A Pattern-based Approach towards Modular Safety Analysis and Argumentation

Maged Khalil, Bernhard Schatz, Sebastian Voss

A Pattern-based Approach towards Modular Safety Analysis and Argumentation

Maged Khalil, Bernhard Schätz and Sebastian Voss

fortiss GmbH - Software and Systems Engineering Dept., Guerickstr. 25, 80805 Munich, Germany
khalil@fortiss.org | schaetz@fortiss.org| voss@fortiss.org

Abstract: Safety standards recommend (if not dictate) performing many analyses during the concept phase of development as well as the early adoption of multiple measures at the architectural design level. In practice, the reuse of architectural measures or safety mechanisms is widely-spread, especially in well-understood domains, as is reusing the corresponding safety-cases aiming to document and prove the fulfillment of the underlying safety goals. Safety-cases in the automotive domain are not well-integrated into architectural models and as such do not provide comprehensible and reproducible argumentation nor any evidence for argument correctness. The reuse is mostly ad-hoc, with loss of knowledge and traceability and lack of consistency or process maturity as well as being the most widely spread and cited drawbacks.

Using a simplified description of software functions and their most common error management subtypes (avoidance, detection, handling, ..) we propose to define a pattern library covering known solution algorithms and architectural measures/constraints in a seamless holistic model-based approach with corresponding tool support. The pattern libraries would comprise the requirement the pattern covers and the architecture elements/ measures / constraints required and may include deployment or scheduling strategies as well as the supporting safety case template, which would then be integrated into existing development environments. This paper explores this approach using an illustrative example.

Keywords: Safety-critical systems, pattern-based design, architectures, safety cases, automotive

1. Introduction

From advanced driver assistance and hybrid drives over autonomous high-speed rail and high precision medical equipment to Unmanned Aerial Vehicles: software-intensive systems that perform safety-critical tasks are increasingly prevalent and pervasive in today's world. Driven by the incessant increase in the number of integrated control units, communication systems and software, managing architectural complexity, let alone mastering it, is becoming an increasingly difficult task. This difficulty in turn translates into a heightened possibility of design and implementation errors going undetected with hazardous consequences. This challenging situation is further exacerbated by the coupling of new development methods being employed for innovative technologies with increasingly complex international safety standards.

Safety standards recommend (if not dictate) performing many analyses during the concept phase of development as well as the early adoption of multiple measures at the architectural design level [1], [2], [3], [4], [5], [6], most of which has become part of the day-to-day business of safety-critical development, yet has to receive adequate tool support. This is particularly true if one wishes to front-load these aspects into an integrated solution environment, in which these (mostly) repetitive tasks can be automated. Dealing with non-functional requirements, especially safety, at later development stages is difficult and highly costly [7], [8], [9].

Because these analyses and architectural measures have to be conducted during the design phase of the systems, they cannot be based on physical prototypes and implemented software, and, hence, have to rely on the design information in terms of requirements, the structural design, functional specifications, and the principled knowledge about the (nominal and potential deviating) behavior of the components used [10]. The field of model-based problem solving [11] has generated theoretical foundations, prototypical solutions, and products for modeling artifacts and natural systems and for using models to solve a variety of tasks, with a major focus on automated diagnosis, but also solutions for failure modeling as well as for automated software FMEA [12], [13].

The use of patterns in safety-critical software development in conjunction with model-based development techniques is documented and well suited for these needs. Patterns of safety cases for well-known problems have been suggested in academic literature as well [14], [15], [20], [23], [24], [25] and [26]. A safety case is "a documented body of evidence that provides a convincing and valid argument that a system is adequately SAFE

for a given application in a given environment", where an argument is "a connected series of claims intended to establish an overall claim." A safety case should communicate a clear, comprehensive and defensible argument that a system is acceptably safe to operate in a particular context [22].

Reuse in a safety-critical context, and particularly the reuse of safety cases, is neither systematic nor adequate [14]. In this paper we will show how reuse is simplified in a safety-critical context through the encapsulation of all information into a library element, along with the corresponding safety case to support it.

Chapter 2 details the problem. Chapter 3 explains our approach and gives an example of a pattern library element, as well as its usage in a wider context. In chapter 4 we discuss the impact of this contribution and, in conclusion, we summarize our work and detail open research questions and future work in Chapter 5.

2. Problem

In practice, the reuse of architectural designs, development artifacts and entire code sequences is widely-spread**,** especially in well-understood domains. This trend holds true for the development of safety-critical products, with well-established architectural measures and safety mechanisms in wide reuse, as is reusing the corresponding safety-case**s** aiming to document and prove the fulfillment of the underlying safety goals. This however, is marred by several problems:

- Safety-cases in the automotive domain are not well integrated into architectural models and as such
- they do not provide comprehensible and reproducible argumentation
- nor any evidence for the correctness of the used arguments.
- Most analyses (FMEA, FTA, etc.) have to be performed at system level, yet the components/ measures / safety mechanisms themselves need to be reused independently,
- and are not tied in any structured manner to other elements needed to provide the relevant context.

The reuse of safety-cases is mostly ad-hoc, with loss of knowledge and traceability and lack of consistency or process maturity as well as inappropriate artifact reuse being the most widely spread and cited drawbacks [14].

The typical reuse scenario would, in this context, involve reusing an architectural measure or design pattern, e.g., homogenous (hardware) duplex redundancy, in which a vulnerable critical single processing channel is duplicated to increase reliability as seen in Fig. [1], or a development artifact, such as a comparator function. If the previous safety case is at all reused, it serves as a detached guide of what needs to be provided to close the case, i.e. serving a prescriptive role, which is better than nothing.

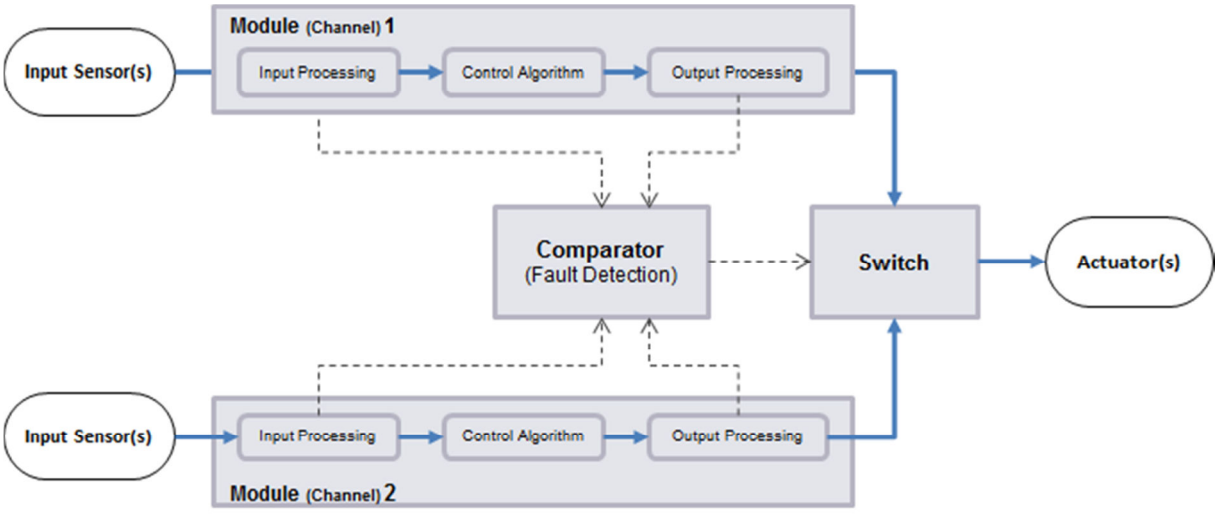

Figure (1): Homogenous (Hardware) Duplex Redundancy Pattern

Yet be it the design pattern or the development artifact, the single item does not tell the entire story. For example, to correctly deploy homogenous redundancy, many other aspects have to be covered:

- one has to define the requirements the pattern fulfills,
- refine the requirements and draw up a specification,
- detail a (logical) component architecture,
- and optimize a deployment strategy that guarantees the duplicate components will not run on the same hardware resource.

This has to be preceded by checking the availability or making the explicit assumption that the system layout allows, for example, a second hardware channel, which is a contextual bit of information. This is not all; to justify reusing this pattern, one would also have to include any tests or information proving that this particular pattern is suitable for the goal it targets.

Finally, all parts comprising this information, as well as their relations, which are more complex than simple tracing, must be captured in a comprehensive and comprehensible manner which should also provide a suitable interface to the environment the reused element will be deployed in.

As such, the reuse of well-understood and trusted design patterns or safety mechanisms cannot be confined to reusing the central artifact alone. Much of the information, in this case highly critical information, remains trapped in the heads of the developer and if mentioned at all retains a high measure of implicitness.

This gives rise to the need of some kind of encapsulation of the reusable safety mechanism (with requirements, specification, components, etc.), along with a minimum set of guarantees that can be achieved at that level and support via a solid argumentation.

3 Approach

Using a simplified description of software functions and their most common error management subtypes (avoidance, detection, handling, ..) it is possible to define a pattern library covering known solution algorithms and architectural measures/constraints in a seamless holistic model-based approach with corresponding tool support. The pattern libraries would comprise the requirement the pattern covers and the architecture elements/ measures / constraints required and may include deployment or scheduling strategies as well as the supporting safety case template, based on the established structure notation known as GSN [16] which would then be integrated into existing development environments. This enables an early analysis of hazards and risks, which is recommended by many safety standards. Subsequently, fault types can be matched both to probable hazards but more importantly to the problem categories they fall into or are most similar to, from a system architecture design viewpoint. Combining this with known architectural constraints and patterns for solving them, we can thus reason about which types of architectural patterns are relevant for the system under analysis. The fault types, along with their requirements, are bundled with solution arguments, comprising components, their (sub-) architectures, deployment plans and schedules, into pattern libraries, which are rounded up by the corresponding safety-case templates or skeletons to provide argumentation for achieving the goals.

Underlying the approach is the consistent use of patterns from the categorization of hazard types, over the abstract modeling of the respective safety concepts, and down to their implementation in the system architecture description, with a focus on providing argument chains in a seamless model-based environment.

In this paper, we mainly focus on the contents of the library of reusable patterns, and the necessary structure of the problem and solution patterns to support a structured description of the argument and provide the necessary seamless integration into a model-based system development environment. While we furthermore sketched the necessary mechanisms for the application of those patterns and the checking of the constructed argument, automation via tool-support is essential to achieve the intended advantages.

Figure (2) gives a sketch of a generic library element with some, not all, possible content, side by side with the supporting safety case view. The left part of the schematic gives possible parts of the reusable pattern, as detailed

in Chapter 2, while the right part shows the corresponding safety case and connections to the relevant artifacts, displayed in a graphical notation using GSN.

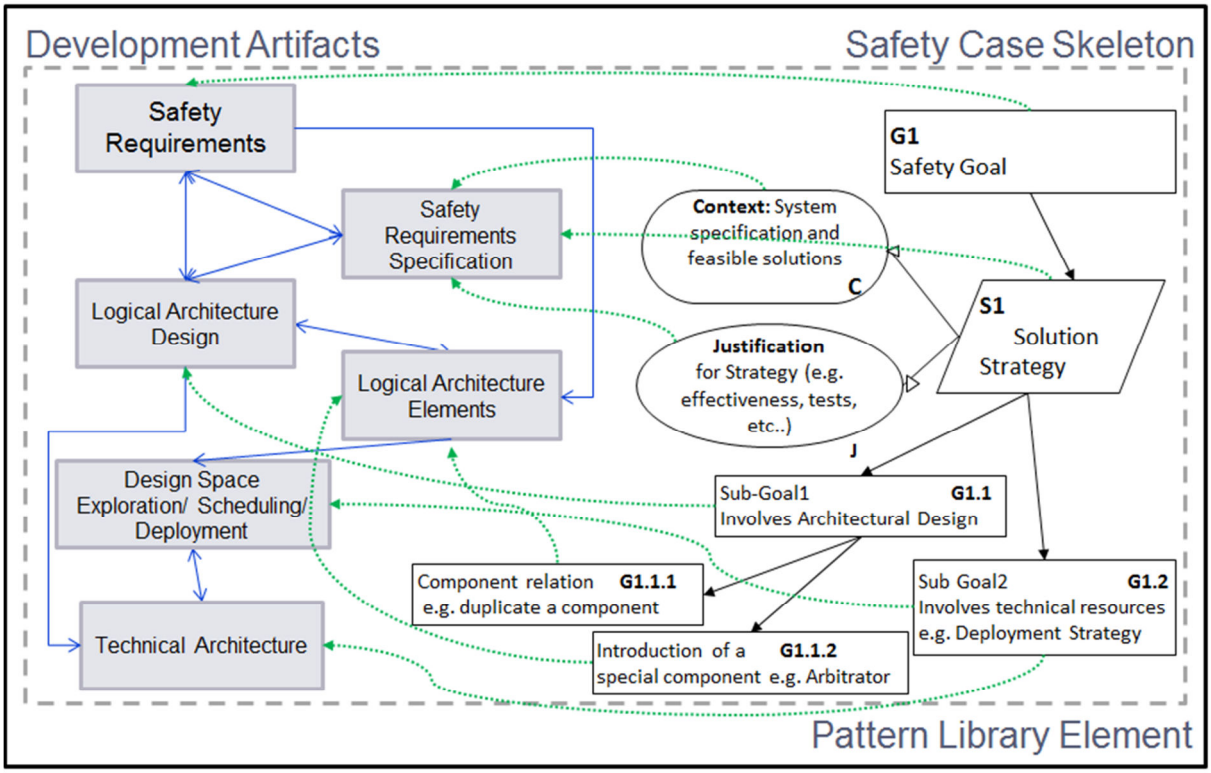

Figure (2): Generic Pattern Library Element

While the left part of the schematic can entirely originate in one seamless tool, as will be shown in our implementation example in Section 3.1, this is not necessary. The development artifacts may reside in multiple repositories and be in varying formats; the binding element is the safety case shown on the right hand side, which gives the story and rationale behind the reusable pattern. This particular aspect is particularly important for reuse in a safety-critical development context.

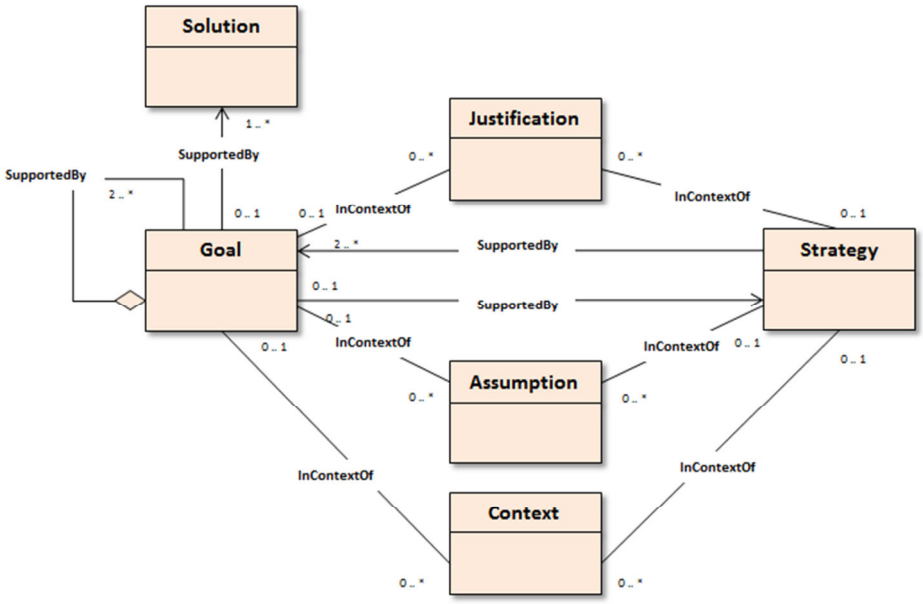

Figure (3): Data Model for GSN Representation

In our work we evaluated several possible ways to express safety cases and settled on the Goal Structuring Notation (GSN), in its Community Standard Version 1.0 [16], as it was the most mature option suited to our needs. GSN is a structured (yet not formal) notation, which provides graphically differentiated basic safety case element types, such as goals, evidence, strategy, justification and so on, as well as a clear description of necessary and allowed connection types between these elements. A data model of safety cases according to GSN is provided in figure (3). A comparison with the Toulmin argumentation pattern used in EAST-ADL 2.1 is provided in [18]. An illustrative implementation example is given in section 3.2.

3.1 Pattern types

The library of patterns can be categorized according to types, such as problem patterns, design patterns, and so on. As patterns may in fact belong to or satisfy more than one category, a cross referencing system using keywords will more likely be a more adequate approach. This will be explored in further work.

The use of patterns in the development of safety-critical products is already in wide spread use. Catalogues exist [26] that discuss highly organized and well-known safety mechanisms, such as comparators or "1outof2" voters. Safety case templates can be generated, stored and reused for many categories of patterns. Some of these patterns are truly abstract and tackle higher system description levels, such as the "High Level Software Safety Argument Pattern Structure" presented in [24], while some can target a certain context, such as the use of Commercial Off The Shelf Components (COTS) in a safety-critical context, as discussed in [25].

Safety case patterns can also be grouped according to their applications, such as:

- Strategy (Problem solving) patterns

The range of problems developers in a domain face, and subsequently the algorithms they favor in solving them, are not unlimited. By identifying recurring paradigms and analyzing them, it is possible to generate templates of their safety cases. The strategy node is the core element of the actual argumentation in a problem pattern safety case template, supporting the justification of goals being fulfilled by sub-goals. By applying strategy patterns, it is possible to build argumentations using only accepted justifications. This way confidence in the correctness of the argument can be increased. Still the appropriateness and correct use of a pattern has to be evaluated before trusting a safety case, but the question of the fundamental validity of each single argumentation step itself need not be argued. On the long run, it is desirable to establish general and domain-specific patterns as a pattern library for a faster and easier creation of safety arguments [15].

Example: Logical transformation

The required safety goal is a logical combination leading to desirable or undesirable situations. It is possible to generate a safety case skeleton that represents the pattern of avoiding a situation in which the constraint is violated, e.g. to keep the gap to the forward car larger than or equal to a minimum safe distance in an adaptive cruise control. This is simplified by transforming the goal into avoiding a gap which is smaller than the safe distance [15], as shown in figure (4).

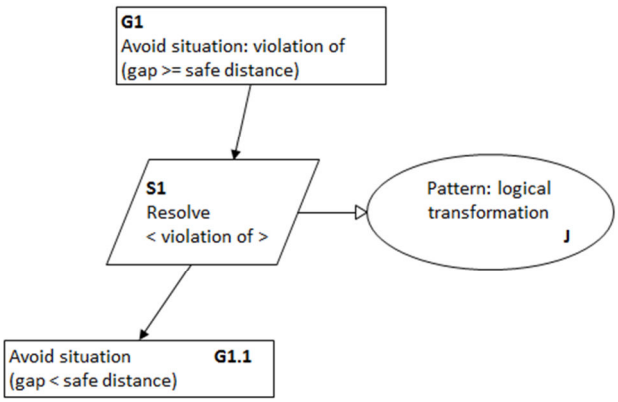

Figure (4): Logical Transformation Pattern for ACC distance control [15]

- Design (Constraints) patterns

Safety standards, such as the ISO26262, require that component architectures, independently of their functionalities, display certain characteristics or adhere to constraints. Some of these such as modularity, simplicity, and an adequate level of granularity and encapsulation are simply good engineering practices aimed at avoiding failures arising from unnecessary complexity.

Other requirements, such as *freedom from interference*, which is the absence of dependent failures (cascading and common cause failures) in safety-critical components [6], are aimed at guaranteeing correct operation of safety-critical functions.

According to ISO26262, there are several ways for components to have freedom from interference. The components can be deployed redundantly, or be functionally diverse (the use of totally different approaches to achieve the same results), or be based on diverse technologies (the use of different type of equipment to perform the same result), or be physically separated such that foreseeable failures do not affect redundant safety-related systems and so on.

Example: Redundancy

The guidelines above have in turn matured into various design patterns, many of which revolve around redundancy and partitioning, which is the separation of functions or component elements to achieve a design, which can be used for fault containment to avoid cascading failures. The design patterns vary in their addressed context, structure, and presented solution and can be categorized in many ways. For example, for redundancy alone, [27] lists 8 hardware patterns, i.e. patterns that contain explicit hardware redundancy, 5 Software patterns, and two combined patterns.

All of the above patterns do however, represent one general paradigm and as such can be covered by a general safety case pattern template as shown in figure (5), namely that of solving the lack of confidence in the safety-critical channel using a redundancy strategy. This general template can in turn be instantiated and subsequently further specified and extended to suit the specific case and used solution, for example using context elements [18], and would for the case of our Homogeneous Duplex Redundancy Pattern example yield the more detailed safety case discussed in the example in section 3.2 and seen in figure (6). The choice of which method or pattern to employ depends on the system constraints, and thus does not tell the whole story on its own.

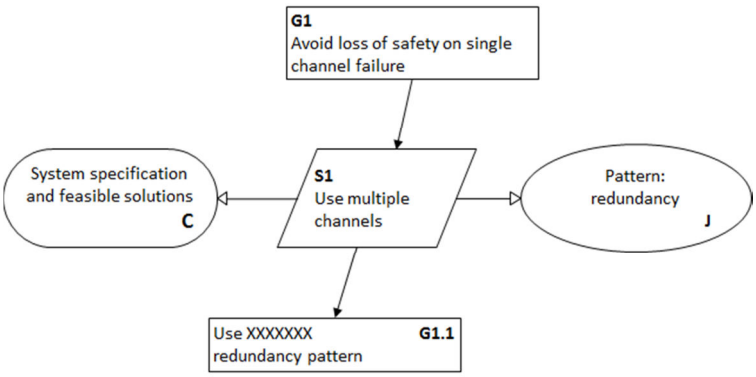

Figure (5): Pattern Library Element: Homogeneous Duplex Redundancy Pattern [18]

As shown by the example in the following section, the approach should lend itself well to any well-understood reusable pattern or component. We will implement further examples in future work.

3.2 Implementation example

We have carried out a specification in the integrated development environment for the ITEA2 SAFE project [17], [18] as well as examined a proof-of-concept in the research CASE tool AUTOFOCUS3(AF3) [19] to support seamless safety-case modeling and partially automated generation in a model-based development environment. While we haven't yet fully implemented a library, preliminary results show the approach is feasible and should scale well. This will be proven at a later stage when safety cases are fully supported in AF3.

It also identified compositionality of argumentation as being a major hurdle and open research question for the successful reuse of safety cases. The underlying approach was well received by the reviewing OEM advisory board industry experts (SAFE project) and published in [18], [20] and [21].

Using the example of homogenous hardware redundancy, introduced in Chapter 2 and shown in figure (1), figure (6) shows an exemplary pattern library element. On the left are snapshots of screen grabs of potential candidate functionalities for the generation of pattern artifacts from the research CASE tool AF3 [19][21][28] and on the right is the corresponding (simplified) safety case.

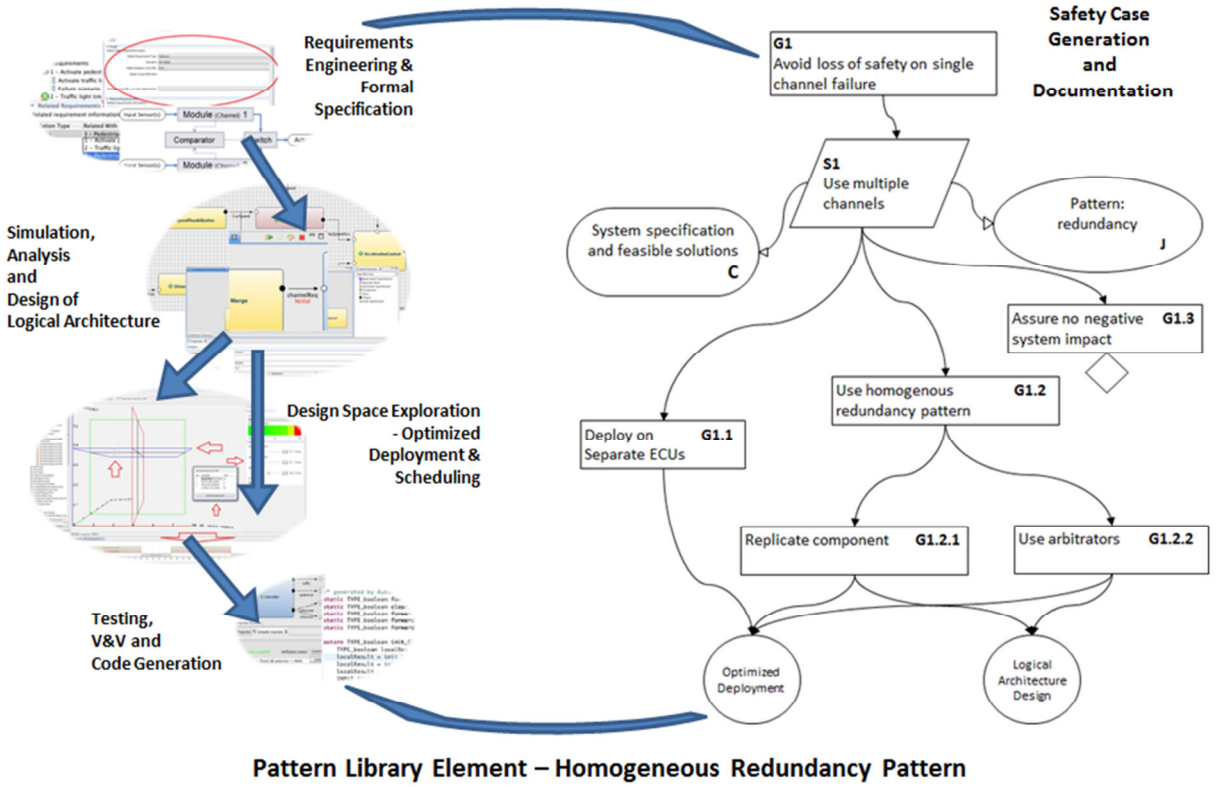

Figure (6): Pattern Library Element: Homogeneous Duplex Redundancy Pattern

This single reusable pattern element would thus contain the requirements and their refinement, specification, logical components and logical architecture design, deployment strategy as well as all relevant tests and their results, held together by the links from all relevant artifacts to the corresponding safety case elements. Using the relations described in the GSN data model described in figure (3), it is possible to perform completeness and consistency checks on the safety case, such as "*solutions must stem from goals and no other elements*" or "*no open goals remain*". All the information required to reuse the pattern is now encapsulated in a clear defensible package, enabling modular argumentation.

3.3 Modular and compositional argumentation

The use of patterns as a way of documenting and reusing successful safety argument structures was pioneered by Kelly [29], who developed an example safety case pattern catalogue which provided a number of generic solutions identified from existing safety cases. Although providing multiple useful generic argument strategies, the catalogue merely represents a cross-section of useful solutions for unconnected parts of arguments. The safety argument pattern approach was further developed by Weaver [23], who specifically developed a safety pattern catalogue for software. The crucial difference being that this catalogue not only specifically targeted software aspects of the system under development, the set of included patterns were specifically designed to connect together in order to form a coherent argument. The solution was, however, not generic.

A number of weaknesses have since been identified with Weaver's pattern catalogue. First, the patterns take a fairly narrow view of assuring software safety, in that they focus on the mitigation of known failure modes in the design. Mitigation of failure modes is important, but there are other aspects of software assurance which should be given similar prominence. Second, issues such as safety requirement traceability and mitigation were considered at a single point in Weaver's patterns. This is not a good approach; as it does not reflect the building up of assurance relating to traceability and mitigation over the decomposition of the software design. Finally, Weaver's patterns have a rigid structure that leaves little scope for any alternative strategies that might be needed for novel technologies or design techniques [24].

A software safety pattern catalogue was also developed by Ye [25], specifically to consider arguments about the safety of systems including COTS software products. Ye's patterns provide some interesting developments to Weaver's, including patterns for arguing that the evidence is adequate for the assurance level of the claim it is supporting [24].

These approaches all focused on generating safety case templates for known patterns, whereas the safety case templates alone are the end result. Our approach encourages the encapsulation of the reusable development artifacts along with the supporting safety cases to enable true and speedy reuse in a safety-critical context.

The cognitive process of employing patterns is a fairly straightforward example of normal human pattern matching; the developer finds a problem or need, which can be expressed in a requirement, namely to eliminate the problem or satisfy the need, and then proceeds to firstly recall similar past problems and subsequently identify the most probable solution, based on his experience. Thus the entry point is a goal, which the developer believes they can satisfy by reusing previously successful solutions. We postulate the same adage holds for employing our patterns library elements. The interface to the system, whose problems the pattern solves, would be at the safety goal boundary.

To illustrate this we return to the example provided in section 3.1 and the corresponding safety case shown in figure (4). The arbitrator we choose to employ can be a COTS HW/SW unit we buy from a trusted supplier who also provides the corresponding detailed safety case for the arbitrator. That (sub-) safety case would then snap on to the larger safety case, shown in figure (4), at the safety sub-goal G1.2.2 it satisfies. The entire safety case in figure (4) would then itself snap into the parent (system) safety case at the G1 goal boundary defining its purpose, and this process would repeat itself hierarchically and cumulatively. As such, satisfying all the system's goals at the interface would enable not only modular but also compositional argumentation. This approach does, however, give rise to at least three as yet open questions:

1- Is the safety goal the most adequate boundary interface for modular compositional argumentation?
2- How do we guarantee that all goals (and sub-goals) have been identified?
   and more importantly,
3- How do we guarantee that the introduced pattern or component has no negative impact on the system?

The answers to these questions lie outside the scope of this paper and are the subject of future work.

4. Impact and use

The design of functional-safety systems is largely driven by best practices – like use of fault monitors, safe states, or redundant paths. These best practices can often be presented in form of patterns – both to describe a possible solution but also to document an argumentation about their contribution to a safety case. The provision of library of such patterns allows the identification and (re-)use of suitable architectural measures and of corresponding components along with their safety cases in a pattern-library based approach that allows compositional argumentation about the achieved safety goals. Thus, it contributes to the optimization of the development with respect to system safety in general, and specifically to safety-critical component reuse, but also COTS (Commercial Of-The Shelf) components. It further more provides support for the "Safety Element out of Context" clause cited in safety standards, such as the ISO26262 [6].

5. Conclusion and future work

This paper presents a concise overview of pattern-based methods for safety analysis and argumentation, a pattern-based approach for the encapsulated reuse and modular argumentation in a safety-critical context and an outlook at tool implementation. We furthermore explored the possibilities and show the opportunities supported by this approach. A holistic pattern-based approach to the construction of safety-cases in a seamless model-based development of safety-critical systems requires several elements, the main constituents of which we have identified. These are:

1- A library of reusable argumentation patterns – both in form of problem patterns (e.g., faults like early, late, or wrong sensor information; temporal or spatial interference between functions) and solution patterns (e.g., error avoidance, detection, mitigation; sensor fault detection and correction mechanisms; partitioning mechanisms) – built from elements of a model-based description of the system under development (e.g., requirements, functions, SW-/HW-components, channels, busses, deployment strategies) as well as GSN safety cases (e.g., goals, solutions, justifications, contexts)

2- A mechanism for the instantiation (e.g., stuck at-/noise-like faults; different filter for fault detection) and application (e.g., linking to the corresponding HW- and SW-elements) of those patterns in a compositional fashion.

3- A mechanism to check the soundness of the constructed argumentation (e.g., no open sub-goals; all context are covered by corresponding system elements) w.r.t. to its structure.

Using the example in Section 3.1, we have discussed the feasibility of (at least partially) handling all three of the above points.

In the next step we are focusing on expanding our pattern library with more implementation examples to test for general application and scalability as well as examining adequate methods for cross-referencing and categorizing patterns. The following stage will tackle the research questions pertaining to the compositionality of argumentation discussed briefly in section 3.3, particularly the identification of an appropriate boundary interface for integrating safety cases.

6. Acknowledgments

The AF3 project is a long term ongoing tools development effort led by the fortiss department of "Software and Systems Engineering". The functionality provided currently by the tool is mainly the result of the joint work of the following persons:

- Florian Hölzl (mailto:hoelzl@fortiss.org): project lead, tooling infrastructure for developing views and modular languages
- Sabine Teufl (mailto:teufl@fortiss.org): MIRA - model-based requirements engineering
- Mou Dongyue (mailto:mou@fortiss.org): MIRA - model-based requirements engineering (partly under the IMES project [33]), model based testing, refinement modelling
- Sebastian Voss (mailto:voss@fortiss.org): design space exploration, GSN-based modelling of safety cases, ASIL allocation, and optimized (saftey-oriented) deployment and schedule generation under the SPES XT [30], RECOMP [31] and ARAMiS [31]projects
- Daniel Ratiu (mailto:ratiu@fortiss.org): user friendly formal specification and verification

Many thanks also to the other members of the AF3 team.

Many thanks also go to the project partners from the SAFE/SAFE-E projects. The concept for the pattern-based approach presented in this work is based on the SAFE and SAFE-E projects. SAFE is in the framework of the ITEA2, EUREKA cluster program Σ!3674. The work has been funded by the German Ministry for Education and Research (BMBF) under the funding ID 01IS11019, and by the French Ministry of the Economy and Finance (DGCIS). SAFE-E is part of the Eurostars program, which is powered by EUREKA and the European Community (ID 01|S1101). The work has been funded by the German Ministry of Education and Research (BMBF) and the Austrian research association (FFG) under the funding ID E!6095. The responsibility for the content rests with the authors.

List of references

[1] European Organisation for Civil Aviation Equipment, EUROCAE ED-79/ ARP-4754: Certification considerations for highly integrated aircraft, 2011.

[2] European Organization for Civil Aviation Equipment (EUROCAE) and United States Department of Defense (DOD), EUROCAE ED-12B/DO-178B: Software considerations in airborne systems and equipment certification, 1992.

[3] United States Department of Defense (DOD), DO-178C: Software considerations in airborne systems and equipment certification, 2012.

[4] European Committee for ElectroTechnical Standardization, EN-50126 Railway Standard, www.cenelec.eu, 2008.

[5] International Electrotechnical Commission, IEC 61508 Standard, "Functional Safety of Electrical/Electronic/Programmable Electronic Safety-related Systems", www.iec.ch/functionalsafety, 2009.

[6] International Standards Organization, ISO 26262 Standard, Road Vehicles Functional Safety, www.iso.org, 2011.

[7] K.K. Breitman, J.C.S.P. Leite, and A. Finkelstein, The World's Stage: A Survey on Requirements Engineering Using a Real-Life Case Study, J. of the Brazilian Computer Soc., vol. 6, no. 1, pp. 13-38, July 1999.

[8] L.M. Cysneiros and J.C.S.P. Leite, Integrating Non-Functional Requirements into Data Model, Proc. Fourth Int'l Symp. Requirements Eng., June 1999.

[9] D.R. Lindstrom, Five Ways to Destroy a Development Project, IEEE Software, pp. 55-58, Sept. 1993.

[10] „Funktionale Sicherheit: Grundzüge sicherheitstechnischer Systeme", Börcsök, J. / VDE-Verlag, 2011

[11] Struss, P. : A Conceptualization and General Architecture of Intelligent Decision Support Systems. MODSIM2011, 19th International Congress on Modelling and Simulation. Modelling and Simulation Society of Australia and New Zealand, December 2011, pp. 2282-2288. ISBN: 978-0-9872143-1-7.

[12] Chris Price and Neal Snooke, An Automated Software FMEA, Proceedings of the International System Safety Regional Conference, Singapore, April 2008

[13] Struss, P., Fraracci, A.: Automated Model-based FMEA of a Braking System. In: 8th IFAC Symposium on Fault Detection, Supervision and Safety of Technical Processes (Safeprocess 2012), Mexico City, 2012

[14] Kelly, Tim P., and John A. McDermid. "Safety case construction and reuse using patterns." 16th International Conference on Computer Safety, Reliability and Security (SAFECOMP 1997). 1997.

[15] Stefan Wagner, Bernhard Schätz, Stefan Puchner, Peter Kock, A Case Study on Safety Cases in the Automotive Domain: Modules, Patterns, and Models, Proc. International Symposium on Software Reliability Engineering (ISSRE '10), IEEE Computer Society, 2010

[16] Tim Kelly, Ibrahim Habli, et al.: Goal Structuring Notation (GSN). GSN COMMUNITY STANDARD VERSION 1, Origin Consulting (York) Limited, on behalf of the Contributors, November 2011

[17] The ITEA2 SAFE Project / The EUROSTARS SAFE-E Project; www.safe-project.eu

[18] The SAFE / SAFE-E Consortium. Deliverable D3.1.3 / D3.4 "Proposal for extension of Meta-model for safety-case modeling and documentation". www.safe-project.eu ITEA2 / EUROSTARS. 2013.

[19] AutoFOCUS 3, research CASE tool, af3.fortiss.org, 2013 fortiss GmbH

[20] M. Khalil "Pattern-based methods for model-based safety-critical software architecture design" ZeMoSS 2013 Workshop at the SE 2013 in Aachen. 2013.

[21] S.Voss, B. Schätz, M. Khalil, C. Carlan "A step towards Modular Certification using integrated model-based Safety Cases" VeriSure 2013.

[22] T. Kelly and R. Weaver: "The Goal Structuring Notation . A Safety Argument Notation". Proc. DSN 2004 Workshop on Assurance Cases, 2004

[23] R. Weaver, "The Safety of Software – Constructing and Assuring Arguments", PhD Thesis, Department of Computer Science, The University of York, 2003.

[24] R. Hawkins, K. Clegg, R. Alexander, T. Kelly, Using a Software Safety Argument Pattern Catalogue: Two Case Studies, in F. Flammini, S. Bologna, and V. Vittorini (Eds.): SAFECOMP 2011, LNCS 6894, pp. 185–198. Springer, Heidelberg, 2011.

[25] F. Ye, "Justifying the Use of COTS Components within Safety Critical Applications", PhD Thesis, Department of Computer Science, The University of York, 2005.

[26] Weihang Wu and Tim Kelly. 2004. "Safety Tactics for Software Architecture Design". In Proceedings of the 28th Annual International Computer Software and Applications Conference - Volume 01 (COMPSAC '04), Vol. 1. IEEE Computer Society, Washington, DC, USA, 368-375.

[27] A. Armoush, "Design Patterns for Safety-Critical Embedded Systems", Ph.D. Thesis, RWTH-Aachen, 2010

[28] S. Voss, B. Schätz, "Deployment and Scheduling Synthesis for Mixed-Critical Shared-Memory Applications". Proceedings of the 20th Annual IEEE International Conference and Workshops on the Engineering of Computer Based Systems (ECBS) 2013.

[29] T. Kelly, "Arguing Safety – A Systematic Approach to Managing Safety Cases", PhD Thesis, Department of Computer Science, The University of York, 1998.

[30] SPES-XT Project Consortium. spes2020.informatik.tu-muenchen.de/spes_xt-home. 2013.

[31] Automotive, Railway and Avionics Multicore Systems – ARAMiS Project Consortium.. www.projekt-aramis.de. 2013.

[32] Reduced Certification Costs Using Trusted Multi-core Platforms – The RECOMP Project consortium. An ARTEMIS Joint Undertaking Project. http://atcproyectos.ugr.es/recomp/

[33] IMES - Integrierte modellbasierte Entwicklung softwareintensiver Systeme. IMES ist ein KMU Innovation BMBF Projekt. http://www.fortiss.org/forschung/projekte/imes/