Towards Systematic Safety System Development with a Tool Supported Pattern Language

Jari Rauhamäki, Timo Vepsäläinen and Seppo Kuikka
Department of Automation Science and Engineering
Tampere University of Technology
Finland
Email: {jari.rauhamaki, timo.vepsalainen, seppo.kuikka}@tut.fi

***Abstract*—Design patterns illustrate qualities and features that would suit well in current understanding of safety system development, design and documentation. However, though a number of design patterns for safety system development have been proposed, the focus has been on individual quality attributes such as fault tolerance and reliability. The systematic use of design patterns in the development process has received less attention. In this paper, we discuss and illustrate extended usage possibilities for design patterns as part of safety system development. We discuss a design pattern language that we are developing to cover, e.g., safety system architecture, scope minimization and co-operation with basic control systems. Use of patterns for documentation purposes, tool support for using patterns, and rationale for the pattern approach are discussed as well.**

Keywords-safety system; software; design pattern; safety standard; tool support

I. INTRODUCTION

Design patterns are a means to systematically promote the re-use of design and proven solutions to recurring problems and challenges in design. Each design pattern represents a general, reusable solution to a recurring problem in a given context. Triplets of problems, contexts and solutions are also the essential pieces of information in patterns. In addition, pattern representation conventions can include, among others, relations to other patterns. With such relations describing, for example, rational orders to use patterns, patterns can be combined to collections and to pattern languages. Depending on patterns, the natures of their solution parts can vary too, for example, from source code templates to text and Unified Modeling Language (UML) illustrations.

Software safety functions are software parts of usually multi-technical systems, the purpose of which is to ensure the safety of controlled processes and plants. Unlike many other software systems, safety systems are developed according to standards. The standards govern the development lifecycle activities, as well as techniques and applicable solutions of such systems. However, although design patterns have been specified also for safety system development, their systematic use has not been researched in the domain. This is surprising because the use of patterns could facilitate both design and documentation activities, which are equally important in safety system development.

In this paper, we address the aforementioned issues. The contributions of the paper are as follows. We rationalize how and why design patterns, which have already shown their value in software development, in general [1], could be especially useful in safety system development. We discuss a design pattern language for safety systems, which has been developed and published iteratively and is to be finalized during DPSafe project in collaboration with Forum for Intelligent Machines (FIMA) in the machinery domain. Lastly, we discuss and rationalize the role of tool support in facilitating the use of patterns and in benefitting from patterns.

The rest of this article is organized as follows. Section 2 reviews work related to design patterns and the use of design patterns in safety system development. Section 3 presents a view on the development of software safety systems and rationalizes why and how design patterns could be beneficial. In Section 4, we discuss a design pattern language for safety system development that has been developed at the Tampere University of Technology. Before conclusions, Section 5 discusses the role of tool support when trying to benefit from patterns.

II. RELATED WORK

The design pattern concept was originally presented by Alexander [2][3] in the building architecture domain to refer to recurring design solutions. In software development, design patterns begun to attract interest after the publication of the Gang of Four (GoF) patterns [4]. Thereafter, collections of design patterns have been gathered and used for various purposes in various domains. Results from their use have included, among others, improvements in quality of code, as well as improved communication through shorthand concepts [1].

Design patterns have also been developed for special purposes and application domains, including critical [5] and distributed [6] control systems. In the functional safety domain, especially, patterns already cover many solutions and techniques that are recommended by standards, such as IEC 61508 [7] and ISO 13849 [8]. For example, related to

 ISBN: 978-1-61208-438-1

architecture design in [7], there are patterns to implement redundancy [9] and recovery from faults [10].

Pattern languages, on the other hand, aim to provide holistic support for developing software systems by using and weaving patterns and sequences of patterns [11]. For embedded safety system development, for example, a large collection of (both software and hardware) patterns for various problems is listed in [5]. However, the multi-technical collection is not regarded as a pattern language, per se.

Partially because of reasons to be discussed in the next section, documentation is of special importance in safety system development. A developer of a software safety system needs to be able to prove the compliance of the application to standards. Otherwise, the application cannot be used in the safety system. However, certifiable safety applications are not made by coincidences but by designing the systems and applications systematically, with certifiability in mind. As such, also the software parts need to be specified (modeled) prior to their implementation. On the other hand, the suitable solutions (patterns) that are used in the applications should already be visible in the models. Otherwise, the use of the patterns would not be documented in the models and valuable information could be lost.

It is thus clear that the systematic use of design patterns in safety application development requires tool support for the patterns already in the modeling phase. This is regardless of whether or not the models can be used in producing (automatically) executable code as, e.g., in Model-Driven Development (MDD). Using and applying patterns in UML, which is currently the de-facto software modeling language, has been addressed in several publications. For example, work has been published to specify patterns in a precise manner [12], to apply patterns to models [13, 14], to detect pattern instances [15, 16] and to visualize pattern instances in models and diagrams [17]. However, without extensions the support for patterns is still weak in UML [18].

III. PATTERNS IN SAFETY SYSTEM DEVELOPMENT

The development of safety functions is governed by standards, such as IEC 61508 [7], IEC 62061 [19], and EN ISO 13849-1 [8]. These standards guide the development of safety systems involving electric, electronic and programmable electronic control systems in their operation. Regardless of the variety of standards, we outline a generic development process for safety systems common to the aforementioned standards. The simplified process is illustrated in Figure 1.

The development process begins by the definition of the concepts and scope of the system to be developed. This includes forming an overall picture of the system and defining the boundaries of the system/machine to be analyzed or made safe. The next step is to carry out a hazard analysis and risk assessment. The role of this phase is centric as only known risks can be consciously mitigated. Otherwise risk mitigation measures have no justification. Typically, risk assessment includes hazard identification, risk estimation and evaluation. The former provides an indicator for the risk and the latter assess the impact of the risk, that is, is the risk

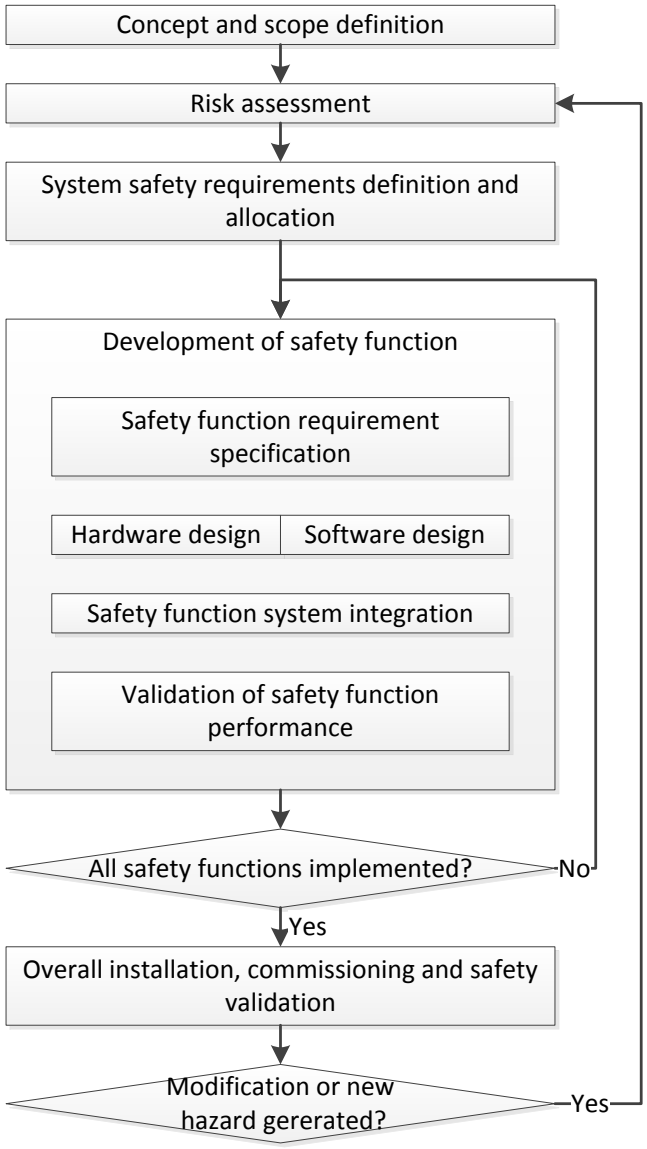

Figure 1. Simplified safety system development process according to EN ISO 13849-1 [8] and IEC 61508 [7]

tolerable or not. Intolerable risks need to be mitigated or made tolerable otherwise.

As the risks are assessed, the requirements considering the system safety can be justifiably made. In this phase, suitable risk reduction methods are selected and their requirements are documented. In the context of this paper it is assumed that the risk reduction method is a protective measure depending on a control system to implement the required functionality. In addition, the allocation of the measures is done. That is, to allocate the measures for dedicated functions.

The next phase is the development (realization in IEC 61508 terminology) of the safety functions allocated in the previous phase. The development process starts with compiling a requirement specification for the safety functions. The specification should include both functional

 ISBN: 978-1-61208-438-1

descriptions, what the functions need to do, and non-functional descriptions, how or within which restrictions the functions need to operate.

Quite often, the non-functional descriptions include the specification of performance or integrity levels for the functions. When the requirement specification is completed, the hardware and software design can begin. In this state the hardware and software parts of the safety function are designed, potentially with separation between the design teams. Thus, hardware and software integration needs to take place along the design process. At this point, a functional entity can be constructed including both the hardware and software to be used in the final system. Finally, the results of the safety function development are verified to match the safety function requirements and required performance/integrity levels. If unimplemented safety functions exist, the development process is reinitialized for the next safety function.

A. Utilization of patterns in safety system development

In the context of safety system development and design, design patterns can be used to capture and provide solution models for techniques and applicable solutions that are recommended and/or required by applicable standards. In this case, a design pattern captures the solution that is used in order to fulfill the requirements and recommendations of a standard. Such design patterns can be linked to the parts of the standards for which the design patterns provide a complete or partial fulfillment or help to achieve to fulfill the standard requirements. This kind of approach also supports building the libraries of named solutions. That is, the patterns support the awareness and usage of the solutions.

One can justifiably argue that standard solutions to recurring problems have been applied in safety system development and other domains of engineering for years – without necessarily calling them patterns. However, their unconscious use may not have eased the task of documenting the systems. Since design patterns provide names for solutions, they can be used in communication, too [1]. Though initially applicable to discussions and face-to-face communication, design patterns can be used as a part of written and diagrammatic documentation. This is achieved by referring to the solution illustrated by a pattern with the name of the pattern that should be both illustrative and related to the application context.

The documentation aspect can be achieved by marking the patterns in, e.g., diagrams that are used as a part of the system documentation. This can enhance traceability between the standard solutions and their practical applications in systems. For a pattern-aware person, this may increase the understandability and traceability of the design decisions, too. To take further advantage of this setup, statistics could be gathered to see which patterns are used the most and in which kind of situations. It can also be noted that the quality attributes understandability and traceability are similarly components of systematic integrity acknowledged by IEC 61508 [7].

Other viewpoints supporting the utilization of design patterns in safety system development include for instance [20]:

- Patterns document well-tried solutions and thus condense experience on proven solutions, which is of special importance in the domain. The approach resembles, for instance, the proven in use concept defined by IEC 61508.
- Patterns can alleviate bureaucracy by providing practical solutions and approaches to fulfil requirements given to safety system development in, for example, standards. Bridging the gap between the requirements and design and implementation eases the burden of designers.
- Patterns create the vocabulary of solutions to domains. Assuming that the patterns are known by both the developer and maintainer of a system, patterns can help to communicate the structural and operational principles of the system. This aspect thus improves the communicability and maintainability of the system.

B. Safety system patterns

In the context of this paper, we are especially interested in design patterns for safety system development, called safety system patterns here. These patterns are, or at least they are meant to be, most useful in the development of (functional) safety systems. This does not indicate that the patterns could not be used for other purposes as well. However, the contexts of the patterns relate them to the safety system development. It is up to the readers or appliers of the patterns to judge whether the solutions are applicable outside the indented contexts of the patterns, too.

It should be noted that a pattern does not necessarily illustrate the cleverest or the most innovative solution or approach to the defined problem. Instead, the preferable approach is to provide proven solutions and approaches that have been utilized successfully in practice, in real projects and systems. This is, on one hand, targeted to provide assurance on the applicability of the solution, for instance, in the eyes of an inspector. On the other hand, the most innovative solutions might promote other quality attributes than simplicity, which is one of the most important driving qualities behind a safety system development.

So, which parts does a safety system pattern consist of? In our work, we have used a slightly modified canonical pattern format [21]. That is, each pattern documents the context, problem and solution. They are complemented with forces, consequences, example, known usages and related patterns, see Figure 2. The triplet of context, problem and solution provides the main framework for the patterns. These aspects should provide sufficient information to apply a given pattern. However, the other aspects, for instance, support the selection of the most suitable pattern and help to identify other potentially applicable patterns. The former aspect is achieved through the definition of forces and consequences. Forces relate to the context, refine the problem, and direct the solution to the one selected to be illustrated on the pattern. On the other hand, consequences

 ISBN: 978-1-61208-438-1

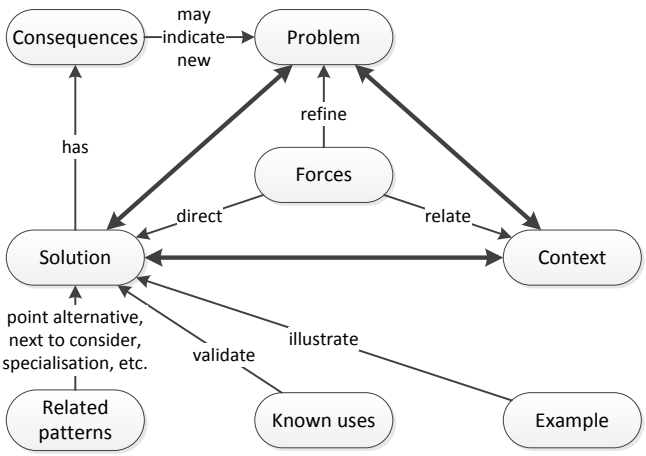

Figure 2. The pattern structure used in our safety system patterns.

provide hints to select a solution proposed by a certain pattern. Presumably one wants to select a pattern or a solution that has the most positive consequences and/or the least negative consequences produced by the solution.

In addition to the mentioned pattern aspects, safety system patterns could be complemented with an aspect indicating the applicable performance level (PL), safety integrity level (SIL), or similar quantity. This is to indicate for which purposes or levels (as defined in standards) the pattern can be used. [21]. For certain patterns or solutions such indicators can be given directly and for others such indicators are indirect or cannot be given at all. For instance, a pattern implementing cyclic execution behavior could be recommended or highly recommended on all safety integrity levels (as defined on IEC 61508-3:2010 table A.2 [7]).

How and where can design patterns then be obtained? Foundationally, design patterns document recurring solutions. The basic assumption is that at least three known usages for a solution need to be obtained to call a solution a design pattern [22]. Keeping this in mind at least the following pattern mining approaches can be considered.

As standards, such as the mentioned IEC 61508 and EN ISO 13849-1, provide requirements considering safety system design and development, they are potential candidates as source information. One potential approach is to take requirement clauses or required techniques or methods and search and provide practical solutions to fulfil the requirements. Depending on the standard and case, the standard may or may not provide instructions on how to actually apply and use required methods, techniques and clauses. Thus treating such elements as problems yields a way to found similar solutions and format them as patterns. For instance, one could consider graceful degradation, which is at least recommended on all SIL levels (as defined by IEC 61508-3:2010 table A.2), and mine patterns to design and implement graceful degradation on software. Using this approach, the integrity (or performance or similar quantity) levels can be directly linked to the patterns.

Literature and similar sources provide a feasible source for pattern mining. Solutions found from different literature sources can be considered pattern input. However, potentially the most credible sources for pattern mining are existing systems and their documentation. In the context of safety system patterns, such sources would be safety systems, their documentation and developers. To provide additional credibility for the mined safety system patterns (at least from the standard point of view), the patterns should be mined from inspected and approved systems. Such merit supports the patterns as the solution has been used as a part of an approved system. It should be noted, however, that a pattern originating from an inspected system does not directly implicate that the new system in which the pattern is applied, would be automatically approved. Nevertheless, such a pattern provides support and trust to believe that the solution is approvable in similar context.

Thus, ideally safety system patterns are mined from existing, inspected, and approved safety systems. As such, the solutions should be applicable on similar integrity level systems and also on lower levels although this is not always the case. Actually, by looking for instance IEC 61508-3 Annex A, this is not always the case. There are methods and techniques highly recommended, e.g., on SIL 3-4 and only recommended on SIL 1-2. Apparently the method or technique is still applicable, but it may be considered too heavy-weight or expensive for the lower integrity levels. To complement this approach, the inspection process and results could be systematically used to document the approved solutions in the form of patterns. During the process, the inspector approves and declines some of the solutions, approaches, and design decisions, which should be considered valuable input for future work. In the end, the inspections cost money and other resources to the customer so it is rational to try to minimize the process and to learn from mistakes and successful designs. Such work would support one of the purposes of patterns in the first place, that is, the systematic reuse of solutions.

IV. A PATTERN LANGUAGE FOR SAFETY SYSTEMS?

First of all, what do we mean by a pattern language? A pattern language is in our case a set of patterns that consider the same domain and are interconnected through relations. According to Eloranta et al., a pattern language is a concept "guiding the designer in building a coherent whole using patterns as building blocks" [6]. In this context, building block mindset, pattern relations and shared domain context between the patterns is seen centric to form the grammar to use the patterns. In practice, the pattern language defines restrictions, rules and suggestions on how to compose the designs of the provided building blocks. [6]. A collection of patterns, in contrast to a pattern language, does not have to have grammar or relations between the patterns.

The relations promote co-usage of the patterns as they guide a designer through the language by providing her with links indicating patterns that can be considered next, alternative, specialized and incompatible solutions related to the pattern that has been recently applied. Although the described approach may ease decision making, it may also narrow the designer viewpoint. A pattern language cannot include all possible solutions and the ones that are included,

do not necessarily introduce the best alternative for a problem or situation under consideration.

One way to utilize the pattern language in design work was described above. The mentioned pattern relation based language walkthrough approach is a rather optimistic view at least if a large context is considered. Safety system development as well as other system development is a process consisting of multiple phases. Covering all of these with a single language of patterns is a large scale problem itself not to mention how to parse a meaningful language by establishing the pattern relations and interconnections. Still, patterns can provide pinpointed solutions to encountered problems and the related patterns may offer ideas during the design process. From our perspective, this is a more feasible use case for a safety system pattern language. To support the usage of the language, the patterns should be, however, grouped so that they resemble the corresponding design phases. That is, architectural patterns would benefit architecture design phase issues and implementation patterns (or idioms) the implementation phase issues.

The safety system design pattern language developed at the Tampere University of Technology has currently some 50 patterns and/or pattern candidates and some of them have been discussed in the workshops of patterns conferences [23]-[27]. (Pattern candidates are initial pattern ideas that do not yet have three known uses, that is, they are under construction. We have found writing pattern candidates an excellent way to communicate the ideas and find new known usages for the pattern candidates.)

In its current state, relations have not been specified for all the patterns of the language, but there are relations between the individual patterns. For example, patterns can specialize more general solutions in stricter contexts. Thus one could say the language lies somewhere between a pattern language and a collection of patterns at the moment. However, our purpose is to develop a full pattern language for safety system development.

We started the work in 2010 and the patterns have been collected, developed and published under various projects such as SULAVA, ReUse, and currently under DPSafe project. In the DPSafe project, we are working with several companies involved one way or another in safety systems design and development in the context of machinery applications. The target of the project is to mine and document design patterns considering software based safety functions and systems as well as gain new known uses for the existing patterns and identified pattern candidates. The participating companies include machinery producers, engineering offices, as well as software houses so there is potential to have different relevant views on the subject.

The patterns are targeted to safety system development. Currently, the language includes patterns and pattern candidates considering, for instance:

- development process
- risk mitigation strategies
- architecture and principles in terms of
  - software
  - hardware
  - system
- co-existence with control system
- scope reduction

In contrast to, for example, redundancy, diversity and other fault tolerance related matters, the sub domains mentioned above seemed to have less attention by pattern community. Thus our purpose is to extend the pattern approach to cover larger part of the safety system development outside the fault tolerance aspect. According to our work carried out in the DPSafe project, there seems to be a clear need for such an approach.

V. On Tool Support for Design Patterns

Whereas some of the benefits of patterns described in Section 3 could be achievable in any case, it is clear that tool support for patterns could increase their benefits significantly. For example, even without tool support, pattern names can become a part of the developer vocabulary [1]. Without a doubt, recurring solutions have also been used in the domain. However, using patterns to improve the traceability of standards solutions, for instance, would certainly benefit from automated functions already during the specification and modeling of the applications. Unfortunately, the support for patterns is in current software modeling tools restricted, at best. The purpose of this section is to discuss opportunities and challenges related to pattern tool support in safety system development. When appropriate, lessons learned from the previous work of the authors [18] will also be provided.

A. On Pattern Modeling

As mentioned, tool support for patterns is currently weak. For example, the pattern concepts of UML, structured collaborations [28], restrict patterns to describe the contents of the UML classifiers only. Thus, elements such as components and packages that would be useful in describing architectural patterns (for instance) cannot be used in patterns in UML [18]. The variety of published patterns in literature, however, covers problems on different levels of design and for various purposes. It cannot be said that all the patterns would be related to classifiers (classes) when all patterns are not even related to software systems. The origin of the (pattern) concept is in building architectures [2, 3] and there are also, for example, multi-technical pattern collections (such as [5]) with both software and hardware aspects. It is thus clear that the UML pattern concepts are currently too restricting, by nature.

With respect to the modeling of multi-technical patterns mentioned above, they could be used in SysML models, which are not restricted to software. However, the use of patterns would not have to be limited to modeling languages at all. For example, patterns could be equally useful in, for example, Computer Aided Design (CAD) tools and software Integrated Development Environments (IDE), in aiding practical design and programming work. Similarly to software engineering, also other engineering disciplines most certainly have recurring problems with known solutions.

While acknowledging this, in our work [18] the focus in developing tool support has been on safety systems and their UML and Systems Modeling Language (SysML) based

 ISBN: 978-1-61208-438-1

modeling in a Model-Driven Development (MDD) context. With new pattern modeling concepts and by integrating them into both UML and SysML, the aim has been to support hardware aspects in addition to software and UML modeling. Safety systems are also systems that are developed and approved as a whole. Good practices and documentation are needed not only for software parts but for all parts of the systems, regardless of their implementation technologies. However, while the developed approach [18] currently allows pattern definitions and instances to consist of practically any modeling elements, the approach suffers from the drawback of not being easily portable to standard tools.

B. On Pattern Instances

In addition to (more or less) formal approaches, e.g., that of UML, modeling tools could support patterns also in an informal manner. Informal support has been developed into, e.g., MagicDraw that enables instantiating patterns from libraries by copying modeling elements. This functionality is not restricted to classifiers as is the case with standard UML. However, copying patterns (informally) can support mainly the aspect of using the solutions and not necessarily using the information about the use of the solutions. Copying model elements may not enable storing information about the elements being part of a pattern instance so that the information could be used for, e.g., documentation purposes.

There is existing research, e.g., [15] and [16], on detecting pattern instances in design models by searching for model structures that are similar to pattern definitions. However, it is questionable whether the use of such work would be an appropriate solution in safety system development. A developer does not use a design pattern by a coincidence. Instead, developers decide to apply patterns because they are facing challenges that they aim to solve with the solutions of the patterns. As such, it is natural that the decisions, which are architectural decisions, should be documented. Why should one try to guess whether a pattern has been applied when the decision could have been explicitly marked in the model when applying the pattern?

Identifying pattern instances based on markings could also be more reliable by nature than trying to detect instances with, for example, the mentioned comparison techniques. When patterns are used in design, they are applied to contexts in which it is feasible to use context specific names and to include additional properties. For example, a non-trivial subject (in an Observer [4] instance) should probably have properties (etc.) that the observer would be interested in. With context specific names, properties and surroundings (in the model), the results of comparisons could be less reliable. However, by marking pattern instances explicitly, the information should be as reliable as documentation is in general. In the end, it would be about the reliability of the developer that marks the pattern instances.

It is thus clear that the information on pattern occurrences should be stored (i.e., the pattern occurrences marked) when they are created. This is also the case in the approach of the authors [18]. Patterns, however, could be in general instantiated both manually and in a tool-assisted manner and the initiatives (to instantiate patterns) could come from either a developer or a tool.

C. On Instantiating Patterns

In a simple, conventional case, pattern instances can be assumed to be always created manually. In this case, it is natural to assume the markings (about the pattern instances) to be created manually, too. Otherwise, a tool would need to – somehow - know about a pattern being applied although the task would be performed by a developer. A tool could also include support for marking the pattern instances - without assisting in the pattern application task itself. However, also in this case the responsibility over the (possibly easily forgotten) marking task should be taken by the developer who knows about the pattern being applied.

Assuming that the pattern application process would be assisted by the tool, also the markings could be on the responsibility of the tool because the tool would know about the application. This thinking has also been used in our work [18]. When patterns are created with an interactive wizard, a developer can justifiably expect the tool to handle the markings. However, markings can be edited (and created) also manually. For example, functions to manually edit markings are needed when deleting or editing a pattern instance.

D. On Initiatives to Instantiate Patterns

In order to *actively* suggest a design pattern to be applied, the tool should have the ability to identify both the context and the problem at hand (in the design task) and to notice that they correspond to the context and problem of the pattern. If the active party was the developer, the tool would not necessarily need to have all the abilities. A set of suggested patterns, to be shown as a response to a user activity for example, could be narrowed down from all possible patterns based on the identification of context or problem. Naturally, with less information, not all the suggestions could be appropriate. However, it would still be up to the developer to make the decision.

Detecting a context of a pattern to match that at hand could be done based on a graph or semantic techniques, for example. However, there could still be challenges in formalizing contexts of many existing patterns that have been defined mainly with text. Identifying a problem, *what the developer would like the system to be like*, could be even more difficult to automate, and prone to errors.

If the active party to initiate an activity to apply a pattern would be the developer, also key words and search functions could be used to filter suggested patterns. This would not be possible if the active party would be the tool, so that the initiative would come prior to any user activity, i.e., prior to typing the key words. In addition, with the key words would come the problem of using different words to describe similar aspects. Nevertheless, key words could provide a sufficiently practical solution for suggesting patterns.

When suggesting patterns to use, a tool could also take advantage on information included - not in the patterns themselves - but in the pattern languages and collections that the patterns appear in. For example, when noticing a pattern

 ISBN: 978-1-61208-438-1

to follow a recently used pattern in a pattern language and the problem of the pattern to match the context at hand, the pattern could be (at least) raised in a list of suggested patterns. Similarly, relations in pattern languages that indicate patterns solving the resulting problems of other patterns could be used in an automated manner to facilitate the work of developers.

In our work [18], pattern suggestions currently based on comparing the patterns that are used in models to collections of patterns that have been formed to correspond to the recommendations of standards. In the domain, this is meaningful since the standards govern and restrict the practical solutions that can (or should) be used by developers. However, the patterns are not yet suggested in any specific phase and the initiative to use patterns comes always from the developer. On the other hand, suggestions do not rely on the identification of either context or problem at hand. This could, however, be a possible future research direction.

In the domain, there can be also competence requirements for developers. As such, it can be assumed that appropriate solutions (patterns) are known by developers and that tool support for suggesting patterns would not even be a necessity. Nonetheless, automated functions can be useful in gathering information on the use of the patterns when there is reliable information about their presence available.

E. On Using Pattern Instances

When pattern instances are reliably detected (marked), the information can be collected from models for analysis purposes or to present it in a tabular, compact form. Especially this can be used to support traceability between solutions and their use, as demonstrated in [18]. Traceability is also a good example property in the (safety) domain because it is a property of systematic integrity and required from safety system development. As discussed in Section 3, the development process of software safety systems and applications consists of phases during which developers should apply appropriate techniques and measures that are to ensure the quality of the applications. Documentation is, though, needed to indicate how and where the techniques and measures have been used.

With pattern marks, it is also possible to automate different kinds of consistency checks, in addition to supporting traceability. For example, it can be made sure that patterns are appropriate for the safety levels required from the safety function or application. Naturally, this requires information on the applicability of the solutions to different levels of safety.

VI. DISCUSSION AND CONCLUSIONS

This paper has discussed the role of design patterns in facilitating the development of software safety systems and applications. Design patterns, which are essentially triplets of contexts, problems and solutions, are a means to systematically re-use design and proven solutions to recurring problems and needs. Their systematic use in the safety system development, however, has not been researched extensively although the re-use of recommended solutions is a general virtue in the domain.

Reasons why design patterns could, in general, benefit safety system development are various. Patterns document proven solutions, which provide designer support on selecting the solution to be used in the safety system under design. Known usages and ideally known usages from inspected and approved systems build this support. Patterns can illustrate practical approaches and solutions to alleviate the requirements considering safety system development given in standards, etc. This eases the burden of the designer by bridging the gap between standards and safety system design and implementation. In relation to this, patterns can be used as a part of documentation.

To provide designers with the patterns to be used in safety system design and development, we have mined and documented a set design patterns and pattern prototypes. The patterns consider various aspects of the safety system design including the development process, architecture, co-existence with basic control systems and scope minimization aspects. The work considering the pattern collection is in progress and current effort is to extend the collection to software based safety functions. New known usages for the existing patterns and pattern candidates are also being collected.

The development of safety systems is a systematic process that is governed by standards. Phases of the process build on information produced in the previous phases so that, for example, safety function requirements are specified to treat previously identified hazards and their associated risks. In the implementation phases of the process, developers are required to apply solutions, techniques and measures that are recommended by the standards and can be assumed to result in sufficient quality. However, in safety system development, it is not enough to apply the required techniques and solutions. Developers need to be able to prove the compliance of the applications to standards. This is where appropriate documentation - including information on the usage of the solutions - is needed.

Clearly, certifiable software parts of safety systems are not built by coincidences but by designing them systematically, with the use of appropriate solutions and techniques. As such, the applications need to be specified prior to their implementation, which usually includes at least their partial modeling. Unfortunately, the support for patterns is in UML, the de-facto software modeling language, restricted at best.

When developing pattern modeling approaches, however, patterns should be specified with dedicated modeling concepts and pattern instances marked in the models. In this way, reliable information on patterns could be used for documentation purposes and to automate consistency checks. In the future, tool support could be developed also for assisting developers in selecting patterns to use. However, this task should perhaps consider not only information included in the patterns themselves but also the information included in pattern languages and collections of patterns. Such collections could then be developed with the requirements of safety standards in mind.

 ISBN: 978-1-61208-438-1

REFERENCES

[1] K. Beck, et al., "Industrial experience with design patterns," in Proceedings of the 18th International Conference on Software Engineering, 1996, pp. 103-114.

[2] C. Alexander, S. Ishikawa, and M. Silverstein, Pattern languages. Center for Environmental Structure, vol. 2, 1977.

[3] C. Alexander, The timeless way of building. Oxford University Press, 1979.

[4] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, 1994.

[5] A. Armoush, Design Patterns for Safety-Critical Embedded Systems. Ph.D. thesis, Aachen University, 2010. Available http://darwin.bth.rwth-aachen.de/opus3/volltexte/2010/3273/pdf/3273.pdf [referenced 25.6.2015].

[6] V. Eloranta, J. Koskinen, M. Leppänen, and V. Reijonen, Designing Distributed Control Systems: A Pattern Language Approach. Wiley Publishing, 2014.

[7] IEC, 61508: functional safety of electrical/electronic/programmable electronic safety-related systems. International Electrotechnical Commission, 2010.

[8] ISO, 13849-1:2006 Safety of machinery - Safety-related parts of control systems - Part 1: General principles for design. International Organization for Standardization, 2006.

[9] B. P. Douglass, Real-Time Design Patterns: Robust Scalable Architecture for Real-Time Systems. Addison-Wesley, 2003.

[10] R. Hanmer, Patterns for Fault Tolerant Software. John Wiley & Sons, 2013.

[11] F. Buschmann, K. Henney, and D. Schimdt, Pattern-Oriented Software Architecture: On Patterns and Pattern Language. John Wiley & Sons, 2007.

[12] R. B. France, D. Kim, S. Ghosh, and E. Song, "A UML-based pattern specification technique", Software Engineering, IEEE Transactions On, vol. 30, 2004, pp. 193-206.

[13] P. Kajsa and L. Majtás, "Design patterns instantiation based on semantics and model transformations", in SOFSEM 2010: Theory and Practice of Computer Science, Springer, 2010, pp. 540-551.

[14] R. France, S. Chosh, E. Song and, D. Kim, "A metamodeling approach to pattern-based model refactoring," IEEE Software, vol. 20, 2003, pp. 52-58.

[15] A. Pande, M. Gupta, and A. K. Tripathi, "A new approach for detecting design patterns by graph decomposition and graph isomorphism," in Contemporary Computing, Springer, 2010, pp. 108-119.

[16] N. Tsantalis, A. Chatzigeorgiou, G. Stephanides, and S. T. Halkidis, "Design pattern detection using similarity scoring," Software Engineering, IEEE Transactions on, vol. 32, 2006, pp. 896-909.

[17] D. Jing, Y. Sheng, and Z. Kang, "Visualizing design patterns in their applications and compositions", Software Engineering, IEEE Transactions on, vol. 33, 2007, pp. 433-453.

[18] T. Vepsäläinen and S. Kuikka, "Safety patterns in model-driven development," The 9th International Conference on Software Engineering Advances (ICSEA 2014), Nice, France, 2014, pp. 233-239. ISBN: 978-1-61208-367-4.

[19] IEC, 62061: Safety of machinery - Functional safety of safety-related electrical, electronic and programmable electronic control systems. International Electrotechnical Commission, 2005.

[20] J. Rauhamäki, T. Vepsäläinen, and S. Kuikka, "Patterns in safety system development", The Third International Conference on Performance, Safety and Robustness in Complex Systems and Applications (PESARO 2013), 2013, pp. 9-15.

[21] B. Appleton, "Patterns and software: Essential concepts and terminology", Object Magazine Online, vol. 3, no. 5, 1997, pp. 20-25.

[22] C. Kohls and S. Panke, "Is that true...?: thoughts on the epistemology of patterns". In Proceedings of the 16th Conference on Pattern Languages of Programs (PLoP '09). ACM, New York, NY, USA, Article 9, 2009, 14 pages. http://doi.acm.org/10.1145/1943226.1943237.

[23] J. Rauhamäki and S. Kuikka, Strategies for hazard management process. The 19th European Conference on Pattern Languages of Programs (EuroPLoP 2014), 9.-13.7.2014, Irsee, Germany, ACM New York, NY, USA 2014. Article 31. DOI: 10.1145/2721956.2721966. ISBN: 978-1-4503-3416-7.

[24] J. Rauhamäki and S. Kuikka, Patterns for Sharing Safety System Operation Responsibilities between Humans and Machines. The VikingPLoP 2014 Conference, 10.-13.4.2014, Vihula, Estonia, 2014. ACM New York, NY, USA, 2014, pp. 68-74.

[25] J. Rauhamäki and S. Kuikka, Patterns for control system safety. The 18th European Conference on Pattern Languages of Program, EuroPLoP 2013, Irsee, Germany, July 10-14, 2013. ACM, 2013, Article 23. DOI: 10.1145/2739011.2739034, ISBN 978-1-4503-3465-5.

[26] J. Rauhamäki, T. Vepsäläinen, and S. Kuikka, Patterns for safety and control system cooperation. In: Eloranta, V.-P., Koskinen, J. & Leppänen, M. (eds.). Proceedings of VikingPLoP 2013 Conference, Ikaalinen, Finland 21.3. - 24.3.2013.Tampere University of Technology. Department of Pervasive Computing. Report 2, 2013, pp. 96-108.

[27] J. Rauhamäki, T. Vepsäläinen, and S. Kuikka, Functional safety system patterns. In: Eloranta V.-P., Koskinen, J., Leppänen M. (eds.). Proceedings of VikingPloP 2012 Conference, 17.-20.3.2012. Tampere University of Technology. Department of Software Systems. Report. Nordic Conference of Pattern Languages of Programs vol. 22, Tampere, Tampere University of Technology. 2012, pp. 48-68. Available: http://URN.fi/URN:ISBN:978-952-15-2944-3.

[28] OMG, Unified Modeling Language Specification 2.4.1: SuperStructure. Object Management Group, 2011.

 ISBN: 978-1-61208-438-1