TOWARDS PATTERNS FOR HANDLING SAFETY CRITICAL ADAPTIVE CONTROL SOFTWARE

André A. Hauge
Department of ICT Risk and Dependability, Institute for Energy Technology, Halden, Norway
Department of Informatics, University of Oslo, Oslo, Norway

Ketil Stølen
Department of Networked Systems and Services, Sintef ICT, Oslo, Norway
Department of Informatics, University of Oslo, Oslo, Norway

Keywords: Pattern format, Adaptive software, Safety.

Abstract: This article puts forward a pattern format for use in the safety critical control domains where adaptable components are part of the control software. The pattern format may be seen as a first step towards establishing a pattern language uniting three interests. The first interest is related to the objective of providing the comprehensibility and usability found in design patterns with respect to communication of solutions to problems that may be solved by means of adaptive control. The second interest is related to the need to make explicit the requirements to be satisfied in order to facilitate instantiation of a design in different safety critical contexts. The third interest is related to the need to provide argumentation for risk being satisfactory reduced. The pattern format supports not only documentation of a technical solution to a recurring problem, but also documentation of the requirements that must be satisfied when instantiating a design in different contexts as well as solutions for how the safety property may be demonstrated.

1 INTRODUCTION

From the perspective of a software safety engineer, evaluating the suitability of a particular adaptive software design or finding the most suitable of a set of relevant designs for use in a specific context is a challenge in itself. Our approach is intended to provide the safety engineer with a pattern language which offers design solutions as well as indication on the requirements that may be associated with the design and solutions to how one may demonstrate safety.

Figure 1 illustrates the relationship between the different constituents of our pattern language for Safe Adaptive Control Software (or SACS for short). We employ the Alexandrian (Alexander et al., 1977) interpretation of what is meant by a pattern language. This article mainly addresses the format and discusses how the format supports establishing the SACS pattern language.

Section 2 provides an overview of the main challenges to be addressed by our pattern language. Section 3 presents the pattern format. Section 4 exemplify how patterns specified according to the format may be combined to form a usable pattern language.

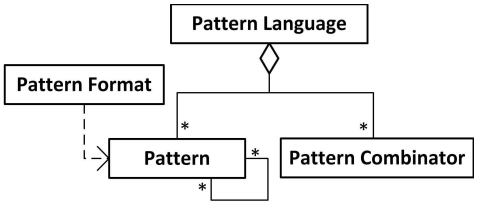

Figure 1: The levels of the language.

2 THE CHALLENGE

Adaptive control software is able to evolve while executing, thereby introducing an additional uncertainty aspect with respect to functional behaviour compared to conventional software systems. The uncertainty is associated with the variability of the system and the variability of the service as an effect of the adaptations that may be experienced at runtime. The variability of

A. Hauge A. and Stølen K..
TOWARDS PATTERNS FOR HANDLING SAFETY CRITICAL ADAPTIVE CONTROL SOFTWARE.
DOI: 10.5220/0003538502110214
In *Proceedings of the 8th International Conference on Informatics in Control, Automation and Robotics* (ICINCO-2011), pages 211-214
ISBN: 978-989-8425-74-4

the systems and the service is governed by the adaptation loop mechanism (Salehie and Tahvildari, 2009) consisting of processes for monitoring, detecting, deciding and acting upon change. From a safety perspective, the challenge is to demonstrate that the system is still safe after each adaptation iteration.

A successful pattern language for use by a safety engineer should be able to communicate: (i) design solutions; (ii) requirements that must be satisfied in order to give confidence that a system may achieve its intended utility and safety integrity; and (iii) how to demonstrate that a system based on a design pattern is sufficiently safe for a given purpose. In order to facilitate such a pattern language, we summarise the following main success criteria for our pattern format:

1) support the specification of patterns in such a manner that a pattern language for safety critical/related adaptive control software may be provided
2) support the specification of patterns for each of the concerns requirements, design and safety case
3) support specification of patterns which build upon each other in such a manner that a user may be provided with choices and detail by the linked patterns

3 MAIN INGREDIENTS OF THE PATTERN FORMAT

Figure 2 gives the basic format of the three types of patterns. In the following we describe these in further detail.

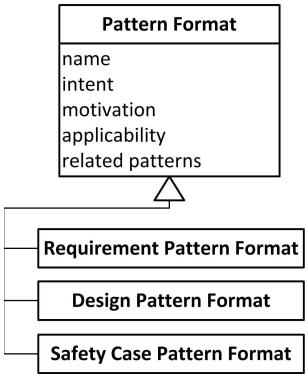

Figure 2: Overall Structure for Pattern Format.

3.1 Requirement Pattern Format

The requirement pattern format as illustrated in Figure 3 is inspired by the problem frames approach by

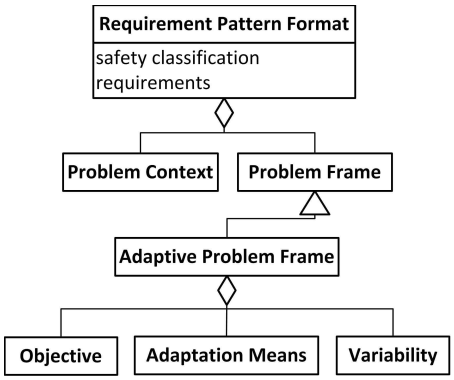

Figure 3: Requirement Pattern Format.

(Jackson, 2001). A requirement pattern in our pattern language address a specified set of problems, or problem domains, which are documented as part of the *problem context*. The *problem frame* part of a pattern is intended to be used to elaborate upon the phenomenons associated with the different types of problem domains identified in the problem context to such an extent that the problem is sufficiently understood. The *requirements* part should specify the sets of requirements which are derived on the basis of the problem frames specifications. A challenge is to capture uncertainty or allowable variability of the service and the system as an effect of adaptations, this is discussed in (Qureshi and Perini, 2009) and (Whittle et al., 2009). An *Adaptive Problem Frame* may be used to elaborate upon the challenges with respect adaptiveness. This type of problem frame requires that the *objective* for introducing adaptivity, *adaptation means* detailing how adaptability may be obtained, and the challenges associated with *variability* of the system and the system service are addressed. Requirements that may be derived the problem frames specifications are detailed in the *requirements* field.

3.2 Design Pattern Format

Our format for expressing design patterns as shown in Figure 4 is inspired by the the format provided by (Gamma et al., 1995), an extension is provided by requiring the explicit description of *safety features*.

Although our language is intended to address adaptive control software, each pattern which make up a part of the language does not necessarily address adaptivity. A design pattern thus might conform to the basic *Design Pattern Format* or the extended *Adaptable Design Pattern Format* as illustrated in Figure 4. An adaptable design pattern should clearly describe the characteristic properties which provide *system variability* and *service variability* as an effect of runtime adaptations.

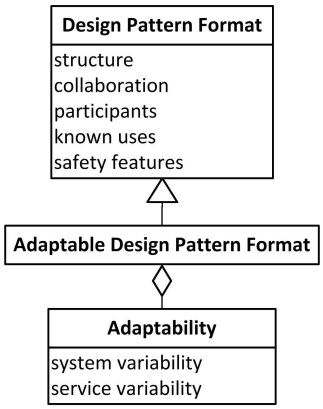

Figure 4: Design Pattern Format.

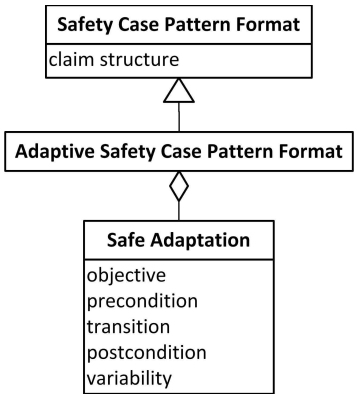

Figure 5: Safety Case Pattern Format.

3.3 Safety Case Pattern Format

A safety case may be structured in different styles (Alexander et al., 2008) and by different means, e.g. graphical approaches like GSN or CAE (Alexander et al., 2008), (Bishop et al., 2004). The intent however is to structure claims, arguments and evidence in such a manner that it may be logically deduced and concluded that the system is sufficiently safe.

For any safety critical or safety related system, the core driver for providing confidence that the system is sufficiently safe is basically to demonstrate two concerns: (i) that the safety requirements specification is sufficient and correct and (ii) that the system satisfies the safety requirements. A safety case pattern should provide the main claims, arguments and indicate the evidence required such that the concerns of (i) and (ii) may be demonstrated.

Adaptive software differs from non-adaptive software in that the properties of the software are expected to change beyond commissioning time. In order to demonstrate that an adaptive system is safe, we must demonstrate that an iteration of the adaptation loop always yields a refinement which is safe within its operational lifetime.

Figure 5 provides the basic format of our pattern for demonstrating the safety of adaptive software. A pattern specified according to the *Adaptive Safety Case Pattern Format* is required to provide a *claim structure* documentation which in general is intended to be used to document the argument in a structured manner. With respect to demonstration of adaptive software safety, the claim structure should be documented in such a manner that it is a logical consequence of the premises provided by these argument parts that adaptation will have no negative effect on safety. There are five important parts to such an argument which are required to be explicitly documented in our pattern format under the field *Safe Adaptation*.

The *objective* should reflect the goals to be achieved by adaptation. The *precondition* is intended to capture the premises of the argument related to the conditions for executing the adaptation loop. Related to the *transition* field, there should be provided a description of the means which assure adapting a system satisfying a precondition will always yield a system satisfying a postcondition. The *postcondition* is intended to capture set of premises which is satisfied when the system is adapted and which guarantees that the system is safe. The *variability* part of the pattern is intended to capture the implied variability of the service and the system which may be expected as an effect of accommodating changes by adapting. It should also be described why this variability will have no negative effect on safety.

4 EXAMPLE

Figure 6 illustrates a map of related patterns as seen from the perspective of an example design pattern named *Trusted Backup*, thus only one generation of links is provided. A map of the complete pattern language may be provided by combining the information contained in the *related patterns* field of individual patterns.

Pattern combinators describe the rules for how patterns may be combined to form a pattern language. Figure 6 exemplify the use of a set of combinators where the following semantics may be assumed:

includes: specifies that a *pattern* may be partially defined by another pattern of the same type. This is used to define that patterns may build upon each other

requires: specifies that a *Design Pattern* should satisfy the requirements in a *Requirement Pattern*

enables: specifies that a specific *Design Pattern*

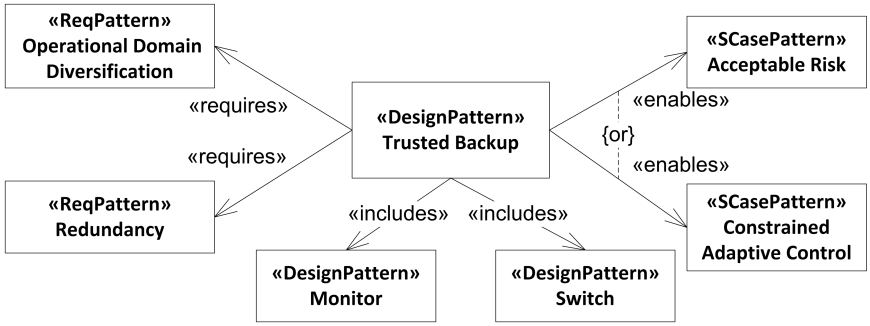

Figure 6: Example.

enables the use of a specific *Safety Case Pattern* in order to argue about the safety property

Where there is a one-to-many relationships between patterns, the following rules may be used in order to detail the relationship:

or: define that the targets represent alternatives where only one is required for completeness

and: define that all the target patterns are required for completeness. This is implied in Figure 6 if not otherwise specified

We assume here that the *Trusted Backup* design pattern describes a configuration of an adaptable controller, a conventional controller, a monitor and a switch component. Both controllers operate in parallel and are granted control privileges according to a switching scheme. A monitor supervises the adaptable controller in order to detect anomalies to its behaviour. The switch grants control to the adaptable controller given that no anomalies are detected by the monitor and the system under control is in a state for which the adaptable controller may not cause harm, otherwise the switch grant control to the conventional controller.

The *Trusted Backup* design specifies the use of on a redundant set of controllers and means for delegation of control according to some type of diversification of the operational state space. The *requires* relationships specify the requirement patterns which must be instantiated and satisfied when instantiating the design pattern, here this is the *Operational Domain Diversification* and *Redundancy* requirement patterns.

The *Trusted Backup* design pattern may include several other design patterns as part of its specification. The *include* relationships of Figure 6 indicate that the pattern include functionality for monitoring and switching and that these are handled in dedicated patterns named *Monitor* and *Switch*.

Depending on the intent for introducing adaptability and other characteristics of the system design, strategies for demonstrating safety may or may not be applicable. If adaptivity is introduced in a design to improve performance, we cannot make an argument of improved safety. In Figure 6 there is illustrated two applicable Safety Case Patterns named *Acceptable Risk* and *Constrained Adaptive Control* where it is sufficient to apply only one for demonstrating safety.

REFERENCES

Alexander, C., Ishikawa, S., and Silverstein, M. (1977). *A Pattern Language: Towns, Buildings, Construction*. Oxford University Press.

Alexander, R., Kelly, T., and McDermid, J. (2008). Safety cases for advanced control software: Safety case patterns. Technical Report FA8655-07-1-3025, Department of Computer Science, Univeristy of York.

Bishop, P., Bloomfield, R., and Guerra, S. (2004). The future of goal-based assurance cases. In *Proceedings of Workshop on Assurance Cases. Supplemental Volume of the 2004 International Conference on Dependable Systems and Networks*, pages 390–395.

Gamma, E., Helm, R., Johnson, R. E., and Vlissides, J. (1995). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley.

Jackson, M. (2001). *Problem Frames: Analysing and Structuring Software Development Problems*. Addison-Wesley.

Qureshi, N. A. and Perini, A. (2009). Engineering adaptive requirements. *International Workshop on Software Engineering for Adaptive and Self-Managing Systems*, pages 126–131.

Salehie, M. and Tahvildari, L. (2009). Self-adaptive software: Landscape and research challenges. *ACM Trans. Auton. Adapt. Syst.*, 4(2):1–42.

Whittle, J., Sawyer, P., Bencomo, N., Cheng, B. H. C., and michel Bruel, J. (2009). Relax: Incorporating uncertainty into the specification of self-adaptive systems. In *17th IEEE International Requirements Engineering Conference RE 2009*, pages 79–88.