will fail if components B, C and D fail. The corresponding options are identical across the two FFSs. The total range of options for allocation can be formed by combining the possibilities from each FFS.

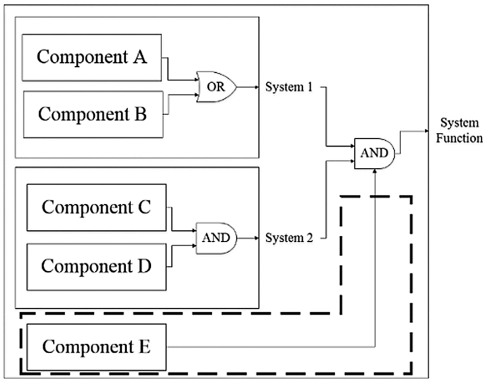

Fig. 3. Example model of an abstract System

Table 1. DAL decomposition options for components.

FFS	FFS 1		
Components	Component A	Component C	Component D
Option 1	A	C	C
Option 2	C	A	C
Option 3	C	C	A
Option 4	B	B	C
Option 5	B	C	B
Option 6	C	B	B

Furthermore, ARP4754-A explicitly states that regardless of the number of functional decompositions, it is important to apply the options that correspond to the DAL allocation of the given top-level FC (i.e. DAL A).

At this stage, we can determine the most cost-effective option by evaluating the cost of implementing each element of the architecture with the given DAL. Table 2 provides cost values for each DAL for illustrative purposes.

Table 2. Cost of DALs for the abstract system.

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

Adding the costs of the component DALs, we identified three groups of allocations with identical costs. For example, allocating DALs C, C, A, C to components A, B, C, D yields one optimal solution of cost 220. The simplicity of this example translates into a rather trivial solution. If we examine a system with twice as many components, then the solution would not be so apparent. The more complex a system is, the more extensive the design space of available options becomes. This renders the exhaustive search for

optimal solutions into an intractable problem for systems of non-trivial scale or complexity.

The combinatorial nature of the problem rendered exhaustive techniques in [22] inadequate for large scale systems. The authors focused instead on metaheuristic techniques. Metaheuristic techniques do not guarantee optimal results, but are known to reliably achieve nearly optimal solutions. Specifically, in [6, 22] the metaheuristic method Tabu Search was adopted for its superior performance when allocating ASILs and DALs respectively. The method initiates with a random, yet feasible, solution of allocated DALs across the system architecture. Then, it iterates through the neighbouring solutions for lower cost allocations. The approach features a memory structure known as a 'Tabu Tenure'. This memory structure registers the recent allocations investigated. The recently registered states are avoided and search moves towards different areas in the design space that might hold better solutions. Finally, the use of an Aspiration Criterion allows the search to select candidate solutions that are better than Tenure's current best solution, ignoring the Tabu Tenure.

3.2 Argument Pattern Instantiation

Figure 4 features a part of the model (i.e. in Fig. 3 Sub-system 1 and its components) and the part of the argument pattern that corresponds to the derived argument structure for these elements in XML (i.e. Fig. 5 G5 and its children nodes). We instantiate the argument pattern from the bottom part of Fig. 4 using the model information from the top part of Fig. 4. The pattern is able to retrieve the information through the use of the text elements in brackets. For instance, in {S} a system is referenced defined earlier in the pattern. Properties of {S} are accessed via the "dot" operator; for instance, the

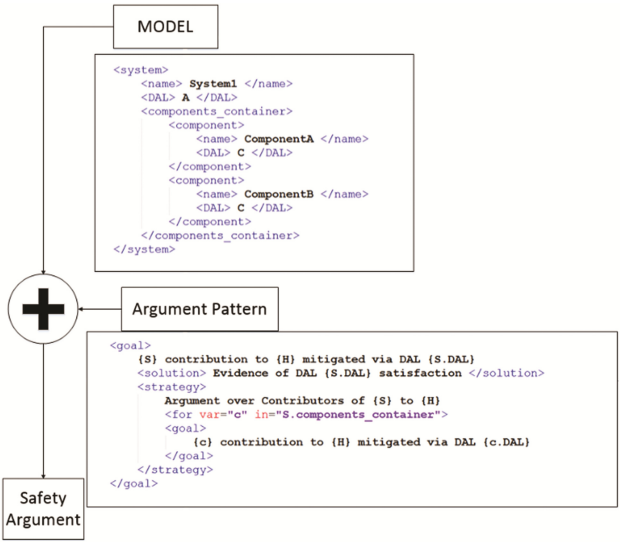

Fig. 4. Model and pattern in XML

system's DAL can be accessed this way. The "for" element pattern enables iteration over the contained elements of the system (i.e. the components). Each component is referenced through the "var" declaration in the pattern. The system's components container acts as the source of each component variable. The approach shown here can be repeated throughout the entire pattern to synthesize argument structures, which can span the entire system architecture.

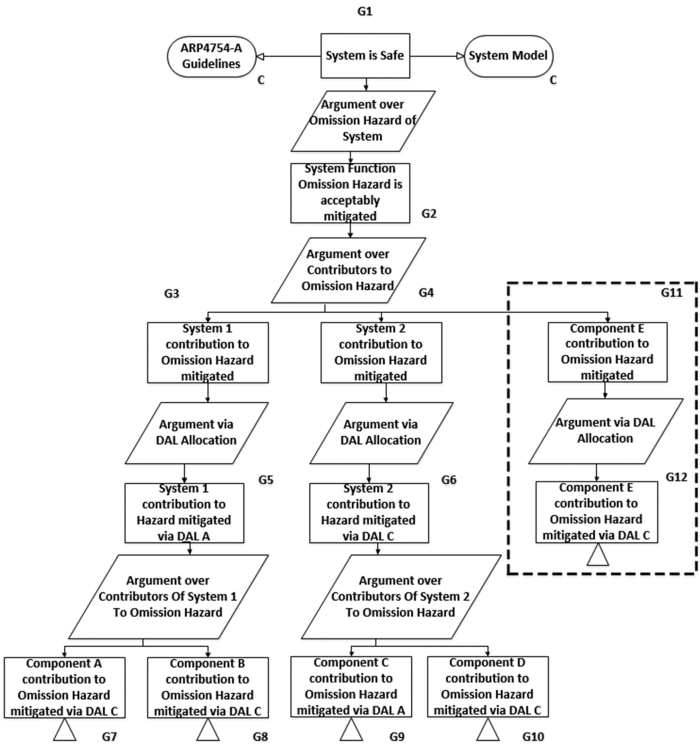

Fig. 5. Abstract safety argument from example

3.3 Argument Structure Generation and Maintenance

Figure 5 shows the safety argument structure produced from the example in Fig. 3 (based on the most cost-optimal option in Sect. 3.1). The argument is constructed with a simplified version of GSN for demonstration purposes. The rectangles represent claims (G1–G10), the parallelograms represent arguments, the circular objects represent evidence and lastly the ovals represent context (C). The directed lines connect and indicate the supported elements. The part of the argument within dotted lines will be added after design change. The argument claims system safety (G1) if all the hazards have been mitigated. This is achieved via DAL assignment to all the elements and/or systems that contribute to the hazard. In this example, further claims and evidence that support

how the components satisfy the assigned DAL are left 'undeveloped' for simplicity and a triangle is placed to indicate that purpose.

Following system evaluation, the engineers may decide to change the design and add the component within the dotted lines to the system. Subsequently, a new safety argument can be constructed to claim system safety. The introduction of the new component changes the results in Table 1 by providing new allocation options.

The argument is updated based on the new most cost-effective option and claims that in addition to the sub-systems, the new component E also contributes to the mitigation of the hazard via its DAL. This option is similar to the allocation governed the argument structure initially, but now has also assigned a DAL C to the new component. Similar to the previous argument, further claims for component E (in this case, undeveloped) would normally follow. In our example, the generated argument structures are only partial and in practice would require appropriate evidence. The type of evidence is defined by the corresponding standards and aims to show that any component requirements, assumptions as well as component independence, fault propagation and fault mitigation have been met in practice. Part of the evidence comprises failure analysis, such as FTA and Failure Mode and Effects Analysis (FMEA), which are susceptible to system changes and frequent reapplication is required. Performing those analyses manually for every design change of the system during the development cycle is a time-consuming and error-prone task. However, model-based techniques such as HiP-HOPS can alleviate this burden by automating the safety assessment.

4 Conclusion

We have demonstrated construction of a safety argument structure from a system model by addressing the decomposition of integrity requirements under the ARP4754-A guidelines. The method connects the safety cases to the design model. Hence safety argument maintenance is more efficient following design changes, as demonstrated in Sect. 3. The progressive use of SILs/DALs throughout the various safety-critical domains means that our approach can be adopted and with little effort expand in those domains. To the best of our knowledge, there is not a similar approach on safety argument generation via the automatic allocations of SILs that applies on a generic array of systems and not only in software-intensive systems as presented in [18].

Currently, the tool is still under refinement while larger case studies are being developed. It is clear that the proposed method cannot create a complete safety case. The latter requires the inclusion of all the supplementary documents such as the DO-178C for software components, DO-254 for hardware components and DO-297 for integrated avionics. Additionally, it requires the incorporation of process-based arguments that provide support for justifying the confidence of the processes utilized to generate the evidence. However, our method does capture and realise a general syllogism of how to argue safety, which is compatible with many contemporary standards. The produced structure argues that the system examined is adequately safe because all hazards identified in a systematic risk analysis are addressed through meeting the appropriately determined safety requirements. This overarching claim is then demonstrated via

satisfaction of the integrity requirements that are assigned to subsystems and components of the system through a sound process of model-based allocation that respects system design, dependencies, and follows industry standards. The argument patterns are one of the key elements in the model-connected safety cases that can be produced by this method. While the user-defined argument pattern stays the same, its instantiation changes every time a new system is considered or the model of the system under examination changes. The benefit of the approach is that changes in the structure of the safety case or the evidence supporting it can be effected in a largely automatic fashion by exploiting the connection of the safety case to the design model.

The evaluation of the method relies on case studies and well-defined criteria. Scalability is one of our main concerns which is implicitly supported by the use of argument modules, the algorithm responsible for the automatic instantiation of the argument pattern and other elements that will enable iteration and recursion. Naturally, the evaluation is being supported by quantitative results, obtained by examining larger case studies with our tool.

Given that the proposed approach can only generate part of the structure and evidence one can expect to find in a typical safety case, it is envisioned that the method will be part of a safety case approach in which some parts of the safety case are manually defined while other parts are connected to design models and are auto-updated as these models evolve.

Acknowledgments. This work was partly funded by the DEIS H2020 project (Grant Agreement 732242).

References

1. Kelly, T.P.: A Systematic Approach to Safety Case Management. SAE International (2003)
2. Kelly, T.P., Weaver, R.: The goal structuring notation – a safety argument notation. In: Proceedings of Dependable Systems and Networks, Workshop on Assurance Cases (2004)
3. Kelly, T.P.: Arguing safety – a systematic approach to managing safety cases. Thesis, University of York (1998
4. Bishop, P., Bloomfield, R.: A methodology for safety case development. In: Proceedings of the Sixth Safety-Critical Systems Symposium on Industrial Perspectives of Safety-Critical Systems, Birmingham, UK (1998)
5. Hawkins, R., Clegg, K., Alexander, R., Kelly, T.: Using a software safety argument pattern catalogue: two case studies. In: Flammini, F., Bologna, S., Vittorini, V. (eds.) SAFECOMP 2011. LNCS, vol. 6894, pp. 185–198. Springer, Heidelberg (2011). doi: 10.1007/978-3-642-24270-0_14
6. Sorokos, I., Papadopoulos, Y., Azevedo, L., Parker, D., Walker, M.: Automating allocation of development assurance levels an extension to HiP-HOPS. In: Lopez-Mellado, E., Ramirez-Trevino, A., Lefebvre, D., Ortmeier, F. (eds.) 5th IFAC International Workshop on Dependable Control of Discrete Systems – DCDS (2015). IFAC-PapersOnLine **48**(7), 9–14
7. Papadopoulos, Y., Walker, M., Parker, D., Rude, E., Rainer, H., Uhlig, A., Lien, R.: Engineering failure analysis and design optimisation with HiP-HOPS. In: Gagg, C., Clegg, R. (eds.) The Fourth International Conference on Engineering Failure Analysis, Part 1 (2011). Eng. Fail. Anal. **18**(2), 590–608

8. Sorokos, I., Papadopoulos, Y., Bottaci, L.: Maintaining safety arguments via automatic allocation of safety requirements. In: Emmanouilidis, C., Iung, B., Macchi, M., Peres, F. (eds.) 3rd IFAC Workshop on Advanced Maintenance Engineering, Services and Technology, AMEST, Biarritz, France (2016). IFAC-PapersOnLine **49**(28), 25–30
9. Origin Consulting (York) Limited: GSN Community Standard Version 1 (2011)
10. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley Professional, Boston (1994)
11. Vesely, W., Goldberg, F., Roberts, N.: Fault Tree Handbook. Nuclear Regulatory Commision, Washington, DC (1981)
12. Vesely, W., Dugan, J., Fragola, J., Minarick, J., Railsback, J., Stamatelatos, M.: Fault Tree Handbook with Aerospace Applications. NASA Office of Safety and Mission Assurance, Washington, DC (2002)
13. ARP4754-A: Guidelines for Development of Civil Aircraft and Systems. SAE Aerospace (2010)
14. Joshi, A., Heimdahl, M., Miller, S., Whalen, M.: Model-Based Safety Analysis. NASA Langley Research Center, Hampton (2006)
15. Fuqua, N.: The applicability of markov analysis methods to reliability, maintainability, and safety. In: Start, vol 10, no. 2 (2003)
16. Basir, N., Denney, E., Fischer, B.: Building heterogeneous safety cases for automatically generated code. In: AIAA Infotech@Aerospace Conference (2011)
17. Sljivo, I., Gallina, B., Carlson, J., Hansson, H., Puri, S.: A method to generate reusable safety case fragments from compositional safety analysis. In: Schaefer, I., Stamelos, I. (eds.) ICSR 2015. LNCS, vol. 8919, pp. 253–268. Springer, Cham (2014). doi: 10.1007/978-3-319-14130-5_18
18. Oliveira, A.: A model-based approach to support the systematic reuse and generation of safety artefacts in safety-critical software product line engineering. Thesis, Instituto de Ciencias Matematicas e de Computacao (2016)
19. Clements, P., Northrop, L.: Software Product Lines: Practices and Patterns. Addison-Wesley, Boston (2001)
20. Hawkins, R., Habli, I., Kolovos, D., Paige, R., Kelly, T.: Weaving an assurance case from design: a model-based approach. In: 16th IEEE International Symposium on High Assurance Systems Engineering, pp. 110–117 (2015)
21. Object Management Group (OMG): Structured Assurance Case Metamodel (SACM), Version 2.0 (2016)
22. Azevedo, L., Parker, D., Walker, M., Esteves, A.: Assisted Assignment of Automotive Safety Requirements. IEEE Softw. **31**(1), 62–68 (2014)