Design Patterns to the rescue: guided model-based reuse for automotive solutions

MAGED KHALIL, Systems & Technology, Chassis & Safety Division, Continental Teves AG & Co. oHG

The reuse of proven solutions (e.g., Safety Mechanisms or architecture designs) for safety-critical applications is considered a good practice for increasing confidence in the system and cutting development cost and time, and is widely-spread in practice. However, reuse in safety-critical applications is mostly ad-hoc, with lack of process maturity or adequate tool support. Moreover, it is difficult to assess the quality or completeness of a reuse process, if there is no "definition of done". In previously published works, we defined a structured "Pattern Library" approach for the reuse of Safety Mechanisms (fault avoidance / error detection and handling) in the automotive domain, elaborating a prototypical tool implementation for both Pattern *Developer* and *User* role. This paper consolidates the previously provided definitions and provides evidence of the effectiveness of the approach via the evaluation of instantiations into a generic research CASE tool (AutoFOCUS3), as well as a domain-adequate automotive safety modeling framework (SAFE Framework). We demonstrate the improvements in the maturity, adequacy and completeness of the reuse, as well as how the approach can be used to guide tool selection, development and/or extension. Finally, we showcase how the approach can generally be applied to other system design reuse problems, via an instantiation into a large scale automotive functional & technical component library at a leading automotive supplier. Given the rise in automotive application complexity and ever shorter development cycles, solution reuse is of paramount importance.

Categories and Subject Descriptors: **General and reference~Reliability** • **Computer systems organization~Dependable and fault-tolerant systems and networks** • Computer systems organization~Embedded systems • **Software and its engineering~Abstraction, modeling and modularity** • Software and its engineering~Designing software • Applied computing~Computer-aided design

General Terms: Reuse

Additional Key Words and Phrases: safety, safety argumentation

ACM Reference Format:

Khalil, M. 2018. Design Patterns to the rescue: guided model-based reuse for automotive solutions. HILLSIDE Proc. of Conf. on Pattern Lang. of Prog. 25 (October 2018), 21 pages.

1. INTRODUCTION

1.1 Background

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-cases aiming to document and prove the fulfillment of the underlying safety goals. A Safety Mechanism is defined by the ISO26262 International Automotive Safety Standard [1] as a "technical solution implemented by E/E functions or elements, or by other technologies (mechanical etc.), to detect faults or control failures in order to achieve or maintain a safe state". Safety Mechanisms are concrete instances of a category of implementation-independent solution descriptions targeting safety. The solutions used in a "Safety Mechanism" are in fact not limited to safety but may also be useful for all manner of dependability or RAMSS – Reliability, Availability, Maintainability, Safety and Security – issues. Wu und Kelly described a very similar grouping of architectural designs or "Tactics" in [3] and proceeded to describe a design pattern and template for capturing them. Tactics capture the abstract principles or primitives of a design pattern.

The primary aim of the approach presented here is make Safety Mechanism reuse within an organization better (more systematic, repeatable, effective) by approaching Safety Mechanisms as if they were Design Patterns. We understand that Safety Mechanisms are strictly not design patterns, and are not aiming at capturing a new kind of Design Pattern. We wish to use the structure provided by the design pattern template and general rigor of defining a design pattern to improve solution component capturing for reuse in a practical setting.

Wu and Kelly used a graphical notation of safety cases to capture the rationale attribute of their tactics template. We will be using the same notation in our example. 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, proving that a system is acceptably safe to operate in a particular context [2]. Safety cases can be expressed graphically, as seen on the right-hand side of figure 2.
The use of patterns in safety-critical software development in conjunction with model-based development techniques is documented and well suited for these needs, for instance [5]. Patterns of safety cases for well-known problems have been suggested in academic literature as well, e.g., for using COTS (Commercial-Off-The-Shelf) Components [34]. The reuse of safety-cases – and Safety Mechanisms in general – is mostly ad-hoc, with the practitioners focusing on the central artifact – a piece of code, an algorithm or a design model – and forgetting that this does not tell the entire story needed for proper reuse. Loss of critical knowledge and traceability, lack of consistency and/or process maturity and inappropriate artifact reuse being the most widely spread and cited drawbacks [4].

1.2 Motivation

Yet be it an architecture description or the software or hardware development artifact, the single item does not tell the entire story. For example, to correctly deploy homogenous redundancy [29], shown in the next figure, many other aspects must be covered:

- one must define the requirements the pattern fulfills,
- refine the requirements and draw up a specification,
- detail a (logical) component architecture,
- optimize a deployment strategy that guarantees the duplicate components will not run on the same hardware resource,
- and finally, show how the usage of this Safety Mechanism contributes to System Safety, i.e., the safety case.

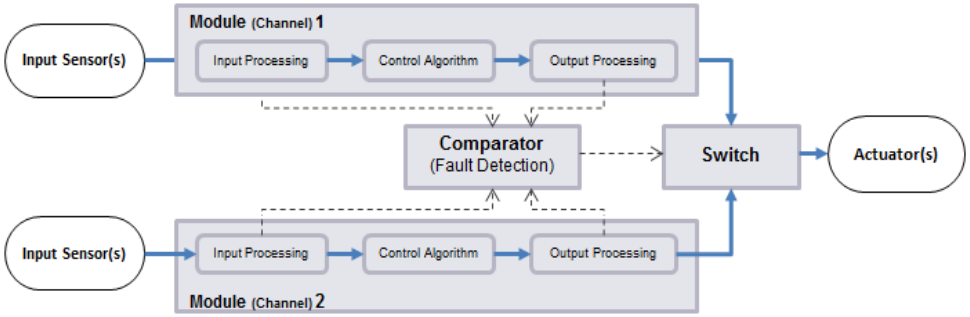

Fig. 1. Homogenous duplex (hardware) redundancy pattern [12]

Doing this in a structured, repeatable and assessable manner is a further problem. To begin with, it is difficult to assess the quality or completeness of a reuse process, if there is no "definition of done". Providing this structure was our first step. But merely defining yet another design pattern attribute template is not the answer to the challenges surrounding the reuse of Safety Mechanisms in practice. The new definition needs to be a part of a more holistic approach, leveraging model-based capabilities to provide tool-supported reuse automation in a Systems Engineering context, e.g., as provided by the Systems Engineering Lifecycle Processes standard ISO15288 [30]. While Safety Mechanisms are not limited to software only, the development of complex Systems has become a software-intensive endeavor in and of itself. Thus to achieve our goals, our approach leverages a combination of software engineering paradigms; chief among them Model-Driven Engineering (MDE) [37] – treating models as first class citizens, component-based software engineering (CBSE) [38], and reuse repositories as presented in [39] & [40].
The next aspect we focused on was the usability and adequacy of the modeling approach for the task at hand. The author has considerable hands-on industrial experience, and has observed first-hand, as well as received plenty of anecdotal evidence to the pervasiveness of the golden hammer syndrome, in industry as well as academia. Practitioners who are long accustomed to a tool or programming language will often see problems and solutions through the prism of their tool of choice, in this case blurring the lines between what is generally describable, and what is describable in their tool/language of choice. There is a noted tendency to bend the language or tool and contort it to address the problem at hand, and view the resultant solution in satisfactory light. Case in point; one only has to observe the sheer number of UML profiles in literature, used to cover every possible activity and purpose, regardless of whether UML is the right tool for the job. In essence, the limits of the

tool's expressiveness become the limits of knowledge documented – and eventually *perceived* – by the user, creating a vicious circle.

We aim to break that circle, in a three-pronged approach.

(1) First and foremost, we do not wish to limit ourselves to, nor do we indeed have any preference for any particular modeling language, framework, or development environment. We do, however, encourage the use of domain-adequate context-rich models, as these will – by definition, and as our results will demonstrate – allow the capturing of more information. Our approach's basis is the capturing of solutions in the exact artifacts which will be (re)used.

(2) Secondly, we encourage practitioners to ask not which reuse aspects they can cover using their tools, but rather which aspects need to be captured by their tools. We want practitioners to look beyond the limitations of their tools and onto the actual scope of the problem at hand. This is the immediate localized impact of the approach.

(3) Finally, we wish to embolden and empower practitioners to cast a skeptical eye on their development tools; asking themselves whether they are indeed covering all the necessary information for their desired product quality, and hence whether they have the right tools. This is the generalized long-term impact of the approach.

These 3 points form the basis of the approach presented in this paper. We focus on capturing all the necessary information we *have to* capture for the reuse, as defined by the pattern template; instead of focusing on the aspects we *can* capture using one tool-chain or another.

1.3 Previous Work

The reuse of Safety Mechanisms can be made both simpler and more robust through the encapsulation of all information into a consistent structured package, which can then be stored as a library element, along with the corresponding safety case to support it, as we demonstrated in previously published works [6] & [11], in which we defined a structured approach for the reuse of Safety Mechanisms (fault avoidance / error detection and handling) in the automotive domain, capturing them in a "pattern library", based on Design Pattern literature. This library uses a simplified description of Safety Mechanisms, covering established solution algorithms and architectural measures/constraints in a seamless model-based approach with corresponding tool support.

The classical approach to capturing Design Pattern - as found in literature - focuses on capturing useful solutions in a generic fashion (via structured prose and diagrams), which is understandable across organization or even domain boundaries. In contrast, our approach focuses on the in situ capturing of the information necessary for the reuse within the artifacts actually used during development used within an organization. The approach provides guidance for the reuse in practice, based on a meta-model covering both development artifacts and safety case elements, which can be instantiated into any existing development environment. At the foundation of the approach, an attribute template was defined, identifying the aspects of a Safety Mechanisms that have to be captured for the reuse to be successful. Initial descriptions of the structure and usage of this reusable library element concept in a seamless model-based development tool are the focus of a previous publication [12]. An extended description of the usage workflow and Pattern "Developer" Role is given in [33].

1.4 Contribution

In this work, we expand on previous definitions of the Safety Mechanism pattern attribute template and corresponding Pattern Library approach, including both a Pattern Developer and a Pattern User role. We will give detailed descriptions of how a can employ the approach in practice to capture Safety Mechanisms. We demonstrate the benefits of our Pattern Library approach for the systematic reuse of technical solutions – as previously developed for Safety Mechanisms – by first instantiating it in a domain-independent research CASE tool (AutoFOCUS3 – AF3) and then in a domain-adequate automotive safety modeling framework (SAFE Modeling Framework). We use the instantiation in AF3 to contrast the perceived completeness and adequacy of the reuse before and after applying our approach, as well as to demonstrate how the approach can be useful in guiding practitioners when extending their existing tools. We use the instantiation in the SAFE Framework to prove the technical feasibility of instantiating the method into more than one development environment and its independence from specific modeling frameworks, as well as to show how the domain adequacy of the underlying framework is positively linked to the adequacy and completeness of the reuse. This point highlights the importance of selecting the right tool and how the approach can be used as a guide in tool selection. Finally, we redefined the attribute catalogue of the pattern library and instantiated the approach into a large scale automotive function & technical component library at a leading Tier 1 automotive supplier (Continental AG).

This instantiation gives first evidence for the general applicability of the approach to other problem types. Deeply rooted in the user-centric nature of Design Patterns, our contribution can be summarized us:

1. With a Design Pattern Library at its heart; a systematic, repeatable and assessable approach – for reusing technical solutions (in the automotive domain), which can be instantiated into any development environment.
2. A demonstration of the feasibility of the approach.
3. Demonstrations of the usefulness of the approach not only for direct reuse, but also as gauge of the adequacy of the employed development environment / tool suite for the reuse.
4. Demonstrations of the usefulness of the approach as a guide for tool extensions / selection.

The structure of this paper is as follows. Section 2 details our approach and the definitions for the pattern template for Safety Mechanisms, and explains the usage of the approach. Section 3 describes the various instantiation case studies and their results. We discuss related work in section 4 and provide observations and lessons learned from applying our approach in section 5, before concluding this paper in section 6.

2. DESIGN PATTERN BASED APPROACH

The reuse of safety-critical automotive solutions is wide-spread, but is marred, however, by several problems [12]:

- 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 safety analyses (STAMP, FMEA, FTA, etc.) have to be performed at system level, yet the components/ measures /Safety Mechanisms themselves need to be reused locally/independently,
- and are not tied in any structured manner to other elements needed to provide the relevant context.

2.1 Safety Mechanism Pattern Library

Using a simplified description of Safety Mechanisms s according to the most common subtypes (avoidance/ detection/ handling) we 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 library comprises the minimum set of elements needed for correct reuse, i.e. the requirement the pattern covers, the specification of how one plans to implement it and the architecture elements/ measures /constraints required as well as the supporting safety case template, based on the established structure notation known as GSN [17], and may include deployment or scheduling strategies, which would then be integrated into existing development environments. This enables an early analysis of hazards and risks, as well as the frontloading of many design aspects, which is recommended by most 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, and other relevant information, 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 as a user-centric aide to practitioners: 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.

To structure the capturing of the Safety Mechanisms, we defined a template, shown in the next table. The selection of attributes is based on an extensive literature survey of Design Pattern templates covering several dozen templates, but is most influenced by the works of the Gang of Four [30], Kelly [4], Douglas [20] and Armoush [29], as well as the POSA Architecture Template by Buschmann et al [54], which is itself based on the works of Riehle and Züllighoven [55].

Table 1 Safety Mechanism Template [12]

Attribute	**Abbr.**	**Meaning**
Name	n	Name of the Safety Mechanism

Intent	i	Immediate purpose of the Safety Mechanism
Motivation	m	Rationale behind using the Safety Mechanism
Applicability	a	Situation and conditions to which the Safety Mechanism can be applied, as well as counterexamples
Structure	s	Representation of the elements of a pattern and their relationship (preferably in graphical form)
Participants	p	Complementary to the *Structure* attribute; provides a description of each of the Safety Mechanism pattern elements, including potential instantiation information.
Collaborations	cl	Shows how the pattern's various elements (e.g., sources of contextual information, argument structure, requirements, logical components) collaborate to achieve the stated goal of the pattern. Moreover, this attribute focuses on capturing any links between the pattern elements that could not be captured clearly and explicitly or were not the focus of the Structure attribute, e.g., dynamic behavior.
Consequences	cn	Captures information pertaining to the instantiation and deployment of the pattern, as well as to the impact it may have on the system.
Implementation	im	Describes the implementation and should include its possible failure modes or dysfunctional behavior as well as any constraints or possible pitfalls
Implications	ic	This characteristic covers the impact of applying the pattern on the non-functional aspects of the system. This may include traits such as reliability, modifiability, cost, and execution time.
Usage Classification	uc	Categorizes the Safety Mechanisms according to their usage (fault avoidance / error detection and handling)
Example	e	Exemplary (preferably graphical) representations of the Safety Mechanisms are especially useful if the previous attributes were not captured graphically, or are very complex. Otherwise optional.
Related Patterns	rp	Provides a listing of Safety Mechanism related to the pattern being documented. This may, for instance, include patterns that are derived from this pattern.

The „Known uses" attribute was purposefully not included in the template, as we are not aiming at capturing a general Design Pattern description, but rather targeting the reuse of a safety mechanism within an organization. That is, a safety engineer develops a new mechanism, and wants to capture it and its documentation, tests, etc. using the exact artifacts used within their company, instead of "structured text and diagrams". Practitioners in a company may wish to list applications or products where a certain pattern is in use and could choose to include this attribute. We however would encourage the inclusion of this information into the mechanisms of the pattern library repository, as a textual list of uses or applications does not provide a persistent link to the applications in which the patterns are deployed, which may need to be updated if the pattern is redesigned or replaced.
The template is the first step towards a systematic capturing of Safety Mechanisms for reuse. We identified the types of development artifacts most likely to contain the information required for each attribute and then proceeded to provide a guide for instantiating the approach into various development environments. The approach provides guidance for the reuse in practice, based on a meta-model covering both development artifacts and safety case elements, which can be instantiated into any existing development environment.
At its essence, the approach for a capturing a solution can be divided into 5 simple steps for the practitioner:

1. Identify the information necessary for a systematic reuse of the solution in practice, and structure this information into an attribute template. E.g., name, applicability, implications ...
2. Identify the types of development artifacts that should/could contain the necessary information. E.g., textual requirements, state charts, graphical safety cases, code…
3. Define a data-model (linking all the development artifacts, capturing the information necessary for reuse – as defined in the attribute template), and use it to set up a "library" mechanism capturing the reusable solution and all its related artifacts.
4. Evaluate whether and to what degree the development environment / tool suite used by the practitioner can actually capture the necessary information.
5. Extend the development environment / tool suite as necessary, using the gaps identified in the evaluation step to both guide and prioritize the tool extension.

More details about the approach, the attribute template, the structure and usage (both Safety Mechanism Developer and User roles) of this reusable library element concept's instantiation into a seamless model-based development tool are provided in [12], and summarized in section 2.4.

Figure 2 gives a sketch of a generic library element comprising development artifact categories. The left part of the schematic gives possible development artifact elements– requirements, specification, and implementation description – of the reusable pattern, while the right part shows the corresponding argumentation artifacts comprising a safety case skeleton – shown exploded for clarity – with connections to the relevant development artifact categories. The argumentation artifacts are displayed exemplarily in a graphical safety argumentation

notation, in this case the Goal Structuring Notation (GSN) [17] introduced by Kelly and McDermid [4]. Both sides together comprise the pattern library element, the building blocks of which are the four artifact categories requirements, specification, implementation and argumentation.
While the left part of the schematic, comprising the various development artifacts necessary for reuse, can entirely originate in one seamless tool – as will be shown in our first use case in section 4 – this is not necessary. As long as the artifacts' relations are well-defined, the development artifacts may reside in multiple tools or 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, supported in-tool by the pattern library mechanism.

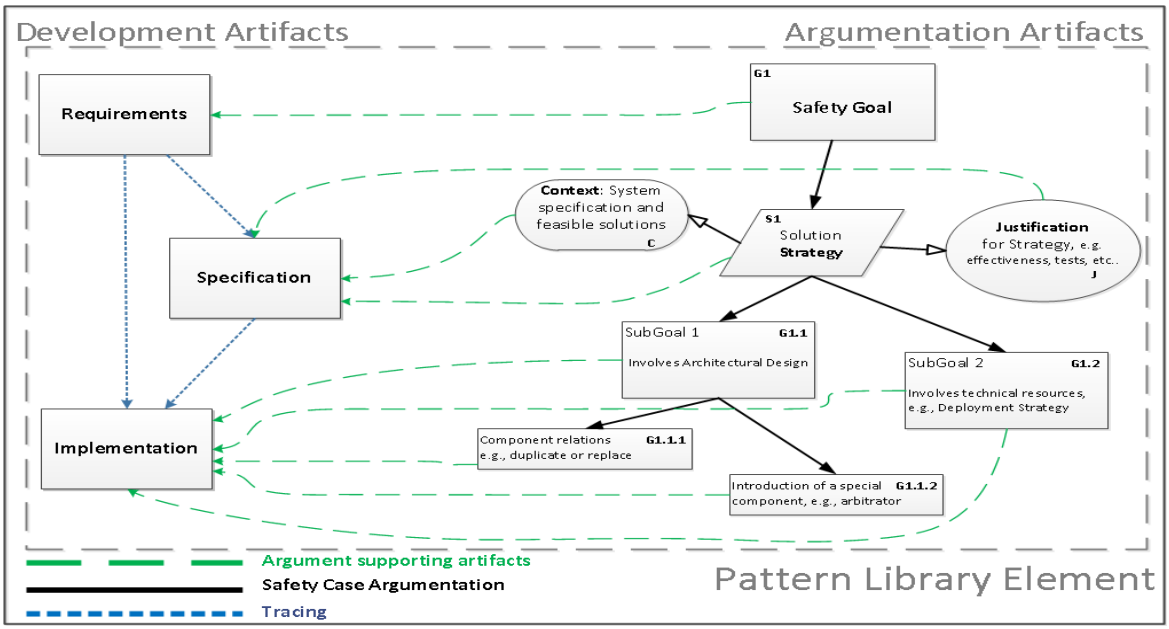

Fig. 2. Generic Safety Mechanism Pattern Library Element [12]

This specific aspect is particularly important for the practical reuse in a safety-critical development context, where each practitioner may have their own disparate tool landscape. This forms a cornerstone of our approach. Depending on the development environment used, the relation between artifact categories will differ according to the definitions provided for how the different category views (if available) interact. The pattern library mechanism can then be in general terms interpreted as a weaving model, which binds the pattern elements together and allows their reuse. Additional mechanisms for the instantiation, also depending on the development environment, may be necessary.
In order to investigate the instantiation of the approach, it is necessary to first provide some basic definitions of the pattern library and its elements, and explore the attributes further.

2.2 Instantiation example of the Pattern Library Approach

The example presented here, including the practical applicability, along with an initial usage description – for the pattern *User* role – and description of the implementation (with screenshots), were previously presented in [12] and are repeated here to help explain the approach presented in this paper. Detailed definitions of the data-model elements and relations shown in Fig. 3, were also given. We also provide usage descriptions for the pattern Developer role, which were published in [33].

Introduction to AutoFOCUS3

AutoFOCUS3 (AF3) is a research computer-aided software engineering (CASE) tool, which allows modeling and validating concurrent, reactive, distributed, timed systems on the basis of formal semantics [7]. Most importantly, AF3 provides a graphical user interface that facilitates the modeling of embedded systems in different layers of abstraction while supporting different views on the system model, including the requirements-, component- and platform- view essential for the description of complex systems. This support for views enables applying paradigms such as "separation-of-concerns" – different views for different stake-

holders/concerns – as well as "divide-and-conquer" – different hierarchy layers allowing the decomposition of engineering problems to facilitate solution, as championed by the IEEE 42010 standard [8].

Introducing the Pattern Library Approach to AF3

An early attempt at reusing architectural patterns in AF3 was presented in [9]. In it, the authors demonstrated how Homogenous Duplex Redundancy (HDR), and Triple Modular Redundancy (TMR) could be integrated as patterns into AF3. Before this work, AF3 had no pattern capability. The approach was successful in demonstrating that design patterns for embedded-system Safety Mechanisms can be supported in AF3 and that they are useful. Practitioners who experimented with the feature gave positive feedback and voiced their interest in having the functionality expanded.

As detailed in more detail in section 3.2, the Safety Mechanism capturing capabilities of Af3 were expanded using our approach. Using the individual evaluation scores for each attribute to identify gaps in the tool as well as to guide and prioritize the extension of AF3, the Safety Mechanism capturing capability was expanded by integrating the Pattern Library approach – presented in the previous section – into AF3. Figure 3 shows the data model of the pattern library approach after instantiation in AF3, which will notably look different if the approach is instantiated in another tool or modeling framework.

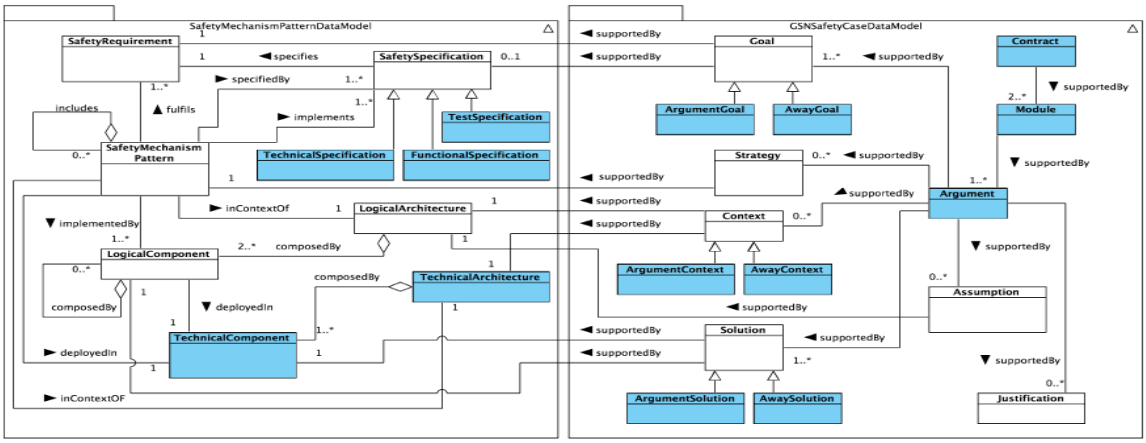

Fig. 3. AF3 Extended Pattern Library Element Data Model [32]

The instantiation leverages preexisting capabilities in AF3 (such as logical architecture description, model-based requirements engineering, model checking, optimized deployments, safety case expression, among many others) to generate the artifacts for the safety mechanism libraries. The implementations demonstrated in [12] covered four patterns: 1oo2Voting, Comparison, RangeCheck, and HomogeneousDuplexRedundancy, and were later extended to more than a dozen patterns [33]. Continuing with the redundancy example introduced in Chapter 1 and shown in figure 1, the following section illustrates the implementation using screenshots.

Safety Mechanism Library Implementation in AF3

The interaction of the safety mechanism with its environment is captured in the LogicalArchitecture element shown in figure 4.

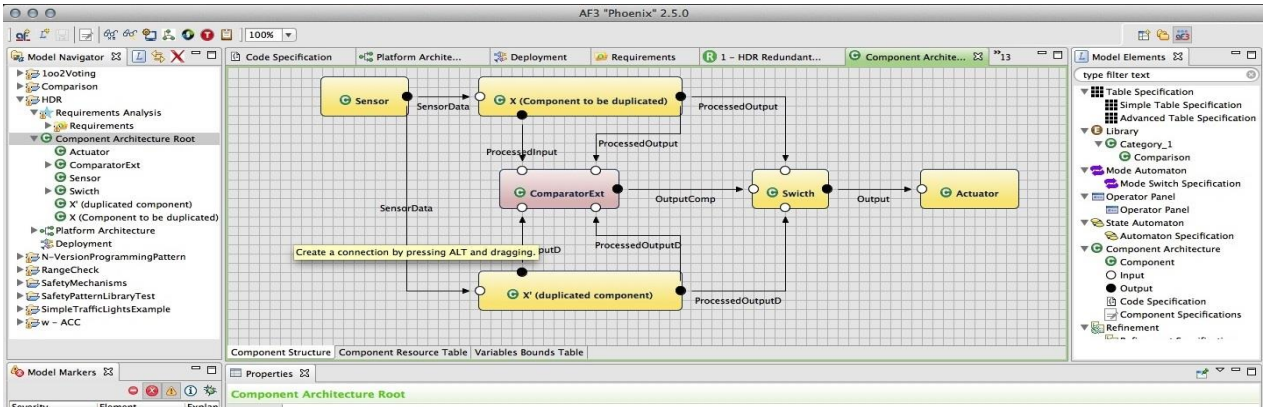

Fig. 4. Implementation Example HDR: LogicalArchitecture [12]

Figure 5 shows the LogicalComponent element of the HomogeneousDuplexRedundancy (HDR) implementation example, which is itself comprised of other logical components, with the corresponding safety case shown collapsed in figure 6.

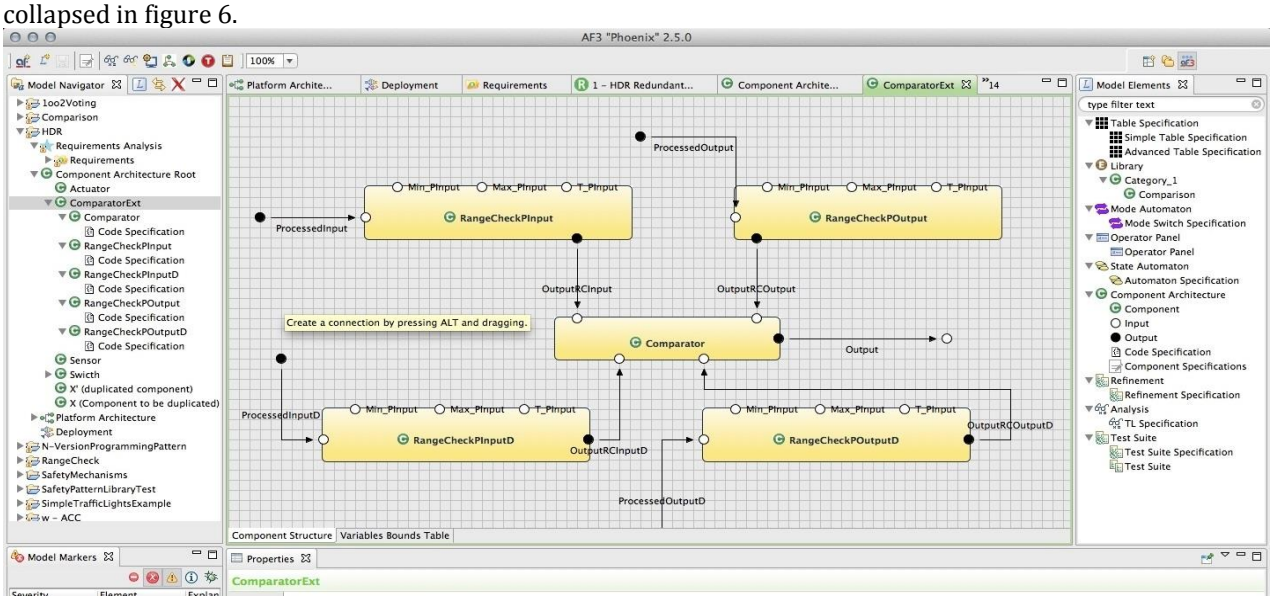

Fig. 5. Implementation Example HDR: LogicalComponent [12]

Using the relations described in the GSN standard [17] and implemented in AF3, 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".

Not shown, due to space limitation but implemented with AF3, are the corresponding requirements or specification of any of the central LogicalComponent elements, e.g., the Comparator. Also possible but not shown, is the formal specification of mechanism behavior using automatons or tables, which then allows for automatic test suite generation as described in [46].

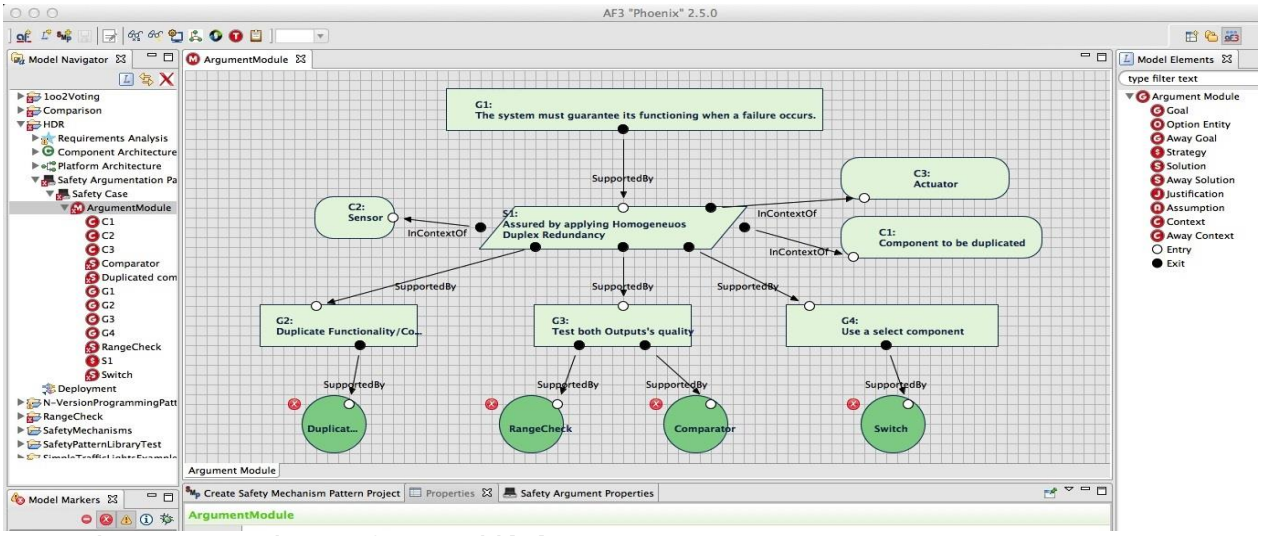

Fig. 6. Implementation Example HDR: SafetyCaseModel [12]

2.3 Capturing Safety Mechanisms in AF3 – the Pattern *Developer* Role

This section aides in understanding the usage of the Pattern Library approach, and reiterates the pattern developer role presented in [33]. A comprehensive analysis of all the Safety Mechanisms found in our literature survey (approx. 40), led us to identify three archetypes of structures for capturing and instantiating Safety Mechanisms in AF3, detailed in [12]. To save a new Safety Mechanism, the *Developer* has to decide which of these

archetype best fits their need. The *Developer* is aided in this task by a wizard, shown in Fig. 5, which also assures that library elements cannot be created without the mandatory information. But first, Fig. 4 shows the steps a developer has to undertake to document a new Safety Mechanism Pattern in AF3, some of which are optional. The steps are explained in more detail in [33].

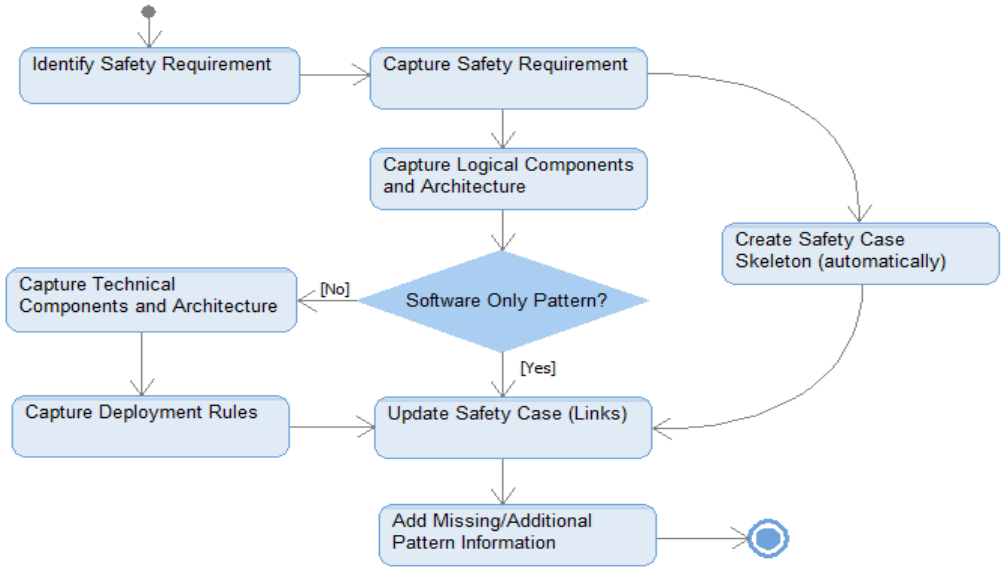

Fig. 7. Workflow steps for Safety Mechanism Pattern *Developer* Role in AF3 [33]

AF3 allows a precise definition of deployment rules to govern the allocation of logical components to computational resources (nodes). Furthermore, it is possible to use the Design Space Exploration capability of AF3 to generate deployments and schedules optimized to fulfill multiple criteria, such as worst case execution time, memory constraints, bus loads or power consumption, but also safety constraints, such as ASIL-decomposition and random hardware failure probabilities, as seen in [35].

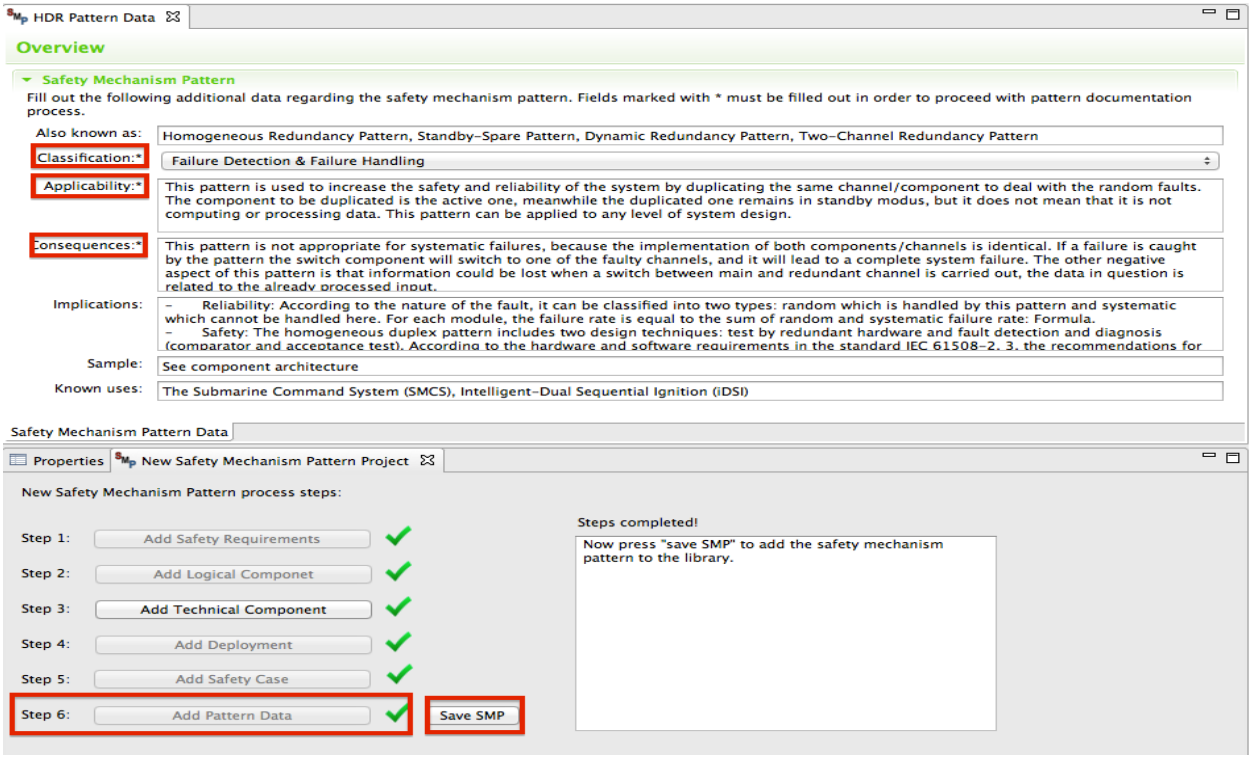

Fig. 8. AF3 Safety Mechanism Pattern *Developer* Wizard Interface [32]

Throughout this process, the *Developer* is guided by a wizard, whose interface can be seen in Fig. 8, which makes sure that all mandatory information, necessary for the correct documentation of the Safety Mechanism Pattern attributes, is provided into the library element. The new Safety Mechanism Pattern can now be saved as an element of the pattern library, where it is accessible to other AF3 pattern *Users*.
As will be seen in the case study evaluations in section 3, although the approach brought significant improvements to AF3's capability of capturing Safety Mechanism Patterns for reuse, some of the required pattern attributes are still nevertheless not satisfactorily captured in AF3's artifacts – especially *implications* and *consequences*. We appended the library mechanism with the capability to add some short textual information to enhance the documentation qualities of these attributes. This is meant as a stop-gap until AF3 capabilities improve enough to allow capturing the information solely in modeling artifacts. These shortcomings factored into our evaluations.

3. CASE STUDIES

3.1 Evaluation scale of case studies

The evaluation of the case study development environments solely along objective criteria is not possible in our context, as the captured information content for each attribute differs depending on the tools used, how the tools are integrated into the development environment, and how the users perceive the tool.
We will thus be evaluating the instantiation of the approach into the case study development environments from a mostly subjective viewpoint. We do not believe this will negatively affect our results, as the purpose of the evaluation is to study the instantiation of the approach in different development environments and its impact on them, and not to actually evaluate the tools themselves.
We will use a scale of 0 to 1 in quarter-point steps to evaluate if, and how well, an attribute is captured in the development environment, and will apply this scale to our evaluations of the exemplary development environments in the next sections, according to the differentiation provided in the following table.

Table 2. Evaluation Scale for Case-Studies

Score	Description
0.00	The attribute cannot be captured in the development environment.
0.25	The development environment has no contingency to capture the attribute, but it can be partially covered by abusing the tool, or is inadvertently captured while documenting another attribute, e.g., capturing motivation, rationale and applicability in a single requirements engineering tool entry.
0.50	The development environment expressly captures the attribute, but using an inadequate mechanism, e.g., a text entry for describing complex structure or component relations. Or the attribute is not captured, despite the development environment having the capability to capture it.
0.75	The development environment has a dedicated mechanism for capturing the attribute, but it misses some information, or is not well integrated with other attribute descriptions, or has other problems affecting the quality of the attribute documentation.
1.00	The development environment captures the attribute well. All necessary information is documented.

3.2 Case Study I: AutoFOCUS3

AF3 and the instantiation of the Pattern Library approach into it were explained in some detail in section 2.2 of this paper.

Instantiation of the Pattern Library approach into AUTOFOCUS3
An early attempt at reusing architectural patterns in AF3, was presented in [9]. In it, the authors demonstrated how Homogenous Duplex Redundancy (HDR) – discussed in section I and shown in Fig. 1, and Triple Modular Redundancy (TMR) could be integrated as patterns into AF3. The approach works by allowing the user to right-click anywhere in the Logical Architecture view of an open pre-existing AF3 model, and select one of the two pattern to use. This then copied empty logical components, essentially a skeleton, conforming to the pattern into the open AF3 model. The user could then copy his actual components into the skeleton provided by the feature.
Before this work, AF3 had no pattern capability. The approach was successful in demonstrating that design patterns for embedded-system Safety Mechanisms can be supported in AF3 and that they are useful. Practitioners who experimented with the feature gave positive feedback and voiced their interest in having the functionality expanded. The quality of the existing Safety Mechanism capturing in AF3 was evaluated, with results shown in the next table.

Table 3. AF3 Pre-Evaluation

Attribute	Evaluation Remarks	Score
Name	Captured in the feature.	1.00
Rationale	Could be captured by AF3, but was not part of the feature.	0.50
Motivation	Could be captured by AF3 along with other requirements, but was not.	0.25
Applicability	Partially captured in the reusable Logical Architecture but needed further details.	0.50
Structure	Captured for the most part in the reusable Logical Architecture.	0.75
Participants	Implicitly captured for the most part in the structure of the reusable Logical Architecture, but would have benefitted from increased elucidation.	0.50
Collaborations	Implicitly & partially captured in the structure of the reusable Logical Architecture, but would have benefitted greatly from argumentation descriptions.	0.25
Consequences	The usage was limited to copying Logical Architectures with predefined instantiation rules, while all other attributes were missing.	0.25
Implementation	The components of the Logical Architecture captured most of the implementation information, except the deployment rules.	0.75
Implications	This part went completely missing, some of the impact on execution time and resources were nevertheless inadvertently captured by AF3's semantics.	0.25
Usage Classification	This part is not handled in AF3	0.00
Example	The graphical interface of AF3 makes the captured Logical Architecture fairly easy to understand, so we can consider it to be an example. Some additional description would have facilitated the understandability of the pattern even more.	0.50
Related Patterns	Not handled in AF3.	0.00
Total Tally (out of 13)		**5.50**
Percentage		**42.3%**

The reuse was mainly limited by the capabilities of AF3 itself at the time and hence that is the primary source of the major drawbacks:

- The approach is limited to copying logical component architectural snippets into existing models, which then have to be filled by the user with their own functional components.
- Updates to the existing patterns could only be carried out (at code level) by the developer.
- Normal users of AF3 could not expand the feature with other patterns.
- The approach was limited to logical components. All other artifacts necessary to support the reuse of the design pattern and capture the necessary attributes were not included in the reusable feature.

AF3 had no support for safety case argumentation. More importantly, it also had no support for a reusable pattern library. The patterns featured here had to be manually hard-coded into the tool, rendering this approach unusable to a normal tool user.
The lack of ability to integrate other artifacts generated in AF3 to support the correct reuse of design patterns in a safety-critical context was also a particularly relevant drawback. Indeed the patterns implemented exemplarily here would have especially benefitted from this support, because they are both variations of the N-Homogenous Redundancy design pattern. Limiting their consideration to logical components greatly reduces the usefulness of the captured pattern, as such descriptions are not helpful, unless one can maintain a link to a deployment strategy that ensures the redundant components are deployed on different hardware resources.

Results of AF3 Instantiation
Using the results of the evaluation to guide the extension of AF3, the tool suite was expanded in two steps. The first, presented in [10], added safety case expression capability to AF3. The second expansion, driven by the author and presented in [11], introduced a pattern library capability that allows the encapsulation of the information necessary for the capturing of Safety Mechanisms according to the Attribute Catalogue presented in Section II. The structure of the AF3 library element is shown in figure 3 in the section 2.2, along with the practical applicability, usage description – for both pattern developer and user roles – and description of the implementation (with screenshots).

The capturing of Safety Mechanisms in AF3 was reevaluated after these changes, with results in the next table.

Table 4. AF3 Re-Evaluation

Attribute	**Evaluation Remarks**	**Score**
Name	Captured in the name of the Safety Mechanism Pattern itself.	1.00
Rationale	Captured in the SafetyRequirement entity and supported by the Safety Case Elements. Somewhat limited by missing dysfunctional view and the generic nature of requirements artifacts in AF3.	0.75
Motivation	Provides the rationale behind using the pattern and is captured in the SafetyRequirement entity, with any additional, implementation-specific information being captured in the SafetySpecification entity, and supported by the Safety Case elements. Also somewhat limited by missing dysfunctional view in AF3.	0.75
Applicability	Describes the architectural context of the safety mechanism as well as instantiation information and is captured by the LogicalArchitecture entity and augmented by the workflow described in section II. Somewhat limited by missing dysfunctional view and support for failure propagation analysis in AF3.	0.75
Structure	This attribute is captured by the entities LogicalComponent and LogicalArchitecture and their relation and supported by data model of the pattern library approach.	0.75
Participants	Captured at an interaction level by the LogicalArchitecture entity and at a structural level by the structure of the Safety Mechanism Pattern library.	0.75
Collaborations	Captured by the structure of the safety mechanism library and the safety case supporting it. Somewhat limited by missing dysfunctional view and failure propagation analyses in AF3.	0.75
Consequences	The parts of this attribute describing which components in the logical architecture or inside the Safety Mechanism's component have to be instantiated or further developed upon usage are captured in the structure model as well as in the workflow described in section II. Limited by missing dysfunctional view and failure propagation analyses in AF3.	0.50
Implementation	Is for the most part captured in the SafetySpecification entity as well as the entities LogicalComponent and LogicalArchitecture and their relation. Somewhat limited by missing dysfunctional view and failure propagation analyses in AF3.	0.75
Implications	Is still not supported very well. Out of the constituent aspects reliability, modifiability, costs and execution time, only the latter two could be improved due to the added ability to capture deployment relations and TechnicalArchitecture models in the pattern library. Reliability analysis requires tool support for dysfunctional description and failure propagation.	0.50
Usage Classification	States category of Safety Mechanisms (Fault Avoidance, Error Detection and Error Containment/Handling) to which the pattern belongs, and is captured as a textual attribute in the library element container, but would benefit from a more general classification scheme which also incorporates failure propagation analysis and allows the automated inheritance of functional and dysfunctional behavior aspects.	0.50
Example	The graphical interface of AF3 makes the captured Logical Architecture fairly easy to understand, so we can consider it to be an example. Some additional description would have facilitated the understandability of the pattern even more. We are considering adding some visual aids to the pattern library wizard to help the user understand how the pattern's elements collaborate to carry out the functionality.	0.75
Related Patterns	Captured in the data model for inclusion relations by the recursive aggregation relation of the SafetyMechanismPattern entity. The implementation is incomplete and ongoing – limiting the propagation of pattern attribute relations to a semi-automatic support.	0.50
Total Score (Out of 13)		**9.00**
Percentage		**69.2%**

The improvement, guided by the Pattern-based Approach evaluation, is significant, with capturing quality jumping from 42% to 69%. We interpret this as a validation of the approach's premise based on predefining the information targeted for capture in the model-based development environment independently of the environment itself. Yet AF3 is a domain-independent tool, with many of the automotive safety domain-specific constructs having to be (forcibly) introduced – if at all possible. It was time to see whether the capturing of Safety Mechanisms would improve in a domain-adequate framework.

3.3 Case Study II: SAFE Modeling Framework

The SAFE Modeling Framework

The SAFE/SAFE-E Project [13] is a publicly-funded European research project that aimed at analyzing the then-emerging ISO26262 Automotive Functional Safety standard and identifying how increasingly complex safety-critical applications could be developed. A simplified overview of the SAFE Modeling Framework (MF) construct is shown in the next figure.

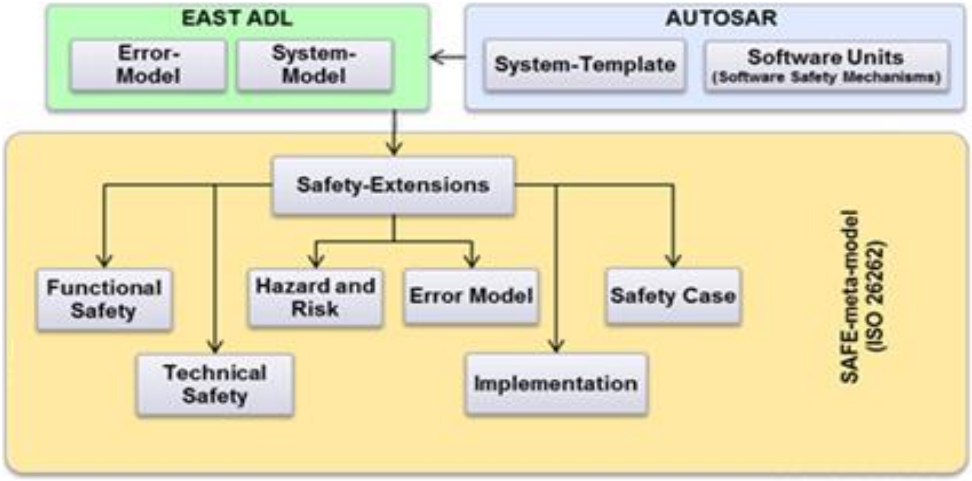

Fig. 7. SAFE Modeling Framework Extensions [47]

The project reused concepts from the architecture description language EAST-ADL [50], as well as the automotive software description framework AUTOSAR [51], extending these where necessary to develop safe automotive software architectures in compliance with ISO26262.
Detailed descriptions of the SAFE Meta-Model and various extensions are provided in publicly available descriptions at [13], such as [14]. The SAFE MF uses the "Tactics" principle introduced by Wu & Kelly in [3] to capture Safety Mechanisms. Combined with Safety Case Extensions to the SAFE MF – carried out by the author and introduced in [52], the Safety Mechanism capturing in the SAFE MF can be described by the following abstract figure.

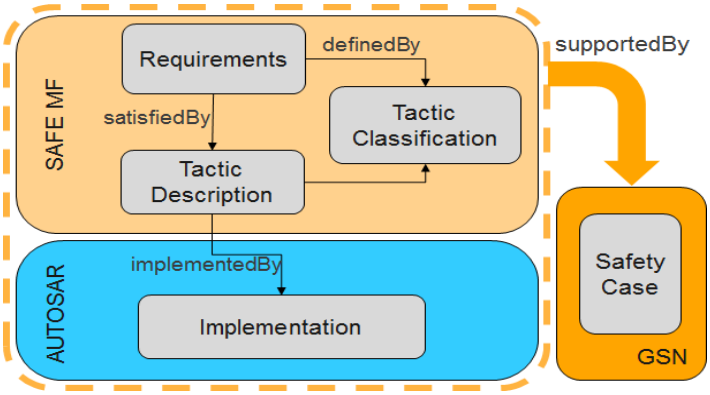

Fig. 8. Abstract Artifact Structure for Tactics Library in SAFE MF [48]

Results of SAFE Modeling Framework Evaluation
Using the attribute catalogue presented in Section II to evaluate the quality of Safety Mechanism capturing in the SAFE MF yielded the following results.

Table 5. SAFE MF Evaluation

Attribute	Evaluation Remarks	Score
Name	This attribute is captured in the name of the Tactic and the AUTOSAR or CHROMOSOME implementation it links to.	1.00
Intent	This attribute is captured using the extensive requirements expression capability of EAST-ADL, extended using SAFE artifacts to cover ISO26262-compliant hazard description, safety concepts, support for malfunction behavior and failure description and propagation.	1.00
Motivation	Provides the rationale behind using the pattern and is captured in the requirements and safety concept artifacts of SAFE, and supported by the Safety Case elements. This combination provides for a strong coverage of this attribute.	1.00
Applicability	Describes the architectural context of the safety mechanism as well as instantiation information and is captured by the Tactic Description category, which includes not only clear links the category of Safety Mechanisms (Fault Avoidance, Error Detection and Error Containment/Handling) the pattern belongs to, but to the actual fault models the patterns addresses. Furthermore, the links from requirement to specification to actual implementation artifacts means this attribute is captured very well.	1.00

Structure	Captured by the Tactic Description category, which clearly defines how the pattern relates to other artifacts. The actual implementations are standard AUTOSAR or CHROMOSOME artifacts, so their internal workings are well-known and documented.	1.00
Participants	Captured in the Tactic Description Category and supported by the safety concept and safety case artifacts in SAFE.	1.00
Collaborations	Captured by the structure of the Tactic element and the safety case supporting it. It is further supported by the direct link to the dysfunctional view and failure propagation artifacts. A precise capturing is somewhat limited by the lack of timing annotations.	0.75
Consequences	The parts of this attribute describing which components in the logical architecture or inside the Safety Mechanism's component have to be instantiated or further developed upon usage are captured in the Tactic description and the included implementation descriptions, which allow a correct instantiation into multiple target implementation environments, such as AUTOSAR. This attribute is largely dependent on tool support, and there are currently no tool implementations covering the full-range of SAFE specification and capability.	0.75
Implementation	Captured very precisely by including implementation-related information in the Tactic Description, and then linking to standard, well-defined and documented implementations in target environments, such as AUTOSAR or CHROMOSOME.	1.00
Implications	Is still not supported completely. Out of the constituent aspects reliability, modifiability, costs and execution time, Reliability analysis could be improved in comparison to the AF3 implementation – evaluated in section III.B – due to the added ability to capture dysfunctional description and failure propagation. A precise capturing is further limited by the lack of timing annotations.	0.75
Usage Classification	Captured in the Tactic Classification artifact, which not only clearly links the pattern to a category of Safety Mechanisms (Fault Avoidance, Error Detection and Error Containment/Handling), but to the actual fault models the patterns addresses. The benefit is nevertheless somewhat limited due to the lack of Pattern Relation link to other Tactics at the meta-model level, or indeed of any description of what the pattern categories mean. This denies the ability for automated inheritance of practical functional and dysfunctional behavior aspects between related patterns in a category.	0.75
Example	Despite the great expressive power of the SAFE meta-model and technology platform, there are still no standard tools implementing the entirety of the SAFE scope. Tactics in SAFE have no actual provision for examples. Assuming the implementing tools will have some graphical capability, the potential for capturing explanatory examples will be very high, with the examples being very detailed if needed, due to the expressive power of the underlying meta-model.	0.75
Related Patterns	The Tactic categorization in SAFE is currently used to ensure consistency between the requirements branch and the specification branch of a Tactic, as seen Fig. (2). It does not contain any Pattern Relation link to other Tactics at the meta-model level, or indeed any description of what the pattern categories otherwise mean. This denies the capacity for automated inheritance of practical functional and dysfunctional behavior aspects between related patterns in a category.	0.50
Total Tally (out of 13)		**11.25**
Percentage		**86.5%**

3.4 Case Study III: Continental COREPA Framework

COREPA Framework

Driven by the incessant rise in the number of integrated control units, communication systems and software components, managing architectural complexity in the automotive industry, let alone mastering it, is becoming an increasingly difficult task. Combining the recent push towards automated and autonomous driving systems with the rich array of functions and components Continental offers, selecting the right set of functions necessary to fulfill the customers' wishes and needs, as well as the corresponding technical components and system design can quickly become a daunting task without adequate methodical and tool support. As a leading automotive systems supplier, Continental's challenges are further increased by the sheer diversity of the designs its numerous customers employ. The situation is exacerbated even further by the fact that these complex functions are not the sole inhabitant of the technical components they are deployed on, but must often harmoniously coexist with other advanced assistance functions and even share resources with them as necessary. A globally distributed research, development and production rounds out the challenges, giving effective knowledge management and transfer a pivotal role in Continental's future as a leading technology company in an ever-changing world.

In line with its strategic focus on Systems Engineering excellence, Continental introduced a seamless model-driven development approach; treating models as first-class citizens where relevant information is generated, not just stored. The approach emphasizes frontloading design and analysis capabilities into the modeling framework. Continental's solution for managing the complexity and number of functions, components and designs in which they can be combined centers around a reusable functional and technical component library, COREPA (COmponent REadiness Preparation by Architecture analysis). COREPA and its Model-based Systems

Engineering framework fully embrace the separation-of-concerns and divide-and-conquer paradigms, drawing on ISO42010 [8] and ISO15288 [30] to define stakeholder viewpoints as well as hierarchy and abstraction layers.

Instantiation of the Pattern Library approach into COREPA

In its infancy, COREPA had the aim of capturing existing solutions (both functional and technical) into models and using these models to analyze synergies in both domains, in order to generate reference architectures for future function bundles. One example could be the set of functions and technical components needed to achieve a 5-star rating in the upcoming Euro NCAP 2020 [49] standard crash test – and their design(s) into a system architecture. In addition, COREPA helps define the technology roadmap for Continental's products, by identifying key future technologies. The reuse of the captured system design solution descriptions – requirements, function descriptions, safety analyses and architecture designs, was a secondary but increasingly important goal. System designs in COREPA are primarily modeled using SysML, with functional components captured in a logical view, and technical components in a technical/physical view.

Assessing the completeness or adequacy of the capturing of solutions in COREPA was a difficult task, because there was no "definition of done", i.e., no standardized structured definition of when a solution or component was described well enough in the models to allow analysis or reuse. Initial estimates by COREPA experts attested between 50-70% adequacy and completeness, albeit admitting that these were "only" estimates.

As part of a push to place the reuse aspect at the forefront of COREPA and increase the overall maturity, we introduced the pattern library method to guide and provide structure for the reuse process. The attribute catalogue used to capture Safety Mechanisms for reuse - as described in Section II – was not adequate for COREPA. A new one had to be defined. The first step involved a reduction of the attribute catalogue to 7 standard attributes, well-known from Design Pattern literature: (Name, Type, Description, Context, Applicability, Behavior, Implications).

An assessment was then carried out, similar to the previous two case studies, to evaluate how well these attributes were covered by the current models. The results were used to guide initial tool extensions to COREPA, including a large undertaking to integrate multi-criteria architecture deployment optimization into the COREPA capabilities, allowing automatic optimized deployment calculations of logical solutions to technical components, leveraging formal Design Space Exploration capabilities and implemented using an SMT solver, the results of which were published in [15]. The results of the evaluation of COREPA reuse adequacy – before and after the tool extensions – are shown in the following table:

Table 6. COREPA Before&After Evaluation Results

Catalog Attribute	**Definition**	**Evaluation before tool extensions**	**Evaluation after tool extensions**
Name	Meaningful name of the item	1	1
Type	Type of the item	1	1
Description	Short statement describing the item	1	1
Context	Exemplary context in which the item has been applied	0.75	1
Applicability	Specific usage limitations	0.25	0.5
Behavior	Description of how the item behaves as black box	0	0.25
Implications	Impact on using this item on non-functional requirements (safety, execution time,…)	0.25	1
Total Tally (out of 7)		**4.25**	**5.75**
Percentage		**60.7%**	**82%**

The result of this preliminary evaluation was pleasing as it showed significant improvement – from 60,7% to 82% capturing quality. Tool improvements and extensions continued.

Using the early success as a motivator, a more comprehensive attribute catalogue defining multiple "reusable COREPA items" was defined, and the old evaluation was then revisited. The exact attributes in this new catalogue – along with the exact descriptions of the COREPA Models – are privileged intellectual property of the Continental Corporation and can thus unfortunately not be included in this paper.

The results of the new evaluation were more sobering; the framework adequacy and completeness evaluation in capturing reusable items – before any of the extensions carried out in the previous 2 years – dropped from

60.7% in the previous evaluation to 39.6%. The result of the older tool extensions – considered in the previous table – and newer ones combined resulted in an evaluation of 72.9%, more clearly demonstrating how much work had been done, and how much still needs to be done. The new attribute catalogue is used not only as a guide for which aspects of the framework to develop next, but also as a gauge for the return on investment any framework development work or tool extension would bring, by analyzing the projected improvement in the framework evaluation vs. the expected effort/cost. An example of such newer framework extensions – providing for complete technical platform synthesis capability, using a DSL especially developed for this application - is provided in [16], where more information is also provided about the COREPA viewpoints and modeling artifacts. Most importantly, the design pattern driven approach of first clearly defining what needs to be captured, then extending the tool and evaluating, is now a standard practice for the COREPA group.

4. RELATED WORK

A recent survey of patterns in safety-critical development provided in [19] demonstrates that patterns tend to focus on clear target types of problems; engineering step (requirements, design, implementation...), category (argumentation pattern, safety / security pattern, design pattern – like redundancy...), as well as abstraction (e.g. software, hardware...), and used these to organize the results. This is useful, yet somewhat too narrow or isolated an approach for an effective reuse of technical solutions in practice. This is because be it an architecture description or the software or hardware development artifact, the single item does not tell the entire story, as discussed in section I. A holistic *Systems* Engineering approach is needed. The survey also observed wide-spread use of model-based techniques, with different modeling methods and languages used in accordance with the needs of safety engineers (mostly graphical notations like UML/SysML and GSN), but also textual patterns.

Zimmer introduced the necessity of defining relationships between related patterns in [44], and Noble built on this work in [45], defining a number of relationships between system design patterns such as *uses*, *refines*, *used by*, *combine*, and *sequence of*. In contrast to our work presented here, Noble defined these relations at a very abstract level, which does not support an expressiveness for detailing what parts of the related patterns are *refined*, *combined* or *used by* other patterns. A precise modelling of pattern relationships is thus not possible.

Combining the power of model-based techniques with the design pattern approach seems a logical step. In fact, Douglass documented his first patterns for real-time embedded systems using UML [20]. The examples are abundant – using UML to capture hardware design process patterns [21] or to document patterns for software product lines [22]; or using models to capture hardware design patterns [23] – yet the fact remains, that the focus was on leveraging the capabilities of model-based techniques for the documentation of the patterns, rather than capturing them for direct reuse in a practical setting.

The Assured Reliability and Resilience Level (ARRL) approach, presented in [24], provides a push in that direction. In essence, ARRL leverages Quality of Service (QoS) paradigms, arguing against Safety Integrity Levels (SIL) and seeking to supplant them with ARRLs, which essentially transform components into a compositional framework, where "ARRL level components carry a contract and the evidence that they will meet this contract given a specific set of fault conditions". This should allow for a safer reuse of components.

While the notion has its charm, there are several limitations to this approach:

- It assumes that the information is available in the models or components it is applying ARRL to; this is often not the case.
- It assumes that a transformation of the information, contracts and guarantees is possible into ARRL, without loss of information. In practical settings, this is not a trivial assumption, as each company will have its own artifacts.
- It requires the adoption of a new framework, which will undoubtedly meet very high resistance from industry, as well as substantial and not entirely drawback-free changes to existing and established safety standards and practices.

Denney and Pai [25] offer automation support for the generation of (partial) safety cases, by considering three inter-related tables – namely one each for hazards, system requirements, and functional requirements – as idealizations of the safety analysis and development processes. The argument structures are then automatically populated by predefined mappings from the tables. The approach is useful for structuring requirements and argumentation, is for the most part formalized, and has functioning tool support [26]. The approach does not, however, directly link design architectures or product artifacts in its function; or address patterns; or how the aspects needed for reuse are covered; nor does it show how the method may work in other fields.

Armengaud [27] proposed a conceptually similar approach for automating the assembly of safety cases. His approach is based on linking ISO 26262 activities and work products – for instance, evidence such as design and test artifacts – by means of information models. However, the approach does not address how these models structure the safety case argument; nor does he address the reuse of patterns or Safety Mechanisms s in safety.
Hawkins et al. present a very pragmatic approach – quite similar to ours – in [28], based on the automatic generation of assurance cases from design artifacts. The approach uses model weaving to link various elements relevant for arguing the safety of architecture design and product information from the AADL [36], with a meta-model of GSN, to generate a safety case argument structure. A main advantage of the approach is that the weaving models are bidirectional; allowing the generation of updated argument structures to match changes in product information, but also allowing the reflection of changes in the argument structure back to the product and architecture design. This facilitates the co-evolution of architecture design and product information with the assurance case.
The approach targets development environments using (or based on) AADL, which is not widely adopted in the automotive domain; and it does not address implementation artifacts or issues. Furthermore, the approach does not address patterns or reuse exactly, but is focused on the generation of assurance cases. The approach can, however, be used to support safety patterns, by adapting it to maintain the co-evolution of the system design and product information side with the argumentation side of a safety pattern library element. In that capacity it is most useful when applied to abstract high-level patterns. For concrete Safety Mechanisms, such as those targeted in our approach, it would be much simpler and more precise to directly link the assurance case elements to the design and implementation artifacts supporting them. In that scope, the Model-weaving approach mentioned above would be too general in its current form, and on the other hand, attempts to make it more precise would probably be over-kill and counter-productive. Its strengths simply lie elsewhere.
More closely related are the works of Hauge [41] and Hamid [40]. Hauge introduces a pattern language for safety design – called Safe Control Systems (SaCS) - with which he proceeds to capture known approaches towards a safe system design. These include both product-based as well as process-based safety concepts enabling a safe-design. Hauge's selection of what a "safety concept design" is builds on the work of Habli and Kelly in [43], in which they introduce safety case patterns for generic safety concepts – i.e., repeatable argument patterns – using GSN. The SaCS language provides good coverage of safety concept design, as demonstrated in the analysis given in [41], and expanded on in [42]. Hamid introduces a holistic approach for model repository-centric reuse of solutions for complex applications. His approach provides a methodology and tool-support for developing model repositories, supporting two categories of users: ''reuse'' producers and ''reuse'' consumers, and is demonstrated via a preliminary prototype that captures security and dependability pattern models. Hamid builds on his previous work in [53], in which he introduces so-called "SEMCO (System and software Engineering with Multi-Concerns support)", which is an integrated repository of modeling artifacts aiming at enabling the co-evolution of models covering various engineering concerns, e.g., safety, security.
Hauge's work targets basic "patterns" in safety concepts, such as "Hazard Analysis", "Risk Analysis", and "Establish System Safety Requirements" and how to combine them into composite "Safety Requirements" patterns. Our work focuses on the practical reuse of the category of technical solutions collected under the term "safety mechanisms", as well as the generalization of the concept for other types of model-based reuse. While Hamid's work offers a holistic and precise approach to building modeling repositories. Its reliance on model transformations, while supporting a more precise definition of his holistic model repositories, can be somewhat cumbersome and inflexible to a practitioner, and this reason made us opt for the relative flexibility of model-weaving. More importantly, both Hauge and Hamid have come up with their own modeling languages and tools. Similar to the ARRL and related approaches presented here, this assumes that practitioners are willing, or even capable, of completely abandoning their domain/industry-established tools in favor of an academic suite with little to no commercial support. In contrast our approach provides a blue-print for practitioners to tailor and instantiate it into their own tool suites for their own exact purposes, while enabling them to evaluate the adequacy of those tools for their individual purposes and providing a guide for eventual tool extensions/replacements.

5. OBSERVATIONS AND LESSONS LEARNED

The expanded definitions of the Safety Mechanism pattern library approach enabled the instantiations of the approach into multiple development frameworks and allowed the evaluations of the case studies. We also demonstrated the usefulness of the approach in general for reuse, as well as its impact on tool selection and development in practice.

5.1 Discipline vs. Design Freedom – The Pattern Library in practice

Expanding the number of Safety Mechanisms considered in previous works to all found in literature, we evaluated approx. 40 and implemented 7 in the AF3 Case-Study, finding none that could not readily be captured using our approach, thus demonstrating that our Safety Mechanism attribute catalogue and Pattern Library approach apply to all categories of Safety Mechanisms.
Our observations with practitioners who tried out the tool led us to increase the rigor and discipline enforced by the tool itself, to both guide and reign in abuses by the users, who – without guidance – were inevitably attracted to "short-cuts", focusing on their central artifact of interest and trying to forego the onerous task of filling in all the other necessary information.
While the wizard shown in section 2.3 has certainly increased this discipline, we feel that a certain sensitization, speaking to the mindset of the practitioners, is necessary and cannot be achieved by tool constraints alone, which when raised beyond a certain threshold led the practitioners to label the tool "unusable" and abandon it altogether.

5.2 Threats to the validity of the Evaluations

There are two main threats to the validity of our results, stemming from our choice of an evaluation scale.

Subjectivity of the Evaluation Scale
Our evaluation of the instantiations of the approach is highly subjective. We accept this, and do not believe this will negatively affect our results, as the purpose of the evaluation is to study the instantiation of the approach in different development environments and its impact on them, and not to actually evaluate the tools themselves; independently or with each other.

Parity of the Attributes' Weights
In our evaluation scale we gave equal weights to the coverage of each attribute, essentially declaring all attributes to have equal importance. We can readily believe this may not be precisely the case. We nevertheless decided to keep the evaluation scale as is; for two reasons.
First and foremost, as all attributes are necessary, we don't believe that the variations between the weights of the individual attribute would have been significantly large so as to skew the results of the evaluation.
More importantly, we don't believe a more accurate representation would have affected the results of the case study evaluations. This is because the evaluations are meant to provide comparisons of the adequacy of a tool suite for solution capturing for reuse before and after the instantiation of our approach – and not to individually evaluate these tools objectively. As the case studies in the next three chapters will show, improvements from using our approach where observed across all attributes and weren't bundled in any repeated attributes, which could have nullified our results given a different weighting.
For applications of the approach to reuse problems where the parity of the attributes' importance is unclear, it would be possible to evaluate these weights using an analysis of their impact on the reuse maturity, or alternatively an expert survey to query the same. Such activities were beyond the scope and purpose of our work.

5.3 Case study evaluation results

Using the case studies in section 3 we demonstrated the following benefits from using the approach:

- A guided capturing of reusable solutions. As an immediate benefit of the approach, practitioners no longer have to wonder if they have captured all the necessary information nor approach the problem in an ad-hoc fashion. Pre-defining the attributes necessary to capture a reusable solution guarantees that practitioners know which information needs to be captured.
- A gauge of if and how well the necessary information is being captured in the current tool-suite. Pre-defining the attributes necessary to capture the reusable solution also means that practitioners will no longer be limited by the current capabilities of their tools, or the way they currently use them, but will look into adapting and expanding their tool-suite to better suite this goal, as they now have a gauge to test against. This is a more abstract, longer-term benefit.
- An aide for selecting better tools. Knowing exactly which information needs to be captured means that practitioners can look beyond the limitations of their current tools, and onto adopting such tools as truly fit their needs. This is the most abstract, long-term benefit of the approach.

Our observations while evaluating the second and third case studies confirm an earlier hypothesis: the more detail-rich and context-adequate or domain-specific a modeling/tool framework, the higher the number of attributes one can capture in its models and the better the quality and degree of attribute capturing, thus increasing the quality of the final product. This hypothesis is supported by our observations while evaluating the second and third case studies. The shortcomings exposed by the first evaluation reflect limitations due to the domain-independent nature of AF3, with many of the missing features being very specific to tools specifically developed for safety-critical automotive embedded systems development.
We interpret these results to be a validation of the approach's premise, based on predefining the information targeted for capture in the model-based development environment in a structured systematic approach – independently of the environment itself.
Importantly – and contrary to related work, our approach features an explicit model-based representation of the reusable solutions within the practical context of their usage and the problems they solve, which can be instantiated into any model-based development environment. This has several advantages, including 1) a better characterization of the problem space addressed by the pattern – better than the textual description otherwise used in pattern templates, 2) a more natural representation of the transformations embodied in the application of the pattern, and 3) a better handle on the selection, rationale and application of the patterns [18]. Our approach is not limited to one type or category of artifact, but targets the capturing of all information necessary for successful system design reuse.
Finally, and most importantly, the COREPA case study demonstrates how the underlying concept of our approach can generically be applied to a model-based reuse-in-practice problem in general, guiding both the systematic reuse as well as the corresponding tool development in a large-scale industrial setting with demonstrated positive impact.

6. CONCLUSION

The design of functional-safety systems is largely driven by best practices – like the use of fault monitors, safe states, or redundant paths. These best practices can often be presented in the form of design patterns – both to describe a possible solution but also to document an argumentation about their contribution to a safety case. Our approach provides for a library of such patterns, allowing the identification and reuse of suitable measures and their corresponding components along with their safety cases. It also guides practitioners into a systematic capturing – in the Pattern *Developer* role – and reuse of technical solutions such as Safety Mechanisms – in the Pattern *User* role.
Beyond the immediate systematic Safety Mechanism reuse aspect, the approach provides developers with a gauge with which to measure not only the quality and completeness with which they are capturing their reusable items, but extends to allow an assessment of the adequacy of the tools they are using for such reuse, and provide a guide for the extension of such tools and or selection of others more adequate, as demonstrated by the case studies shown in this paper.
Finally, and most importantly, we have shown that the approach can be generally applied to other reuse endeavors in a large-scale industrial setting with demonstrated positive impact. This work is ongoing.

Acknowledgment
I am very grateful for the effort and input from my PLoP Mentor, Eduardo Fernandez. My paper improved substantially during the shepherding phase. My gratitude also goes to all my fellow members of writer's workshop group D (the best one); especially Rebecca Wirfs-Brock, Lise Hvatum, Helene Finidori and Julio Moreno Garcia-Nieto.
The initial specification of this approach stems from and was carried out in the integrated development environment for the ITEA2 SAFE project and the EUROSTARS SAFE-E project, with proof-of-concept implementation in the research CASE tool AUTOFOCUS3 (AF3). AF3 is the result of the hard work of many researchers and employees of the Software and Systems Engineering Department at the fortiss Institute in Munich, Germany.

REFERENCES

[1] International Standards Organization, ISO 26262 Standard, Road Vehicles Functional Safety, www.iso.org, 2011.
[2] T. Kelly and R. Weaver: "The Goal Structuring Notation . A Safety Argument Notation". Proc. DSN 2004 Workshop on Assurance Cases, 2004

[3] 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.

[4] 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.

[5] Stefan Wagner, Bernhard Schatz, 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.

[6] M. Khalil “Pattern-based methods for model-based safety-critical software architecture design” ZeMoSS 2013 Workshop at the SE 2013 in Aachen, Germany. 2013.

[7] AutoFOCUS 3, research CASE tool, af3.fortiss.org, 2018 fortiss.

[8] ISO/IEC/IEEE 42010:2011, Systems and software engineering — Architecture description. www.iso.org.

[9] C. Carlan. “Implementierung unterschiedlicher Redundanzkonzepte zur automatischen Generierung entsprechender logischer Strukturen für ein sicheres Verhalten gemischt-kritischer Systeme im CASE-Tool AutoFocus3“. Bachelor Thesis (German). Technische Universität München. Faculty of Informatics. Chair of Software and Systems Engineering. 2012. http://download.fortiss.org/public/carlan/BA_Carmen_Carlan.pdf

[10] S.Voss, B. Schatz, M. Khalil, C. Carlan “A step towards Modular Certification using integrated model-based Safety Cases” VeriSure 2013.

[11] M.Khalil, B. Schatz, S. Voss. “A Pattern-based Approach towards Modular Safety Analysis and Argumentation”. Embedded Real Time Software and Systems Conference (ERTS2014) – Toulouse, France. 2014.

[12] Khalil M., Prieto A., Hölzl F. (2014) A Pattern-Based Approach towards the Guided Reuse of Safety Mechanisms in the Automotive Domain. In: Ortmeier F., Rauzy A. (eds) Model-Based Safety and Assessment. Lecture Notes in Computer Science, vol 8822. Springer, Cham.

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

[14] The SAFE Consortium. Deliverable D3.5c “SAFE Meta-Model: System, SW, HW reference meta-model definition”. www.safe-project.eu ITEA2. 2014.

[15] Johannes Eder, Sergey Zverlov, Sebastian Voss, Maged Khalil, and Alexandru Ipatiov. 2017. “Bringing DSE to life: exploring the design space of an industrial automotive use case”. In 2017 ACM/IEEE 20th International Conference on Model Driven Engineering Languages and Systems (MODELS). https://doi.org/10.1109/ MODELS.2017.36

[16] Johannes Eder, Sergey Zverlov, Sebastian Voss, Alexandru Ipatiov, and Maged Khalil. 2018. “From Deployment to Platform Exploration: Automatic Synthesis of Distributed Automotive Hardware Architectures”. In 2018 ACM/IEEE 21st International Conference on Model Driven Engineering Languages and Systems (MODELS). In review.

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

[18] Hafedh Mili & Ghizlane El-Boussaidi. “Representing and applying design patterns: what is the problem?” Model Driven Engineering Languages and Systems, Lecture Notes in Computer Science. Volume 3713, pp 186-200, 2005.

[19] Gleirscher, Mario & Kugele, Stefan. (2016). A Study of Safety Patterns: First Results. 10.13140/RG.2.2.23347.22562.

[20] B. P. Douglass. “Doing Hard Time: Developing Real-Time System with UML, Objects, Frameworks, and Pattern”. Addison-Wesley, New York. 1999.

[21] R. Damaševicius and V. Štuikys. “Application of UML for hardware design based on design process model”. In ASP-DAC ’04: Proceedings of the 2004 Asia and South Pacific Design Automation Conference, pp. 244–249, Piscataway, NJ, USA, 2004. IEEE Press.

[22] F. Rincon, F. Moya, J. Barba, and J. C. Lopez. “Model reuse through hardware design patterns”. In DATE ’05: Proceedings of the conference on Design, Automation and Test in Europe, pages 324–329, Washington, DC, USA, 2005. IEEE Computer Society.

[23] H. Gomaa and M. Hussein. "Model-based software design and adaptation". In Proceedings of the 2007 International Workshop on Software Engineering for Adaptive and Self-Managing Systems, p. 7. IEEE Computer Society. 2007.

[24] E. Verhulst, and B. H.C. Sputh. "ARRL: A Criterion for Composable Safety and Systems Engineering". Workshop SASSUR (Next Generation of System Assurance Approaches for Safety-Critical Systems) of the 32nd International Conference on Computer Safety, Reliability and Security SAFECOMP. 2013.

[25] E. Denney and G. Pai. "A lightweight methodology for safety case assembly." In Computer Safety, Reliability, and Security. pp. 1-12. Springer Berlin Heidelberg. 2012.

[26] E. Denney, G. Pai, and J. Pohl. "Advocate: An assurance case automation toolset". In proc. of Workshop SASSUR (Next Generation of System Assurance Approaches for Safety-Critical Systems) of the 31st International Conference on Computer Safety, Reliability and Security SAFECOMP. pp 8-21. 2012.

[27] E. Armengaud. "Automated safety case compilation for product-based argumentation". Embedded Real Time Software and Systems Conference (ERTS 2014), Toulouse, France. 2014.

[28] R. Hawkins, I. Habli, D. Kolovos, R. Paige, and T. Kelly. "Weaving an Assurance Case from Design: A Model-Based Approach". 16th IEEE International Symposium on High Assurance Systems Engineering – HASE2015. Florida. 2015.

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

[30] ISO/IEC/IEEE 15288:2015. Systems and software engineering -- System life cycle processes Standard. https://www.iso.org

[31] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. “Design Patterns: Elements of Reusable Object-Oriented Software”. Addison-Wesley, Boston, MA, USA, 1994.

[32] A. A. Prieto Rodriguez. “Exploration of a pattern--based approach for the reuse of safety mechanisms in embedded systems”. Master’s Thesis. Technische Universität München. Faculty of Informatics. Chair of Software and Systems Engineering. 2014.

[33] Maged Khalil. “Pattern libraries guiding the model-based reuse of automotive solutions”. 2018. 10th System Analysis and Modelling (SAM) conference. Copenhagen. Denmark. Accepted.

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

[35] S. Zverlov, M. Khalil and M. Chaudhary. "Pareto-efficient deployment synthesis for safety-critical applications". Embedded Real Time Software and Systems Conference (ERTS 2016), Toulouse, France. 2016.

[36] AADL. Architecture Analysis and Design Language. SAE International Standard AS-5506 Ver. 2.1. 2102. www.aadl.info. 2018.

[37] D. Schmidt, Model-driven engineering, IEEE comput. 39 (2) (2006) 41–47.

[38] I. Crnkovic, M.R.V. Chaudron, S. Larsson, Component-based development process and component lifecycle, in: Proceedings of the International Conference on Software Engineering Advances, ICSEA 2006, IEEE Computer Society, 2006, p. 44.

[39] W. Frakes, K. Kang, Software reuse research: Status and future, IEEE Trans. Softw. Eng. 31 (7) (2005) 529–536.

[40] Hamid, Brahim "A Model-Driven Approach for Developing a Model Repository: Methodology and Tool Support." (2017) Future Generation Computer Systems, vol. 68. pp. 473-490. ISSN 0167-739X

[41] A. Hauge and K. Stølen, "An analytic evaluation of the SaCS pattern language—Including explanations of major design choices", PATTERNS 2014, 79-88.

[42] A. A. Hauge. "SaCS: A Method and a Pattern Language for the Development of Conceptual Safety Designs" Doctoral Dissertation. Series of dissertations submitted to the Faculty of Mathematics and Natural Sciences, University of Oslo. No. 1568. ISSN 1501-7710. 2014.

[43] I. Habli and T. Kelly, "Process and Product Certification Arguments – Getting the Balance Right," SIGBED Review, vol. 3, no. 4, 2006, pp. 1–8.

[44] W. Zimmer, "Relationships between Design Patterns," in Pattern Languages of Program Design. Addison-Wesley, 1994, pp. 345–364.

[45] J. Noble, "Classifying Relationships between Object-Oriented Design Patterns," in Proceedings of Australian Software Engineering Conference (ASWEC'98), 1998, pp. 98–107.

[46] The SAFE-E Consortium. Mou Dongyue. Deliverable D4.3 "Specification of plug-in for test suite generation". EUROSTARS. 2014.

[47] J. Kemmerich and M. Khalil (Eds.). ITEA/EUROSTARS SAFE / Safe-E Project. Deliverable D6.b / D6.2b "Methodology and application rules Documentation". The SAFE / Safe-E Consortium. www.safe-project.eu. 2014.

[48] The SAFE / SAFE-E Consortium. ITEA/EUROSTARS SAFE / Safe-E Project. Deliverable D3.1.3b / D3.4b "Updated proposal for extension of Meta-model for safety-case modeling and documentation". The SAFE / Safe-E Consortium – www.safe-project.eu. 2014.

[49] The European New Car Assessment Programme. www.euroncap.com

[50] EAST-ADL (Electronics Architecture and Software Technology - Architecture Description Language). www.east-adl.info

[51] AUTOSAR (AUTomotive Open System ARchitecture). www.autosar.org.

[52] The SAFE / SAFE-E Consortium. ITEA/EUROSTARS SAFE / Safe-E Project. Deliverable D3.1.3b / D3.4b "Updated proposal for extension of Meta-model for safety-case modeling and documentation". The SAFE / Safe-E Consortium – www.safe-project.eu. 2014.

[53] B. Hamid, SEMCO Project, System and software Engineering for embedded systems applications with Multi- COncerns support, http://www.semcomdt.org.

[54] Buschmann et al. "Pattern-Oriented Software Architecture, Volume 4, A Pattern Language for Distributed Computing.". ISBN: 978-0-470-05902-9. John Wiley & Sons Ltd, The Atrium, Southern Gate, Chichester, West Sussex PO19 8SQ, England 2007.

[55] D. Riehle and H. Züllighoven. "A Pattern Language for Tool Construction and Integration Based on the Tools and Materials Metaphor". In Pattern Languages of Program Design, J.Coplien and D. Schmidt, Eds. pp. 9-42. Reading, MA. Addison-Wesley. 1995.