SaCS: A Method and a Pattern Language for the Development of Conceptual Safety Designs

Doctoral Dissertation by

André Alexandersen Hauge

Submitted to the Faculty of Mathematics and Natural Sciences at the University of Oslo in partial fulfilment of the requirements for the degree Philosophiae Doctor (Ph.D.) in Computer Science

June 2014

Series of dissertations submitted to the
Faculty of Mathematics and Natural Sciences, University of Oslo
No. 1568

ISSN 1501-7710

Cover: Hanne Baadsgaard Utigard.
Printed in Norway: AIT Oslo AS.

Produced in co-operation with Akademika Publishing.
The thesis is produced by Akademika Publishing merely in connection with the thesis defence. Kindly direct all inquiries regarding the thesis to the copyright holder or the unit which grants the doctorate.

Abstract

Flight control systems, railway interlocking systems, and nuclear reactor protection systems are examples of safety critical systems from different industrial domains. A safety critical system within any of these domains requires some type of acceptance from a safety authority prior to commissioning. The minimum prerequisite for achieving acceptance is to comply with relevant normative requirements from regulations and standards. Safety standards and guidelines typically define the safety objectives to be met by a system and by the process of developing the system.

In this thesis we present a method and a pattern language called Safe Control Systems (SaCS) for development of conceptual safety designs. By a conceptual safety design we mean an early stage specification of system requirements, system design, and safety case for a safety critical system. The SaCS method consists of: (1) The SaCS process – a process for the systematic application of patterns as development support. (2) The library of SaCS patterns – a collection of patterns providing guidance on effective solutions to different challenges relevant when developing conceptual safety designs. The library is structured into patterns for requirements capture, system design and safety assurance in the form of a safety case. (3) The SaCS pattern language – a language for defining patterns and for specifying the application of patterns for safety design conceptualisation. The three artefacts are complementary and their integration represents a combined approach to pattern-based development.

The patterns in the library represent safety engineering best practices inspired by international safety standards and guidelines. Applying patterns according to the SaCS process supports establishing the evidence that the conceptualisation of systems is being performed according to a suitable process and according to accepted practices. The pattern language supports the specification of patterns and the documentation of their use.

The SaCS method has been evaluated in three different studies: (1) Study 1 – a case study on safety design conceptualisation of a nuclear power plant control system; (2) Study 2 – a case study on safety design conceptualisation of a railway interlocking system; (3) Study 3 – an analytic evaluation of the suitability of the SaCS pattern language for its intended task.

The experiences and results from the different evaluations indicate that the SaCS method facilitates the development of conceptual safety designs by systematically combining and applying patterns as development support.

Acknowledgements

This work has been conducted and funded within the OECD Halden Reactor Project (HRP), an international research project managed by Institute for energy technology (IFE). I am grateful to IFE, my employer, for supporting me in conducting the PhD research.

I want to thank my main supervisor Ketil Stølen for always providing insightful and to the point review of presented work. I owe Ketil my deepest gratitude for the many hours spent in discussions and commenting on the work I have presented. It has been an enriching experience and a privilege to be a doctoral student of Ketil Stølen.

I also wish to thank my second supervisor, Bjørn Axel Gran, for supporting me in pursuing a PhD in the first place and for the read through and commenting on my thesis.

I would like to thank the guest scientist in the DIGIT and EMERGENCY projects at SINTEF that also have commented on my work at different research meetings, these are: Fabio Massacci, Christian Damsgaard Jensen, Audun Jøssang, late William Winsborough, Jonas Landgren, and Isabelle Simplot-Ryl.

I have on many occasions presented my work at SINTEF to Aida Omerovic, Birger Møller-Pedersen, Olav Skjelkvåle Ligaarden, Tormod Vaksvik Håvaldsrud, Erik Gøsta Nilsson, Amela Karahasanovic, Bjørnar Solhaug, and Gyrd Brændeland. Thanks for the many interesting discussions and for the valuable feedback.

I would like to thank everyone at IFE for their support and for providing a very good working environment. I want to thank all my colleagues at the Department of Software Engineering that has provided their feedback at research meetings, on articles and reports. A special thanks goes to Terje Sivertsen for the thorough review and the helpful comments on different works. Lots of thanks to Silvia Henriksdóttir, Sizarta Sarshar, Christian Raspotnig, Vikash Katta, Rune Fredriksen, John Eidar Simensen, Jan Erik Farbrot, Terje Johnsen, Michael N. Louka, and Alf Ove Braseth for the discussions and the feedback on different topics of my work.

I would like to thank my friends and family for all support and their kind words of encouragement. Most of all I would like to thank my wife Eva-Marie and my two daughters Thea Adelin and Jamilla. I love you more than anything, without you there is nothing.

List of original publications

1. André Alexandersen Hauge and Ketil Stølen. **Syntax & Semantics of the SaCS Pattern Language**, Technical report HWR-1052, OECD Halden Reactor Project, Institute for energy technology, Halden, Norway, 2013. The report represents an extension and consolidation of the paper published in *Proceedings of the 18th Conference on Pattern Languages of Programs (PLoP'11)*, ACM, 2011.

2. André Alexandersen Hauge and Ketil Stølen. **A Pattern-based Method for Safe Control Conceptualisation Exemplified Within Nuclear Power Production.** In *Proceedings of the 31st International Conference on Computer Safety, Reliability and Security (SAFECOMP'12)*, pp. 13-24, LNCS 7612, Springer, 2012.

   We have included the full technical report, HWR-1029 rev 2, OECD Halden Reactor Project, Institute for energy technology, Halden, Norway, 2014.

3. André Alexandersen Hauge and Ketil Stølen. **Developing Safe Control Systems using Patterns for Assurance.** In *Proceedings of the 3rd International Conference on Performance, Safety and Robustness in Complex Systems and Applications (PESARO'13)*, pp. 1-8, IARIA, 2013.

   We have included the full technical report, HWR-1037 rev 2, OECD Halden Reactor Project, Institute for energy technology, Halden, Norway, 2014.

4. André Alexandersen Hauge and Ketil Stølen. **An Analytic Evaluation of the SaCS Pattern Language – Including Explanations of Major Design Choices.** In *Proceedings of the 6th International Conference on Pervasive Patterns and Applications (PATTERNS'14)*, pp. 79-88, IARIA, 2014.

The publications 1-4 are available as Chapters 9-12 in Part II of the thesis.

Contents

List of Figures

List of Tables

Part I

Overview

Chapter 1

Introduction

A safety critical system must not only *be* safe, it must also be *demonstrated* safe, and typically be *accepted* as being sufficiently safe for its intended purpose by a relevant safety authority prior to commissioning [160]. By safe, we do not mean in an absolute sense. No system is free from risk and thus absolutely safe [113], not even those systems that are developed according to the highest standards.

The safety property of a programmable system is affected by the different development practices applied during its life-cycle [28,93]. The principles for determining what is sufficiently safe and what engineering practices shall be applied in order to develop systems that are inherently safe and that may be accepted as suitable differ among experts. International safety standards are developed by means of a consensus among domain and safety experts and reflect the current best practices for development of safety critical systems within an industrial domain [30, 86, 93] that may understood as accepted. Even if a system has been developed in compliance with relevant standards, this does not imply that the intended safety level of the system developed has been achieved. Catastrophic accidents still occur as in the case of the 2011 Fukushima Daiichi [45] nuclear disaster or the 2011 collision of two high-speed trains close to Wenzhou in China [112]. Compliance to a standard only implies that a system has been developed according to the principles that are accepted as yielding safe systems within a domain.

Acceptance for the commissioning of a safety critical system is based on a safety authority having achieved confidence in the the system being sufficiently safe for its intended purpose. In 2009, the Norwegian National Rail Administration discontinued its project to deploy a computer-based interlocking system named Merkur, designed by ABB, after spending 420 million NOK (Norwegian Krone) on the project and not achieving safety acceptance [139]. The Norwegian Railway Inspectorate rejected the request to put the Merkur system into operation due to substantial weaknesses in the elementary documentation [139, 155], including the requirement specification and the safety demonstration documentation. Thus, failure to produce basic documentation correctly can be costly, as in the case of the Merkur system.

A system failure can potentially endanger human life or the environment and render a system unsafe. A system failure is the effect of one or more errors propagating to the system level [14]. An error is a manifestation of a latent fault [14]. Feiler [53] indicates that 70% of the faults detected during development of software-based systems are introduced in the early phases of the development life-cycle, during the phases for specifying requirements, architecture and design. In general, a fault or an error

becomes increasingly more costly to mitigate the later in the development process it is detected [12,21,53]. Risks associated with the potential erroneous operation of a system may be effectively reduced in the early phases of development by avoiding the potential causes of failure, the faults, to be introduced and by incorporating fault tolerance in the system design to mitigate the effects of a failure [14].

Developing systems in compliance with the requirements and recommendation given in standards is costly. Amey [7] reports that developing software compliant to Level A of DO-178B/ED-12B [47] is five times as costly as developing non-critical software. Wong et al. [175] compares software safety standards from different domains with respect to cost-effectiveness and identifies that there is no consensus across domains on how to achieve safety in a cost-effective fashion. The authors also reviewed several large development projects with respect to the cost-effectiveness of the practices applied in order meet safety objectives. They conclude that safety objectives and cost-effectiveness objectives are best met by applying company-specific safety engineering best practices throughout the development life cycle.

Regulations [50] and standards [28–30] commonly give detailed requirements on the objectives to be fulfilled during system development, and to a lesser degree provide guidance on how to apply methods and techniques in order to fulfil objectives. Guidelines [52, 144] on the other hand may bridge this gap and offer guidance on the application of different methods and techniques in order to satisfy objectives. A challenge with providing guidance on different topics within the format of a traditional guideline is completeness; it is not possible to cover all topics. Guidelines are typically documented in the form of a report. They are therefore expanded with new knowledge through revisions.

Based on the challenges introduced above we see the need for a new type of development guidance, a guidance that describes safety engineering best practices and effective solutions in a manner that may be efficiently applied. The guidance should support the early phases of system development and provide assurance that safety objectives are met. The guidance should also be easy to expand with new knowledge as this is gained. In addition, it should be possible to tailor the guidance to domain-specific and company-specific needs. We envision a body of knowledge represented as guidance on safety engineering best practices and solutions on every aspect of system development in a library such that guidance on individual topics can be efficiently combined, applied, and tailored to accommodate specific needs.

The notion of patterns [5,26,55] has been used for decades as a means to describe the essence of a recurring problem and to define the very core of a solution to that problem. Pattern collections and languages define a body of knowledge on effective solutions within a particular domain. Patterns generalise known solutions to commonly recurring problems and are usually presented in an format that is easy to understand and apply. The use of patterns for communicating engineering best practices and design concepts is widespread. The pattern format is flexible in the sense that pattern collections and languages are found in a wide range of different domains and for different purposes. A pattern definition is usually scoped to define one concept only. Thus, the pattern format supports a separation of concerns. A number of concerns may be addressed by a collection of complementing patterns. The problem of combining patterns in order to solve an overall problem is addressed by pattern languages. A pattern language defines the systematic application of several patterns. The many attractive features of patterns and pattern languages as presented above led us to pursue a pattern-based

solution.

In the following sections we state the overall objective of our work and give an overview of the main contributions. Then, the structure of the thesis is presented.

1.1 Objective

As stated in the introduction, the notion of patterns [5,26,55] has been used for decades as a means to describe the essence of a recurring problem and the very core of a solution to that problem. Pattern languages [5] can serve as a tool for expressing a combination of patterns. Each individual pattern specified in the language describes a solution to an isolated problem and the combination of results from the application of several patterns solves the overall problem.

The literature on patterns defines solutions to problems on a variety of topics relevant to safety critical systems development, e.g., fault tolerant software design [68], or safety argumentation [102]. In the pattern literature there is a lack of effective approaches for precisely specifying the combined use of several patterns for addressing a development challenge [79]. In order for a pattern-based development approach to be useful within development of safety critical systems, it should be possible to integrate patterns on product-oriented development challenges and solutions, e.g., patterns for hardware design and software design, with patterns on process-oriented challenges and solutions, e.g., patterns for hazard identification or risk assessment. It is not necessarily sufficient to present the system design alone in order to provide confidence that the system in question is suitable for its intended use. Safety is a property of a system achieved through a systematic development process where the system design is one of several results. The system design is a result from the design phase and is very important as it defines the components, interfaces and other main characteristics of a system. With the intention of addressing the early stages of development, the focus of this thesis is to support the stages leading to the definition of conceptual safety designs by providing guidance to the definition of system solutions that are suitable for their intended use. Guidance in the form of patterns represents the core of our approach.

The overall objective of this thesis is to develop a method and a pattern language that is:

1. Well-suited for developing conceptual safety designs.
2. Applicable within an industrial context within acceptable effort.
3. Comprehensible for its intended users.

The intended target users are system engineers, safety engineers, hardware and software engineers.

1.2 Contribution

The main contributions of this thesis is the Safe Control Systems (SaCS) method and its supporting pattern language that has been developed to meet the overall objective stated above. The SaCS method facilitates the development of conceptual safety designs for safety critical systems.

The SaCS method consists of the SaCS process, the library of SaCS patterns, and the SaCS pattern language. In the following three subsections we give a brief overview of each of these artefacts.

1.2.1 The SaCS process

The SaCS method offers a process for systematically selecting and instantiating patterns as well as documenting how patterns are applied to support development.

The most important documents established during the development of a safety critical system design are the requirement specification and the design specification. The requirement specification should define the required properties of the system. The design specification describes how the requirements shall be fulfilled technically.

In order to provide confidence that the design of the system under development actually defines a safe system, another important document is the safety case. The purpose of a safety case is to explicitly define why a system under consideration is sufficiently safe for its intended purpose, e.g., by demonstrating that the safety requirements are correct and sufficient and that the safety requirements are satisfied by design. Further, the specification of the safety demonstration should be initiated at the very beginning of system development and systematically maintained during the life-cycle of the system.

The SaCS process defines a sequence of activities for systematic selection and application of patterns. The selection process ensures that the patterns are applied in a suitable order. In addition, activities are defined for documenting the composition and application of patterns.

1.2.2 The library of SaCS patterns

The library of SaCS patterns consists of so-called basic SaCS patterns. Basic patterns are categorised into six different kinds. There are basic patterns for, e.g., eliciting requirements, assessing risk, deriving system design, and establishing the safety argumentation.

Although basic SaCS patterns are of different kinds, they are all described as variations over a common and simple format. Each pattern is uniquely identified, clearly defines the problem that is addressed, and provides a description of a solution to the problem. The patterns in the library provide guidance on a number of topics. Depending on the problem in being addressed, guidance is given in different forms, e.g., a solution to a design challenge is described by an abstract definition of a design, a solution to an assessment challenge may be described as a process.

The effective use of patterns depends on the ability of the user to match problems occurring in the real world with, if available, a pattern that describes a solution to the problem. Secondly, the effective use of patterns depends on the ability of the user to correctly apply the pattern.

The patterns in the library are defined at different levels of abstraction; some are very specific and some generic. Every pattern includes an instantiation rule that characterises guidance on its instantiation. The patterns in the SaCS library are also defined in a manner that allows them to be applied as stand-alone entities. However, the intention is that basic patterns are applied in a specific sequence decided by the

user. In order to select patterns in a proper sequence, the *related patterns* sections of the basic patterns may be consulted.

1.2.3 The SaCS pattern language

A pattern expressed in the SaCS pattern language may be a basic pattern, as represented by the patterns in the library, or a composite pattern. A composite pattern is a structure of patterns where patterns are combined by operators. The SaCS pattern language includes a notation for defining how patterns are combined into composite patterns. The rules for combining patterns forms the basis for defining new patterns as compositions of predefined patterns. The notation of the SaCS pattern language may also be used to define how patterns are applied during development and thus offers the ability to document the application of SaCS.

The notation for defining composite patterns is graphical and serves several purposes. It is graphical to facilitate communication and discussion among users on the proper application of patterns for problem solving. The notation allows the documentation of the application of patterns in a development context. The notation also facilitates the documentation of the reusable pattern compositions that may be applied in different contexts.

A composite pattern defines a structure of patterns where each pattern in the structure is represented abstractly by an icon and a unique identifier. The full definition of a referenced pattern can be found using the identifier as a key. A composite pattern is parametrised in terms of inputs and outputs. Each pattern in a composite is also parametrised. The notation supports defining reusable pattern structures. The patterns in a composite are connected by relations. The relations models the expected data flow and dependencies between patterns when they are applied in a context. Relations operate on the parameters of the patterns that are combined. A specific application of a pattern may be captured by connecting its formal parameters with the actual documents these represent in a given context. A composite pattern can define the instantiation order of patterns and in this sense guide and document the process of developing conceptual safety designs.

A composite pattern can be instantiated according to the instantiation rules associated with each basic pattern it is composed of and according to the rules of composition.

1.3 Structure of the thesis

This thesis is based on a collection of 4 research papers and split into two parts. Part I provides the context and an overall summary of the work, while Part II contains the research papers. Each paper is self-contained and can therefore be read separately.

Part I is organized into the following chapters:

Chapter 1 - Introduction presents the main objective of this thesis, the research domain, the contribution, and the structure of this thesis.

Chapter 2 - Problem characterisation provides background material and argues why our main objective is worth pursuing. In addition, success criteria for our main artefacts are formulated.

Chapter 3 - Research method discusses a method for technology research and how this method has been applied in this thesis.

Chapter 4 - State of the art presents the state of the art literature of relevance for our main artefacts. In Section 7.2 of Chapter 7 we conduct a more detailed discussion of our results with respect to this state of the art. This more detailed discussion has been postponed to Chapter 7 because it depends on the detailed description of our artefacts.

Chapter 5 - Achievements: the overall picture provides an overview of the SaCS method and its main ingredients.

Chapter 6 - Overview of research papers provides publication details of each research paper.

Chapter 7 - Discussion argues to what extent artefacts satisfy the success criteria stated in Chapter 2. In addition, important design decisions are discussed, and it compares our approach to other approaches from the literature.

Chapter 8 - Conclusion presents conclusions and possible areas of future work.

Chapter 2

Problem characterisation

In this chapter, we present some background material and motivate in more detail why the overall objective defined in Section 1.1 is desirable. Furthermore, we decompose and refine the overall objective into a set of success criteria that we would like our artefacts to fulfil.

Section 2.1 clarifies the terminology and main concepts that are used throughout this thesis. Section 2.2 details success criteria for each of the three artefacts.

2.1 Conceptual clarification

To avoid ambiguity, and consequently misinterpretation, we define the different terms that are used throughout the thesis. The terms that we apply or define are emphasised in a **bold** font.

2.1.1 Safety

This thesis addresses the development of systems that are required to be dependable. We address especially the safety property of dependable systems. **Dependability** is by Avižienis et al. [14] defined as *"the ability to deliver service that can justifiably be trusted"*. The authors further define dependability to be be composed of [14]: **availability**: defined as the *"readiness for correct service"*; **reliability**: defined as *"continuity of correct service"*; **safety**: defined as *"absence of catastrophic consequences on the user(s) and the environment"*; **integrity**: defined as *"absence of improper system alterations"*; and **maintainability**: defined as the *"ability to undergo modifications and repairs"*.

Leveson [113] points out that no system is free from risk and thus absolutely safe. Thus, safety is commonly associated with the notion of risk. In the generic safety standard IEC61508 [93], the notion of **risk** is defined as *"combination of the probability of occurrence of harm and the severity of that harm"*. A **harm** [93] is defined as *"physical injury or damage to the health of people or damage to property or the environment"*, a **hazard** is defined as *"potential source of harm"*.

Safety is considered in relation to a specific system. This thesis addresses **programmable electronic systems** [93], defined as systems *"for control, protection or monitoring based on one or more programmable electronic devices, including all elements of the system such as power supplies, sensors and other input devices, data highways and other communication paths, and actuators and other output devices"*.

Some sources differentiate between different kinds of what we broadly term as a safety critical system and operate with the terms safety system and safety related system as in the nuclear standard IEC61226 [91]. The former term is restricted to systems important to safety and that can be classified as a safety system; the latter term denotes applications that are important to safety but not part of a safety system. In IEC61508 [93], the term safety related system is used to denote a system that implements the required safety functions in order to achieve or maintain a safe state.

In order to avoid misunderstandings, we avoid using terms that are defined differently in different domains and use the term safety critical system to denote any system, independent of the nature of the function it provides, that by failing might negatively impact safety. More precisely, we apply the definition given by Knight [106] that defines a **safety critical system** as *"those systems whose failure could result in loss of life, significant property damage, or damage to the environment"*. Thus, we make the distinction between those systems that may negatively affect safety in the event of failure, independent of their function, and those systems that may not affect safety.

The criticality of different types of systems is classified differently within different industrial domains. Within railway [30], it is a classified in terms of required safety integrity level (SIL) of a system function. Within aviation [49], it is classified into an associated development assurance level (DAL), while the nuclear power production domain [91] applies a categorisation where category A denotes safety functions and where category B and C denote safety related functions. Common to the different classification schemes is that all reflect upon the concept of risk, either by classifying a system or a system function with respect to the potential severity of failure or by its importance in assuring safety. The classification provides guidance on how risk should be reduced.

Given that the risks associated with a safety critical system are properly reduced to a tolerable level, we generally accept the residual risk as a trade-off of the utility provided by the system. **Tolerable risk** [93] is defined as *"risk which is accepted in a given context based on the current values of society"*. **Residual risk** [93] is defined as *"risk remaining after protective measures have been taken"*.

A system provides its service as long as no failures are experienced. A **failure** is defined by Avižienis et al. [14] as *"an event that occurs when the delivered service deviates from correct service"*. The authors term the *"deviation"* (from correct service) an **error** and the *"adjudged or hypothesized cause of an error"* as a **fault**.

2.1.2 Pattern

Patterns are defined for different purposes and within diverse application domains, e.g., for supporting software design [55], or buildings design [5]. According to Buschmann, Henney, and Schmidt [25] a pattern for software architecture *"describes a particular recurring design problem that arises in specific design contexts, and presents a well-proven generic scheme for its solution. The solution scheme is specified by describing its constituent components, their responsibilities and relationships, and the way in which they collaborate"*.

According to Alexander et al. [5] a pattern for buildings architecture *"describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing the same twice"*.

In this thesis, we use the format of a pattern for describing solutions to recurring problems on a number of different issues, e.g., system design, requirements elicitation, development processes, assessment methods, and safety case argumentation. The main purpose of a pattern is to describe the core elements in a recurring problem and, respectively, the core elements of its solution in a comprehensible and easy to use format. The definition of design pattern as given in ISO 24765 [95] captures our interpretation pattern in the broader sense, thus, we apply the following definition stating that a **pattern** is *"a description of the problem and the essence of its solution to enable the solution to be reused in different settings"*.

2.1.3 Pattern language

A **pattern language** is a language for specifying patterns making use of patterns from a vocabulary of existing patterns and defined rules for combining these.

In the pattern language of Alexander et al. [5] for buildings construction, patterns are organised and applied in a sequence based on defined connections between already existing patterns. Patterns presented early in the sequence are at a high level of abstraction and are complemented by the patterns presented later in the sequence. The organisation of the patterns and the connections defined between these are essential.

2.1.4 Safety assurance

Within the context of developing safety critical systems, safety assurance refers to the process of providing confidence that a system is sufficiently safe for its intended purpose. In the standards literature there are fundamental differences in how safety assurance is achieved [6, 103, 119]. One difference is between the so-called *process assurance* based approach and the *product assurance* based approach [6]. Safety standards with a strong process focus, e.g., the CENELEC standards [28–30], emphasise safety assurance by applying prescribed development processes and tasks. Safety standards with a strong product focus, e.g., the Defence Standard 00-56 [123], emphasise safety assurance on the basis of well-structured and reasoned safety arguments targeting the product specifically [6].

Safety assurance from a process assurance perspective may involve the application of a particular development process (e.g., the process indicated in Fig. 2.1) as well as the application of specific methods, techniques (e.g., selected techniques from Table 2.1) and tools. Furthermore, safety assurance is achieved on the basis of justifiable evidence for a system under consideration being suitable and developed according to acceptable safety engineering principles. The evidence accumulated during the system life-cycle provides the required confidence that safety is assured.

The EN50126 standard [28] addresses the specification and safety demonstration of railway signalling systems and defines a process, based on the system life-cycle, to achieve safe systems. Safety assurance of railway applications within the European Union involves showing compliance to the development process and the recommendations defined within EN50126 [28] and associated standards. Fig. 2.1 describes the typical life-cycle of a railway system as it is defined in EN50126 [28]. The life-cycle depicted in Fig. 2.1 is comparable to the life-cycle described in the generic standard IEC61508 [93] that is used as a foundation for standards within different domains. The life-cycle is defined as a sequence of phases where each phase defines tasks that shall

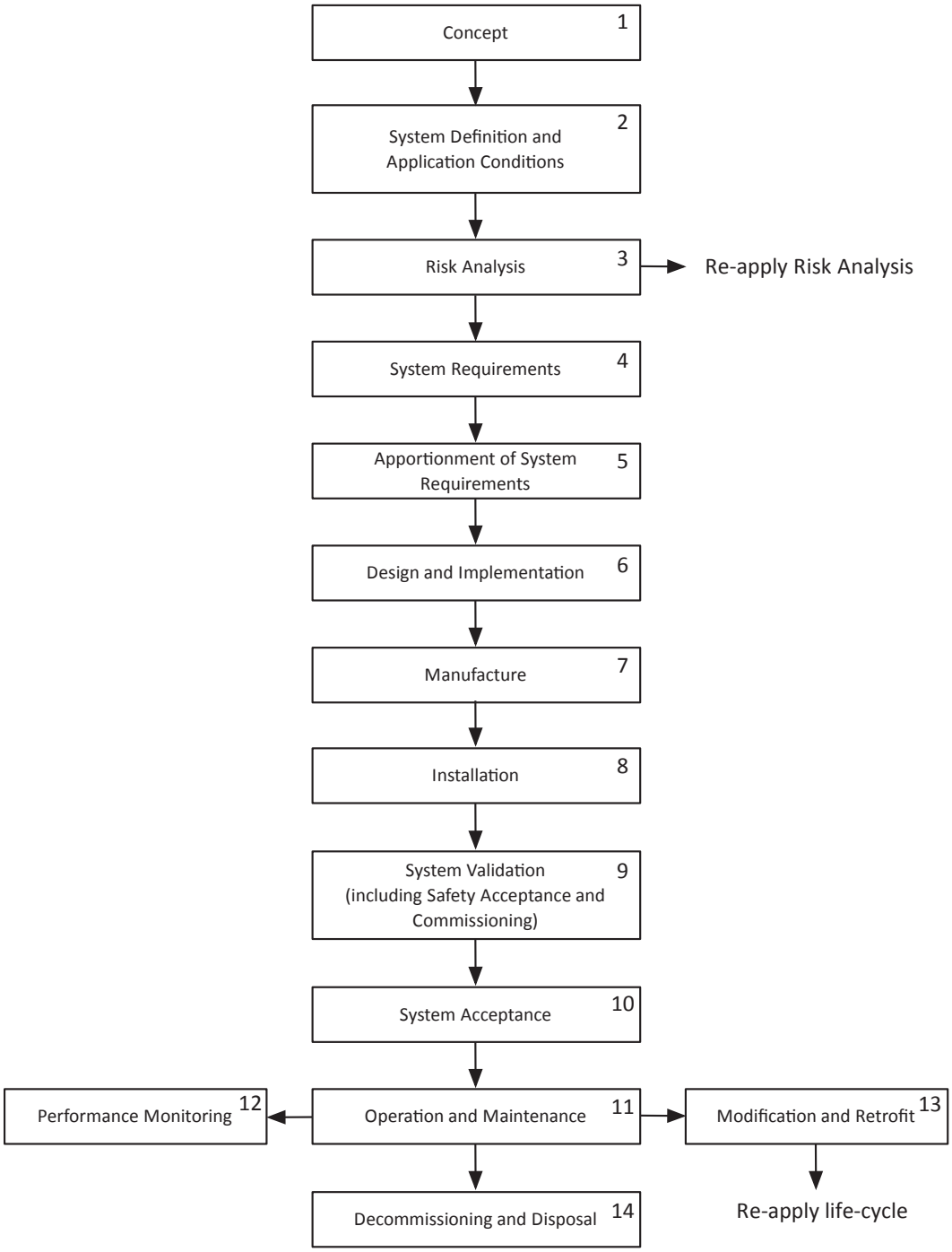


Figure 2.1: System life-cycle from EN50126 [28]

Assessment Technique	SWSIL 0	SWSIL 1	SWSIL 2	SWSIL 3	SWSIL 4
1. Checklists	HR	HR	HR	HR	HR
2. Static software analysis	R	HR	HR	HR	HR
3. Dynamic software analysis	–	R	R	HR	HR
4. Cause consequence diagrams	R	R	R	R	R
5. Event tree analysis	–	R	R	R	R
6. Fault tree analysis	R	R	R	HR	HR
7. Software error effect analysis	–	R	R	HR	HR
8. Common cause failure analysis	–	R	R	HR	HR
9. Markov models	–	R	R	R	R
10. Reliability block diagram	–	R	R	R	R
11. Field trial before commissioning	R	HR	HR	HR	HR
Requirement 1. One or more techniques shall be selected to satisfy the software safety integrity level being used.					

Table 2.1: Modified version of "Table A.9 – Software assessment (clause 14), assessment techniques" in Appendix A of EN50128 [29]

be performed. EN50126 [28] details the associated tasks for each phase indicated in Fig. 2.1. The joint set of phases and tasks covers the life of the system from initial concept to disposal.

The EN50128 [29] standard specifies the process and requirements for development of software for railway control and protection systems. EN50128 complements EN50126. Table 2.1 exemplifies the recommendations given on techniques for software assessment as given in Appendix A of EN50128 [29]. Different techniques for assessing the software are given a recommendation for use depending on a target software safety integrity level (SWSIL). In Table 2.1, the symbol "HR" means that the technique is highly recommended, "R" means recommended, and "–" means that the technique or measure has no recommendation for or against being used.

Habli [65] argues that there is no consensus on a universal approach to the development and assessment of safety-critical systems in the standards from different industries, domains, and regions but identify nonetheless some important similarities and states *"it is common for a safety standard to cover basic safety activities such as hazard and risk analysis, safety requirements allocation, and safety evidence provision"*.

We apply the definition given by the European Commission [51] defining **safety assurance** as *"all planned and systematic actions necessary to provide adequate confidence that a product, a service, an organisation or a functional system achieves acceptable or tolerable safety"*.

2.1.5 Safety case

The notion of a safety case is used in this thesis interchangeably with the term safety demonstration. A **safety case** according to Defence Standard 00-56 issue 4 [123] is *"a structured argument, supported by a body of evidence that provides a compelling, comprehensible and valid case that a system is safe for a given application in a given operating environment"*.

A safety case according to the railway standard EN 50129 [30] is a structured justification document that provides a convincing and valid argument that a system is adequately safe for a given application in a given environment. The documentation representing the safety case shall according to [30] be structured into the following parts: definition of the system; evidence on quality management; evidence on safety management; evidence of functional and technical safety; evidences in the form of references to dependent safety cases; a summary of all the evidences into a conclusion.

Independent of how a safety case is structured, it presents the body of evidences in a documentation form with the intention of providing confidence that the system is sufficiently safe for its intended purpose.

A safety case should ideally be created and maintained in parallel with the activities on system development and assessment, rather than postponed to the end of the development life-cycle [102]. Delaying the creation of the safety case to the later phases may result in inability to justify safety and give proper rationale for choices that potentially require costly design and implementation changes [34], or not achieving acceptance to put the system into operation [139].

2.1.6 Conceptual safety design

Design is defined by IEEE in [94] as *"(1) The process of defining the architecture, components, interfaces, and other characteristics of a system or a component. (2) The results of the process in (1)"*. Inspired by this, in this thesis we define **system design** as the architecture, components, interfaces, and other characteristics of a system.

A system design matures during the development process and may be represented as a concept in the early phases of development (see Fig. 2.1). The system design is mature and should define the main characteristics of a system when the implementation phase is reached. It is essential to capture the relevant requirements for the system to be developed. By relevant requirements we mean requirements that define the intended system functions and constraints, its intended use, how it shall be operated, how it shall be maintained, etc. Thus, in addition to the technical specification of the system design, we argue that it should be complemented with a specification of the requirements that motivated its definition and that describes the required properties of the system in question.

Unless it can be demonstrated that a safety critical system is sufficiently safe for application within its intended context, it will not be accepted for operation. In order to avoid re-engineering in the later stages, demonstration of safety should be a major concern from the very beginning of development. The importance of arguing that safety objectives are met is highlighted by the 2006 Nimrod XC230 aircraft accident review report [66]. Haddon-Cave concludes in the report [66] that a main contributor to the accident was a series of crucial design flaws that resulted in mid-air fire. The report also concludes that the safety case for the aircraft represented the best opportunity to identify these serious design flaws, but it was riddled with errors: *''If the Nimrod Safety Case had been properly carried out, the loss of XV230 would have been avoided"*. To reflect this, we define a **conceptual safety design** to mean an early stage specification of system requirements, system design, and safety case for a safety critical system. In the following, we elaborate on this definition based on the UML class diagram in Fig. 2.2.

As illustrated by Fig. 2.2, a conceptual safety design is basically a triplet consisting

of a specification of requirements *R*, a specification of system design *D*, and a specification of a safety case *S*. An implementation *I* satisfies a conceptual safety design *(R, D, S)* if there exists a safety case *SC* in accordance with *S* that demonstrates that *I* is safe with respect to the specifications of *R* and *D*.

Although the objective of a safety case is to argue that the implementation is sufficiently safe to apply in its intended context, this thesis only addresses the activities leading to the specification of a conceptual safety design. In the same way that a system design is refined into its implementation, the safety case specification is refined into a safety case.

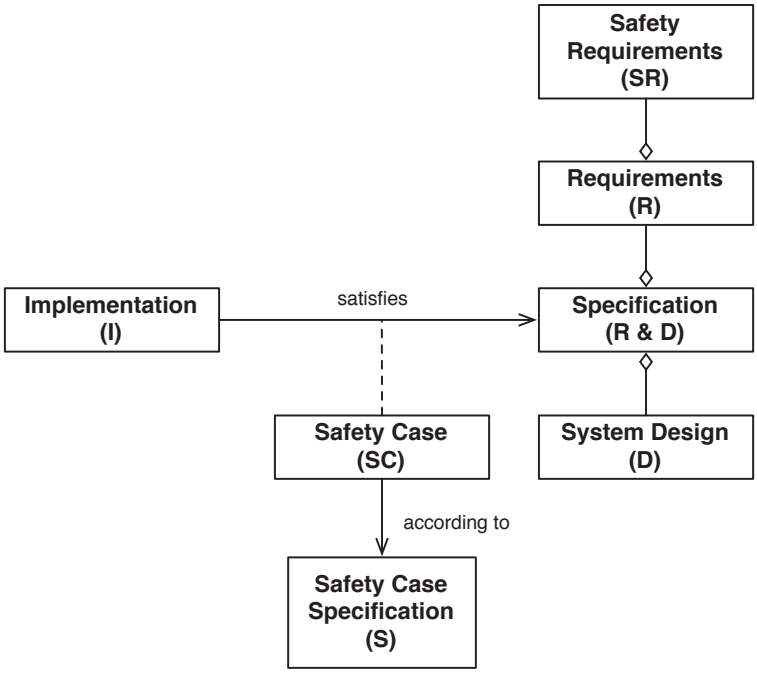


Figure 2.2: Relationships between an implementation (I) and elements of a conceptual safety design (R,D,S)

2.2 Success criteria

In Chapter 1 we outlined some of the challenges of developing safety critical systems and argued that there is a need for a new kind of guidance on effective solutions to these challenges. Further, we justified the suitability of a pattern-based solution to satisfy this need. The overall objective of the thesis, as stated in Section 1.1, is to develop a method and a pattern language that is:

1. Well-suited for developing conceptual safety designs.

2. Applicable within an industrial context within acceptable effort.

3. Comprehensible for its intended users.

To meet the overall objective, we have developed the SaCS method consisting of three main artefacts, namely:

1. The SaCS process.

2. The library of SaCS patterns.

3. The SaCS pattern language.

The process (1) guides the user in systematically selecting, applying, and documenting the use of patterns; the library (2) contains knowledge on effective solutions to recurring development challenges that may be used as development support; and the language (3) facilitates pattern specification and documentation of how patterns are applied in a given context. Further, the combined use of (1), (2), and (3) facilitates the development of conceptual safety designs.

In the following sections, we characterise more precisely the expectations for the three artefacts in terms of a set of success criteria.

2.2.1 The SaCS process

The purpose of the SaCS process is to facilitate development of conceptual safety designs of safety critical systems by a systematic application of patterns. The process should describe how the method should be applied, hence the criterion:

Success criterion 1 *The SaCS process provides satisfactory guidance for its intended users, which are system engineers, safety engineers, hardware and software engineers.*

The intended result of the process is a conceptual safety design according to the definition given in Section 2.1.6. The definition describes the main characteristics of a conceptual safety design but not what qualifies it as useful. A conceptual safety design is useful if it specifies the characteristics of a system in accordance with safety objectives at a sufficient level of detail, and is easy to use. On the basis of this we define the following success criterion:

Success criterion 2 *The application of the SaCS process results in conceptual safety designs that are: a) in accordance with safety objectives; b) at a sufficient level of detail; and c) easy to use.*

With respect to the overall objective of providing a pattern-based method that is *applicable* within industrial contexts to an acceptable effort, we formulate the following success criterion:

Success criterion 3 *The SaCS process is cost efficient.*

2.2.2 The library of SaCS patterns

The purpose of the library of SaCS patterns is to offer guidance on effective solutions to known challenges commonly recurring in the context of conceptual safety designs of safety critical systems. In order to offer a library of patterns that is well-suited to this purpose, the available patterns must define relevant solutions to relevant challenges. Thus, we define the following criterion:

Success criterion 4 *The library of SaCS patterns consists of patterns that describe effective solutions to recurring challenges within conceptual safety design.*

Each pattern in the library should be described in a format that facilitates efficient application of the guidance offered by the patterns. Once a user of SaCS has identified the essence of a problem and a suitable pattern for addressing the problem, it must be easy to apply. In addition, it should be easy to combine patterns. Hence, we define the following success criterion:

Success criterion 5 *The library of SaCS patterns consists of patterns that may be efficiently and effectively applied individually or in combination.*

Potential users of the method may want to formalise their company specific best practices as patterns by extending the the SaCS library, hence the criterion:

Success criterion 6 *The library of SaCS patterns is easy to extend.*

2.2.3 The SaCS pattern language

As stated in Section 2.1.3, a pattern language is a language for specifying patterns from a vocabulary of existing patterns and defined rules for combining these. The existing patterns in SaCS reside in the library. The expectations to the vocabulary of existing patterns are covered by the success criteria presented in Section 2.2.2.

Another expectation of the SaCS pattern language is that it shall facilitate the specification of composite patterns for conceptual safety design. Hence, the criterion:

Success criterion 7 *The SaCS pattern language is sufficiently expressive to specify composite patterns that may be easily instantiated into conceptual safety designs.*

An important part of providing safety assurance in many industrial domains where safety is a concern, e.g., within nuclear, railway and aviation, is to provide evidence that a suitable development process has been followed. In a pattern-based method for conceptualisation of safety critical systems, there should be support for documenting how patterns are applied. In order to fulfil the overall objective of providing a method that is applicable within an industrial context, we want the SaCS pattern language to support an easy way of documenting the application of patterns in a development context. The SaCS pattern language should facilitate specifying the combined use of patterns from the library in a manner that allows composite patterns to be used in different contexts. Hence, the criterion:

Success criterion 8 *The SaCS pattern language facilitates the specification of composite patterns in a context.*

A composition of patterns specified by the SaCS pattern language may include a large number of relationships between patterns from the library and in this sense potentially represent a complex specification. The complexity of a specification may be more manageable if it is modularised into a combination of several less complex specifications. By less complex, in this context, we mean a combination of several specifications where each specification has fewer patterns and fewer relationships than the original non-modularised specification. Modularisation may also facilitate separation of concerns, interchangeability of specification items, as well as reuse. This motivates the following success criterion:

Success criterion 9 *The SaCS pattern language facilitates modularity and reuse.*

In order to facilitate a common understanding of a composition of patterns from the library, each pattern in the library, as well as the rules for composition, must be easy to understand. Hence, the criterion:

Success criterion 10 *The SaCS pattern language is well-suited to express patterns that are easily understandable for its intended users, which are system engineers, safety engineers, hardware and software engineers.*

Chapter 3

Research method

In this chapter, we briefly present a method for technology research and describe how we have applied this method in our research. This chapter is influenced by similar discussions presented in the thesis of Hogganvik [83], Ligaarden [116], Omerovic [135], and Refsdal [141].

This thesis is rooted within the field of computer science, and there are different opinions on whether computer science may rightfully be called science [23,38,39,69,161].

Brooks [23] argues that computer science is not a science but rather an engineering discipline concerned with *making things*. Brooks further argues that the discovery of a new fact or a new law within the research fields that classify as sciences is itself an end that are worthy of publication, whereas the development of novel designs in computer science is not a proper end but merely tools to reach an end.

Abelson and Sussman [1] support the view of computer science not being a science. A definition of what is science and what disqualifies computer science as science is however not given. They relate computer science to mathematics and argue that mathematics provides a framework for precisely describing *what is*, while computation represents a framework for precisely describing *how to*.

Authors in support of computer science representing a science are in majority in the literature. Denning et al. [39] define computer science as *"the systematic study of algorithmic processes – their theory, analysis, design, efficiency, implementation and application – that describe and transform information"*. Denning claims [38] that computing research is filled with examples of scientific research where hypothesis are formed and experimentally tested according to scientific principles. Denning dismisses the viewpoint of computer science not being science merely due to its study of man-made artefacts and states that "computer science studies information processes both artificial and natural".

Tichy [161] considers computer science to be the study of information processes and therefore a science. Tichy argues that computer scientists should put more emphasis on experiments to test their theories. He claims that the traditional scientific method for observing phenomena, formulating explanations, and testing is also applicable to the field of computer science although the entity that is investigated is information rather than energy or matter.

Hartmanis [69] argues that advances in computer science often results from demonstration rather than from experimentation as is common within physical science and states *"It is the role of the demo to show the possibility or feasibility to do what was thought to be impossible or not feasible"*.

3.1 A technology research method

The research described in this thesis may be classified as technology research. Technology research concerns creating new artefacts or artefacts that are better, faster, safer or in some way improve on existing artefacts [81, 118, 153]. Classical research aims to gain new knowledge about the world in the form of new or refined principles and laws [153].

Within technology research focusing on information systems, Hevner [81] identifies two research paradigms, termed design-science and behavioural-science research, which are defined and motivated along the following lines. The behavioural-science paradigm is concerned with development and evaluation of theories to explain and predict human or organisational behaviour and has its roots in natural science research methods, or classical research methods to use the term in [153]. The design-science paradigm seeks to extend the boundaries of human and organizational capabilities by creating new and innovative artefacts. The design-science paradigm has its roots in engineering and the sciences of the artificial and is fundamentally a problem solving paradigm.

Fig. 3.1 is adopted from Solheim and Stølen [153] and illustrates the main steps in performing technological research as well as the associated question that is sought answered at each step.

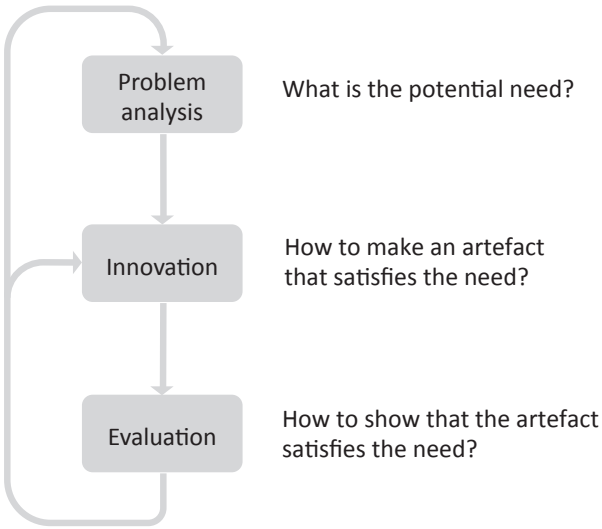


Figure 3.1: Main steps in technological research (adopted from Solheim and Stølen [153] and modified)

As described in [153] and illustrated in Fig. 3.1, technological research is performed by iterating over the following main steps:

- *Problem analysis* – concerned with identifying the (potential) need for a new or improved artefact. The need may be characterised in the form of a set of success criteria.
- *Innovation* – concerned with invention, creation or establishment of an artefact

that satisfies the identified needs. The overall hypothesis is that the new artefact satisfies the needs.

- *Evaluation* – concerned with the evaluation of the new artefact with the intention of finding out whether the need is satisfied by the artefact as hypothesised. The researcher formulates predictions on the basis of hypothesis to test the hypothesis. If the predictions are not falsified the confidence in the validity of the hypothesis increases.

If the hypothesis is falsified, the previous steps may be reconsidered.

3.2 Strategies for evaluation

According to McGrath [120], there are eight common strategies for evaluation of predictions. There is however no single evaluation method according to McGrath that provides results that are valid across populations (strong on generality), provides very precise measurements (strong on precision), and at the same time is performed in environments that are very similar to reality (strong on realism). The researcher has to choose, based on the strengths and weaknesses of the different evaluation methods and the questions that are required answered, how the different kinds of methods should be combined.

The eight different strategies are presented in Fig. 3.2. The evaluation strategies fall into four groups, corresponding to the inner circle of Fig. 3.2, depending on the differences in the environment in which the evaluations are performed. The three single letters with a bold font in Fig. 3.2 denote the point of maximum concern with respect to providing results strong on either precision, generality, or realism.

The most common evaluation strategies according to McGrath are:

- *Field experiment* – an experiment carried out in a natural environment, but in which the researcher intervenes and manipulates a certain factor.
- *Field study* – a direct observation of natural systems, with little or no interface from the researcher.
- *Experimental simulation* – a laboratory test simulating a relevant part of the real world.
- *Laboratory experiment* – gives the researcher a large degree of control and the possibility to isolate the variables to be examined.
- *Qualitative interview* – a collection of information from a few selected individuals. The answers are more precise than those of a survey, but cannot be generalized to the same degree.
- *Survey* – a collection of information from a broad and carefully selected group of informants.
- *Non-empirical evidence* – argumentation based on logical argumentation and common sense.
- *Computer simulation* – operating on a model of a given system.

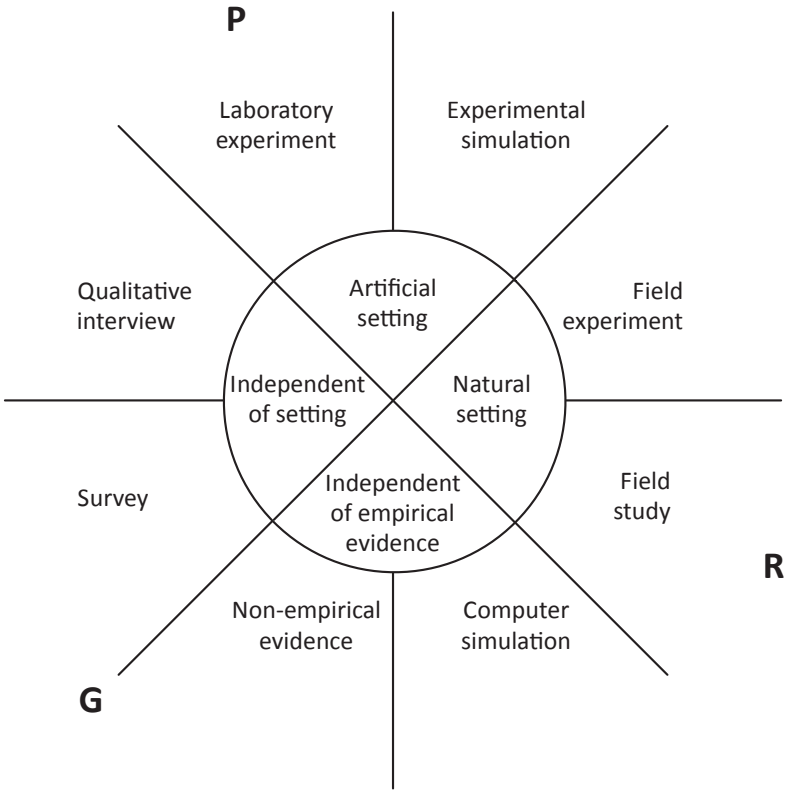


Figure 3.2: Evaluation strategies (adopted from McGrath [120])

It is desirable to maximise precision, realism and generality simultaneously but, as argued by McGrath, this is not possible with one single research strategy. For example, the field study is strong on realism but not necessarily on precision; a laboratory experiment is strong on precision but not necessarily on realism. Different and complementing evaluation strategies can be applied in order to compensate for any lack of precision, generality or realism of one strategy.

In the following sections, we describe how we have applied the technology research method outlined in Section 3.1, including an outline of the different and complementing evaluation strategies that have been applied.

3.3 How we have applied the research method

Fig. 3.3 is adopted from Refsdal [141] and modified to illustrate our research method. An iterative process was applied, in which the artefacts were continuously developed and improved based on the experiences gained. Our research method is similar to the method outlined in Section 3.1 where Fig 3.1 depicts a process that iterates over the problem analysis, innovation, and evaluation activities.

In this thesis, the final results from the problem analysis are summarised in Chapter 1 and Chapter 2. Chapter 1 presents the challenges that motivated the research and defines the overall objective. Chapter 2 defines success criteria characterising what is required to satisfy the need. The final results from the innovation activity are summarised in Chapter 5. The final results from the overall evaluation are summarised in Chapter 7. Chapter 8 concludes on the basis of the evaluation results.

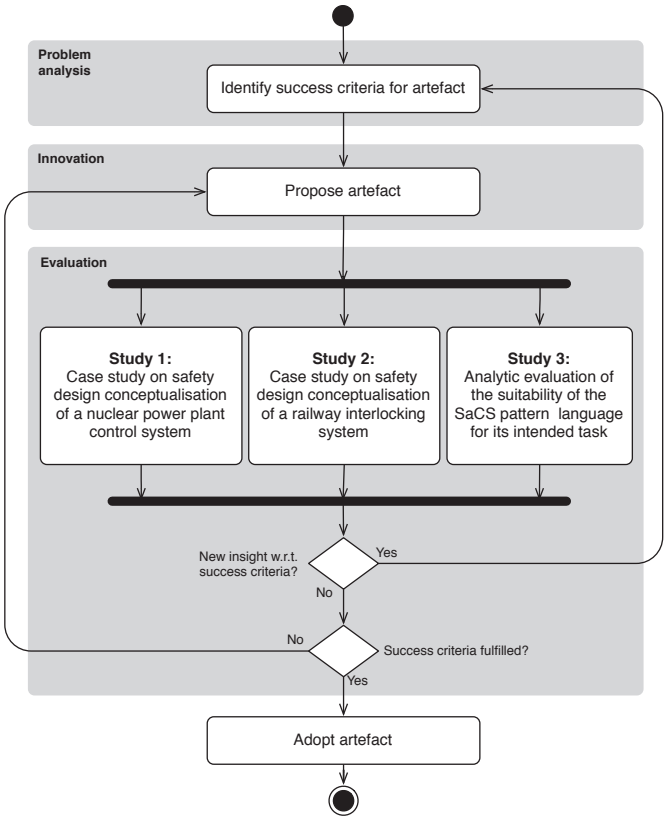


Figure 3.3: Applied research method (adopted from Refsdal [141] and modified)

Fig. 3.3 distinguishes between three studies. They are briefly described below:

- *Study 1: Case study on safety design conceptualisation of a nuclear power plant control system.* The case study concerns the suitability of SaCS within the nuclear domain. It addresses a control system for operating a power producing reactor in load following mode. A reactor operating in a load following mode is controlled

such that the production of its outputs, electricity, is scaled to accommodate fluctuations in power demand during the day. Experiences from this case study are detailed in Paper 2 [74].

- *Study 2: Case study on safety design conceptualisation of a railway interlocking system.* The case study concerns the suitability of SaCS within the railway domain. It addresses an interlocking system for safely controlling train movements through a train station with two tracks and a level crossing that are connected to neighbouring trains stations with a single track. Experiences from this case study are detailed in Paper 3 [75].

- *Study 3: Analytic evaluation of the suitability of the SaCS pattern language for its intended task.* The evaluation was performed by adopting SEQUAL [108,134] to assess the quality of the SaCS pattern language. Results of the evaluation are detailed in Paper 4 [76].

The case studies referred to as Study 1 and Study 2 may be seen as a variant of the *field experiment* in the McGrath classification, a strategy that scores high on realism. According to Eisenhardt [43], the case study approach is *"especially appropriate in new topic areas"* and describes how to build theories from case study research. Yin [176] states: *"A case study is an empirical inquiry that investigates a contemporary phenomenon in depth and within its real-life context, especially when the boundaries between phenomenon and context are not clearly evident."*. Study 3 is most closely related to *non-empirical evidence* in the McGrath classification.

The three artefacts presented in this thesis have also been evaluated against the literature, the results of which are presented in Chapter 4 and the discussion in Section 7.2.

Fig. 3.4 details the relationship between the three different studies and the artefacts proposed. The two case studies, Study 1 and Study 2, address all three artefacts. Study 3 primarily addresses Artefact 3.

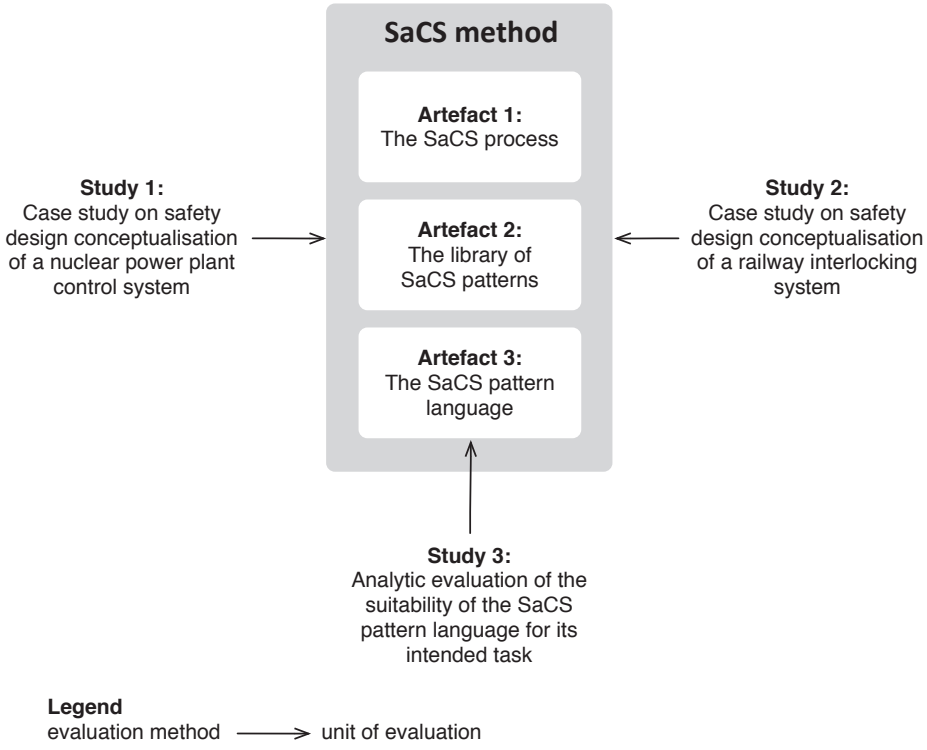


Figure 3.4: Relationship between evaluation studies and research artefacts

Chapter 4

State-of-the-art

In this chapter, we give a general overview of state-of-the-art literature of relevance for the three artefacts developed in this thesis. The relevant literature is classified with respect to topics instead of per artefact. For a more detailed discussion on the relationship of our artefacts to the state-of-the-art, the reader is referred to Section 7.2. We have structured this chapter into four sections:

- *Section 4.1 Pattern-based development* – SaCS is a method that makes use of a pattern language, and this section addresses the relevant literature on patterns. Section 4.1.1 addresses notable pattern collections of relevance to SaCS. Section 4.1.2 addresses the literature on pattern languages. Section 4.1.3 addresses formats for defining patterns.

- *Section 4.2 Standards and guidelines* – SaCS is intended to be used for conceptualisation of safety critical systems, and this section addresses relevant literature defining acceptable and recommended practices for achieving safe systems within different domains. International safety standards and guidelines are commonly established by a consensus between experts from different countries and thus describe practices that may be seen as being widely accepted within an industrial domain. Section 4.2.1 addresses generic safety standards that may be applied independent of domain. Section 4.2.2 addresses the railway domain. Section 4.2.3 addresses the nuclear domain. Section 4.2.4 addresses the aviation domain.

- *Section 4.3 Safety engineering* – SaCS facilitates conceptualisation of safety critical systems, and this section addresses safety engineering literature on effective solutions within the field of safety engineering. Section 4.3.1 addresses relevant literature on system design. Section 4.3.2 addresses approaches for safety demonstration. Section 4.3.3 addresses methods and techniques for safety assurance.

- *Section 4.4 Modelling* – SaCS offers a graphical notation for specifying composition of patterns and their application. SaCS also offers a library of patterns where each pattern expresses a solution graphically. Section 4.4.1 addresses relevant literature on modelling. Section 4.4.2 addresses the literature on generic principles of visualisation.

4.1 Pattern-based development

The initial work on pattern languages is described by Alexander et al. [3–5] as an approach to buildings architecture and construction. The notion of patterns was later introduced to the software engineering community by the work of Beck and Cunningham [17] and after that by Gamma et al. [55] with their popular book *"Design Patterns: Elements of Reusable Object-Oriented Software"*. The patterns presented in [17,55] are organised more in the form of pattern collections rather than pattern languages. The notion of patterns is currently adopted by several disciplines within system engineering. Here we will report on the pattern approaches relevant with respect to the development of SaCS.

4.1.1 Pattern collections

The engineering methods, techniques, and tools applied at different stages of development vary with the development challenges that are addressed. Thousands of patterns for development of software based systems exists [79] on diverse topics in different pattern collections and pattern languages. Of special relevance to SaCS are pattern approaches that support requirement elicitation, systems design, and establishing safety cases, as these are constituents of a conceptual safety design according to the definition in Section 2.1.6.

Jackson [98] presents the problem frames approach that supports requirements analysis and elicitation. A generic set of problem diagrams are adapted to model the main problem domains appearing in a development context and their interactions with a system under development. Jackson identifies five different problem frames that represents classes of problems, and describes a solution for each class of problem. Once a user has identified different problem domains, interactions, and classes of problems in the context of the analysis, the user systematically elicits requirements for the system. Although the problem frames approach is useful for detailing and analysing a problem and thereby detailing requirements, the problem classes presented in [98] are defined at a very high level of abstraction.

The use of boilerplates [84, 150] for requirement specification is a form of requirement templates but nonetheless touches upon the concept of patterns. The boilerplate approach helps the user phrase requirements in a uniform manner and to detail these sufficiently. Although boilerplates may be useful for requirement specification, the focus of this thesis is more towards supporting requirement elicitation.

Withall [174] describes 37 requirements patterns for assisting the specification of different types of requirements. The patterns are defined at a low level; the level of a single requirement. Each pattern describes how a particular type of requirement shall be defined. The patterns of Withall may be useful, but as with the boilerplates approach, the patterns are intended primarily to support the specification of requirements rather than their elicitation.

Patterns on design and architecture of software-based systems are presented in several pattern collections. One of the well-known pattern collections is described by Gamma et al. [55] on recurring patterns in the design of software based systems. Hanmer [68] presents a collection of patterns for the development of fault tolerant systems. Buschmann et al. presents in [24–26, 105, 146] a five volume series on Pattern-Oriented Software Architecture (POSA). In addition, there are different online collections, such

as the Portland Pattern Repository [35] or the repository provided by the Hillside Group [82]. The Lecture Notes in Computer Science (LNCS) series on Transactions on Pattern Languages of Programming (TPLOP) [130–132] give insight into notable recent developments within the community of pattern developers. Without a doubt, the different pattern collections and languages for system design and architecture represent a deep insight into effective solutions on different aspects of software based systems development. However, there are two major challenges to the effective application of existing patterns for the development of safety critical systems: a) The need to justify the application of one design/architecture/technique pattern over another; and b) The need to provide evidence of the correct application and combination of the knowledge contained in the different patterns. Design choices should be founded on the needs for assuring and arguing safety, and otherwise follow well-established principles for good design. The choice of applying one design pattern over another should be based on a systematic process to establish the need in order to ensure that design choices are not left unmotivated.

The motivations for a specific design choice are founded on the knowledge gained during the development activities applied prior to design specification. Gnatz et al. [57] outline the concept of process patterns as a means to address the recurring problems and known solutions to challenges arising during the development process. The patterns of Gnatz et al. are not tailored for development of safety critical systems. Thus, their patterns do not necessarily reflect development practices relevant to our domain. Fowler [54] presents a catalogue of 63 analysis patterns. The patterns do not follow a strict format but represent a body of knowledge on analysis described textually and supplemented by sketches.

While process patterns and analysis patterns may be relevant for assuring that the development process applied is suitable and leads to well-informed design choices, Kelly [102] describes patterns for arguing that safety is sufficiently addressed. Kelly presents patterns that support the safety demonstration in the form of safety-case patterns. The safety-case patterns express how safety may be argued and are represented textually and by the use of the Goal Structured Notation (GSN) [61]. GSN is a notation for explicitly representing the elements of a safety argument in a graphical model. The safety-case patterns presented by Kelly [102] are representative for how we want to address the safety demonstration concern in our approach.

From the literature study on different pattern approaches, it is obvious that a great body of knowledge is captured in the form of patterns or similarly easy to use formats. A challenge is to effectively combine and apply the knowledge captured in these diverse pattern collections. Furthermore, pattern collections tend to contain patterns on different concepts defined in a manner where it is unclear how the knowledge contained in patterns may be combined. In a survey on software pattern practices, Henninger and Corrêa [79] states *"software patterns and collections tend to be written to solve specific problems with little to no regard about how the pattern could or should be used with other patterns"*.

The intention of pattern languages is to facilitate the integration of patterns as support for solving problems. We now go on to address pattern languages in Section 4.1.2.

4.1.2 Pattern languages

The pattern language of Alexander et al. [5] consists of 253 patterns on the architecture of buildings. The connections between patterns are defined by a sequential arrangement of the patterns. The sequential arrangement also represents the intended order for applying the patterns as well as how a grammar is approached in the pattern language. The patterns are defined from large scale solutions (organising towns and buildings) towards smaller scale pattern solutions (rooms and walls).

Aguiar and David [2] present a pattern language as guidance to developers in documenting object-oriented frameworks. The pattern language is more a process than a language in the sense that patterns have interdependencies and the set of patterns defines a sequence in the style first proposed by Alexander et al. [5].

Guerra et al. [62] present a pattern language for metadata-based frameworks. The interconnectivity of the patterns is presented in a simple sketch providing guidance to the user on the relationship between patterns in terms of arrows and short sentences explaining how patterns can be used in a sequence. Although some guidance is given on the expected sequence of patterns, it is not clear how patterns should be integrated.

Vainsencher and Black [166] present a pattern language supporting a single code model for the program analyses and tools that might be applied to the code. An illustration shows the relationship between challenges occurring in the intended context of the language and the patterns that may be applied to solve the challenges. A development is approached by a sequence of generally defined challenges and choices for how challenges may be solved by patterns. The approach represents a process with solution choices more than a language.

Hentrich and Zdun [80] present a pattern language as support for designing Service-Oriented Architectures (SOA). The patterns in the language address conceptual and architectural design issues. The approach is similar in style to [2,62,166]. The interconnections between patterns are indicated as loosely defined dependencies and expected sequences.

Siddle [148] presents an interactive approach to pattern selection. The approach is defined as a means to support software design education and learning. The author exemplifies the approach through a sequence of steps that might guide a user in exploring and selecting patterns that are appropriate for application in a given context.

Henninger and Corrêa [79] identify in their survey a lack of inter-pattern dependencies in software pattern collections and that little to no regard has been taken to specify how patterns should and could be used in pattern combinations. One exception identified by the authors is the pattern relations of Noble [129].

Zimmer [178] describes the need to define relationships between design patterns in order to efficiently combine them. Noble [129] builds upon the ideas of Zimmer and defines a number of relationships such as *uses, refines, used by, combine*, and *sequence of* as means to define relationships between design patterns. A challenge with the relations defined by Noble is that these may only specify relations on a very high level. The relations are not able to detail what part of a pattern is *used, refined*, or *combined* and relate that with a part described in a combined pattern. Thus, the approach does not facilitate a precise modelling of relationships.

Bayley and Zhu [16] define a formal language for pattern composition. The authors argue that design patterns are almost always to be found composed with each other and that the correct application of patterns thus relies on precise definition of the

compositions. A set of six operators is defined for the purpose of specifying pattern compositions. Formalised relationships expressed between design patterns described by Gamma et al. [55] are used to provide examples of the language in use.

Smith [151] presents a catalogue of elementary software design patterns in the tradition of Gamma et al. [55] and proposes the Pattern Instance Notation (PIN) for expressing compositions of patterns visually. The PIN notation can be seen as a visual language for expressing patterns. The notation uses simple rounded rectangles to abstractly represent a pattern and its associated roles. Connectors define the relationships between patterns. The notation is comparable to the UML [137] collaboration notation.

4.1.3 Pattern formats

Table 4.1 indicates the format of patterns in different pattern collections and languages. The intent of a pattern as presented in the work Alexander et al. [5] is to disseminate and document design knowledge and represent a vocabulary for expressing ideas on design. Each pattern in the language described in [5] is defined according to a common format that is easy to understand for a human user, through a sequence of sections containing textual descriptions and sketches.

Patterns are commonly described by a sequence of descriptive sections, named or unnamed, adopted from the initial pattern format defined by Alexander et al. [5]. Henninger and Corrêa [79] identify that although there are similarities, almost every pattern collection they surveyed used different pattern formats. The authors further identify that attempts at standardising pattern formats have not been successful as different types of patterns serves different needs.

Table 4.1 compares the formats of patterns in some of the commonly cited pattern collections [5,26,55] and some more recent pattern collections [2,62,68,102,140,143,151, 179]. The different pattern formats are clearly rooted in the tradition of the patterns described by Alexander et al. [5] where each pattern has a *name*, describes a *problem*, presents a *solution* to the described problem, and further indicates *related patterns*.

[5] (1977) Alexander et al. – Buildings architecture
[55] (1995) Gamma et al. – Software design
[26] (1996) Buschmann et al. – Software architecture
[68] (2007) Hanmer – Software fault tolerance
[102] (2008) Kelly – Safety argumentation
[2] (2011) Aguiar and Gabriel – Frameworks documentation
[179] (2011) Zimmermann et al. – Process description
[151] (2012) Smith – Software design
[143] (2013) Rüping – Data migration
[62] (2013) Guerra et al. – Metadata-based frameworks
[140] (2013) Ratzka – User interface interactions

	[5]	[55]	[26]	[68]	[102]	[2]	[179]	[151]	[143]	[62]	[140]
Name	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Also known as		✓			✓			✓		✓	
Intent		✓			✓		✓	✓			
Motivation		✓			✓			✓			
Context	✓		✓			✓	✓			✓	✓
Applicability		✓			✓			✓			
Problem	✓		✓	✓		✓	✓		✓	✓	✓
Forces						✓	✓		✓	✓	✓
Solution	✓		✓	✓		✓	✓		✓	✓	✓
Process steps							✓				
Structure		✓	✓		✓			✓		✓	
Participants		✓			✓			✓		✓	
Collaborations		✓			✓			✓			
Dynamics			✓							✓	
Consequences		✓	✓		✓		✓	✓		✓	✓
Benefits									✓		
Liabilities									✓		
Rationale											✓
Implementation		✓	✓		✓			✓			
Sample code		✓						✓			
Example			✓		✓	✓	✓		✓	✓	
Example resolved			✓								
Variants			✓								
Known uses			✓	✓		✓	✓			✓	✓
Related patterns	✓	✓		✓	✓		✓	✓		✓	✓
See also			✓								

Table 4.1: The format of patterns as found in the literature

4.2 Safety standards and guidelines

International safety standards and guidelines have been important sources for consolidating our pattern-based approach with accepted development practices. The standards and guidelines presented in this section have been developed on the basis of consensus among domain and safety experts from different countries. Thus, the standards and guidelines presented reflect broadly accepted practices for development of safety critical systems. The intention of a pattern is to express a generalised solution of well-proven practices for solving commonly recurring problems.

A commonality between the safety standards and guidelines addressed in the following is the focus on the need to assure functional safety. Functional safety is the part of the overall safety of a system or equipment that depends on the system or equipment operating correctly in response to its inputs. Further, functional safety is achieved when every specified safety function is carried out and the level of performance required of each safety function is met. A difference between the safety standards and guidelines addressed in the following is the categorisation scheme applied to functions as a measure of their relative importance to safety. Although different categorisation schemes are applied in different domains, their use is very similar as it provides a basis for imposing requirements for risk reduction.

4.2.1 Generic

The standard IEC 61508 [93] is an international and generic safety standard titled *Functional Safety of Electrical/Electronic/Programmable Electronic Safety-related Systems* that is applicable to development of safety critical systems in all kinds of industry. The IEC 61508 standard has been adopted into domain specific variants in different industrial domains, e.g., automotive [96], railway [29] and for development of nuclear power plant systems [86], and as such is influential on how safety critical systems are developed within several domains.

The viewpoint of the IEC 61508 [93] standard, as of its domain specific variants, is that an absolutely safe system is not possible to achieve; there will always be risks associated with the operation of a system. A key concept is the notion of functional safety; achieved when every specified safety function is carried out and the level of performance required of each safety function is met.

The standard describes the process of steps, known as the safety life-cycle, to be performed in order achieve functional safety. The safety life-cycle includes activities that shall be performed in the different phases of the system development, such as defining the initial concept, hazard and risk analysis, overall safety requirements, etc. until the last phase that addresses decommissioning and disposal.

The standard requires an assessment of the risks associated with the operation of the system under development in its context and a classification of the system with respect to the required safety integrity level (SIL) of the functions performed by the system. Based on the allocation of SIL there are defined requirements on the application of means for risk reduction.

4.2.2 Railway

The CENELEC standards EN 50126 [28], EN 50128 [29], and EN 50129 [30] represent a railway implementation of IEC 61508 [93]. EN 50126 [28] defines the general approach to specification and demonstration of reliability, availability, maintainability, and safety (RAMS) of the total railway system. EN 50129 [30] addresses system safety in signalling systems while EN 50128 [29] addresses the methods and techniques that provide software that satisfies requirements to safety integrity.

The standards are highly-process oriented, meaning that they define requirements for applying certain processes, performing specific activities, and documenting results associated with different phases of the safety life-cycle of systems. The SIL concept is adopted from IEC 61508 and represents a measure, in the form of a categorisation of the criticality of the functions provided by systems on safety, used for specifying safety integrity requirements for the safety functions performed by systems. The categorisation of SIL is performed on the basis of hazard and risk analysis. Depending on SIL there are imposed constraints and given recommendations on the methods and techniques to be applied during system development as well as how to document safety objectives being met.

Fig. 2.1 in Section 2.1.4 illustrates the system life-cycle as it is presented in EN 50126 [28]. The system life-cycle in EN 50126 is relatively similar to the life-cycle described in other safety standards presented in Section 4.2.

Another influence from the railway domain is the framework known as Common Safety Methods (CSM) described in the European Union (EU) regulation presented in [50]. A guidance to the application of CSM regulation is provided in [52]. CSM addresses the harmonisation of the practices for operation and approval of railway products within the borders of the EU. The framework describes the process to be used across all EU member states when assessing significant changes to railway systems that affect safety. The CSM framework and associated guideline is particularly strong on describing the general practices for achieving safe systems and the acceptable strategies for arguing that safety objectives are met.

4.2.3 Nuclear

The standard IEC 61513 [86] represents a nuclear domain application of IEC 61508 and describes the general requirements for nuclear Instrumentation and Control (I&C) systems. IEC 61513 is a system level document covering all categories of nuclear safety I&C functions. The functions to be performed by I&C systems are assigned to categories according to their importance to safety. The importance of a function to plant safety is identified on the basis of the anticipated consequences in the event of its failure to perform required operation when needed and the consequences in the event of a spurious actuation. Different requirements are imposed on I&C systems based on their importance in assuring nuclear power plant safety such that, e.g., more demanding requirements are put on systems that are required to provide reactor shutdown when needed than those systems that are responsible for controlling the reactor within operational bounds.

IEC 61226 [91] describes a classification of I&C functions into category A, B, C, and otherwise not classified functions. Category A functions either play a principal role in the achievement or maintenance of plant safety or are functions whose failure could

directly lead to accident conditions which may cause unacceptable consequences if not mitigated by other category A functions. Category B functions play a complementary role to the category A functions in the achievement or maintenance of plant safety. Category C functions play an auxiliary or indirect role in the achievement or maintenance of plant safety. IEC 60880 [90] defines requirements for category A software. IEC 62138 [87] defines requirements for category B and C software.

The purpose of the classification described in IEC 61226 [91] is to provide a mechanism for imposing graded requirements for the development of I&C systems depending on the importance of their functions to achieve or maintain plant safety. As in the other industrial domains, different safety objectives are required to be met depending on the classification of functions and thus the systems performing the functions.

4.2.4 Aviation

The ARP4754 [49] is a guideline to the process of developing aircraft systems. ARP4754 is supported by the standards ED-12B [47] and ED-80 [48] on the process of developing the software and hardware parts of systems, respectively.

ARP4754 [49], ED-12B [47], and ED-80 [48] are highly process-oriented standards. ARP4754 [49] defines a Development Assurance Level (DAL) classification scheme consisting of five categories, A-E, for classifying aircraft functions with respect to the criticality of the functions on safety. The allocation of DAL dictates the required development process rigour that shall be applied in order to assure a proper risk reduction. The ARP4761 [144] is a guideline for conducting the safety assessment process and for the application of different methods for assessing aircraft systems with respect to safety.

The classification of a DAL for a function leads to the application of graded requirements for the software and hardware parts of a system implementing the function. ED-12B [47] describes software requirements. ED-80 [48] describes hardware requirements. The concept of DAL may be seen as similar to the concepts of SIL within railway and the categorisation applied for nuclear power plant I&C systems. The different concepts are similar in the sense that different kinds of requirements are imposed on systems depending on a classification of the importance of the functions performed by systems on safety. Contrary to the railway industry, the nuclear industry and the aviation industry have so far not adopted the SIL concept from IEC 61508 [93] directly.

4.3 Safety engineering

4.3.1 Designing for safety

To design a system that is fault free is generally accepted as being impossible [113]. Common strategies within safety engineering for reducing the risk of system failure are to avoid faults being introduced in the first place or remove them once detected, if that is possible; secondly to mitigate the potential negative effects of faults by fault tolerance techniques.

Avižienis et al. [14] describe the relationship between fault and failure and the fundamental relationship between threats, mitigations, and their effect on achieving dependable systems. Dependable systems are, according to the authors [14], the result of the combined utilisation of the following techniques: fault prevention; fault removal;

fault forecasting; and fault tolerance. None of these techniques are sufficient alone, but a mix of these is required to attain dependable systems.

Fault prevention may be supported by different methods and techniques for assuring high quality of the end product. One means for avoiding faults related to out-of-range values is to choose a strongly typed programming language. Faulty data type casting was one of the causes that led to the Ariane 5 accident [117]. Another means may be to apply formal methods in order to mathematically specify and reason about the software [22]. Fault detection and removal may be supported by, e.g., dynamic and static testing techniques [109]. Different types of predictive techniques and safety monitoring may support fault forecasting. Fault tolerance may be supported by design such that errors occurring at a component or subsystem level are mitigated before propagating to the system level.

Redundant hardware modules are commonly applied in safety critical systems. Different traditional and well-established design principles are descried by Storey in [156,157]. The Dual Modular Redundant (DMR) [156] design contains replicated modules that operate in parallel on the same input in order to provide protection against random hardware failure. A voting mechanism compares the outputs of the different modules and takes appropriate action upon disagreement. The replication of modules provides protection against random hardware failure based on the assumption that it is unlikely that two identical modules will experience random failure at the same time. In order to reduce the likelihood of common cause failure, a higher degree of dissimilarity between redundant hardware modules may be achieved by, e.g., choosing hardware modules from different suppliers in redundant channels. In order to provide further protection against random hardware failure, the number of redundant modules may be increased. The Triple Modular Redundancy (TMR) [156] design and the M-out-of-N [156] design is comparable to the DMR arrangement in that there is a replication of modules and a mechanism for voting that allows continued safe operation of a system despite module failure. The difference is the increased number of redundant modules, which is more costly. The benefit is an increased protection against random faults and the ability to continue operation in the event of module failure as the voter may detect which module failed. The standby-spare arrangement [156] is comparable to DMR but where the voting mechanism is replaced with a module for switching operational control between a primary and a secondary control module on the basis of information from a fault detection unit.

Contrary to the hardware parts of a system, the software always fails systematically and not randomly due to wear and tear. A software module will fail each time the conditions for executing its faulty code are satisfied. Given two identical software modules operating in parallel, both modules will fail provided that the operating conditions are similar as both modules will continue to execute their identical faulty code. In order to achieve protection against software faults in redundant channels software diversity may be introduced.

Avižienis [13] describes the N-version technique that involves using several different implementations of a program as a means to achieve tolerance to failure in any of the individual versions. The assumption is that software errors will be experienced as statistically-uncorrelated random events in independently-implemented programs. The different versions all attempt to implement the same specification and therefore should produce the same result for the same input. In the absence of a disagreement between the software modules, the unanimous answer is passed to its destination. The action

taken given a disagreement between modules depend on the number of versions/variants used. For a two-version system, the disagreement between the versions represents a fault condition where the system cannot tell which version is incorrect. Given a transient, the problem may be resolved by repeating the calculation. If a system contains three or more versions, the effect of an error in one version may be masked by some form of voting mechanism. The main disadvantages of N-version programming is its cost of implementation. Knight and Leveson [107] question the independence claim of different versions of the same specification on the basis of an experiment with N-version programming. In the experiment, 27 independently developed versions of a program was subjected to one million tests. The tests revealed that the programs were individually extremely reliable but also that the number of tests in which more than one program failed was substantially more than expected.

The Recovery Block (RB) [8] technique uses some form of error detection to validate the operation of a software module. If an error is detected in a primary module then a secondary software module is used. The RB scheme is based on the use of acceptance tests. These tests may have several components and may, for example, include checks for runtime errors; excessive execution time; and calculation errors. The overall system can execute a redundant module in one of two ways given the failure of an acceptance test, which are by backward recovery or by forward recovery. By backward recovery it is meant that the state of the system is reset to a recovery point from which the redundant module starts executing. By forward recovery it is meant that the redundant module must continue execution from the current execution point. The effective implementation of the error detection mechanism and acceptance tests are essential for the successful implementation of the RB technique.

A way to achieve safety is to utilise adaptable systems [11]. Adaptable systems are able to evolve during operation, thereby providing resilience against unforeseen changes that may occur while executing. The ability to adapt introduces an additional uncertainty aspect with respect to the correctness of the function provided compared to conventional software-based systems that do not evolve by themselves. Adaptable systems are proposed or explored in many different domains, e.g., health, nuclear power production, space exploration, and military applications, for different kinds of safety and mission critical control tasks as indicated in the papers [18, 110, 127, 149, 152, 158, 159]. The motivation is to increase performance by applying techniques for adapting the behaviour of some control function to handle unforeseen or otherwise uncertain factors. Evidently there are challenges with respect to determinism and predictability of adaptable systems since the behaviour of these types of systems will change over time as adaptations are effectuated. In critical applications where there may be unacceptable consequences of certain failures, potential negative effects due to erroneous adaptations must be controlled in order to facilitate utilisation. There are open questions with respect to how to assess the safety of such systems as indicated in [152] and [158]. Tallant [158] identifies the non-determinism of intelligent and reasoning systems as a challenge to current practices on verification and validation that further slows down the utilisation of these types of systems for performing safety critical control.

From a safety perspective, the main challenge with utilising adaptable systems is to demonstrate that the system is still safe after each change. The service offered by an adaptable system may vary over time as an effect of the adaptations that may be experienced at runtime. This reduces the ability to predict the behaviour of the system. The variability of an adaptable system and its service is governed by the adaptation

loop mechanism [145] consisting of processes for monitoring, detecting, deciding and acting upon change. The challenges associated with the variability of the service and the system as an effect of adaptations are discussed in [126] and [172].

In order to provide assurance before commissioning that system adaptations will not negatively affect safety, there are basically two choices: (i) assure that the system is not able to adapt into a hazardous configuration, or (ii) assure that that no change is effectuated during operation unless it may be confirmed there that are no adverse effects of the change. The two choices are basically different in that choice (i) establishes the safety argument once and for all before commissioning, while choice (ii) necessitates on-job assessment of change as an integral part of the safety argument. Kurd [110] describes the SCANN (Safety Critical Artificial Neural Network) model and how it can be applied in order to satisfy performance goals as well as safety goals. The SCANN incorporates constraints as part of the network model in order to constrain adaptation in the manner of choice (i) above. Kurd [110] and also Taylor [159] discuss several methods found in the literature addressing different aspects of the life-cycle important for verification and validation of neural networks. Both authors identify rule extraction techniques as a promising approach for demonstrating that the knowledge inherently contained in a neural network is consistent with requirements.

A survey of rule extraction techniques is provided by Darbari in [37]. Darbari [37] identifies that many of the approaches to rule extraction is in the form of a search problem where a space of candidate rules are explored and confirmed by testing of individual candidates against the network in order to establish valid rules.

Although rule extraction techniques may offer a promising approach for assessing what a neural network has learnt and to establish safety tests, this thesis pursues strategy (ii) above in the first case study documented in Paper 2 (described in Chapter 10) by an adaptation of the ideas of Sha [147] on dissimilar redundant systems for controlling the uncertainty related to adaptable systems learning. A DMR fault tolerant design solution is described in our second case study documented in Paper 3 (described in Chapter 11) founded in traditional design principles to achieve fault tolerance.

4.3.2 Safety demonstration

The two basic concerns that must be addressed in order to achieve confidence that a safety critical system is sufficiently safe for its intended purpose are: (1) that the defined safety objectives are correct and adequate; and (2) that the safety objectives are satisfactory satisfied. A safety demonstration should in order to successfully convince someone assessing that a system is sufficiently safe for operating in its intended context provide the claims, arguments and evidences required such that the concerns (1) and (2) are convincingly demonstrated met.

A safety demonstration can be structured in different styles and by different means, e.g., by graphical approaches like Goal-Structure Notation (GSN) [6] or Claims-Arguments Evidence (CAE) notation [20], by Bayesian approaches [60], by natural language structured according to a template as in [30], or simply presented as a coherent body of documents stemming from the system life-cycle whose combination acts as demonstration of safety being sufficiently addressed, to name a few. The intent, however, is to structure claims, arguments and evidences in such a manner that it convinces some actor reviewing the collection of evidences that the system is sufficiently safe for its intended use.

Approaches for demonstrating that safety objectives are satisfactory met may be divided into: implicit approaches and explicit approaches. Bishop et al. [20] argue that safety justifications in the past tended to be implicit and rule-based when arguing compliance to standards. The authors find implicit approaches suitable in stable environments where best practice is supported by extensive experience but not being flexible in accommodating technology advances. Due to the inherent lack of flexibility in the rule-based approaches, a more explicit and goal- based safety justification approach has emerged according to the authors.

Leveson [114] divides safety assurance methods into two general types: (1) prescriptive approaches; and (2) goal-setting approaches. Leveson [114] view is in line with Bishop et al. [20] on goal-setting approaches emerging within different domains. Prescriptive approaches (called rule-based in [20]) typically represented in standards and guidelines have a long lasting tradition within domains developing safety critical systems and may be divided [64,114] into process-based approaches and product-based approaches. In standards and guidelines with a strong process focus, safety assurance is to a large degree required to be evidenced in the application of recommended or mandatory development practices. In standards and guidelines with a strong product focus, safety assurance is to a large degree based on product specific evidences.

The result of showing compliance to a standard, e.g., IEC 61513 [86], may be seen as a way of performing an implicit safety demonstration where a system is developed according to practices prescribed by a consensus of an international community of experts on what yields safe systems. Thus, a demonstration of compliance also implies that the procured system is developed according to acceptable practices and thus may be seen as sufficiently safe for operating in its intended context. The position of seven nuclear regulators presented in [160] gives a warning to this view and states *"standards and national rules reflect the knowledge and consensus of experts; the fulfilment of these requirements may not be sufficient to assure safety in all cases. They usefully describe what is recommended in fields such as requirements specification, design, verification, validation, maintenance, operation, etc. and contribute to the improvement of safety demonstration practices".* The authors further argue [160] that any combination of approaches for the demonstration of safety may be used and may also be conditioned on the provision of evidence for compliance to laws, regulations, and standards as well as conditioned on certain specific residual risks are acceptable and that certain safety properties are achieved.

Fig. 4.1 illustrates the three main approaches that may be integrated to form an overall safety justification strategy as identified in [160] and [20].

The three main approaches for arguing safety as indicated in Fig. 4.1 differ in focus as follows:

1. *Goal-based approach*: justifications based on evidences of certain safety properties being achieved [160].

2. *Rule-Based approach*: justifications based on evidences of compliance with a set of agreed rules, laws, and standards [160] .

3. *Risk-informed approach*: justifications based on evidences of certain specific residual risks are acceptable [160].

In the railway domain, EN 50129 [30] insists on a structured safety justification document known as a safety case. The role of the safety case [30] is: *"The safety case*

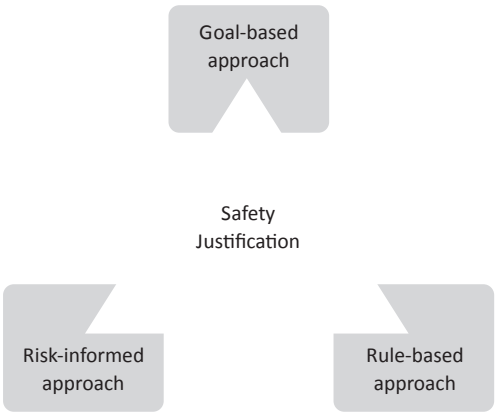


Figure 4.1: Main safety justification approaches (adopted from Bishop, Bloomfield, and Guerra [20] and modified)

forms part of the overall documentary evidence to be submitted to the relevant safety authority in order to obtain safety approval for a generic product, a class of application or a specific application".

In the nuclear domain on the other hand, IEC 61513 [86] does not require a safety case to communicate a safety demonstration. The view on how a safety demonstration should be used is also somewhat different compared to the railway. The position of seven nuclear regulators as given in [160] is that *"A safety demonstration addresses the properties of a particular system operating in a specific environment. It is therefore specific and carried out on a case-by-case basis, and not once and for all".* Thus, to obtain safety approval for a generic product is not generally accepted within the nuclear domain as safety approval of some product is given on its specific use in a specific context.

Habli [65] argues that the traditional free text representation of safety argumentation in the safety domain is problematic when communicating complex safety arguments involving stakeholders with different concerns. Habli identifies the Goal Structured Notation (GSN) [61, 101, 102, 154] as a means to improve expressiveness and deal with the complexity of safety arguments. GSN is a graphical notation for defining a safety argumentation that is inspired by Toulmins theory of argumentation [162].

The main elements of GSN are presented in Fig. 4.2 where an overall claim is decomposed via sub-claims into evidences. Goal elements are used to express claims. Context elements are used to describe the context of terms expressed in a claim. Strategy elements are used to express a specific strategy for the decomposition of a claim. Solution elements are used to reference documentation that shall be perceived as evidences.

4.3.3 Safety assurance

Independent of the means applied for structuring and presenting a safety demonstration, as addressed in Section 4.3.2, there is a need to establish evidence that provides the necessary safety assurance. What is necessary depends on what is needed to convince some safety authority that safety is assured. A first test is to convince oneself.

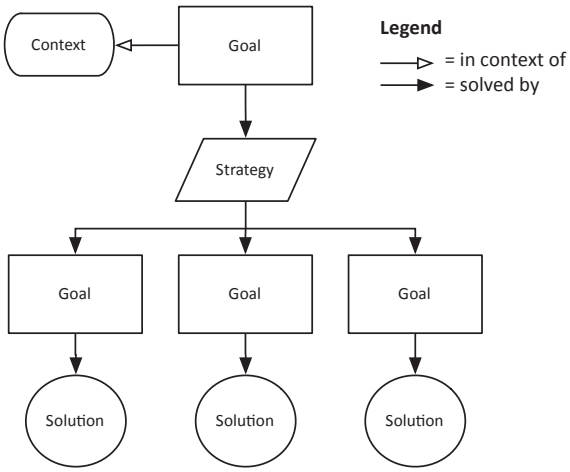


Figure 4.2: Main graphical elements in GSN (adopted from [61] and modified)

Safety assurance may be achieved on the basis of evidence on the appropriate use of processes, methods, techniques and tools for developing and assessing a system under consideration according to acceptable safety engineering principles. What is acceptable depends on the task or the problem that is addressed and the context of application.

Fig. 2.1 presents the system life-cycle as it is presented in the railway standard EN50129 [30]. The system life-cycle is described as a process of phases. EN50129 also provides guidance to the appropriate selection of techniques to support the different phases of the process. Annex E of EN50129 provides recommendations for the application of methods and techniques relevant to the phase identified as risk analysis, namely phase 3 in Fig. 2.1. A prerequisite for achieving safety assurance is to establish the safety objectives associated with the system under development. Some safety objectives may be defined on the basis of the results from hazard and risk analysis. The following techniques are highly recommended within EN50129 as support for failure and hazard analysis of SIL 4 systems: Preliminary Hazard Analysis (PHA); Fault Tree Analysis (FTA); Markov analysis; Failure Modes Effects (Criticality) Analysis (FMEA/FMECA); HAZard and OPerability studies (HAZOP); and Common Cause Failure (CCF) analysis. Another recommended technique is Event Tree Analysis (ETA). In the following, each of these techniques will be briefly presented.

Preliminary Hazard Analysis (PHA) [46] is a hazard identification technique. PHA is typically applied in the early stages of development when detailed technical information about the system under consideration is not available, e.g., prior to the specification of the design. The objective is to identify the hazards that are relevant with respect to the system under development, determine the potential causes of the hazards, the effects of hazards, their probability of occurring, and establish initial requirements to avoid or control hazards. The identification step may be supported by a list of known hazards. The results of the PHA are usually reported by using a worksheet.

Fault Tree Analysis (FTA) [89] is a deductive analysis technique that starts from the definition of a hazard and investigates deductively what can cause this hazard.

Each of the identified potential causes are further investigated in order to deduce their causes. This procedure continues until every derived cause is related to some initial basic event for which there is no reason to perform further investigation. The causal relationships between basic events, intermediate causal events, and the hazard are typically presented in a tree-like structure called a fault tree. The hazard investigated represents the top-node in the tree, the basic events represent the leaf-nodes in the three, and the intermediate events are represented as the other nodes in the tree. Different types of logical gates between the nodes describe the relationships between hazards, causes and basic events.

Markov analysis [9] is a stochastic mathematical method for analysing the reliability and availability of systems whose components exhibit dependencies. In Markov analysis, a system is modelled as a number of states with transitions between the states. It can be challenging to apply Markov analysis as the number of discrete system states gets large due to the rapid growth of the workload for analysing the system.

Failure Modes Effects (Criticality) Analysis (FMEA/FMECA) [88] is a stepwise procedure for systematically assessing the potential failure modes of a system and their effects. Once a system is defined in a manner such that its function and structure may be understood, each part is assessed with respect to it potential failure modes and their resulting effects on the rest of the system. For each failure mode investigated, it is typically assumed that it is the only failure mode that exists at a given time. Thus, the FMEA approach is effective for identifying single failure points but not failure combinations.

HAZard and OPerability studies (HAZOP) [165] is a qualitative technique for identifying hazards based on discussions carried out by a multi-disciplinary team during meetings. The meetings are facilitated by applying guide-words as support for investigating the system effect of deviations from intended operation of different system parts.

Common Cause Failure (CCF) analysis [46] is a technique for assessing in what way a system may experience failure as an effect of multiple components failing due to a single common cause within a specified time.

Event Tree Analysis (ETA) [92] is a technique for modelling the consequences of initiating events by an inductive investigatory process. A set of event chains is modelled starting from an initiating event with logical branches describing potential subsequent events. The result is a tree-like structure modelling the causal relationships between an initiating event and risks by causal chains of events.

The techniques presented above are widely applied in different industrial domains as support for hazard and risk analysis. The result of applying these techniques does not provide safety assurance alone. The result may be used to identify where efforts for risk reduction should be allocated, e.g., by providing a basis for defining safety objectives. Thus, depending on the result of the risk analysis, the development team chooses the appropriate techniques, methods, and tools in order to reduce identified risk to an acceptable level. What is an acceptable level depends on the criticality of the functions that will be performed by the system and what is defined as acceptable within a particular domain. At some point during development, the risks are analysed and understood to such a degree that means for risk reduction may be proposed. Safety assurance is achieved when the proper set of risk reductions is applied such that safety objectives may be argued to be satisfactorily met.

4.4 Modelling

4.4.1 Modelling dependencies and flows

Krogstie [108] discusses different general abstraction mechanisms and perspectives used in conceptual modelling. Krogstie divides modelling languages according to the core phenomena or concepts that are represented and focused on in a language, termed the *perspective* of the language. A survey of state-of-the-art approaches for conceptual modelling is presented and divided into eight different perspectives. The modelling perspectives can be briefly summarised as:

1. *Behavioural*: The main phenomena are states and transitions between states [108].

2. *Functional*: The main phenomena is the transformation, e.g., an activity transforms inputs to outputs. The terms typically used are function, process, activity, and task [108].

3. *Structural*: The main phenomena support the description of the static structure of a system. The constructs of structural languages are often termed entity, or otherwise termed object, concept, thing and phenomena with some differences in the associated semantics [108].

4. *Goal and Rule*: The main phenomena are goals and rules where rules influence the actions of a set of actors [108].

5. *Object*: The main phenomena of object-oriented languages are similar to those found in most object-oriented programming languages. The terms typically used are class, object, type, instance, inheritance, event, interface, etc. [108].

6. *Communication*: The main phenomena in this category of languages support the modelling of how persons cooperate within work processes through their conversations and through mutual commitments taken within them [108].

7. *Actor and Role*: The main phenomena in this category of languages support the modelling of relationships between actors, otherwise called agents, and their role [108].

8. *Topological*: The main phenomena in this category of languages support the modelling of the topological ordering between different concepts, e.g., the modelling of where different tasks shall be performed [108].

Krogstie [108] states that in many cases it can be difficult to classify a specific modelling approach to one perspective as the perspectives are somewhat related, and follows up with a discussion on other classification schemes and their drawbacks. More importantly, no language supports every perspective. Thus, the language for expressing concepts within a particular domain should be selected on the basis of the modelling need. We present below a short summary of a few modelling languages supporting one or more of the modelling perspectives 1-8 presented above.

The Unified Modeling Language (UML) [137] is a general-purpose modelling language that is well known and commonly used within the field of software engineering.

UML has a rich set of language constructs and consists of different kinds of diagrams that supports the specification of the behaviour and structure of a system. UML profiles may be defined in order to adapt UML to particular needs, e.g., the need to express patterns that may be combined.

Jackson [98] describes the Problem Frames (PF) approach as support for requirements analysis and elicitation. Jackson offers a notation for scoping the main concerns relevant to a system under consideration for performing its intended task. The notation offers constructs for identifying the different problem areas and the entities that occur in a specific system context as well as the interfaces between problem areas and entities. When the problem is framed, the user continues with an analysis of the problem through a sequence of steps. The analysis results in the definition of the requirements that must be satisfied in order for the system under consideration to satisfy its goals.

Gane and Sarson [56] present the Data Flow Diagram (DFD) approach for expressing the data flow through an information system. The notation includes symbols for expressing processes, stores (data and material), flows, and external entities. An external entity interacts with a system and may represent an individual, an organisational actor, or a technical actor. A system is represented as a network of processes, stores and external entities connected by flows.

The Business Process Model and Notation (BPMN) [136] is a notation for business process modelling that supports different kinds of users from business analysts, technical developers, to those managing and monitoring business processes. The basic elements in BPMN are different kinds of flow objects, connecting objects, swimlanes, and artefacts. The conceptual modelling of the application of patterns could take the form of a BPMN-like diagram in which flow objects may represent patterns, connectors may represent relations between patterns, and artefacts may represents pattern instances or the outcome of applying a pattern.

Chen [31] describes the Entity-Relationship (ER) model for describing database design. The basic elements in a model are: entity, relationship, and attribute. The model constructs are used to describe the relationships between data.

Hull and King [85] present the Generic Semantic Modelling (GSM) language. GSM supports the semantic modelling of data in the sense that it models the primitive data types, the constructed data types (e.g., generalisations and aggregations of other data types), and attributes. GSM also supports the specification of derived classes and the modelling of the relationships between different types of data.

Halpin [67] presents the Object Role Modelling (ORM) language for designing database models and the procedures for creating, transforming, mapping, and querying the models.

Gulla [63] presents the Phenomena, Process, and Programs (PPP) environment where PPP models are created by four sub-languages for expressing: process model (PrM), phenomenon model (PhM), process life description (PLD), and user interface description (UID).

Rumbaugh et al. [142] present the Object-Modeling Technique (OMT), a predecessor to UML, for software modelling and design. There are three main types of models in OMT: object model; dynamic model; and functional model. Object models are comparable to UML class diagrams. Dynamic models are comparable to UML state machine diagrams and activity diagrams. Functional models describe flow in terms of processes, data store, data flow and actors.

Winograd and Flores [173] present a graph structure for the modelling of a conversation for action. A state transition diagram in the form of a graph describes the speech act between parties attempting to coordinate actions with one another. The approach could be relevant for describing a connected set of patterns where each vertex in the graph represent a pattern and the edges between could represent different relationships or interactions.

Yu and Mylopoulos [177] present an integrated set of languages known as i* as support for the early phases of system modelling, in the understanding of the problem domain. The i* is developed for modelling and reasoning about organizational environments and their information systems with respect to capturing the actors, tasks, goals, resources, and dependencies that occurs in these environments.

Gordijn and van der Raadt [59] present the e^{3}value modelling language for inter-organisational modelling. The purpose of the language is to model how objects of value are created, exchanged, and consumed by actors.

Gopalakrishnan et al. [58] present simple annotations to UML activity diagrams for differentiating between the places where activities are performed. The activities in an activity diagram are given different colours. A legend provides a mapping between each of the colours used in a diagram and the associated geographical places in which activities are performed.

4.4.2 Principles of visualisation

The key activities performed by a reader in order to draw conclusion about a diagram are, according to Larkin and Simon [111]: searching; and recognising relevant information. In order to aid the user in these activities there exists a solid basis of knowledge represented in the literature. Visualisation of qualitative information for effective human information processing is addressed by Ellis [44], Healay and Enns [77], Katz [100], Lidwell et al. [115], Norman [133], Tufte [163, 164], Ware [170], and Wertheimer [171], to name a few.

Ellis [44] provides an extensive overview of the Gestalt principles of perception which build upon classic work from Wertheimer [171] and others. Gestalt principles build upon theory of human perception and the tendency of the human mind to naturally perceive whole objects on the basis of object groups and parts[1].

Katz [100] presents patterns on designing information and exemplifies their use on hundreds of examples of how to facilitate ease of perception and clarity in visualisations.

Lidwell, Holden, and Butler [115] present 125 patterns of good design evidenced within theory and empirical research on visualisation. The patterns describe principles of designing visual information for effective human perception. The patterns are defined on the basis of extensive research on human cognitive processes.

Norman [133] presents principles in the design of everyday objects. Some of the principles are generally applicable within design, such as the principle of natural mapping and the principle of achieving high visibility. The natural mapping principle concerns the arranging of objects controlled and the objects used for controlling. Given a natural mapping order of controllers and controlled objects, the effect is a reduced need for memorising the relationship between objects. The principle of high visibility

[1] In Fig. 4.1, the arrangement of shapes makes the mind perceive a triangle encapsulating the words "safety justification" although there is no triangle. The phenomena is within Gestalt known as the principle of closure [44, 171].

concerns designing items such that those features of the designed item that should get immediate attention receive it, e.g., one look at an item reveals its state and the alternatives for action.

Healey and Enns [77] survey theories and empiric research on visual attention and visual memory. The authors discuss the limited ability of humans to remember image details and to deploy attention. Further, they argue that the limitations to the capabilities of humans for attention and memory have significant consequences for visualisation and should be mitigated by producing images that are salient, memorable, and that guide attention to important locations within the data.

Tufte addresses [164] the display of quantitative information in charts and diagrams. Although the work mainly covers the display of quantitative data, some generally applicable principles to the use of lines, symbols, words, shading and colour for reducing visual complexity and highlighting the data of interest are addressed. In [163], Tufte presents general principles of graphical presentation of quantitative data as well as qualitative data thoroughly evidenced in the literature.

Ware [170] gives a detailed analysis of the current theory of the mechanics of human visual cognitive processes and provides guidance on the use of graphic design as a tool to enhance perception.

A principle of good design addressed by several authors [15, 115, 128] is to balance the need for performance by the importance of preference in designing solutions. Lidwel et al. [115] give the example of the Dvorak keyboard layout, estimated to provide 30% better performance and fewer errors in typing than the QWERTY keyboard layout. Although the Dvorak layout was introduced more than 50 years ago, people prefer the familiar QWERTY layout. Thus, solutions showing higher performance than well-established solutions are not necessarily accepted due to preference.

Chapter 5

Achievements: the overall picture

In this chapter, we give an overview of the three main artefacts resulting from our work and how they are integrated into the SaCS method.

The SaCS method consists of the following three artefacts:

1. *The SaCS process*: defines the process for systematically applying SaCS patterns to support the development of conceptual safety designs.

2. *The library of SaCS patterns*: defines 26 basic SaCS patterns on best practices for conceptual safety design categorised into six different kinds. A user defines composite SaCS patterns on the basis of patterns in the library. The user can extend the library by defining additional basic and composite SaCS patterns.

3. *The SaCS pattern language*: defines how to express basic SaCS patterns and includes a graphical notation for specifying composite SaCS patterns.

In Section 5.1, we start by presenting the SaCS method. Thereafter, Section 5.2 presents the SaCS process, Section 5.3 presents the SaCS pattern language, while Section 5.4 presents the library of SaCS patterns. Lastly, Section 5.5 exemplifies the SaCS method.

5.1 The SaCS method

Fig. 5.1 presents how the SaCS method integrates the three artefacts of this thesis, resulting in the SaCS method for conceptual safety design. In Fig. 5.1, the associations denote in what way the three artefacts are integrated. In this integration, an assumed user of the SaCS process makes use of the patterns within the library as guidance to problem solving. Furthermore, the user of the process makes use of the language for expressing a solution to a problem in the form of a pattern in order to extend the library. The formal syntax of the language is used to support the specification of patterns. The structured semantics is used to support understanding what is expressed by a pattern.

Depending on the complexity of the problem that needs to be solved in a given context, and to the extent available patterns can be used to solve the problem, the user chooses whether to address the problem with a single basic pattern or rather by a combination of several patterns. The classification structure for the patterns denotes the different kinds of patterns offered by the library. As a basic pattern provides

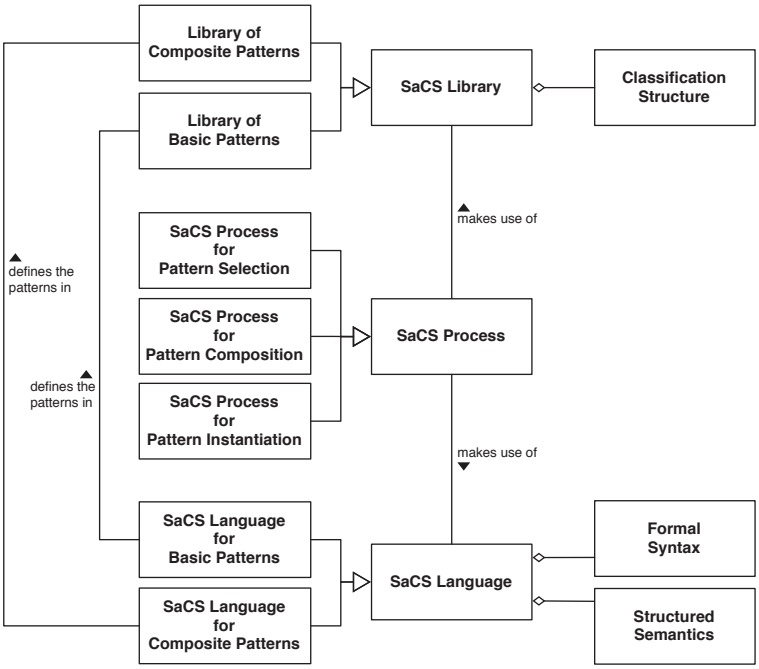


Figure 5.1: UML class diagram showing the integration of the artefacts into the SaCS method

guidance on a specific problem-solution concept with a limited scope, the use of several and complementary kinds of basic patterns is necessary for conceptual safety design. The combination of several basic patterns for problem solving facilitates separation of concerns.

A composite pattern is expressed graphically and specifies how several patterns are combined. The visual presentation facilitates the discussion between different kinds of users on how conceptual safety design is intended to be approached with patterns as guidance. The instantiation of a composite that combines suitable patterns supporting the specification of requirements, system design, and safety case in a given context, is used to produce the conceptual safety design.

5.2 The SaCS process

Fig. 5.2 presents the SaCS process. The SaCS process interleaves the three main activities, pattern selection, pattern composition, and pattern instantiation, each of which is described in the following subsections.

The intention of the interleaving is to allow flexible use. A user can choose to strictly follow the sub-activities of pattern selection before performing pattern composition, and finally, pattern instantiation, if that is found suitable. Another approach is to iterate over pattern selection, pattern instantiation, and pattern composition, one pattern at a time, until every relevant pattern has been applied.

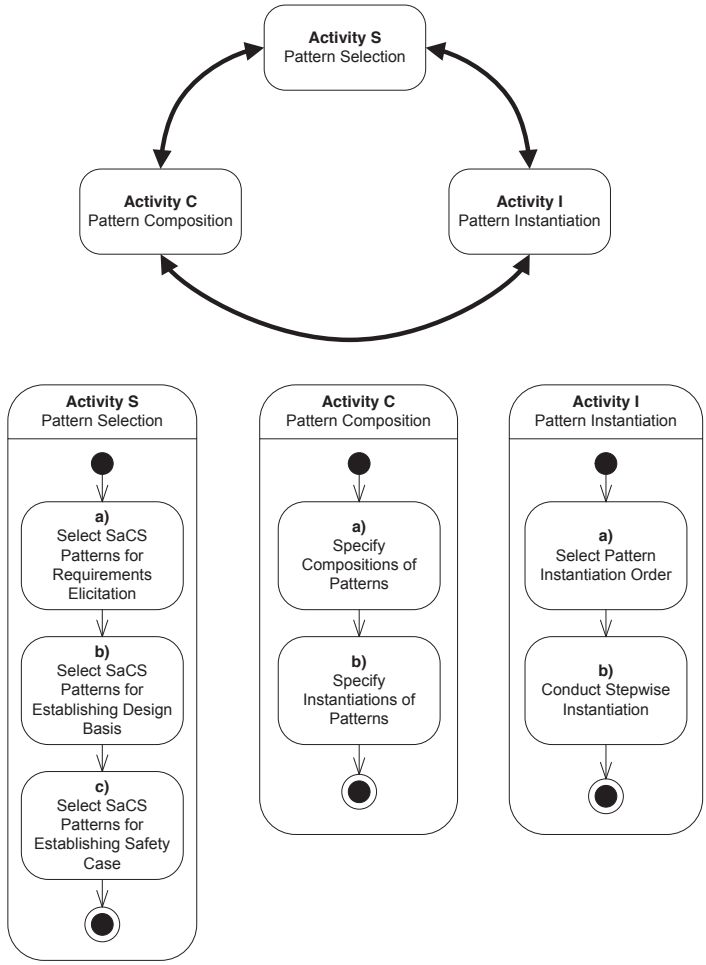


Figure 5.2: The main activities of the SaCS process

5.2.1 Pattern selection

Fig. 5.3 presents a specific kind of relationship between a few of the basic patterns in the library and is a fragment of a larger visualisation. The visualisation is used to support pattern selection and is referred to as a pattern selection map. The kind of relationship that is visualised is the order in which a user is expected to consider the relevancy of a pattern for application in a given context. Patterns are ordered in the map, partly as a sequence and partly grouped into choices between alternatives. The ordering of patterns in the map supports the process for pattern selection as presented in Fig. 5.2 where a user firstly selects patterns supporting requirements elicitation, secondly selects patterns supporting system design, and thirdly selects patterns supporting the construction of a safety case.

In Fig. 5.3, the round icons classify patterns. The identifiers adjacent to the round icons are names of patterns in the library. The arrows suggest the order in which

patterns should be considered for application. A user traverses the pattern selection map in the order indicated by the arrows and considers whether a pattern is relevant or not for application by inspecting its definition. The diamonds represent choices. The patterns below a diamond represent the alternatives associated with a choice where more than one pattern can be selected.

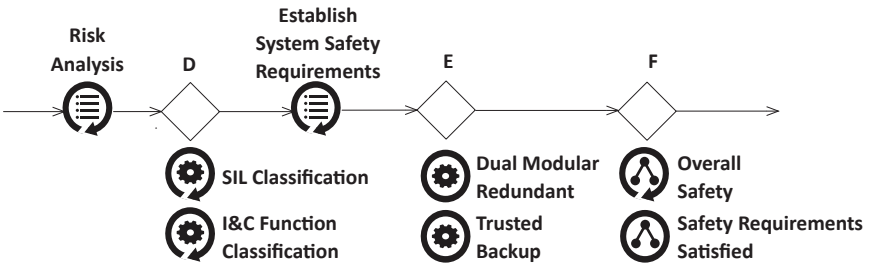


Figure 5.3: A fragment of a larger map of available patterns in the library supporting pattern selection

Six different kinds of basic patterns may be selected from two main groups: process assurance and product assurance patterns. In Fig. 5.3, icons enclosed by an arrow are process assurance patterns. Icons enclosed by a circle are product assurance patterns. Product assurance as well as process assurance patterns are further classified into requirement patterns, solution patterns, and safety case patterns. The process assurance patterns define, e.g., processes for performing certain safety engineering activities and documenting results, effective methods for addressing specific development challenges, and how to argue safety on the basis of applying acceptable development processes. The product assurance patterns define very specific system properties, e.g., the requirements for a system that shall perform a specific function, various effective system designs, and safety arguments from a product oriented development perspective.

5.2.2 Pattern composition

A composite pattern is expressed graphically and is used to specify how the patterns selected as guidance for conceptual safety design are combined and applied. Furthermore, the visual presentation facilitates the discussion between participants involved in conceptualisation on whether the patterns selected and their combination is appropriate as support for solving the different kinds of challenges that arise in a given context. Once the users agree on the specific combination of patterns that shall be applied to support conceptualisation, and have documented this plan within a composite pattern, relevant users perform stepwise instantiation of the patterns within the composite to produce results. A composite is instantiated according to the rules for composition. The patterns within a composite are instantiated according to their associated rules. In order to document more precisely how patterns are applied, the user can annotate a composite to indicate the documents that were used as input to pattern instantiation as well the documents produced as a result of pattern instantiation.

5.2.3 Pattern instantiation

The user must interpret a pattern in order to correctly apply it in a relevant context. Each basic pattern defines its instantiation rule that guides the user on how to transform input to output by the use of the pattern definition. Furthermore, each pattern explicitly defines its input and output parameters. A description of how a pattern is expected to be applied in a sequence with other patterns can be found in the definitions of those basic patterns where this is relevant.

Guidance on the proper instantiation of composite patterns is represented by the rules for composition as defined within Paper 1. In short, patterns in SaCS are combined with others through operators, otherwise called relations, which act on the parameters of patterns. As the output of a pattern cannot be produced before the input required for pattern instantiation is available, and relations define the expected work flows or dependencies between related patterns, the instantiation order can to a large degree be deduced from the defined relations. In cases where it is unclear which pattern should be prioritised or patterns are expected to be instantiated in parallel, the user can provide additional guidance by using annotations to indicate the expected instantiation order of the patterns within a composite.

5.3 The SaCS pattern language

The SaCS pattern language is used to specify SaCS patterns and their use. While a basic pattern is intended to describe a concept for how to solve an elementary problem within conceptual safety design with a limited scope, a composite pattern is intended to describe a solution to a more complex problem as a combination of simpler patterns.

We present the syntax for basic and composite patterns separately as the formats are different. The structured semantics is approached similarly for both basic and composite patterns.

In Section 5.3.1, the syntax of basic SaCS patterns is presented. Thereafter, we present in Section 5.3.2 the syntax of composite patterns. In Section 5.3.3, we describe our approach to structured semantics for SaCS patterns.

5.3.1 Syntax of basic SaCS patterns

Table 5.1 presents the overall format of basic SaCS patterns in terms of identifying the named sections of a pattern description and their expected content. The format is defined in accordance with the practice for documenting patterns as addressed in Section 4.1.3 and can be seen as a variant of the format of Alexander et al. [5].

Fig. 5.4 presents the illustration given in the "Pattern Signature" section of the pattern named *Hazard Identification* [74]. Within the definition of a composite pattern,

Figure 5.4: The signature of the Hazard Identification pattern (from Paper 2 [74])

Section	Content
Name	A unique name identifying the pattern.
Pattern Signature	A short description of the basic information associated with a pattern, such as its categorisation, the name and type of every input and output parameter, and a presentation of how the pattern shall be referred to graphically within a composite SaCS pattern.
Intent	A short overview of the problem that is addressed together with a description of how the pattern provides a beneficial solution for solving the problem.
Applicability	A short description of the typical situations in which the pattern may be applied.
Problem	Identifies and describes the core of problem that is addressed by the pattern.
Solution *	Describes the essence of a solution that addresses the defined problem. The solution description identifies how the inputs to the pattern are used as part of the solution to produce the expected outputs of the pattern when it is applied in a context. * There are variants of the heading depending on the pattern category. The headings are: *Problem Frame Analysis Solution* for product assurance requirement patterns, *Design Solution* for product assurance solution patterns, *Process Solution* for process assurance requirement patterns, *Method Solution* for product assurance solution patterns, *Argument Structure Solution* for process assurance safety case patterns and product assurance safety case patterns.
Instantiation Rule	Defines a rule for the correct instantiation of the pattern in terms of defining what the expected output is.
Related Patterns	Describes the relationships to other SaCS patterns.
Known Uses	Describes examples of the pattern in use.

Table 5.1: The format of basic SaCS patterns

the pattern *Hazard Identification* can be referenced graphically as presented in Fig. 5.4. The illustration identifies the name of the pattern, its categorisation, as well as the name and categorisation of each of its input and output parameters according to the syntax of SaCS [73]. The remainder of the pattern definition describes how the inputs, i.e., ToA (short for Target of Assessment), Src (short for Source), IdHz (short for Identified Hazards) are used in order to derive some result, i.e., HzLg (short for Hazard Log). Specifically, the "Solution" section within a basic pattern definition expresses a generic solution that facilitates the transformation of inputs to outputs.

Fig. 5.5 presents a UML activity diagram with the addition of SaCS specific annotations that represents the process solution proposed within *Hazard Identification* for identifying hazards.

The SaCS specific annotations are represented by: the dotted drawn frame that encapsulates the activity diagram; the dotted drawn boxes that appears on the dotted drawn frame; and the arrows that are connected to the dotted drawn boxes. The

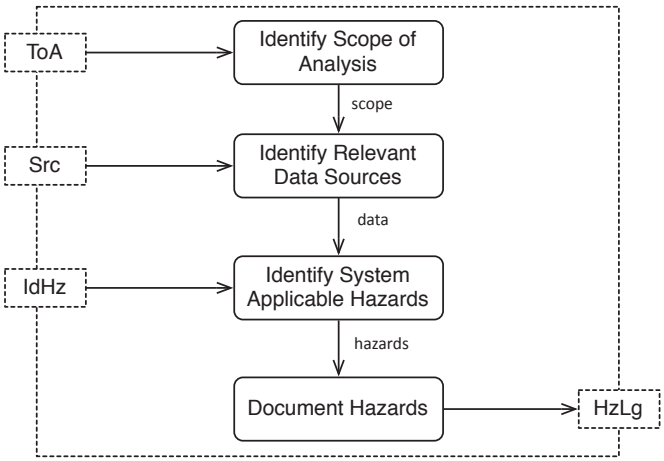


Figure 5.5: Process solution described in the Hazard Identification pattern (from Paper 2 [74])

dotted drawn boxes represent either an input or an output. The identifier within a box names a parameter. An arrow pointing away from a box indicates that the identified parameter is an input. An arrow pointing towards a box indicates that the identified parameter is an output. The remaining diagram elements in Fig. 5.5 represent the process solution in terms of a UML activity diagram. The SaCS specific annotations in Fig. 5.5 indicate in what way inputs are related to the different activities for performing hazard identification and how the result of one of the activities in this process represents an output of applying the pattern. Each activity in the diagram is further defined textually in the pattern definition. The textual description defines how the different inputs are used to perform the activities they are associated with. The textual description also details how the output is intended to be produced. The parameters that can be identified from Fig. 5.5 correspond to the parameters of the pattern signature presented in Fig. 5.4.

The SaCS specific annotations in Fig. 5.5 define a relationship between the parameters of the pattern and elements of the pattern solution that should be interpreted as follows:

- *ToA* is an input to the activity *Identify Scope of Analysis*.
- *Src* is an input to the activity *Identify Relevant Data Sources*.
- *IdHz* is an input to the activity *Identify System Applicable Hazard*.
- *HzLg* is an output of the activity *Document Hazards*.

The syntax for describing basic patterns in SaCS facilitates precise models of pattern compositions by demanding input and output parameters to be explicitly defined and thereby offering the possibility to easily connect patterns through their parameters.

5.3.2 Syntax of composite SaCS patterns

Fig. 5.6 exemplifies the use of a subset of the different graphical elements that may be applied for specifying a composite pattern. In Fig. 5.6, coloured arrows and text are added in order to identify the different kinds of SaCS specific graphical elements that are entirely visualised in different shades of grey. A composite consists of a declaration, which may be thought of as a signature of the composite, and its content. A horizontal line separates the declaration from the content. In the declaration, the composite pattern is given its name and the input and output parameters of the composite pattern are declared. In the content part of the composite, each pattern that is involved in the combination of patterns is identified as well as the relations between patterns. Relations connect patterns.

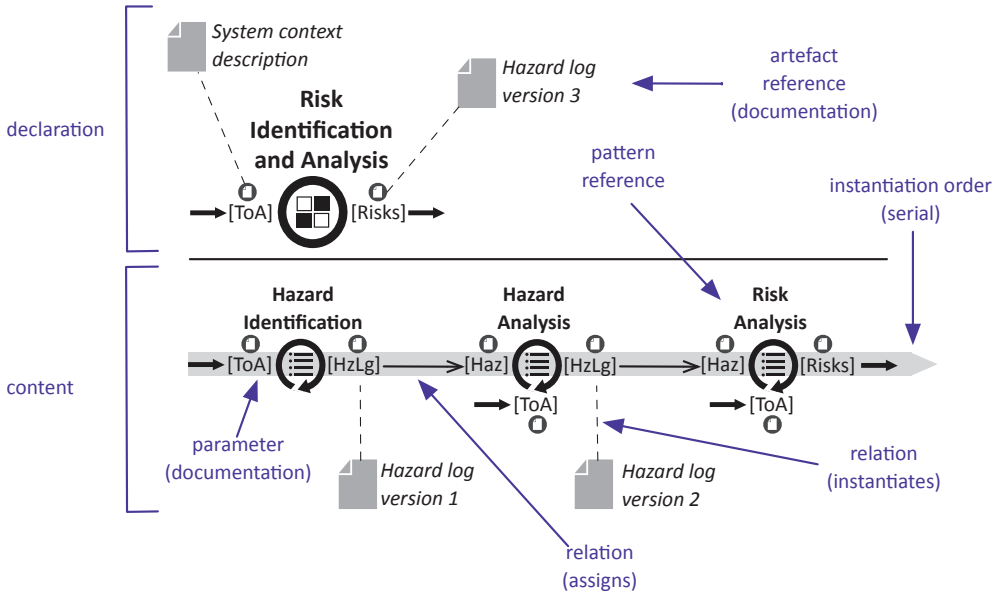


Figure 5.6: The main elements of a composite pattern explained

The input to *Risk Identification and Analysis* is listed inside square brackets, i.e., *ToA* (short for Target of Assessment). The arrow pointing towards the brackets symbolises input. The output of the pattern is also listed inside square brackets, i.e., *Risks*. The arrow pointing away from the brackets symbolises output. An icon placed adjacent to a parameter identifier denotes its type. The parameters *ToA*, *Risks*, *HzLg* (short for Hazard Log), and *Haz* (short for Hazards) in Fig. 5.6 are denoted as *documentation* parameters. Parameters are defined with either local or public accessibility. The inputs and outputs of a composite are always defined with public accessibility; symbolised with a thick black arrow.

The composite in Fig. 5.6 is annotated to indicate a particular instantiation of parameters by connecting them with references to development artefacts. An identifier provides a reference to a unique development artefact; an icon placed adjacent to the identifier classifies what kind of artefact that is referenced. We have assumed the existence of four development artefacts in Fig. 5.6. The belonging documentation of a

referenced development artefact may be found using the identifier as a key. A dotted drawn line represents an *instantiates* relation and connects an artefact to the parameter it gives value or represent. The following relationships are defined between development artefacts and parameters in Fig. 5.6:

- The document *System context description* instantiates *ToA*.
- The document *Hazard log version 3* instantiates *Risks*.
- The document *Hazard log version 1* instantiates *HzLg* (the output parameter of *Hazard Identification*).
- The document *Hazard log version 2* instantiates *HzLg* (the output parameter of *Hazard Analysis*).

The content part of *Risk Identification and Analysis* contains references to patterns and relations combining patterns. Patterns are referenced by their identifiers; icons placed adjacent to the identifiers for patterns classifies what kind of pattern that is referenced. Inputs and outputs of referenced patterns are denoted similarly in the content part of a composite as in the declaration. A parameter of a contained pattern is, if not otherwise specified, defined with local accessibility. A one-to-many relationship exists between inputs in the declaration part of a composite and similarly named inputs with public accessibility in the content part. The relationship is such that when *ToA* of *Risk Identification and Analysis* is instantiated (i.e., given its value by the defined relation to *System context description*) then every correspondingly named input parameter with public accessibility contained in the composite is also similarly instantiated. A one-to-one relationship exists between an output parameter in the declaration part of a composite and a correspondingly named output parameter with public accessibility in the content part. The relationship is such that when *Risk* of *Risk Analysis* is produced then *Risk* of *Risk Identification and Analysis* is similarly produced.

The arrows connecting patterns in the content part of *Risk Identification and Analysis* represent two instances of the *assigns* relation. Relations may be used to define a specific combination of patterns and then operates on the parameters of the respective patterns that are combined. The *assigns* relations also indicate the instantiation order as an output from the instantiation of a pattern must be available before it can be used as an input to the instantiation of a connected pattern. The *assigns* relations defined within *Risk Identification and Analysis* express that:

- The output *HzLg* of the pattern *Hazard Identification* is assigned to the input *Haz* of the pattern *Hazard Analysis*.
- The output *HzLg* of the pattern *Hazard Analysis* is assigned to the input *Haz* of the pattern *Risk Analysis*.

A serial order for instantiating patterns is indicated by the grey wide arrow in Fig. 5.6 where patterns close to the start of the arrow shall be instantiated prior to patterns close to the tip. However, the instantiation order visualisation is redundant and may be removed as the expected order may be deduced from the *assigns* relations.

Fig. 5.7 exemplifies a modularised specification where the composite *Define Safety Requirements* models a relationship between the composite *Risk Identification and*

Analysis defined in Fig. 5.6 and the basic pattern named *Establish System Safety Requirements*. It is easy to modularise a pattern specification by defining within separate composite a subset of the total number of combined patterns, and further combine specification fragments into a complete pattern structure.

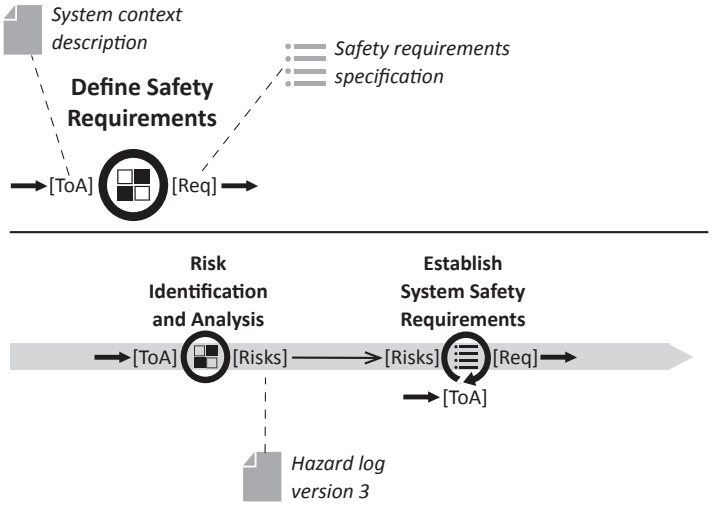


Figure 5.7: Example on referencing a composite pattern within the definition of a composite pattern

The notation for expressing composite patterns consists of the following main modelling elements:

- *Pattern reference*: Fig. 5.8 presents the icons for the different kinds of patterns defined in SaCS. A pattern reference consists of a unique identifier in a **bold** font and an icon classifying the pattern referenced.

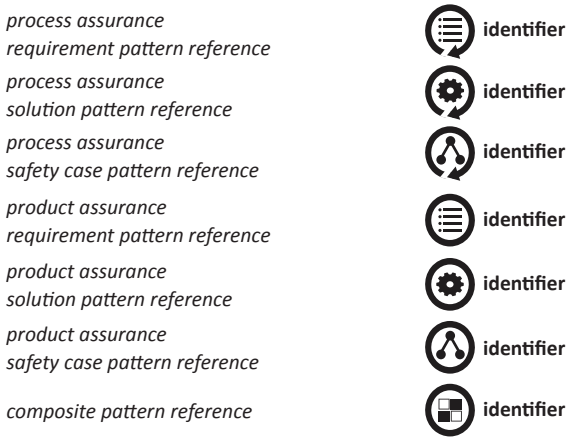


Figure 5.8: The icons for the different kinds of pattern references in SaCS

- *Parameter*: Fig. 5.9 presents the icons for the different kinds of parameters defined in SaCS. A parameter consists of an identifier and an icon classifying the parameter. Within a composite, the parameters of a pattern are listed inside square brackets and placed adjacent to the icon that classifies the pattern. The *documentation parameter* is a general classification and represent a parameter that cannot be classified as a *requirement parameter*, *design parameter* or *safety case parameter*.

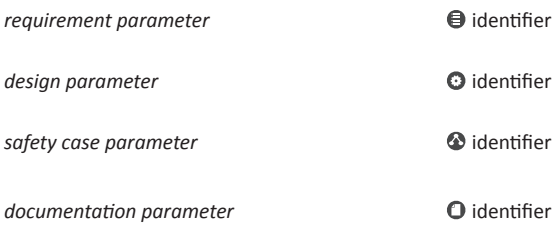


Figure 5.9: The icons for different kinds of parameters in SaCS

- *Relation*: Fig. 5.10 presents the symbols for different kinds of relations defined in SaCS. A relation denotes a relationship between elements in a composite pattern. The *instantiates* relation is used to associate an artefact with a parameter indicating that the artefact instantiates the parameter. The *assigns* relation models a data flow between patterns where the output of one pattern is used as an input to a second pattern. The *combines* relation is used to denote that the outputs of the patterns that are related are combined into a set consisting of the union of all outputs. The *details* relation is used to denote that an output of a pattern is detailed by the output of a related pattern. The *satisfies* relation is used to denote that an output of a pattern (typically represented by a requirement parameter) is satisfied by the output of a related pattern (typically represented by a design parameter). The *demonstrates* relation is used to denote that an output (typically represented by a safety case parameter) demonstrates safe the output of a related pattern (typically represented by a design parameter).

Figure 5.10: The symbols for the different kinds of relations in SaCS

- *Artefact reference*: Fig. 5.11 presents the different kinds of artefact references defined in SaCS. An artefact reference consists of an unique identifier and an icon classifying what kind of artefact that is referenced. Artefact references are used for denoting a specific representation of parameters.

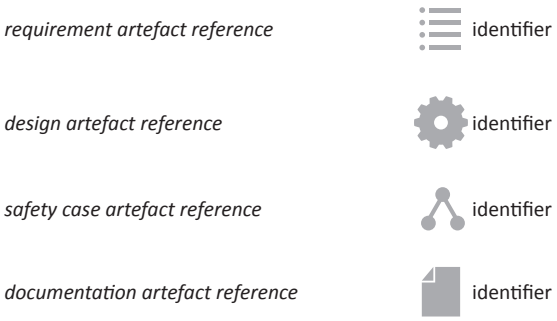


Figure 5.11: The icons for different kinds of artefact references in SaCS

- *Instantiation order*: A composite pattern may include symbols as guidance to the user on the proper instantiation order of patterns. In the serial instantiation case of Fig. 5.12 there are two composite patterns A and B where A shall be instantiated before B. A general rule is that patterns placed closer to the starting point of the arrow are instantiated prior to patterns placed close to the tip of the arrow. In the parallel instantiation case of Fig. 5.12, no specific ordering of the patterns A and B is assumed and thus the respective pattern references are placed on two separate arrows.

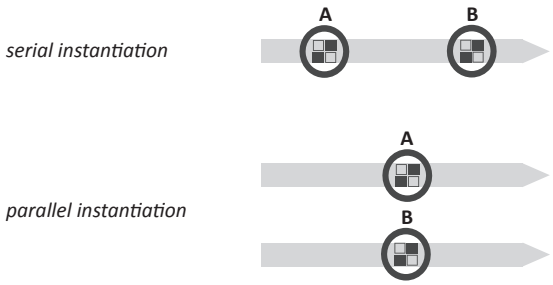


Figure 5.12: The symbols for the different kinds of instantiation orders in SaCS

5.3.3 Structured semantics

A structured semantics is defined in order to facilitate a common understanding of the meaning of patterns on the basis of their definition and rules for translation.

A detailed description of the syntax and semantics of the SaCS pattern language is presented in Paper 1 (described in Chapter 9). We outline the procedure here.

The method for providing the semantics of SaCS patterns is adapted from [36] and is performed in two steps:

1. the translation of a SaCS pattern into its textual syntax; and

2. the translation of the textual syntax into a meaningful text in English.

The textual syntax of SaCS patterns is defined by the use of Extended Backus-Naur Form (EBNF) [97]. The semantics as a sentence or paragraph in English is provided by functions that take fragments of a pattern defined by its textual syntax as input and provides the meaning in English as output.

The textual syntax for pattern references of the same type as the one named *Establish System Safety Requirements* in Fig. 5.7 is expressed in Fig. 5.13.

In Fig. 5.13, a term representing a name of the graphical element is given to the left followed by the associated graphical syntax of the named element. The right pointing arrow should be interpreted "translates to" and separates the graphical syntax from the textual syntax (described in the Extended Backus-Naur Form). The term *identifier* is not defined but can be assumed to be an alphanumeric string. A terminal symbol (written in a **bold** font to increase readability) is used to define a syntactical element.

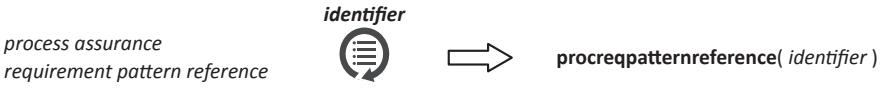


Figure 5.13: Example on the correspondence between graphical and textual syntax of a pattern reference in SaCS

The symbols ⟦ ⟧ below denote a semantic rule and represents a function that takes a fragment of a pattern defined according to its textual syntax as input and provides a sentence or a paragraph in English as output. A semantic rule for the syntactical element presented in Fig. 5.13 is:

$$⟦\ \textbf{procreqpatternreference}(identifier)\ ⟧ = \text{the process pattern } identifier$$

Fig. 5.14 presents a fragment of the composite pattern in Fig. 5.6 and the corresponding translation of this fragment in terms of a textual syntax expressing the relationship modelled between the two patterns *Hazard Identification* and *Hazard Analysis*.

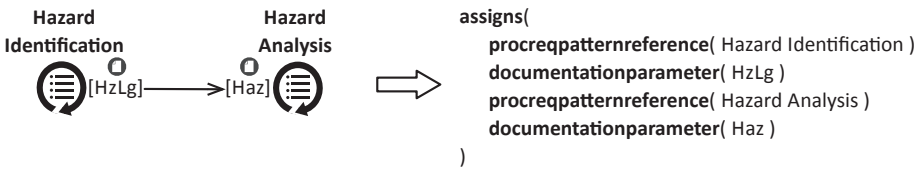


Figure 5.14: A fragment of the composite in Fig. 5.6 and associated textual syntax

Below there is given a semantic rule for translating an *assigns* relation as exemplified in Fig. 5.14 into a paragraph in English. In the semantics, the term *pr* represents a *pattern reference* (note that the *process assurance pattern reference* presented above is one specific kind of *pattern reference*), *p* represents a parameter, and *id*( ) represents a function that takes parameters of different kinds as input and provides as output the associated identifier. A semantic rule for the *assigns* relation is defined as:

$$[\![\ \textbf{assigns}(pr_1, p_1, pr_2, p_2)\ ]\!] = \begin{array}{l} id(p_1) \text{ is assigned to } id(pr_2) \text{ where:} \\ id(p_1) \text{ is the output of } [\![\ pr_1\ ]\!]. \\ id(p_2) \text{ is input to } [\![\ pr_2\ ]\!]. \end{array}$$

By applying the semantic rules provided for translating the *assigns* relation and the *process assurance requirement pattern reference* on the textual syntax presented in Fig. 5.14 we get the following English paragraph (slightly formatted to increase readability):

> *HzLg* is assigned to *Haz* where:
>
> – *HzLg* is the output of the process pattern *Hazard Identification*.
>
> – *Haz* is input to the process pattern *Hazard Analysis*.

The semantics of a complete SaCS pattern is translated into its meaning as understandable paragraphs in English in a similar manner as exemplified above on the *assigns* relation by translating different syntactical elements systematically.

5.4 The library of SaCS patterns

We introduce the library by presenting in Section 5.4.1 the definition of a basic pattern as an example of its content. Thereafter, Section 5.4.2 discusses the content of the library more generally.

5.4.1 An example definition of a pattern from the library

In the following, a slightly formatted version of the pattern *Establish System Safety Requirements* documented in [74] is reproduced. The pattern is described as a sequence of named sections according to the format outlined in Table 5.1. The section names are presented in a **bold** font. The section "Known Uses" (see Table 5.1) is optional according to the detailed syntax of basic SaCS patterns (see Paper 1). In the following pattern definition, known uses is not defined.

Name: Establish System Safety Requirements

Pattern Signature: *Establish System Safety Requirements* is defined with the signature illustrated in Fig. 5.15. In Fig. 5.15, the following abbreviations are used for denoting the parameters of the pattern:

- *ToA* is short for Target of Assessment.
- *Reg* is short for Regulations.
- *Risks* is not abbreviated; represents the documentation of the risks associated with the application of *ToA* in its intended context.
- *Req* is short for Requirements.

Intent: Support the specification of system safety requirements *Req* on the basis of a risk-based approach. The safety requirements describe the required measures to be satisfied by the system *ToA* to assure the necessary safety integrity. The general approach for defining safety requirements is to define them on the basis of the result

Figure 5.15: Establish System Safety Requirements – Pattern Signature

of a risk assessment *Risks*, especially the mitigations identified as means to reduce risk to an acceptable level. The pattern describes the general process of capturing the requirements that must be satisfied in order to assure safety.

Applicability: The *Establish System Safety Requirements* pattern is intended for the following situations:

- When the system under construction may negatively affect the overall system safety.
- When there are identified measures that can mitigate identified risks and can be used as input to the specification of safety requirements.

Problem: The main aspects relevant to address when establishing the safety requirements are:

- *Characteristics*: To define the system characteristics to be satisfied such that the occurrence of unwanted events are minimised or avoided.
- *Functions*: To define the safety functions that assures safe operations.
- *Constraints*: To define the functional constrains that sufficiently delimit potentially hazardous operations.
- *Environment*: To define the operational environment that ensures safe operations.
- *Compliance*: To define the requirements that are required to be satisfied in order to comply with laws, regulation and standards, as a minimum the mandatory requirements related to assurance of safety. These requirements include requirements on applying some specific development process, perform certain activities, or make use of specific techniques.

Process Solution: Fig. 5.16 illustrates the *Establish System Safety Requirements* process specified using a UML activity diagram.
The input parameters associated with the activity diagram may be interpreted as follows:

- *ToA* (Target of Assessment): represents the target system for which safety requirements should be established.
- *Reg* (Regulations): represents any source of information describing mandatory or recommended practices (e.g. as provided in laws, regulations or standards) valuable for identifying risk reducing measures.

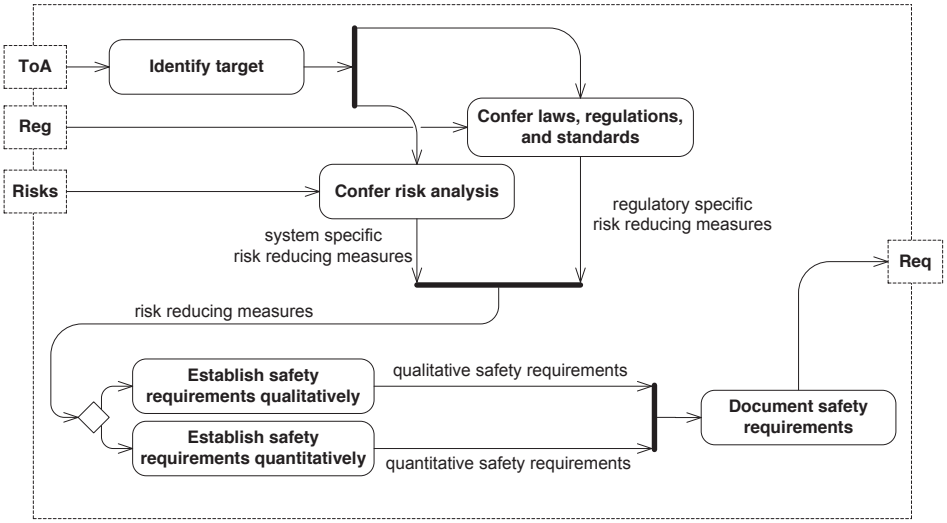


Figure 5.16: Establish System Safety Requirements – Process Flow

- *Risks*: represents risks associated with the target system.

The main activities serve the following purpose:

- *Identify target*: the intent of the activity is to identify *ToA*. The description of the target should as a minimum include a definition of the system and its boundaries, its operational profile, functional requirements, and safety integrity requirements.

- *Confer laws, regulations, and standards*: the intent of the activity is to capture all relevant data (requirements for risk reducing measures) from relevant sources (normative references) in order to outline the set of risk reducing measures that shall be met by compliance. Each source is inspected in order to identify, as a minimum, the mandatory risk reducing measures that shall be met in order to be compliant.

- *Confer risk analysis*: the intent of the activity is to capture all the relevant data on risk analysis of the system that is under construction in order to outline the system specific risk reducing measures that shall be met.

- *Establish safety requirements qualitatively*: the intent of the activity is to define safety requirements on the basis of those identified risk reducing measures required applied, and which can be demonstrated fulfilled with qualitative reasoning.

- *Establish safety requirements quantitatively*: the intent of the activity is to define safety requirements on the basis of those identified risk reducing measures required applied, and which can be demonstrated fulfilled with quantitative reasoning.

- *Document safety requirements*: the intent of the activity is to detail all relevant information with respect to the requirements in a system safety requirements specification. For each requirement defined in the requirement specification, information detailing what influenced its definition should be provided, e.g., the associated risks, assumptions, calculations, and justifications.

Instantiation Rule: An artefact *Req* (see Fig. 5.15 and Fig. 5.16) is the result of a process that instantiates the *Establish System Safety Requirements* pattern if:

- *Req* is a set of requirements.
- *Req* is a result of applying a process illustrated in Fig. 5.16 and described in Section "Process Solution". The process is initiated by an activity on describing the target *ToA*. Once a description of the target system and its operational context is provided, the next activities shall identify the risk reducing measures to be applied to the target by conferring relevant laws, regulations and standards as well as the result of target specific risk analysis for guidance. Once all the relevant risk-reducing measures are identified, these shall be used as a basis to define the requirements to be met by the target system or by the process to be followed while developing the target. The requirements are defined quantitatively or qualitatively depending on the nature of the risk reducing measure that is addressed. The requirements are documented in a requirement specification *Req*.
- Every requirement of *Req* is traceable to relevant risks (identified by the instantiation of *Risks*), and/or regulatory requirements (identified by the instantiation of *Reg*).
- Every requirement of *Req* is justified such that any assumptions, calculations, and assessments that support the specification of the requirement as a safety requirement are provided.

Related Patterns: The *Establish System Safety Requirements* pattern is related to other patterns in the following manner:

- May succeed the *Risk Analysis* pattern that supports identifying risks. The *Establish System Safety Requirements* may be applied as support for defining the requirements to be fulfilled in order to reduce risk to an acceptable risk level.
- May be used in order to detail requirements for the design that is a result of an instantiation of a design pattern.

5.4.2 General description of the content in the library

The library consists of 26 predefined patterns, known as basic pattern, providing guidance on different aspects of conceptual safety design. The library also consists of any composite pattern that a user includes in the library. In the following, we give a short presentation of a few of the basic patterns in the library and explain how they can be combined.

Establish Concept is a pattern that describes the process of establishing an initial specification of the purpose of a system under construction and its intended use. Although *Establish Concept* can be suitable to apply as guidance for defining an initial

system concept description containing an overview of the intended operating context, main functionality, and intended use of a given system, a more detailed assessment must be performed in order identify whether there are some safety constraints that needs to be addressed. While *Hazard Identification* provides guidance on the process of identifying hazards associated with the use of a system in its intended context, *Hazard Analysis* provides guidance on the process of finding the potential causes of hazards. Hazard identification and analysis can be supported by several different methods, the essence of two well known methods within safety engineering are captured in the patterns *FMEA* (short for Failure Modes and Effect Analysis) and *FTA* (short for Fault Tree Analysis). The result from hazard analysis forms a basis for identifying risks. While *Risk Analysis* defines the process of defining the risks associated with the use of a system in a given context, *Establish System Safety Requirements* (fully described in Section 5.4.1) defines the process of specifying safety requirements on the basis of the results from risk analysis and other sources such as laws, regulations, and standards. The system requirements drive the system design. An important subset of the system requirements is the safety requirements. The safety requirements are especially important as they specify what needs to be met in order to achieve a sufficiently safe system. The choice of design pattern to use as a design basis should be motivated by the needs as defined by the requirements. Furthermore, once a system design is derived, it must be assured as a minimum that the designed system satisfies relevant safety requirements. Regarding guidance on system design, two patterns are offered named *Dual Modular Redundant* and *Trusted Backup*, but more are easily found in the pattern literature. Several kinds of strategies can be applied for arguing that the system under construction is sufficiently safe for its intended purpose. *Safety Requirements Satisfied* facilitates the specification of the structure for a safety case arguing that every safety requirement is sufficiently addressed.

The patterns outlined above provide guidance on different and complementing concepts relevant for developing safety critical systems. Although each pattern can be used standalone, the patterns in the library are defined with the intention of being used together.

5.5 SaCS exemplified

Fig. 5.17 presents an example of the integration of the three artefacts of this thesis into the SaCS method. In Fig. 5.17, arrows are used to indicate the flow between the use of the SaCS process, Artefact 1, and the use of Artefact 2 and 3 as support in performing the activities of the process. Filled arrows indicate the inputs and outputs from the application of the SaCS method. The dotted drawn arrow indicates that a user may choose to feed the composite pattern provided as a result of applying SaCS back into the library of patterns.

The SaCS method is in the following explained by exemplifying the steps (1) to (17) from Fig. 5.17.

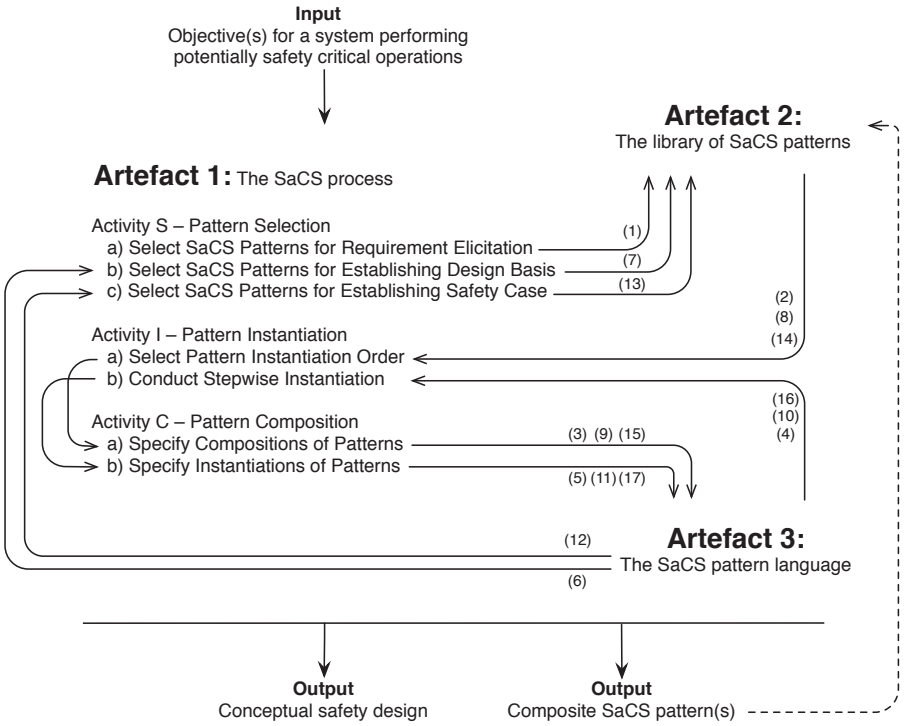


Figure 5.17: Example integration of the three artefacts into the SaCS method

Fig. 5.18 presents references to development documents that are available to an assumed user in the following example. *Level crossing concept description* is a document assumed to define the main functionality of an interlocking system for separating train movements from road traffic movements at a railway level crossing. Furthermore, *Hazard log* is a document assumed to describe the result from an initial hazard analysis of the level crossing concept. There are patterns in the library that facilitates the definition of an initial concept as well as hazard identification and analysis, but we nevertheless assume the presence of these documents as a starting point in the exemplification of the use of the SaCS method. The end result is expected to be a conceptual safety design of a railway level crossing interlocking system.

Figure 5.18: Representation of references to development documentation

Step (1): Fig. 5.19 illustrates the fragment of a map for pattern selection presented earlier with the addition of annotations to indicate in which steps of this example we use the map. A user traverses the pattern selection map in the order as indicated by the arrows and considers whether a pattern is relevant for application by considering its definition.

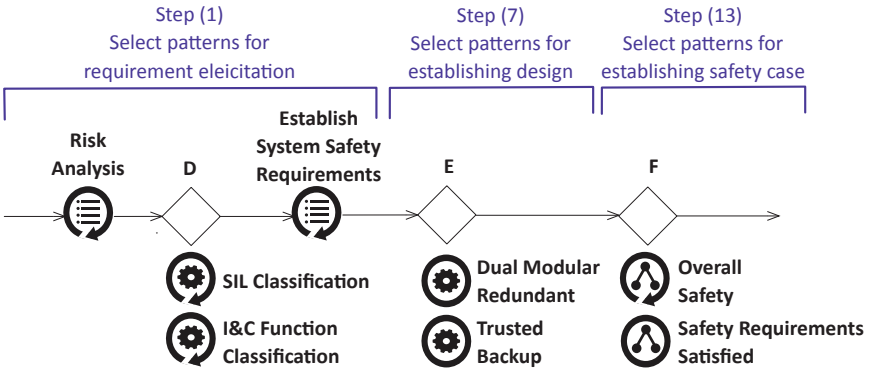


Figure 5.19: Pattern selection map – revisited

By the use of Fig. 5.19, the user starts the pattern selection activity from the left and identify *Risk Analysis* as the first pattern that should be considered as support. By inspecting the definition of *Risk Analysis*, the user identifies that the pattern describes the process of performing risk analysis on the basis of the results from hazard analysis. As a hazard log is already present, the user selects the pattern as support for the classification of risks. In choice D, two patterns are identified that offer guidance on alternative methods for the classification of functions, named *SIL Classification* (SIL is short for Safety Integrity Level) and *I&C Function Classification* (I&C is short for Instrumentation and Control), respectively. The user selects *SIL Classification* as support as it defines an approach to the classification of functions commonly applied within the railway domain whereas *I&C Function Classification* is applicable within the nuclear power production domain. The pattern *Establish System Safety Requirements*

was presented in details earlier. The pattern uses the result from risk analysis as input to the specification of safety requirements. The user selects these three patterns as support for the elicitation of requirements in Step (1), but will return to the selection map in Step (7) and Step (13) related to the selection of patterns for establishing design and safety case.

Step (2)-(3): Fig. 5.20 presents how an assumed user combines the three patterns selected in the previous step according to the syntax of the SaCS pattern language. The order in which these patterns should be instantiated is indicated in the pattern selection map presented in Fig. 5.3. The order can also be found by inspecting the "Related Patterns" sections of the pattern definitions. In Fig. 5.20, the order is indicated by the wide grey arrow in the background indicating that *Risk Analysis* and *SIL Classification* can be instantiated in parallel and prior to *Establish System Safety Requirements*. As described earlier, the symbols [ ] indicate a parameter list. The identifiers within these symbols identify parameters. The parameters are abbreviated as follows: *ToA* is short for Target of Assessment, *Haz* is short for Hazards, *Req* is short for Requirements, *FncCat* is short for Function Categorisation, *ClsCr* is short for Classification of Criticality, and *Risks* is not abbreviated. The small icons adjacent to parameters classify the parameters. The thin arrows represents three instances of an *assigns* relation. The semantics of an *assigns* relation was explained in Section 5.3.2.

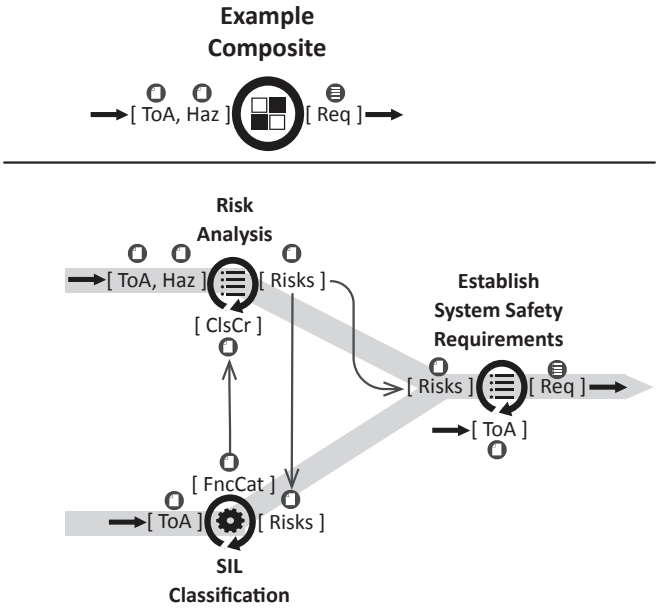


Figure 5.20: Composite that can be instantiated into a specification of requirements

Step (4)-(5): Fig. 5.21 is identical to Fig. 5.20 with the addition of annotations indicating the instantiation of patterns. In Fig. 5.21, the instantiation of the composite is documented such that *Level crossing concept description* represents the documentation associated with the input parameter *ToA* and *Hazard log* is associated with the input *Haz*. We have assumed that the user has instantiated the composite to produce three different documents. The outcome of instantiating the composite is defined as

being represented by a requirements specification known as *Safety requirements specification*. Furthermore, two intermediate documents to the requirements specification is produced where *Classification of functions* is associated with the output parameter *FncCat* of *SIL Classification* and *Risk analysis results* is a document associated with the output *Risks* of *Risk Analysis*.

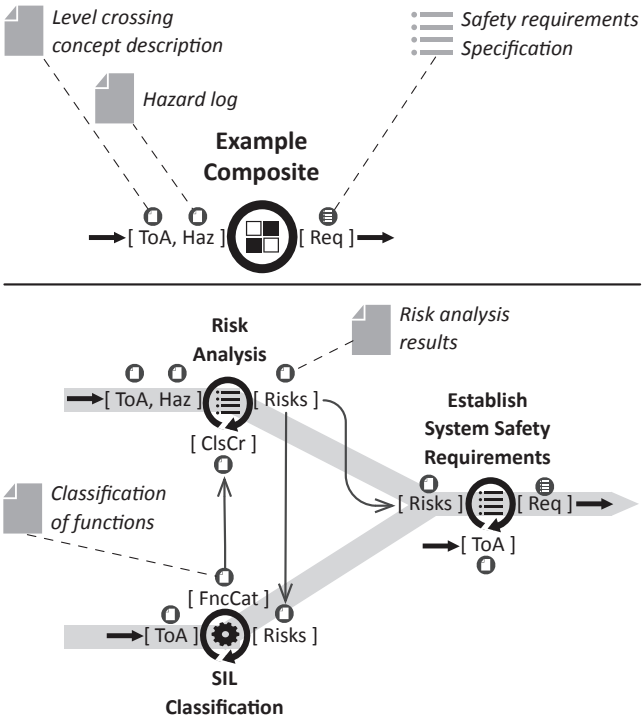


Figure 5.21: Composite specifying its instantiation into a specification of requirements

Step (6)-(8): Once the user has specified the requirements for the system under construction, enough information should be available for selecting an appropriate design pattern to use as a basis for establishing the system design. Thus, in Step (7) the user continues the pattern selection activity from where it was temporarily stopped in Step (1). In the pattern selection map presented in Fig. 5.19, a choice E follows *Establish System Safety Requirements*. In choice E, the design patterns *Dual Modular Redundant* and *Trusted Backup* are indicated as alternatives. We assume that the user find *Dual Modular Redundant* to offer the most beneficial design after an evaluation of the ability of the respective design solutions described within these two patterns to satisfy requirements. The user selects *Dual Modular Redundant* as support for system design.

Step (9)-(11): We postpone Step (9) and Step (11) for later in order to avoid unnecessary repetitions of similar pattern compositions. In Step (10), we assume that the user makes use of the requirements derived earlier as input for detailing a system design supported by *Dual Modular Redundant* selected in the previous step. We further assume that the user instantiate *Dual Modular Redundant* according to its instantia-

tion rule to produce a specification that can be referenced graphically as presented in Fig. 5.22.

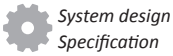


Figure 5.22: Representation of a reference to a design specification

Step (12)-(14): Once the system design is established, enough information should be available for the user to start the detailing of how safety will be argued. Thus, in Step (13) the user continues the pattern selection activity from where it was temporarily stopped in Step (7), leading to choice F in Fig. 5.19. In choice F, the user inspects the pattern definitions of the different proposed patterns. We assume that the user finds *Overall Safety* as a suitable starting point. The pattern provides guidance on arguing safety from a quality management, safety management, as well as a technical safety perspective. The railway standard EN 50129 [30] requires that the three perspectives are explicitly addressed in a safety case for railway signalling systems.

Step (15)-(16): While we postpone Step (15) for later in order to avoid unnecessary repetitions, we assume in Step (16) that the user makes use of the system design derived earlier as a definition of the target for which a safety case shall be defined. We further assume that the user instantiate *Overall Safety* selected in the previous step according to its instantiation rule to produces a safety case that can be referenced graphically as presented in Fig. 5.23.

Figure 5.23: Representation of a reference to a safety case

Step (17): Fig. 5.24 extends Fig. 5.21 to also specify the use of the patterns *Dual Modular Redundant* and *Overall Safety*. In Fig. 5.24, the parameters of the patterns introduced are abbreviated as follows: *S* is short System, *ToD* is short for Target of Demonstration, and *Case* is short for Safety Case. The *satisfies* relation (see Fig. 5.10) expresses that a design *S* (represented by the *System design specification*) shall satisfy the requirements in *Req* (represented by *Safety requirements specification*). Furthermore, *S* of *Dual Modular Redundant* represents the target of demonstration as defined by the *assigns* relation connecting *S* with *ToD* of *Overall Safety*. The outcome *Case* (represented by *Safety case specification*) of *Overall Safety* is related to *S* of *Dual Modular Redundant* with a *demonstrates* relation. The *demonstrates* relation expresses that *Case* is a safety demonstration for *S*.

In the example, the SaCS method is assumed applied for developing a railway level crossing system concept where the composite pattern expressed in Fig. 5.24 represent one of the results. A second result is the conceptual safety design. The conceptual safety design is the result of instantiating the composite pattern. In the example, the triple that represents the conceptual safety design is assumed represented by the documentations referred to within Fig. 5.24 as *Safety requirements specification*, *System design specification*, and *Safety case specification*.

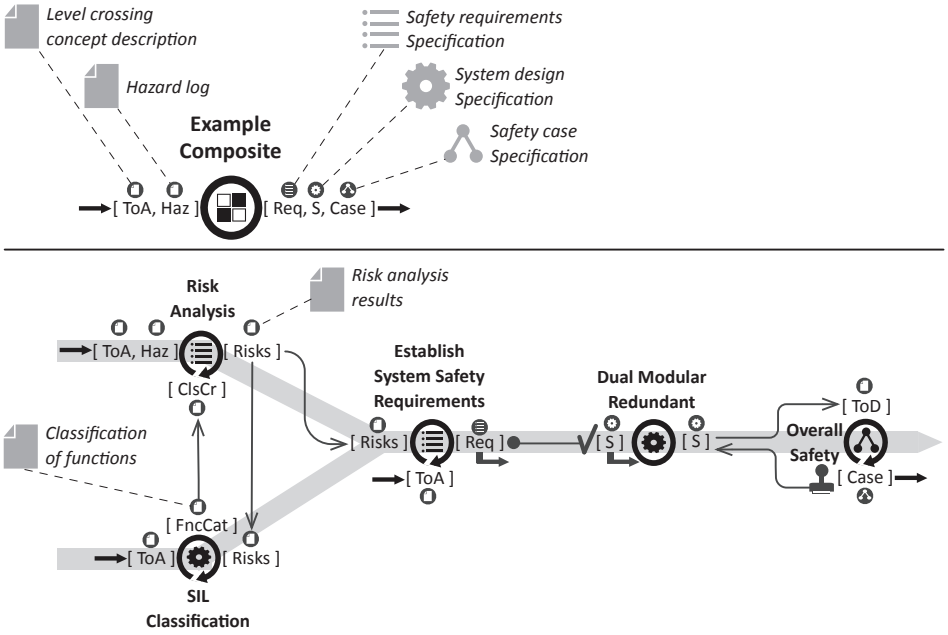


Figure 5.24: Composite specifying its instantiation into a conceptual safety design

Chapter 6

Overview of research papers

The main results of our work are documented in the papers presented in Part II. This chapter provides publication details for each of these papers.

6.1 Paper 1: Syntax & Semantics of the SaCS Pattern Language

Authors: André Alexandersen Hauge and Ketil Stølen.

Publication status: Technical report HWR-1052, OECD Halden Reactor Project, Institute for energy technology, Halden Norway, 2013. The report represents an extension and consolidation of the paper titled "SACS: A Pattern Language for Safe Adaptive Control Software" published in *Proceedings of the 18th Conference of Pattern Languages of Propgrams (PloP'11)* [70].

My contribution: André Alexandersen Hauge was the main author, responsible for about 90% of the work.

Main topics: The report provides a detailed description of the syntax and semantics of the SaCS pattern language. The syntax of SaCS patterns is described by the use of Extended Backus Naur Form (EBNF). The semantics is described as a systematic translation of the textual syntax of SaCS patterns into paragraphs in English. The translation of SaCS patterns into their meaning in English is exemplified on a number of patterns where each step of the translation process is detailed.

6.2 Paper 2: A Pattern-based Method for Safe Control Conceptualisation Exemplified Within Nuclear Power Production

Authors: André Alexandersen Hauge and Ketil Stølen.

Publication status: A short version of the paper with the same title is published in the *Proceedings of 31st International Conference on Computer Safety, Reliability and Security (SAFECOMP'12)* [71]. The full version, which is included in this thesis, is published as a technical report, namely HWR-1029 rev 2, OECD Halden Reactor Project, Institute for energy technology, Halden, Norway, 2014.

My contribution: André Alexandersen Hauge was the main author, responsible for about 90% of the work.

Main topics: The paper demonstrates and presents experiences from the use of the SaCS method and pattern language in a case on the conceptualisation of a safety design for a load following reactor control system. In countries where a large share of the consumed electricity comes from nuclear power plants, there is a need for controlling the electricity production in accordance with demand that is varying with time, referred to as operating in a load following mode. An adaptable system is conceptualised in the report. Adaptability is here introduced as a means to automatically calibrate the controller performing control rod control during operation in order to accommodate fuel burn up. The result is described as a set of requirements for the load following controller, a technical design of the load following controller system and an outline of a safety case for demonstrating that a system based on the conceptual safety design is sufficiently safe for its purpose.

6.3 Paper 3: Developing Safe Control Systems Using Patterns for Assurance

Authors: André Alexandersen Hauge and Ketil Stølen.

Publication status: A short version of the paper with the same title is published in the *Proceedings of the 3rd International Conference on Performance, Safety and Robustness in Complex Systems and Applications (PESARO'13)* [72]. The full version of the paper, which is included in this thesis, is available as HWR-1037 rev 2, OECD Halden Reactor Project, Institute for energy technology, Halden, Norway, 2014.

My contribution: André Alexandersen Hauge was the main author, responsible for about 90% of the work.

Main topics: The paper demonstrates and presents experiences from the application of the SaCS method and pattern language in a case on the conceptualisation of a safety design for a railway interlocking system. A railway interlocking system is a typical safety critical system. A failure in a railway interlocking system may lead to unacceptable consequences. The conceptual safety design is built systematically in manageable steps by the application of SaCS. Each step is exemplified. The resulting conceptual safety design is described as a specification of a set of requirements for the interlocking system, a system design defined in accordance with requirements, as well as an outline of a safety case for demonstrating that a system based on the conceptual safety design is sufficiently safe for its purpose.

6.4 Paper 4: An Analytic Evaluation of the SaCS Pattern Language – Including Explanations of Major Design Choices

Authors: André Alexandersen Hauge and Ketil Stølen.

Publication status: The paper is published in the *Proceedings of the 6th International Conference on Pervasive Patterns and Applications (PATTERNS'14)* [76].

My contribution: André Alexandersen Hauge was the main author, responsible for about 90% of the work.

Main topics: The paper presents an analytic evaluation of the SaCS pattern language for the development of conceptual safety designs. A framework for evaluating modelling languages is used to conduct the evaluation. The quality of a language is within the framework expressed by six appropriateness factors. A set of requirements is associated with each appropriateness factor. The extent to which these requirements are fulfilled are used to judge the quality. The fulfilment of the requirements formulated for the SaCS language is discussed on the basis of the theoretical, technical, and practical considerations that were taken into account and shaped the SaCS language.

Chapter 7

Discussion

This chapter is divided into two sections. In Section 2.2, we defined the success criteria that should be fulfilled by each invented artefact in order for our overall objective to be met. In Section 7.1, we discuss to what extent the invented artefacts satisfy each success criterion. In Chapter 4, we gave an overview of state-of-the-art of relevance for this thesis. In Section 7.2, we characterise in more detail how our work extends and improves this state-of-the-art.

7.1 Fulfillment of the success criteria

The overall objective of this thesis as stated in Section 1.1 is to develop a method and a pattern language that is:

1. Well-suited for developing conceptual safety designs.
2. Applicable within an industrial context within acceptable effort.
3. Comprehensible for its intended users.

Section 2.2 presents the expectations in terms of success criteria for each of the three artefacts developed in this thesis in order to meet the overall objective. In the following subsections, the fulfilment of each success criteria presented in Section 2.2 is discussed.

7.1.1 The SaCS process

Success criterion 1 *The SaCS process provides satisfactory guidance for its intended users, which are system engineers, safety engineers, hardware and software engineers.*

Fig. 5.17 presents the SaCS process and how it can be applied together with the library of SaCS patterns and the SaCS pattern language as support for conceptual safety design. Fig. 5.17 shows that the process defines three main activities with sub-activities. As explained in Section 5.1, the SaCS process interleaves these three main activities; hence, many correct sequences for applying the process exist.

The application of the process has been thoroughly exemplified in two case studies within different domains. Paper 2 (described in Chapter 10) describes the application of the SaCS process in a case study on the conceptualisation of a nuclear power plant control system. Paper 3 (described in Chapter 11) describes the application of the SaCS

process for conceptualisation of a railway interlocking system. Each paper describes the application of each step of the SaCS process and its result in terms of a conceptual safety design according to definition.

We believe that the steps of the SaCS process as presented in Section 5.1 should be easily understood by the intended users of the SaCS process due to their expected formal background and experience within engineering disciplines, as well as due to the simplicity of the process itself. Moreover, both Paper 2 and Paper 3 provide extensive guidance on the application of each of the activities of the SaCS process and describe how pattern selection is supported by a pattern selection map, how pattern composition is supported by the pattern language, and how pattern instantiation is supported by instantiation rules. Paper 1 (described in Chapter 9) details the syntax and semantics of the SaCS pattern language and also provides guidance on the pattern composition step.

Fig. 5.3 presents a fragment of a larger pattern selection map supporting the systematic selection of patterns from the library. The patterns referenced in the pattern selection map are ordered in a sequence. The sequence of patterns also includes choices. In the sequence, a choice follows a pattern for which there exists several patterns that are natural to consider to support development. In a choice, more than one pattern can be selected as support. The different patterns in SaCS, as well as their sequence as presented in the pattern selection maps in Paper 2 and Paper 3, are influenced by the safety literature. One of the influences is the railway standard EN 50126 [28], and another is the generic safety standard IEC 61508 [93]. The system life-cycle of EN 50126 is presented in Fig. 2.1 as a sequence of phases. The first six phases in the life-cycle presented in Fig. 2.1 are to a large degree represented in the sequence of SaCS patterns in Fig. 4 of Paper 2. The first pattern in the sequence presented in Fig. 4 of Paper 2 is named *Establish Concept* and supports the specification of an initial system concept for a system under development. This pattern should be instantiated prior to patterns supporting a detailed elicitation of requirements, safety assessment, system design, and safety argumentation. This practice is inline with safety literature such as the standards EN 50126 and IEC 61508. The instantiation of *Establish Concept* in a context should provide a result that guides the user in selecting the most suitable pattern at the next selection point in Fig. 4 of Paper 2. The user needs to inspect the definitions of candidate patterns in order decide whether a pattern is applicable for solving the challenges of the user. The patterns are organised in a sequence in the selection map that to a large degree mirrors the early stages of the development processes promoted by highly regarded safety standards and guidelines.

Although we argue that the SaCS process is comprehensible for a user unfamiliar with SaCS on the basis of the guidance provided in Paper 1, Paper 2 and Paper 3, this has not been evaluated. Empirical evaluation is needed in order to validate if the SaCS process is comprehensible for its intended users in general.

Based on the above discussion, we believe that the simplicity of the SaCS process and the extensive documentation demonstrating its practical use provide sufficient guidance for its intended users, but further suggest that this should be empirically evaluated.

Success criterion 2 *The application of the SaCS process results in conceptual safety designs that are: a) in accordance with safety objectives; b) at a sufficient level of detail; and c) easy to use.*

As previously mentioned, the SaCS process has been applied to two different cases documented in Paper 2 (described in Chapter 10) and Paper 3 (described in Chapter 11).

Regarding a): Section 10 of Paper 2 and Section 10 of Paper 3 describe the three different specifications that represent the conceptual safety design in each case. In both cases:

- Section 10.1 contains a description of the safety objectives in the form of safety requirements. The safety requirements are established by the application of a similarly named composite identified as *Safety Requirements*. The composite describes the systematic application of selected patterns from the library as support for the elicitation and specification of safety requirements.
- Section 10.2 contains the description of a system design that accommodates the safety requirements. Although the system design is defined in accordance with the safety requirements, it is mainly the safety case presented in Section 10.3 that expresses to which extent the safety requirements are satisfied.
- Section 10.3 contains a description of a safety case that outlines the main arguments for claiming that the safety requirements are met.

The three specifications serve different purposes. The system requirements specification captures the objectives of the system under construction. The system design specification represents an early stage technical description of a system that fulfils objectives. The safety case expresses how the safety objectives are fulfilled.

Regarding b): A conceptual safety design is an early stage specification according to the definition in Section 2.1.6. The conceptual safety designs presented in Paper 2 and Paper 3 are triplets where each part is described in easy to understand formats. The requirements specification part is expressed textually. The system design part is expressed textually and with UML [137] models. The safety case part is expressed with the GSN [61] notation. Although textual specifications of requirements, GSN, and UML models in general should be understandable for the intended users of SaCS, our specifications can, of course, lack understandability and be defined with an insufficient level of detail. We have not tested the specifications on potential users of SaCS in order to investigate if the specifications are easy to understand and at a sufficient level of detail, but rather provide the specifications themselves in the reports.

Regarding c): We recognise that it may be difficult at an early stage to evaluate if the design is easy to use in the sense of being easy to implement or refine. However, we believe that the specifications are informative enough for experts to judge whether the concepts described within Paper 2 and Paper 3 are feasible and detailed enough to use as a starting point for further refinement. The conceptual designs presented in Paper 2 and Paper 3 can be refined by repeating the SaCS process and iterating over pattern selection, pattern instantiation, and pattern composition until each system design is in an implementable state.

The requirements specification part consists of requirements that are uniquely identified. Any requirement may be detailed or rephrased to define another version of the requirement. Requirements may be added to the specification easily by adding a unique requirement reference and the associated requirement text.

UML class diagrams, component diagrams, sequence diagrams, and state machine diagrams (with composite states), as well as textual descriptions, are used for expressing

the system design parts of the conceptual safety designs. Some suggestions for a further detailing of the system designs using UML are given within the papers. However, the main features of the proposed system designs should be easy to understand from the specifications that are provided. Dzidek et al. [42] present a controlled experiment investigating the benefits and costs of using UML during the maintenance and evolution of a real non-trivial system. Professional developers were used as subjects. The control group had no UML documentation. The experiment showed that the subjects in the UML group had, on average, a 54 percent increase in the functional correctness of changes and 7 percent overall improvement in design quality. Cruz-Lemus et al. [33] present the result of five empirical studies of the understandability of UML statechart diagrams with composite states. Although the results are not completely conclusive, composite states seem to be helpful for acquiring knowledge from a diagram. The effect of the use of composite states for memorisation and understanding is not clear.

The safety case specification is described by GSN [61]. The GSN community standard [61] has been developed, and has matured, over several years by a consensus approach between a large group of experts from academia and industry. GSN offers constructs for modularising a safety argument such that existing argument structures can be easily reused. The scope of the safety cases presented in the reports can be easily extended, e.g., by adding nodes to the existing tree structure that represents the safety case, adding sub-trees to the structure, or making the existing safety case a sub-tree in a larger argument structure. We have limited the scope of the safety cases to only provide details on selected issues. The undeveloped parts of each safety case are explicitly denoted within the argument structure as undeveloped goals. The undeveloped goals of a safety case facilitate further refinement. The initial safety case indicates how safety is intended argued and should be maintained throughout the life-cycle of the system. In this sense, the safety case is a living document and needs a continuous refinement. As far as evidence for the suitability of a system under construction for its intended use can be provided in the early stages, the safety cases presented in Paper 2 and Paper 3 at least provide traceability between the safety objectives and the features of the design as far as these are specified to show that the objectives are met.

Textual descriptions, UML models, and GSN models are widely used notational forms for the kind of early stage specifications that constitute a conceptual safety design. In this sense, the notations used for expressing the conceptual safety designs presented in Paper 2 and Paper 3 should be easy to use. The satisfaction of a), b), and c) are also discussed within Paper 2 and Paper 3. The discussions in Paper 2 and Paper 3 represent self-evaluations as the method has been applied in both cases by the main author and the designer of SaCS. However, each step in the application of the SaCS process is documented in a manner we believe is sufficient to allow others to perform an independent evaluation of the feasibility of the SaCS approach.

Success criterion 3 *The SaCS process is cost efficient.*

We will discuss cost efficiency of the SaCS process in terms of estimates of the effort required in order to conduct each step of the process. We find it fair to argue that the SaCS process is cost efficient if it provides intended results when applied in a relevant context without wasted resources, e.g., time or money.

Table 7.1 presents the expected time in hours for an experienced user to conduct the different steps of the SaCS process in collaboration with a stakeholder on a repre-

sentative case. The estimates represent our experience from applying the SaCS process in Study 1 (described in Chapter 10) and Study 2 (described in Chapter 11).

Activities and sub-activities		**User**		**St.h.**	
		P	M	P	M
Preparation:	Define overall objective and system context	**15**	**5**	**10**	**5**
Activity S:	**Pattern Selection**	**3**	**3**	**0**	**3**
Activity S.a	Select SaCS Patterns for Requirements Elicitation	1	1	0	1
Activity S.b:	Select SaCS Patterns for Establishing Design Basis	1	1	0	1
Activity S.c:	Select SaCS Patterns for Establishing Safety Case	1	1	0	1
Activity C:	**Pattern Composition**	**4**	**4**	**0**	**4**
Activity C.a:	Specify Compositions of Patterns	3	2	0	2
Activity C.b:	Specify the Instantiations of Patterns	1	2	0	2
Activity I:	**Pattern Instantiation**	**49**	**5**	**0**	**5**
Activity I.a:	Select Pattern Instantiation Order	1	0	0	0
Activity I.b:	Conduct Stepwise Instantiation	48	5	0	5
After-work:	Document the conceptual safety design	**40**	**0**	**0**	**0**
Total	(the process)	**56**	**12**	**0**	**12**
		68		12	
			80		
Total	(the process. preparations and after-work)	**111**	**17**	**10**	**17**
		128		27	
			155		

Table 7.1: The expected effort in hours for applying the SaCS process

In Table 7.1, estimates of the expected effort required to conduct activities are given in the columns named "User" and "St.h", representing the effort required of a user and a stakeholder, respectively. We distinguish in our estimates between the time spent in meetings "M" and the time spent preparing "P" for meetings or conducting work tasks. We do not distinguish between the different user roles (e.g., modeller or domain expert) or kinds of users (e.g., system engineer or safety engineer) in the estimates nor do we distinguish between the efforts required if there are one or more users. The process does not require several users with complementing competences, although this would be beneficial as a development problem could be addressed from multiple perspectives by users with in-depth expertise from different fields of engineering. We recognise that if there are multiple users collaborating to produce a conceptual safety design, some overhead hours are required to reach a mutual agreement on which patterns should be applied and how. In this sense, some additional hours can be expected for each of the activities related to the effort required to reach a consensus within a group. The stakeholder is more likely to represent a development project stakeholder or responsible rather than a customer, at least a person who is familiar with the SaCS method as well as concepts for developing safety critical systems. We have separated the effort required for preparation and after-work from the effort required conducting the process.

In Fig. 5.17, it is indicated that the SaCS process takes as a starting point one or more defined objectives for a system performing potentially safety critical operations.

In the initial stage (Preparation) prior to the application of the SaCS process, we believe that 10 hours is a reasonable estimate of the effort required by the stakeholder for compiling the necessary information relevant to the conceptualisation work. The user and stakeholder should be able to agree on the overall objective, scope, and context of the system under construction within 5 hours of meetings. We estimate that the user needs 15 additional hours to define more precisely the overall objective and the intended context of the system under construction as well as the scope of the conceptualisation work.

The user conducts pattern selection (Activity S) starting with the defined objective for the conceptualisation work as the initial input. An experienced user is expected to identify quickly the applicability of the patterns offered as support for conceptualisation on the basis of the pattern definitions; the pattern selection map that supports efficient selection; and the low number of patterns that can be selected. We estimate that each of the sub-activities (S.a, S.b, and S.c) of pattern selection can be performed in less than an hour, but, as we only provide estimates in whole hours, a 1 hour estimate is given for preparation in each of the sub-activities of pattern selection. The selection of patterns proposed by the user should be discussed with the stakeholder in order to confirm that the selection is viable. We estimate that the user and stakeholder can reach an agreement on the selection of patterns within meetings of 1 hour for each of the sub-activities (S.a, S.b, and S.c) of pattern selection.

Some effort is required for specifying the composition of patterns (Activity C). A composite pattern can be seen as a model of the interconnection of patterns. Each pattern within a composite has a predefined signature; thus, guidance to the representation of a pattern within a composite is available from the definitions of the patterns that are used. We estimate that 3 hours is enough for specifying the combination of patterns (Sub-activity C.a) as support for conceptual safety design. When the user annotates the composite patterns used for conceptual safety design with symbols indicating how each pattern is instantiated (Sub-activity C.b), the composites are already defined and a small amount of time is needed to add the additional information. We estimate that less than an hour is needed to update the composites with identifiers and a classification for each documentation used or produced during development, in addition to symbols relating these documents to the respective parameters they represent. The application and documentation of patterns for problem solving should be communicated to the stakeholder. We estimate that both the user and the stakeholder need 2 hours for discussing the composition of patterns for problem solving (Sub-activity C.a) and 2 hours for discussing the documentation of the instantiation of patterns (Sub-activity C.b).

The effort required to perform pattern instantiation (Activity I) in a specific context depends on the number of patterns applied; to which level of rigour the guidance within the patterns are applied; and the level of detail that is expected from the results from pattern instantiation. The effort required for selecting the pattern instantiation order (Sub-activity I.a) is expected to be less than 1 hour and is insignificant compared to the effort required to perform instantiation (Sub-activity I.b). Each basic pattern contains a section defining related patterns that can be used to identify the expected order among several patterns. An order for patterns is also indicated in the pattern selection map used in the pattern selection activity as well as in the composites resulting from the pattern composition activity. The expected effort associated with the activity on stepwise instantiation of patterns (Sub-activity I.b) on the other hand significantly

influences the effort required for patterns instantiation (Activity I). While some pattern may be rather quick to perform due to their limited scope, others require more effort. 12 basic patterns were applied as support for conceptual safety design in both Study 1 and Study 2. We estimate that a user needs 4 hours on average for instantiating a pattern. Thus, we estimate a user needs 48 hours for conducting the stepwise instantiation of patterns in the cases described in Study 1 and Study 2. We estimate that both the user and stakeholder need 5 hours for discussing the assumptions and simplifications that are reasonable to apply when instantiating patterns, as well as the interpretation of different concepts (Sub-activity I.b).

After all the steps of the SaCS process have been conducted, the user combines the results in a document that describes the conceptual safety design. We estimate that 40 hours is sufficient for preparing the documentation that explains the conceptual safety design. An experienced user is expected to use a total of 68 hours for applying the SaCS process for conceptual safety design as presented in Paper 2 and Paper 3, and a total of 128 hours if preparations and after-work are also included. The stakeholder is estimated to use a total of 12 hours in meetings with the user during the SaCS process, and a total of 27 hours if preparations and after-work are also included. The total effort, including the effort of the user as well as the stakeholder, is estimated to 155 hours.

7.1.2 The library of SaCS patterns

Success criterion 4 *The library of SaCS patterns consists of patterns that describe effective solutions to recurring challenges within conceptual safety design.*

The full definition of the patterns in the library can be found in the appendices of Paper 2 (described in Chapter 10) and Paper 3 (described in Chapter 11). Each pattern in the library is defined according to a sequence of named sections containing textual descriptions and illustrations. In Section 2.1.2, a pattern is defined as a description of a problem and the essence of its solution to enable the solution to be reused in different settings. The common format of the patterns in the library includes a section named "Problem" that contains a description of the problem or challenge that is addressed by the pattern. A section named "Solution" contains a description of the associated pattern solution. The "Known Uses" section provides examples of known instantiations or otherwise details the source that inspired its definition. The pattern definitions clearly describe a relationship between a problem and its solution. Thus, we find it reasonable to delimit the discussion to whether the patterns address challenges or problems that recur within conceptual safety design as well as whether the solutions described can be accepted as effective. That the overall format of basic patterns is suitable is argued in the discussion related to fulfilment of Success criterion 5.

The knowledge confined within SaCS patterns is extracted from many sources, but is also, to a large extent, traceable to a few important and influential sources within the field of development of safety critical systems. The international safety standards and guidelines presented in Section 4.2 as well as the safety engineering literature presented in Section 4.3, represent some of the most important sources of inspiration in designing the library of patterns. We regard international safety standards and guidelines as particularly suitable sources of inspiration as these are:

- developed and matured over many years;

- defined on the basis of a consensus between an international group of domain and safety experts;
- defined according to established processes for quality assurance within highly regarded organisations;
- defined with the intention of addressing recurring challenges within development of safety critical systems;
- defined with the intention of describing acceptable solutions for developing safety critical systems.

We find it fair to argue that a pattern that expresses a concept in accordance with highly regarded international safety standards and guidelines or otherwise authoritative documents within a domain, expresses a practice that is commonly accepted. Each pattern has undergone a review by at least 2 experts with extensive knowledge and experience within research on the development of safety critical systems, one of which has also been actively involved in safety standardisation work for many years. The knowledge captured within the patterns in the library reflects the knowledge within the safety literature in the following manner:

1) *Establish Concept*: The pattern captures the essence of the first phase in the system life-cycle presented in EN 50126 [28] and in IEC 61508 [93] that is simply named "Concept". The phase is in the standards concerned with how to establish the purpose and constraints associated with a system under development.

2) *Hazard Identification*: The pattern describes a process for identifying hazards in accordance with the practise defined in EN 50129 [30]. The pattern captures the hazard identification part of the phase named "Hazard and risk analysis" of the safety life-cycle presented in IEC 61508 [93]. The identification of hazards is essential for later steps concerned with the definition of safety requirements.

3) *Hazard Analysis*: The pattern describes a process for identifying the potential causes of hazards in accordance with the practise defined in EN 50129 [30]. The patterns in 2), 3), and 4) captures the intent expressed in the life-cycle phase named "Hazard and risk analysis" in IEC 61508 [93].

4) *Risk Analysis*: The pattern describes a process for assessing risk in accordance with the practises defined in EN 50129 [30] and IEC 61508 [93].

5) *Establish System Safety Requirements*: The pattern describes a process for specifying safety requirements inspired by the fourth phase in the system life-cycle presented in EN 50126 [28] named "System Requirements". In EN 50129 [30], the safety requirements are defined on the basis of results from hazard identification and analysis, risk assessment, and the classification of functions. Thus, the patterns in 2), 3), 4), 5), and 9) may be used as a set of complementing patterns supporting the elicitation of safety requirements in a manner comparable to the practice described in EN 50129. In a similar manner, the fourth phase of the safety life-cycle presented in IEC 61508 [93] is named 'Overall safety requirements" and represents a phase that is concerned with the specification of requirements on the basis of hazard and risk analysis.

6) *FMEA*: The pattern captures the essence of the Failure Modes and Effects Analysis (FMEA) method. The FMEA method is widely used within domains developing safety critical systems and is thoroughly described in IEC 60812 [88].

7) *FTA*: The pattern captures the essence of the Fault Tree Analysis (FTA) method as it is described in IEC 61025 [89].

8) *I&C Functions Categorisation*: The pattern captures the method for classifying nuclear Instrumentation and Control (I&C) functions as it is defined within IEC 61226 [91].

9) *SIL Classification*: The pattern captures the railway approach to the classification of functions as it is defined in EN 50128 [29], applicable for software, and EN 50129 [30], applicable for system functions.

10) *Variable Demand for Service*: The pattern is highly specialised to support requirements elicitation in development scenarios as addressed in the first case study presented in Paper 2. The case study describes a very specific development challenge; the pattern describes a solution to that challenge. The pattern is not defined on the basis of the knowledge confined within the safety standards and guidelines literature. It describes, however, a systematic approach for requirements elicitation according to principles for effective requirements engineering. We cannot argue that the pattern describes an effective solution to a recurring challenge within conceptual safety design although it was effectively applied in the case described in Paper 2.

11) *Station Interlocking Requirements*: The pattern captures the essential requirements for building interlocking systems in Norway as defined by the Norwegian Rail Authority in the technical rules JD 550 [99].

12) *Level Crossing Interlocking Requirements*: The pattern captures the essential requirements for building level crossing systems in Norway as defined by the Norwegian Rail Authority in the technical rules JD 550 [99].

13) *Trusted Backup*: The pattern describes a system design concept enabling the utilisation of adaptable control systems for safety critical control tasks by the use of a variant of the Simplex architecture proposed by Sha [147]. Sha refers to the Boeing 777 flight control system as an example of a system that uses the Simplex architecture in practise.

14) *Dual Modular Redundant*: The pattern defines a variant of a generic design solution [156] consisting of two redundant controllers and a voting unit that is implemented in numerous kinds of systems for different kinds of task. The NSB-94 interlocking system used by the Norwegian Railway Authority for many years at different railway stations in Norway uses a configuration of PLC systems according to the concepts described by the pattern.

15) *Overall Safety*: The pattern defines a structure for providing an overall system safety demonstration in a manner that is comparable to the overall structure required for safety cases as presented in EN 50129 [30].

16) *Technical Safety*: The pattern describes a structure for arguing safety with a focus on technical aspects and represents a variant of Part 4 of the safety case required by EN 50129 [30] addressing issues related to technical safety.

17) *Code of Practice*: The pattern defines a structure for arguing that safety objectives are met on the basis of the application of well-proven practices. The strategy of arguing safety on the basis of the application of a code of practice is a strategy expressed in the European Regulation on common safety methods within the railway industry [50] and its associated application guideline [52].

18) *Cross Reference* [74]: The pattern describes a structure for arguing that a system satisfies safety objectives on the basis of a comparison between the system in question with a similar and already accepted system. The strategy is expressed in the European Regulation on common safety methods within the railway industry [50] and its associated application guideline [52].

19) *Explicit Risk Evaluation*: The pattern describes a structure for arguing that a target system is sufficiently safe on the basis of risk being sufficiently addressed. The strategy is expressed in the European Regulation on common safety methods within the railway industry [50] and its associated application guideline [52].

20) *Safety Requirements Satisfied*: The pattern describes a structure for arguing that a target system is sufficiently safe on the basis of evidence for safety requirements being satisfied. The practice of demonstrating system safety on the basis of demonstrating that safety requirements are satisfied is one of the core principles of EN 50129 [30] and IEC 61508 [93].

21) *Deterministic Evidence*: The pattern describes an argument structure where a claim is supported by evidence that demonstrates the claim is fully predictable. One example of the need for relying on deterministic evidence is related to the recommendation expressed in Table A.12 within Appendix A of EN 50128 [29] where it is expressed that the software code of SIL 4 systems is expected to contain no dynamic objects, no dynamic variables, and no conditional jumps.

22) *Assessment Evidence*: The pattern describes an argument structure where a claim is supported by evidence derived on the basis of the application of a suitable assessment method. One example of the need to argue that a suitable assessment techniques has been applied may be seen in Appendix A of EN 50128 [29]. Table A.9 within Appendix A provides recommendations on the application of specific techniques for assessing software depending on their software safety integrity level (SWSIL) classification.

23) *Process Quality Evidence*: The pattern describes an argument structure where the evidence of compliance to a particular process, as well as evidences of the quality of the process, assures a claim being met.

24) *Process Compliance Evidence*: The pattern describes an argument structure where the evidence of compliance to a process that is argued widely known as providing effective results assures a claim being met. A typical use of this strategy may be to claim that, e.g., the software in a given system is developed according

to the practices described in a relevant software standard [29, 90] and thus is developed according to acceptable practices.

25) *Probabilistic Evidence*: The pattern describes an argument structure where the evidence supporting a claim is derived on the basis of probabilistic methods. Table 2.1 in Section 2.1.4 is from Appendix A of EN 50128 [29]. Table 2.1 identifies some of the recommended techniques that may be used to derive probabilistic results such as reliability block diagram, fault tree analysis, and Markov models.

26) *Basic Assumption Evidence*: The pattern describes an argument structure where a claim is supported by a form of rationale or a justified assumption such that no further evidence is required. In any kind of argumentation there are some axioms that are used as base facts. The assumptions used as a basis in assuring and justifying that safety objectives are met should be justified [28–30, 93, 160].

Based on the above, we conclude that the patterns in the library describe effective solutions to recurring challenges within conceptual safety design.

Success criterion 5 *The library of SaCS patterns consists of patterns that may be efficiently and effectively applied individually or in combination.*

The basic patterns in the library are defined with the intent of describing the core of a problem and its solution in a format that is simple, easy to understand, and that makes the solutions described easy to apply. We argued earlier that the patterns may be seen as expressing solutions commonly accepted as effective, as the patterns mirror the knowledge contained in highly regarded safety standards and guidelines. However, a format that facilitates human interpretation is required in order for a user to efficiently and effectively apply the patterns.

The library consists of basic SaCS patterns that are defined in a format comparable to the format of patterns in the literature. In the literature, patterns are commonly described as a sequence of named sections containing textual and graphical descriptions. The overall format of basic SaCS patterns is represented by a sequence of named sections with headings as outlined in the left column of Table 5.1, in the right column a short description of the expected content given beneath each heading is given. A comparison of the formats for expressing patterns in some recent, as well as commonly cited, pattern approaches is given in Section 4.1.3. Many different pattern formats exist, however, the comparison shows that the recent pattern formats, to a large extent, are variations of the pattern formats presented by Alexander et al. [5] in the 70's or Gamma et al. [55] in the 90's. A popular format is that provided by Gamma et al. [55], made known through their book on software design patterns. The format is simple, easy to understand, and most certainly easy to apply as evidenced through their widespread use within software development and within the literature. However, the format [55] is tailored specifically for capturing patterns for software design.

Although the sections in the format of Alexander et al. [5] are unnamed, we have chosen a similar sectioning in basic patterns, but have named each section. The format for basic patterns extends the format in [5] with the addition of the sections "Pattern Signature", "Intent", "Applicability", and "Instantiation Rule". The additional sections are added for efficiency reasons especially. The format in [5] is a suitable basis as it is simple, well-known, and generally applicable for specifying patterns of different

kinds. The pattern signature, intent, and applicability sections of basic patterns are documented in such a manner that the context section part of the format of Alexander et al. is not needed. Within the section named "Pattern Signature", the pattern is shortly summarised in terms of a listing of the expected inputs and outputs of the pattern and its classification. An illustration is also included. The illustration presents how the pattern can be referenced graphically within a composite pattern. The parameters of a pattern represent its interfaces that allow it to be easily connected with other patterns within a composite. The inputs identify the information required from the context in order to apply the pattern. The outputs identify what is the expected outcome of applying the pattern in a context. The "Intent" section contains a short description of what is the intent of the pattern and in what way the solution described is beneficial. A set of rules expressed under the heading "Instantiation Rule" provides guidance to the user on what classifies an instantiation of the pattern. The remaining sections contain textual descriptions and illustrations as in [5]. The input and output parameters of a pattern are explicitly defined in the beginning of a pattern description and are consistently used throughout. The inputs required in order to apply a basic pattern do not have to be provided as a result of the application of another pattern. As long as the required input is provided, a pattern can be applied standalone according to its instantiation rule in order to produce the defined output.

Paper 2 and Paper 3 demonstrate the practical use of the SaCS pattern language in two different case studies. Each paper thoroughly describes how each pattern selected from the library is applied in order to derive results. The papers also describe how several patterns can be applied in combination.

Success criterion 6 *The library of SaCS patterns is easy to extend.*

Although the library currently only consists of basic patterns, it can be easily extended by defining and including in the library new basic patterns or new composite patterns.

Paper 1 (described in Chapter 9) details the syntax and semantics for SaCS patterns. The syntax is defined in Extended Backus-Naur Form (EBNF) [97]. EBNF is a meta-syntax that is widely used for describing context-free grammars. Besides the syntactical description of the different kinds of patterns, Paper 1 also provides several examples in which patterns from the library are translated into their textual syntax in EBNF as well as their semantics. That a newly defined pattern is understandable for a user can be checked by translating it into its semantics, which is presented as a set of paragraphs in English. The translation procedure is shortly outlined in Section 5.3.3 and should be simple to perform.

Paper 2 (described in Chapter 10) and Paper 3 (described in Chapter 11) demonstrate the practical use of the SaCS pattern language for defining composite patterns based on the basic patterns in the library. Papers 1, 2, and 3 provide an extensive guidance on how to define SaCS patterns. Each of the patterns presented in Paper 2 and Paper 3 are defined according to the syntax presented in Paper 1.

7.1.3 The SaCS pattern language

Success criterion 7 *The SaCS pattern language is sufficiently expressive to specify composite patterns that may be easily instantiated into conceptual safety designs.*

A conceptual safety design is an early stage specification of system requirements, system design, and safety case for a safety critical system. What is meant by a conceptual safety design being an instantiation of a composite pattern is defined in both Paper 2 and Paper 3 as follows: *"A conceptual safety design instantiates a SaCS composite pattern if each element of the triple can be instantiated from the SaCS composite pattern according to the instantiation rules of the individual patterns within the composite and according to the rule for composition".* Paper 1 defines the syntax and semantics of the SaCS pattern language that includes the rules for composition. Each basic pattern defines its instantiation rule. The basic patterns are defined in the appendices of Paper 2 and Paper 3. Furthermore, both Paper 2 and Paper 3 describe the application of the SaCS method in two cases, including the use of the SaCS pattern language as support for pattern composition and instantiation. Study 1 is documented in Paper 2 and is concerned with the conceptualisation of a nuclear power plant control system. In Paper 2, the sections 4, 5, 6, 8, and 9 present the composite patterns defined for the case and their instantiation into a conceptual safety design of a nuclear power plant control system. Study 2 is documented in Paper 3 and is concerned with the conceptualisation of a railway interlocking system. In Paper 3, the sections 5, 6, 7, 8, and 9 present the composite patterns for the case and their instantiation into a conceptual safety design of a railway interlocking system. In both cases, it was possible to model the application of patterns and derive a conceptual safety design using the SaCS language. We refer to the demonstration of the applicability of the SaCS pattern language, and the respective discussions on the fulfilment of success criteria for each case as presented in Paper 2 and Paper 3, as support for claiming that the language at least was sufficiently expressive for conceptual safety design in the two case studies. As mentioned earlier, the SaCS method was tested out in the two cases by the authors. Thus, the discussions on the successfulness of SaCS method and the pattern language in the cases presented in Paper 2 and Paper 3 represent self-evaluations. However, each step in specifying the composite patterns and instantiating these into conceptual safety designs is documented. We believe this is sufficient to allow others to perform an independent evaluation of the feasibility of the SaCS method and the pattern language.

Paper 4 (described in Chapter 12) presents an analytical evaluation of the quality of the SaCS pattern language for conceptual safety design as a complement to the experience based evaluations described by Paper 2 and Paper 3. A framework for analysing languages known as the SEmiotic QUALity (SEQUAL) framework [108] is used as a basis for the evaluation. The quality of a language is within the framework evaluated with respect to six different appropriateness factors. A set of requirements is associated with each appropriateness factor. The extent to which these requirements are fulfilled is used to judge the quality.

The SEQUAL framework is based on semiotic theory (the theory of signs) and was chosen as support for the analytic evaluation as it is a general framework applicable to different kinds of languages [108] whose usefulness has been confirmed in experiments [125]. Mendling et al. [121] describe two dominant approaches in the literature for evaluating the quality of modelling approaches: (1) top-down quality frameworks; (2) bottom-up metrics that relate to quality aspects. SEQUAL is the most prominent of the top-down quality frameworks according to the authors. Mendling et al. [121] also discuss a number of bottom-up metrics approaches. According to the authors, several of these contributions are theoretic without empirical validation.

The appropriateness factors used to judge the quality of the SaCS pattern language for conceptual safety design are:

a) *Domain appropriateness*: by domain appropriateness it is meant that the language should be able to represent all concepts in the domain [108].

b) *Modeller appropriateness*: by modeller appropriateness it is meant that there should be no statements in the explicit knowledge of the modeller that cannot be expressed in the language [108].

c) *Participant appropriateness*: by participant appropriateness it is meant that the conceptual basis should correspond as much as possible to the way individuals who partake in modelling perceive reality [108].

d) *Comprehensibility appropriateness*: by comprehensibility appropriateness it is meant that the participants in the modelling should be able to understand all the possible statements of the language [108].

e) *Tool appropriateness*: by tool appropriateness it is meant that the language should have a syntax and semantics that a computerised tool can understand [108].

f) *Organisational appropriateness*: by organisational appropriateness it is meant that the language should be usable within the organisation it targets such that it fits with the work processes and the modelling required to be performed [108].

In the following, we briefly explain the associated evaluation criteria for each of the appropriateness factors a) to f) above and summarise the discussion on their fulfilment. We refer to Paper 4 for the detailed discussion.

Regarding a), the criteria associated with the domain appropriateness factor concerns whether the language: includes the concepts representing best practices within conceptual safety design; and supports the application of best practices for conceptual safety design.

In the design of the SaCS language, we have as much as possible selected keywords and graphical symbols in the spirit of leading literature within the area. Section 2.1 defines some of the main terms from the safety literature used throughout this thesis. In Section 7.1.2, each of the currently available basic patterns is identified and briefly presented with traceability to some of the sources within the safety literature that inspired their definition. That the language supports the application of best practices is represented by the available patterns in the library as well as the language constructs for combining these into composite patterns. This indicates that we are at least able to represent a significant part of the concepts of relevance for conceptual safety design.

Regarding b), the criteria associated with the modeller appropriateness factor concerns whether the language: facilitates tacit knowledge externalisation; and supports the modelling needs within conceptual safety design.

Although we believe that the available patterns in the library represent a significant part of the concepts of relevance for conceptual safety design, only 26 patterns are currently defined. The limited number of basic patterns currently available limits what can be modelled in a composite pattern. Defining additional basic patterns will provide a better coverage of the tacit knowledge that can be externalised. A comparison of

classic and recent formats in the literature is provided in Section 4.1.3. The format of basic patterns is based on the well-known format of Alexander et al. [5] and is presented in Section 5.3.1. The format of basic patterns differs from the format presented in [5] by having the additional sections "Pattern Signature", "Intent", "Applicability", and "Instantiation Rule". The signature, intent, and applicability sections of basic patterns are documented in such a manner that the context section provided in [5] is not needed. The format in [5] is a suitable basis as it is simple, well-known, and generally applicable for specifying patterns of different kinds.

We admit that there can be relevant tacit knowledge that is not easily externalised in the SaCS language. However, the opportunity for increasing the number of basic patterns makes it possible to at least reduce the gap.

Regarding c), the criteria associated with the participant appropriateness factor concerns whether: the terms used for concepts in the language are the same terms used within safety engineering; the symbols used to illustrate the meaning of concepts in the language reflect these meanings; the language is understandable for people familiar with safety engineering without specific training.

In Section 7.1.2, the different concepts expressed in the form of basic patterns were shortly outlined and traced to the safety literature. Activities such as hazard identification and hazard analysis [52], methods such as fault tree analysis [89] and failure mode effects analysis [88], system design solutions including redundant modules and voting mechanisms [156], and practices like arguing safety on the basis of arguing that safety requirements are satisfied [160], are all well known safety engineering practices that may be found in different standards and guidelines [30, 86, 93]. Moreover, as already pointed out, keywords and terms have all been selected based on leading terminology within safety engineering. In this sense, the SaCS language facilitates representing the application of best practices within safety design and mirrors leading international standards and guidelines on development of safety critical systems.

Regarding d), the criteria associated with the comprehensibility appropriateness factor concerns whether: the concepts and symbols of the language differ to the extent they are different; it is possible to group related statements in the language in a natural manner; it is possible to reduce model complexity with the language; the symbols of the language are as simple as possible with appropriate use of colour and emphasis.

Principles supporting effective human interpretation of visualisations as presented in Section 4.4.2 have shaped how composite patterns are expressed in the language, especially the Gestalt principles of perception [115, 171]. Paper 4 explains how several of the classic Gestalt principles has been applied in the language such as *Similarity* for expressing a degree of relatedness between concepts, *Figure-Ground* for separating the objects of focus from the background, *Proximity* for indicating relatedness by placing related objects close to each other, and *Uniform Connectedness* for indicating close relationships by connecting objects. An example of the application of the visualisation principle of *categorisation* [115] is also exemplified in that the SaCS pattern language facilitates reducing the number of relations drawn between patterns when these are similar, thus reducing model complexity. The philosophy behind the use of grey-scale symbols and how simple symbolism is achieved is also explained. As already pointed out, the comprehension of individual patterns and pattern compositions is supported by the use of terms commonly applied within the relevant industrial domains. We find the use of commonly applied safety terms and visualisation principles for effective human interpretation sufficient to facilitate comprehension.

Regarding e), the criteria associated with the tool appropriateness factor concerns whether the language has: a precise syntax; and a precise semantics.

The syntax and semantics of the SaCS pattern language are defined in Paper 1 (described in Chapter 9) and developed in order to facilitate human interpretation and use. The syntax is defined in the Extended BackusNaur Form (EBNF) [97] notation, a meta-syntax that is widely used for describing context-free grammars. Although a computerised tool has not been developed that supports pattern specification and translating patterns defined according to its syntax into its semantics, this should be possible to define on the basis of the syntax and semantics presented in Paper 1.

Regarding f), the criteria associated with the organisational appropriateness factor concerns whether the language: is able to express the desired conceptual safety design when applied in a safety context; ease the comprehensibility of best practices within conceptual safety design for relevant target groups like system engineers, safety engineers, hardware and software engineers; and is usable without the need for costly tools.

Organisations developing safety critical systems are assumed to follow a development process in accordance with the requirements defined in relevant standards. Wong et al. [175] reviewed several large development projects and software safety standards from different domains with respect to cost-effectiveness and concludes that although standards provide useful and effective guidance, safety and cost-effectiveness objectives are successfully met by effective planning and by applying company best practices that are defined in accordance with safety engineering best practices throughout the development life-cycle. SaCS patterns may be defined, applied, and combined in a flexible manner to support company best practices and domain-specific best practices.

Success criterion 8 *The SaCS pattern language facilitates the specification of composite patterns in a context.*

Fig. 5.23 is identical to Fig. 5.21 with the addition of annotations indicating the instantiation of parameters. The annotations for specifying the instantiation of parameters as presented in Fig. 5.23 exemplifies how a composite pattern is specified in a context by the SaCS pattern language. The input parameters identify what kind of information is expected from the context in order for a user to apply the pattern. The output parameters identify the expected outcome of applying a pattern in a context. The user applies the pattern definition in its context according to its instantiation rule in order to provide the output. A symbol and an identifier representing a development artefact used or produced during pattern instantiation is within a composite pattern specification connected with the parameter of the respective pattern it represent.

Paper 1 describes the syntax of composite patterns, which clearly shows how to explicitly define the parameters of a pattern. The syntax also clearly defines the rules for specifying the instantiation of parameters. The structured semantics describes the rules for interpreting parameterised patterns with or without annotations indicating the instantiation of parameters.

Fig. 5, Fig. 10, Fig. 11, Fig. 13, Fig. 20, and Fig. 21 of Paper 2 are all composite patterns annotated with indications of their instantiation in the context represented by Study 1. Fig. 19 and Fig. 21 of Paper 3 specify a composite pattern and the instantiation of this composite, respectively, for addressing the challenges that occur in the context represented by Study 2.

Success criterion 9 *The SaCS pattern language facilitates modularity and reuse.*

Modularity is defined by ISO/IEC/IEEE [95] as *"the degree to which a system or computer program is composed of discrete components such that a change to one component has minimal impact on other components"*. The SaCS pattern language facilitates the specification of composite patterns that can be instantiated into conceptual safety designs. It seems fair to argue that the SaCS pattern language facilitates modularity and reuse if:

a) changes to a pattern within a composite have minimal impact on the other patterns within the composite as well those parts of the conceptual safety design that are not produced on the basis of the changed pattern; and

b) a composite pattern or the conceptual safety design that instantiates the composite can at least partly be reused in another context.

Regarding a), we argue that the parameterisation of SaCS patterns and the composition of patterns by the different kinds of relations minimise the impact of change.

The input and output parameters of a SaCS pattern define the interface that allows it to be easily connected with other patterns. Relations connect patterns and operate on the parameters of the connected patterns. An input parameter identifies the expected information to be provided when instantiating a pattern in its context. An output parameter identifies the expected outcome of pattern instantiation. The pattern definition provides the user with guidance on how to produce the outputs. A change to a pattern will not affect other patterns within a pattern composite unless the change affects the interface or how the user produces the outputs upon pattern instantiation.

The patterns in the library represent the predefined "components" that the user combines to build composite patterns. The composites defined by a user can be used as a basis for specifying new composite patterns. Fig. 5.7 exemplifies a modularised specification where the composite *Define Safety Requirements* uses the composite *Risk Identification and Analysis* defined in Fig. 5.6 as part of its specification. The language facilitates modularity by offering patterns and operators for combining patterns into structures that are reusable. Paper 1 (described in Chapter 9) describes the syntax and semantics of the SaCS patterns language and provides several examples on pattern composition. Paper 2 (described in Chapter 10) and Paper 3 (described in Chapter 11) demonstrates the application of the SaCS pattern language in two different cases resulting in modularised composite patterns as well as modularised conceptual safety designs.

Regarding b), we argue that the parameterisation of SaCS patterns and the composition of patterns by the use of relations facilitate pattern interchangeability and reuse. Furthermore, we argue that the result of pattern instantiation can be reused to the extent the different contexts for which a composite pattern are instantiated are similar.

Fig. 13 of Paper 2 defines a composite pattern named *Safety Requirements* as well as how it was instantiated to support eliciting safety requirements in Study 1. Fig. 10 of Paper 3 defines a composite pattern to support eliciting safety requirements in Study 2. The two composites define approaches to the process of eliciting safety requirements that are to a great extent similar. The similarly specified pattern compositions facilitate

a partial reuse, e.g, by reusing parts of Fig. 13 of Paper 2 for specifying Fig. 10 of Paper 3. Another option is to define a generally applicable composite. A part of the process for eliciting safety requirements as represented in Fig. 13 of Paper 2 and Fig 10. of Paper 3 is risk identification and analysis. In Fig. 5.21, a sequential use of the patterns *Hazard Identification*, *Hazard Analysis*, and *Risk Analysis* is defined. In Section 7.1.2, we argued on the basis of the literature that these patterns represent safety engineering best practices and also that this sequence is promoted within the literature.

Although the composites used to derive safety requirements used in Study 1 and Study 2 are to a large extent similar, the result of pattern instantiation cannot be expected to be reused from one context to another when the contexts are different. However, it is expected that the conceptual safety design presented in Study 2 can be reused to a large extent given a context similar to the one presented, e.g., development of a railway interlocking system at a railway station with a similar: overall system; track configuration; objects to be controlled, and operational profile.

Success criterion 10 *The SaCS pattern language is well-suited to express patterns that are easily understandable for its intended users, which are system engineers, safety engineers, hardware and software engineers.*

We find it fair to argue the language expresses patterns that are easily understandable for its users if:

a) the pattens have a precise syntax and semantics; and

b) the meaning of patterns can be derived by a user in a short time and without the need of a costly tool; and

c) the patterns are defined with notations optimised for human information processing and otherwise follow established principles of human cognition.

Regarding a): Paper 1 (described in Chapter 9) details the syntax and semantics of the SaCS pattern langauge. The syntax of the SaCS language is defined in the EBNF [97] notation. A structured semantics for SaCS patterns is offered in the form of a schematic mapping from pattern definitions, via its textual syntax in EBNF [97], to English. The non-formal representation of the semantics supports human interpretation. The presentation of the semantics of patterns as a text in English was chosen in order to aid communication between users, possibly with different technical backgrounds, on how to interpret patterns.

Regarding b): Paper 1 (described in Chapter 9) offers a procedure for translating patterns into their semantics in English. The procedure is exemplified on several patterns in Paper 1 by the systematic application of translation rules. Although the procedure for the translation of a pattern into its semantics may be automated, it is currently represented as a set of translation rules. The translation of a pattern into its semantics requires no tool beyond the rules defined in Paper 1 and should require only a few minutes to perform, even without training.

Regarding c): Moody [124] defines 9 principles for designing cognitive effective visual notations optimised for human understanding. The principles are synthesised from theory and empirical research from a wide range of fields. We firstly introduce the set of principles. Secondly, we exemplify and discuss the implementation of the principles in the SaCS pattern language. The principles are:

1) *Semiotic clarity*: concerns to which extent there is a correspondence between semantic constructs and graphical symbols [124].

2) *Perceptual discriminability*: concerns to which extent different symbols are distinguishable [124].

3) *Semantic transparency*: concerns to which extent the meaning of a symbol can be inferred from its appearance [124].

4) *Complexity management*: concerns to which extent the visual notation is able to represent information without overloading the human mind [124].

5) *Cognitive integration*: concerns to which extent there are explicit mechanisms supporting the integration of information from different diagrams [124].

6) *Visual expressiveness*: concerns to which extent the full range of and capacities of visual variables are used [124].

7) *Dual coding*: concerns to which extent text is used to complement graphics [124].

8) *Graphic economy*: concerns to which extent the number of different graphical elements are cognitively manageable [124].

9) *Cognitive fit*: concerns to which extent visual dialects are used to support different tasks and audiences [124].

Fig. 7.1 presents how the principles 1-9 outlined above, as well as three of the Gestalt [44, 115, 171] principles of visual perception, are implemented in a slightly revised version of the composite explained in Section 5.3.2. In Fig. 7.1, the coloured elements represent the annotations used for explaining the implementation of principles, while the remaining elements are SaCS notation. Below we give a further description of the implementation of the 9 principles.

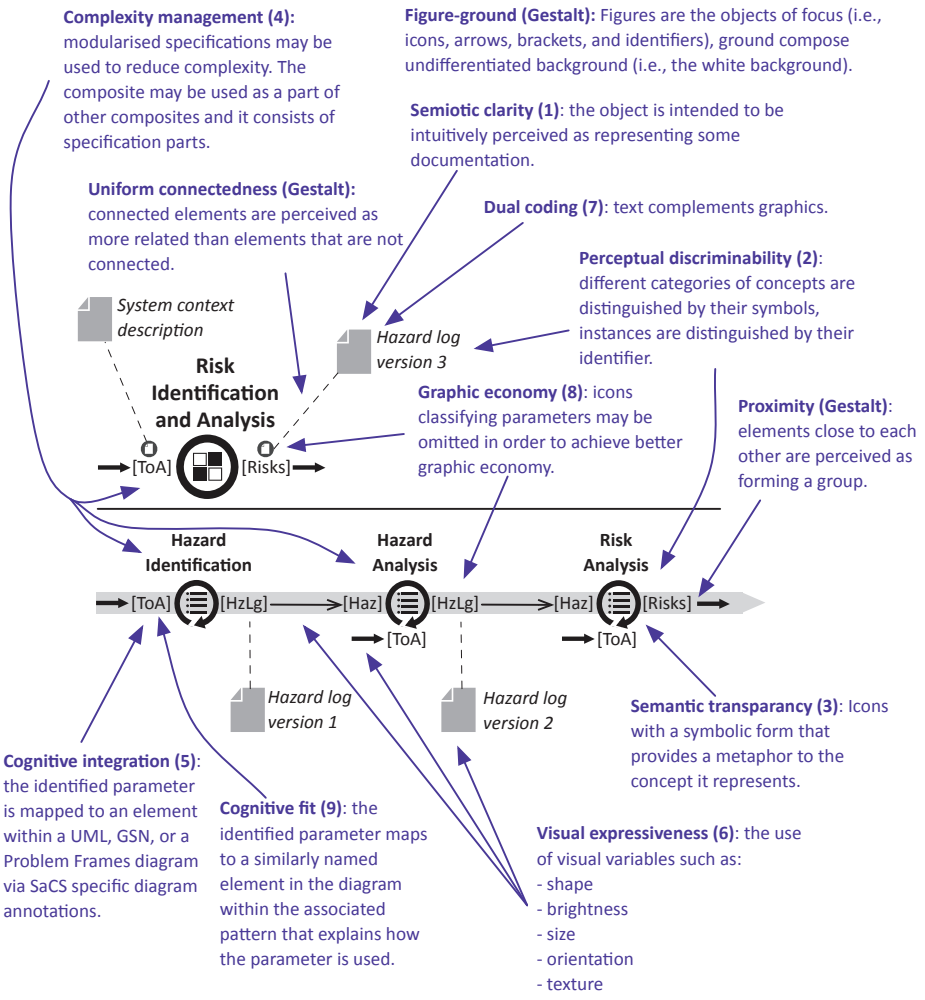


Figure 7.1: Example composite pattern with annotations showing the implementation of principles for cognitive effective visual notations

Regarding 1): We have deliberately applied a symbol deficit approach, which influences semiotic clarity [124]. Several concepts are expressed in natural language rather than by symbols, e.g., names of patterns and parameters. Hence, there is not a one-to-one mapping between concepts and symbols. We believe an approach where all concepts are symbolised with dedicated symbols is counter-productive. A one-to-one mapping between concepts and symbols would likely lead to symbol overload as well as violate the Gestalt principle of simplicity [115]. There is also a limit of human capacity to process information that motivates a small number of symbols. The limit, according to Miller [122], is seven plus or minus two elements. When the number of elements increases past seven, the mind may be confused in correctly interpreting the information. Thus, the number of symbols should be kept low in order to facilitate effective human information processing. We have defined a small set of symbols. The different icons are used as classifiers instead of being associated with one specific concept. However, any lack of clarity with our symbol deficit approach is compensated by the use of other principles such as dual coding explained later.

Regarding 2): We have approached perceptual discriminability [124] by using the Gestalt principle of similarity to balance the need for similarity with the need for dissimilarity. According to Lidwell et al. [115], the principle of similarity is such that similar elements are perceived to be more related than elements that are dissimilar. The use of the similarity principle can be seen applied to the icons for the different contained patterns in Fig. 7.1. Although each referenced pattern has a unique name, their identical icons indicate relatedness. Different kinds of patterns are symbolised by the icons in Fig. 5.8. The icons are of the same size with some aspects of similarity and some aspects of dissimilarity such that a degree of relatedness may be perceived. An icon for a pattern reference is different in shape and shading compared to an icon used for artefact reference (see Fig. 5.8 and Fig. 5.11). Thus, an artefact and a pattern should be perceived as representing quite different concepts. Textual differentiation is approached with different typefaces and font size.

Regarding 3): Lidwell et al. [115] describe iconic representation as *"the use of pictorial images to make actions, objects, and concepts in a display easier to find, recognize, learn, and remember"*. The authors describe four forms for representation of information with icons: similar, example, symbolic, and arbitrary. We have primarily applied the symbolic form to identify a concept at a higher level of abstraction than can be achieved with the similar and example forms. We have also tried to avoid the arbitrary form where there is little or no relationship between a concept and its associated icon. Section 5.3.2 presents the main icons in SaCS and identifies the result of our efforts to achieve semantic transparency [124].

Regarding 4): Several different mechanism are used for complexity management [124] in SaCS. According to Lidwell et al. [115], hierarchical organisation is the simplest structure for visualising and understanding complexity. The SaCS language offers different kinds of hierarchical organisation, e.g., the *details* relation may be used to specify that one pattern is detailed by another, and a composite pattern can use other composites as part of its specification. The latter provides a mechanism for modularisation.

Regarding 5): Different kinds of basic patterns are integrated as support for conceptual safety design in SaCS. We have applied UML [137], GSN [61], and Problem Frames [98] to support modelling different kinds of concepts within basic patterns. These three languages are widely known, and no single language exists that serves

the different modelling needs these notations offer. In addition, a principle of good design [15,115,128] is to balance the need for performance by the importance of preference in designing solutions. However, the choice of several languages challenges cognitive integration [124]. Kim et. al [104] argue within their theory on cognitive integration of diagrams that in order for a multi-diagram representation to be cognitively effective, a mechanism that supports conceptual, as well as perceptual, integration must be explicitly included. The SaCS-specific mechanism that allows cognitive integration is explained in Section 5.3 and is represented by the notation that maps parameters to diagram elements of different kinds.

Regarding 6): The degree of visual expressiveness [124] is defined by the number of visual variables used in a language. Bertin [19] identifies 8 visual variables divided into 2 planar variables and 6 retinal variables. The planar variables are horizontal and vertical position. The retinal variables are shape, size, brightness, orientation, texture, and colour. Every visual variable besides colour is used either to encode information or attract the visual attention of the user to what is important. A general principle within visualisation according to Lidwell et al. [115] is to use colour with care as it may lead to misconceptions if used inappropriately. The authors points out that there is no universal symbolism for different colours. As colour blindness is common we have applied different shades of grey in visualisations. Moody [124] points out that although color is one of the most effective visual variables it should not be used as the sole basis for distinguishing between symbols, but rather for redundant coding. In this sense, colour coding can be added to our models for redundant coding and for emphasising particularly interesting information that requires immediate attention.

Regarding 7): Paivio [138] argues within the dual coding theory that text and graphics together is a more effective carrier of information than using them separately. In SaCS, text is solely used for identifiers, e.g., identifiers for parameters, patterns, and development artefacts. Icons provide visual cues to what an entity represent. We believe this is a suitable strategy as identifiers are used in the verbal communication between users to name the entities that are discussed.

Regarding 8): As mentioned earlier we have deliberately applied a symbol deficit approach in composite patterns, which reduces the number of graphical symbols and positively effect graphic economy [124]. We have used the dual coding principle to balance text with symbols where symbols classify entities and text provides supplementing information. Information is primarily textual in basic patterns. Basic patterns provide detailed guidance on different concepts that is difficult to fully capture graphically; thus, diagrams are used in basic patterns to provide supplementing information.

Regarding 9): The intended users of SaCS represent different engineering disciplines and roles. Depending on task and audience, different kinds of information representation should be used according the cognitive fit theory [124]. In order to achieve an overall effective visual representation, visual dialects suited to the individual tasks and audiences should be integrated rather than providing one representation for all purposes. A requirements engineer is expected to be aware of the Problem Frames [98] notation. A system engineer, hardware engineer, or a software engineer is expected to be aware of the UML [137] notation. A safety engineer is expected to be aware of the the GSN [61] notation. The SaCS pattern language integrates these visual dialects and facilitates the communication between users on the development of conceptual safety design.

7.2 How our contributions relate to and extends state-of-the-art

In this section we will discuss how our artefacts relate to and extend the state-of-the-art presented in Chapter 4. The reason for having this additional discussion on state-of-the-art is that our artefacts is first presented in Chapter 5.

In the following sections, the influence of our artefacts on state-of-the-art is discussed separately for each artefact.

7.2.1 The SaCS process

The process of applying patterns as support for buildings architecture is found in the work of Alexander et al. [3–5] from the 70's. A wide audience of software developers adopted the concept of design patterns, especially from the mid 90's with the introduction of the influential book by Gamma et al. [55] on patterns for software design. The patterns in [55] are organised more like a collection rather than as a sequence of patterns as is the case in [5]. Although a number of different formats for presenting patterns and pattern languages [2, 62, 68, 102, 140, 143, 179] exist, current practices for expressing patterns may be traced to the early works represented in [5] or in [55]. Typically, a pattern expresses knowledge on a proven design concept (or some other kind of solution concept) to a recurring design problem (or some other kind of problem). Moreover, the inter-dependencies between patterns are typically expressed informally or are defined by the sequence in which the patterns are presented. Henninger and Corrêa identify in their survey [79] a lack of pattern approaches supporting the specification of inter-dependencies between patterns and how patterns are combined. The process of the SaCS method improves the state-of-the-art by bridging the activities on selecting patterns, instantiating patterns, and specifying the application of patterns systematically. The SaCS pattern language supports the process of specifying reusable pattern compositions as well as specifying how patterns can be applied in a given context for documentation purposes.

7.2.2 The library of SaCS patterns

There is a lack of pattern approaches targeting the development of safety critical systems specifically, with some notable expeditions [10, 32, 68, 102]. A problem with the mentioned approaches is that these only provide patterns addressing one development perspective/discipline, e.g., embedded design solutions [10], systems architecture [32], software fault tolerance [68], or safety argumentation [102]. In order to consolidate our patterns with best practices for the development of safety critical systems we have used international safety standards and guidelines as inspiration, e.g., [30, 86, 93]. When doing so, we identified and expressed patterns that are not represented in the literature as patterns, but nevertheless represent patterns of safety engineering. The patterns of SaCS represent different branches of knowledge that combined supports development of conceptual safety designs.

A challenge to the effective combined use of patterns in pattern collections is that these are not necessarily defined in a manner that makes it easy to use several together [79]. It is common to detail the relations to other patterns within the definition of a pattern as indicated in Table 4.1. In the literature, the relationships to other patterns

is often detailed in a section named "'Related Patterns". The format of basic SaCS patterns is presented in Table 5.1 and includes a related patterns section. The related patterns sections of patterns in the literature does not normally present indications on how to combine patterns, but more typically provides suggestions for further reading. Thus, its content typically supports pattern selection more than pattern composition, as is also the case with SaCS patterns.

Each pattern in the library is defined to address one concept only. Thus, the library of patterns provides development guidance that facilitates separation of concerns. Every pattern in the library explicitly defines its input and output parameters in order to facilitate precise modelling of a connected structure of patterns using the SaCS pattern language. A flexible solution for connecting patterns is sought by using relations that operate on the parameters of patterns. The relations may be used for expressing flows and dependencies between patterns where the parameters represent the connection points.

7.2.3 The SaCS pattern language

The approach to pattern-based development by the SaCS pattern language involves more than applying patterns merely as support for specifying the technical aspects of a system design. Six different kinds of basic patterns are offered. The patterns address diverse topics, e.g., requirements elicitation, safety assessment, design, and safety demonstration. While some basic patterns provide a description within their definition of an expected sequence in the manner of the pattern language of Alexander et al. [5], other patterns are organised more as in the catalogue of Gamma et al. [55] where different patterns represent alternative solutions to a common problem or address problems not directly related.

The language facilitates the specification of workflows, dataflows, and dependencies in a structure of patterns by the use of relations. A relation in the language operates on the parameters of patterns and thus provides a more precise modelling of relationships than the relations defined by Noble [129], which operates on patterns and not pattern parts. Smith [151] presents a catalogue of elementary software design patterns in the tradition of Gamma et al. [55] and the Pattern Instance Notation (PIN) for expressing compositions of patterns graphically. Connectors define the relationships between patterns. The connectors operate on the defined roles of patterns. The notation of Smith [151] and several other notations [27, 40, 169] for expressing patterns graphically use UML as its basis. The notations are simple, but target the specification of software. In SaCS, patterns for different kinds of aspects are combined.

With respect to the modelling of workflows, Dumas and Hofstede [41] discuss the use of UML [137] activity diagrams for the purpose. Aalst et al. [167] describe workflow patterns and also how Petri nets [168] may be use to model these patterns. UML activity diagrams and Petri nets could be candidate methods for expressing a composition of patterns graphically. A challenge is, of course, that UML activity diagrams, Petri nets, and other kinds of established notational forms mentioned in Section 4.4.1 have a specific purpose and semantics that do not fully accommodate what is intended to be expressed when specifying a structure of patterns in SaCS.

UML collaboration diagrams can be used to graphically present patterns of system structure in the form of objects and interaction by the use of associations or messages [78]. Collaborations can be suitable for modelling some concepts expressed in basic

SaCS patterns that address the structure of systems. A basic pattern can address concepts other than structural concepts. Hence, there is challenge to give reasonable semantics for a structure of patterns modelled as a network of collaborations when the collaboration environment is not applicable to all concepts that are modelled.

In our literature search on patterns, pattern languages, and modelling approaches, we have identified a lack of languages or notational forms that serve our purpose fully. Thus, we have chosen to develop our own language that is not a variant of an established language but specific for our purpose inspired by established languages for modelling and principles for good visualisation.

Chapter 8

Conclusion

This chapter concludes Part I. In Section 8.1, we summarise what has been achieved in terms of new artefacts. Thereafter, Section 8.2 summarises the evaluation of the artefacts. Finally, Section 8.3 describes some possible directions for future work.

8.1 Developed artefacts

In order to develop a safety critical system that can be accepted as being sufficiently safe for its intended use, the system needs to be designed for safety. This entails that safety thinking is part of the development process from the very beginning. To embed safety thinking into each step of the conceptualisation process, starting from the idea stage until a mature system design that can be argued to satisfy safety objectives is reached, requires informed choices. Furthermore, effective guidance is needed on acceptable safety engineering approaches that can be efficiently applied in order to make informed choices and effectuate them in a cost-efficient manner. There is also a need for a common language in order to reach a common understanding and consensus within a development team, likely consisting of people with diverse responsibilities and competences, on what should be the guiding safety principles for safety design conceptualisation.

By using a pattern language as support, different kinds of experts can more easily communicate on how they want to resolve diverse challenges using well-proven patterns as guidance. The applicability of the patterns expressed within the language, and the expressiveness of the language, as well as its ease of use for combining patterns, are essential for success. As different kinds of users, such as system engineers, safety engineers, hardware engineers, and software engineers, need to combine their knowledge on well-proven concepts within their respective field of expertise to reach the end goal, their common language must be able to express these diverse concepts. For this reason, the pattern language must be able to express diverse concepts, such as, processes, techniques, and solutions for requirements elicitation, system and software design, safety demonstration, verification and validation. A language that can express individual patterns on highly specialised topics and still be able to combine these in a flexible manner that also can be easily implemented would contribute to the efficient and effective conceptualisation of systems.

This thesis contributes to the development of conceptual safety design of safety critical systems by providing a pattern-based approach. The overall objective of this

thesis as presented in Section 1.1 is to develop a method and a pattern language that is:

1. Well-suited for developing conceptual safety designs.
2. Applicable within an industrial context within acceptable effort.
3. Comprehensible for its intended users.

To this end, we have developed the SaCS method, which is supported by the SaCS pattern language. More specifically, the artefacts developed in this thesis facilitate the conceptualisation of safe system designs by integrating the following main contributions into the SaCS method:

1. *The SaCS process*: defines the process for systematically applying SaCS patterns to support the development of conceptual safety designs.
2. *The library of SaCS patterns*: defines 26 patterns categorised into six different kinds describing best practices for conceptual safety design.
3. *The SaCS pattern language*: defines how to express basic SaCS patterns and includes a graphical notation for specifying pattern compositions.

8.2 Evaluation of artefacts

We have conducted two large case studies (presented in Paper 2 and in Paper 3) within different domains as part of the evaluation of each artefact of this thesis, as well as its integration into the SaCS method. We have also conducted an analytical evaluation (presented in Paper 4) of the quality of the SaCS pattern language (presented in Paper 1) as support for conceptual safety design as a complement to the two case studies.

While Section 2.2 characterises more precisely the success criteria for each invented artefact in order for the overall objective of this thesis to be met, Section 3.3 describes our strategy for evaluating whether the artefacts satisfy the success criteria. In the McGrath [120] classification of evaluation approaches, our case studies can be viewed as variants of what is termed a *field experiment*. The analytical evaluation can be seen as a variant of what is termed *non-empirical evidence* [120] although it partly builds on the case studies. The *field experiment* and the *non-empirical evidence* represent approaches that score highly on realism and generality, respectively. We recognise that further empirical evidence is needed in order to validate whether the SaCS method is feasible as support for its intended users in general.

The demonstrations of the applicability of the SaCS method in the two case studies provide indications on its feasibility for conceptual safety design. Each pattern applied in the cases is defined according to a formal syntax. Each step in the process of selecting, modelling the use of, and instantiating patterns in the cases, is thoroughly documented. As argued in Section 7, the method can also be expected to provide useful results with reasonable effort.

The analytical evaluation of the SaCS pattern language is supported by a framework for evaluating the quality of languages for conceptual modelling. The result indicates that a significant part of the concepts of relevance for conceptual safety design is

represented by the carefully selected terms, concepts, and terminology. Furthermore, the result indicates that principles of good design for visualisation, to support human comprehension, are applied in the definition of the graphical language. It is also argued that tool support can be provided, to support pattern specification and comprehension, on the basis of the defined syntax and the structured semantics of the language. The result also indicates that some relevant tacit knowledge may not be easily externalised as the SaCS language is today; however, the opportunity to increase the number of basic patterns makes it at least possible to reduce the gap.

8.3 Directions for future work

There are a number of interesting possibilities for further work. We present some of these in this section. Most importantly, empirical evaluations of SaCS are needed in industrial development projects in order to assess its practical application further. Another direction is to develop tool support.

A tool could support pattern selection by offering an interactive process that presents the set of available patterns and provides a mechanism for efficiently filtering the set to only account for those relevant to the context. Given a tool that captures the interaction and prompt for a rationale for the different choices of the user, the process for pattern selection and rationale for choices can be automatically documented by logging interactions. This would provide a thorough documentation of the motivations for the different choices taken during development.

A tool could support pattern specification by, for example, performing automatic checks on the consistency of composite pattern specifications on the basis of the syntax presented in Paper 1. Furthermore, a mechanism for automatically generating the meaning of composite patterns could be provided on the basis of the structured semantics also presented in Paper 1.

A tool could support pattern instantiation by providing templates for the specification of requirements, system design, and safety case, as well as other development artefacts. Traceability could be supported by generating trace links when a template is applied between a pattern and the artefact produced upon pattern instantiation.

The library should be increased with more patterns on relevant topics within conceptual safety design in order to provide better coverage of the concepts that are commonly applied within the development of safety critical systems. The number of patterns in the library can be increased by importing patterns expressed in the literature, such as, patterns on requirement specification provided by Whithall [174], patterns on software fault tolerant software as described by Hanmer [68], or patterns on safety case development provided by Kelly [102]. The main challenge is not to define best practices on different aspects of development in the form of patterns, but rather to combine the knowledge scattered in different pattern approaches efficiently and effectively.

In addition to extending the library with patterns on the core concepts of safety critical systems development, the library could also include a collection of predefined composite patterns. The library of composite patterns should include those pattern structures that are recurring in the intended application domains of SaCS, e.g., composite patterns supporting the implementation of IEC 61508 [93], such that instantiation as specified ensures compliance.

There is a need for effective mechanisms for indexing, filtering, searching and cross-

referencing patterns given a large number of patterns in a library. A solution could be in the form of a dedicated search engine. The search engine could base its search on a rigid categorisation of patterns and interact with the user through questions and answers, or by an interactive graphical model. Another solution could be to define the "Related Patterns" section of basic patterns according to a strict syntax. The user of the language could search for relevant patterns in a manner like searching a word list to find synonyms and antonyms. A WordNet (`wordnet.princeton.edu/`) like semantic network of patterns could increase the utilisation of patterns by providing a platform for sharing knowledge and present this knowledge in a consistent format.

The language defines a limited number of operators to model relations between patterns. There is likely to be a need for additional relations if the library is extended with more patterns as well as new categories of patterns. Furthermore, the development of additional relations should include a categorisation such that the fundamental kinds of pattern relations are represented.

A composite pattern can be seen as a network where the patterns represent nodes and relations represent the vertices. The network of patterns can be extended with information on cost estimates. The vertices and edges could be annotated with knowledge on the expected effort required in order to instantiate patterns. An algorithm could perform calculation on the network. If there are optional elements or alternative paths in a network, the algorithm could calculate all possibilities and provide a comparison such that a user could make informed decisions on what to include and what to exclude in the planning of activities within an acceptable cost limit.

The language could be extended with additional annotations for indicating who is responsible for what, as well as the intended role of different actors involved in pattern instantiation. The symbols used for indicating the instantiation order of patterns provide a very rough model of time. Additional information could be introduced for denoting the expected start and duration for applying the different patterns within a composite, as support for detailed planning of different development activities. Some kind of visualisation could also be introduced for indicating the location of an activity, if this information is relevant, e.g., that a certain test procedure shall be performed at the customer site contrary to the vendor site.

As more information is added to a composite pattern, there is a danger that the model will become more difficult to interpret rather than more informative. Introducing layering, where a base model is supplemented with additional visual information as selected by a user, could be used to mitigate information overload.

Bibliography

[1] H. Abelson and G. J. Sussman. *Structure and Interpretation of Computer Programs*. MIT Press, 2nd edition, 1996.

[2] A. Aguiar and G. David. Patterns for Effectively Documenting Frameworks. In J. Noble, R. Johnson, P. Avgeriou, N. Harrison, and U. Zdun, editors, *Transactions on Pattern Languages of Programming II*, volume 6510 of *LNCS*, pages 79–124. Springer, 2011.

[3] C. Alexander. *The Oregon Experiment*. Oxford University Press, 1975.

[4] C. Alexander. *The Timeless Way of Building*. Oxford University Press, 1979.

[5] C. Alexander, S. Ishikawa, and M. Silverstein. *A Pattern Language: Towns, Buildings, Construction*, volume 2. Oxford University Press, 1977.

[6] R. Alexander, T. Kelly, and J. McDermid. Safety Cases for Advanced Control Software: Safety Case Patterns. Technical Report FA8655-07-1-3025, Department of Computer Science, University of York, 2008.

[7] P. Amey. Correctness by Construction: Better can also be Cheaper. *CrossTalk: The Journal of Defense Software Engineering*, 15(3):24–28, 2002.

[8] T. Andreson and P. A. Lee. *Fault Tolerance: Principles and Practice*. Springer, 1990.

[9] J. D. Andrews and T. R. Moss. *Reliability and Risk Assessment*. ASME Press, 2nd edition, 2002.

[10] A. Armoush, F. Salewski, and S. Kowalewski. Design Pattern Representation for Safety-Critical Embedded Systems. *Journal of Software Engineering and Applications*, 2(1):1–12, 2009.

[11] K. J. Åström and B. Wittenmark. *Adaptive Control*. Dover Publications, 2nd edition, 2008.

[12] A. Aurum and C. Wohlin. *Engineering and Managing Software Requirements*. Springer, 2005.

[13] A. Avižienis. The N-Version Approach to Fault-Tolerant Software. *IEEE Transactions on Software Engineering*, 11(12):1491–1501, 1985.

[14] A. Avižienis, J.-C. Laprie, B. Randell, and C. Landwehr. Basic Concepts and Taxonomy of Dependable and Secure Computing. *IEEE Transactions on Dependable and Secure Computing*, 1(1):11–33, 2004.

[15] R. W. Bailey. Performance versus Preference. In *Proceedings of 37th Annual Meeting of the Human Factors and Ergonomics Society*, volume 37, pages 282–286. SAGE Publications, 1993.

[16] I. Bayley and H. Zhu. A Formal Language for the Expression of Pattern Compositions. *International Journal on Advances in Software*, 4(3):354–366, 2012.

[17] K. Beck and W. Cunningham. Using Pattern Languages for Object-Oriented Programs. In N. K. Meyrowitz, editor, *Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA'87)*. ACM, 1987.

[18] A. Beda, P. M. Spieth, T. Handzsuj, P. Pelosi, N. Carvalho, E. Koch, T. Koch, and M. Gama de Abreu. A Novel Adaptive Control System for Noisy Pressure-controlled Ventilation: A Numerical Simulation and Bench Test Study. *Intensive Care Medicine*, 36(1):164–168, 2010.

[19] J. Bertin. *Semiology of Graphics: Diagrams, Networks, Maps*. University of Wisconsin Press, 1983.

[20] P. Bishop, R. Bloomfield, and S. Guerra. The Future of Goal-based Assurance Cases. In *Proceedings of Workshop on Assurance Cases: Best Practices, Possible Obstacles, and Future Opportunities. Supplemental Volume of the 2004 International Conference on Dependable Systems and Networks (DSN'04)*, pages 390–395. IEEE Computer Society, 2004.

[21] B. W. Boehm and P. N. Papaccio. Understanding and Controlling Software Costs. *IEEE Transactions on Software Engineering*, 14(10):1462–1477, 1988.

[22] J. Bowen and V. Stavridou. Safety-critical Systems, Formal Methods and Standards. *Software Engineering Journal*, 8(4):189–209, 1993.

[23] F. P. J. Brooks. The Computer Scientist as Toolsmith II. *Communications of the ACM*, 39(3):61–68, 1996.

[24] F. Buschmann, K. Henney, and D. C. Schmidt. *Pattern-Oriented Software Architecture Volume 4: A Pattern Language for Distributed Computing*. Wiley, 2007.

[25] F. Buschmann, K. Henney, and D. C. Schmidt. *Pattern Oriented Software Architecture Volume 5: On Patterns and Pattern Languages*. Wiley, 2007.

[26] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal. *Pattern-Oriented Software Architecture Volume 1: A System of Patterns*. Wiley, 1996.

[27] H. Byelas and A. Telea. Visualization of areas of interest in software architecture diagrams. In *Proceedings of the 2006 ACM Symposium on Software Visualization (SoftVis'06)*, pages 105–114. ACM, 2006.

[28] CENELEC. Railway Applications - The Specification and Demonstration of Reliability, Availability, Maintainability and Safety (RAMS), EN 50126. European Committee for Electrotechnical Standardization, 1999.

[29] CENELEC. Railway Applications – Communications, Signalling and Processing Systems – Software for Railway Control and Protection systems, EN 50128. European Committee for Electrotechnical Standardization, 2001.

[30] CENELEC. Railway Applications – Communications, Signalling and Processing Systems – Safety Related Electronic Systems for Signalling, EN 50129. European Committee for Electrotechnical Standardization, 2003.

[31] P. P.-S. Chen. The Entity-relationship Model – Toward a Unified View of Data. *ACM Transactions on Database Systems*, 1(1):9–36, 1976.

[32] T. Crenshaw. *Architecture Reference Models for Early Safety Analysis*. PhD thesis, University of Illinois at Urbana-Champaign, USA, 2008.

[33] J. A. Cruz-Lemus, M. Genero, M. E. Manso, S. Morasca, and M. Piattini. Assessing the Understandability of UML Statechart Diagrams with Composite States – A Family of Empirical Studies. *Empirical Software Engineering*, 14:685–719, 2009.

[34] R. J. Cullen. Safety as a Design Tool. In R. F. Cox, editor, *Proceedings of the 1996 Safety and Reliability Society Symposium (SaRS'96)*, Safety and Reliability Society, 1996.

[35] W. Cunningham. Portland Pattern Repository. Accessed 1 May, 2014 from `http://c2.com/ppr/`.

[36] H. E. I. Dahl, I. Hogganvik, and K. Stølen. Structured Semantics for the CORAS Security Risk Modelling Language. Technical Report STF07 A970, SINTEF ICT, Oslo, Norway, 2007.

[37] A. Darbari. Rule Extraction from Trained ANN: A Survey. Technical Report WV-2000-03, Department of Computer Science, Dresden University of Technology, Germany, 2000.

[38] P. J. Denning. Is Computer Science Science? *Communications of the ACM*, 48(4):27–31, 2005.

[39] P. J. Denning, D. E. Comer, D. Gries, M. C. Mulder, A. Tucker, A. J. Turner, and P. R. Young. Computing as a Dicipline. *Communications of the ACM*, 32(1):9–23, 1989.

[40] J. Dong, S. Yang, and K. Zhang. Visualizing Design Patterns in Their Applications and Compositions. *IEEE Transactions on Software Engineering*, 33(7):433–453, 2007.

[41] M. Dumas and A. H. M. ter Hofstede. UML Activity Diagrams as a Workflow Specification Language. In *Proceedings of the 4th International Conference on The Unified Modeling Language, Modeling Languages, Concepts, and Tools*, volume 2185 of *LNCS*, pages 76–90. Springer, 2001.

[42] W. J. Dzidek, E. Arisholm, and L. C. Briand. A Realistic Empirical Evaluation of the Costs and Benefits of UML in Software Maintenance. *IEEE Transactions on Software Engineering*, 34(3):407–432, 2008.

[43] K. M. Eisenhardt. Building Theories From Case Study Research. *The Academy of Management Review*, 14(4):532–550, 1989.

[44] W. D. Ellis. *A Source Book of Gestalt Psychology*. The Gestalt Journal Press, 1997.

[45] EPRI. Fukushima Daiichi Accident – Technical Causal Factor Analysis. Technical Report 1024946, Electric Power Research Institute, Palo Alto, California, USA, 2012.

[46] C. A. I. Ericson. *Hazard Analysis Techniques for System Safety*. Wiley, 1st edition, 2005.

[47] EUROCAE. Software Consideration in Airborne Systems and Equipment Certification, ED-12B, 2nd edition. European Organisation for Civil Aviation Equipment, 1992.

[48] EUROCAE. Design Assurance Guidance for Airborne Electronic Hardware, ED-80. European Organisation for Civil Aviation Equipment, 2000.

[49] EUROCAE/SAE. Certification Considerations for Highly-Integrated or Complex Aircraft Systems, ED-79/ARP 4754. European Organisation for Civil Aviation Equipment, 1996.

[50] European Commission. Commission Regulation (EC) No 352/2009 on the Adoption of Common Safety Method on Risk Evaluation and Assessment. Official Journal of the European Union, 2009.

[51] European Commission. Commission Regulation (EC) No 1035/2011 of 17 October 2011 Laying Down Common Requirements for the Provision of Air Navigation Services and Amending Regulations (EC) No 482/2008 and (EU) No 691/2010 (Text with EEA Relevance). Official Journal of the European Union, 2011.

[52] European Railway Agency (ERA). Guide for the Application of the Commission Regulation on the Adoption of a Common Safety Method on Risk Evaluation and Assessment. Retrieved Mai 1, 2014, from `http://www.era.europa.eu/`, 2009.

[53] P. H. Feiler. Challenges in Validating Safety-Critical Embedded Systems. *SAE International Journal of Aerospace*, 3(1):109–116, 2010.

[54] M. Fowler. *Analysis Patterns: Reusable Object Models*. Addison-Wesley, 1996.

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

[56] C. Gane and T. Sarson. *Structured Systems Analysis: Tools and Techniques*. Prentice Hall, 1979.

[57] M. Gnatz, F. Marschall, G. Popp, A. Rausch, and W. Schwerin. Towards a Living Software Development Process based on Process Patterns. In *Proceedings of the 8th European Workshop on Software Process Technology (EWSPT'01)*, volume 2077 of *LNCS*, pages 182–202. Springer, 2001.

[58] S. Gopalakrishnan, J. Krogstie, and G. Sindre. Adapting UML Activity Diagrams for Mobile Work Process Modelling: Experimental Comparison of Two Notation Alternatives. In P. Bommel, S. Hoppenbrouwers, S. Overbeek, E. Proper, and J. Barjis, editors, *The Practice of Enterprise Modeling*, volume 68 of *Lecture Notes in Business Information Processing*, pages 145–161. Springer, 2010.

[59] J. Gordijn, E. Yu, and B. van der Raadt. E-service Design Using i* and e^{3}value Modeling. *IEEE Software*, 23(3):26–33, 2006.

[60] B. A. Gran. Use of Bayesian Belief Networks when Combining Disparate Sources of Information in the Safety Assessment of Software-based Systems. *International Journal of Systems Science*, 33(6):529–542, 2002.

[61] GSN Working Group. GSN Community Standard, Version 1.0. `http://www.goalstructuringnotation.info/documents/GSN_Standard.pdf`, 2011. Accessed 2013-09-01.

[62] E. Guerra, J. T. de Souza, and C. T. Fernandes. Pattern Language for the Internal Structure of Metadata-Based Frameworks. In J. Noble, R. Johnson, U. Zdun, and E. Wallingford, editors, *Transactions on Pattern Languages of Programming III*, volume 7840 of *LNCS*, pages 55–110. Springer, 2013.

[63] J. A. Gulla. A General Explanation Component for Conceptual Modeling in CASE Environments. *ACM Transactions on Information Systems (TOIS)*, 14(3):297–329, 1996.

[64] I. Habli and T. Kelly. Process and Product Certification Arguments – Getting the Balance Right. *SIGBED Review*, 3(4):1–8, 2006.

[65] I. M. Habli. *Model-Based Assurance of Safety-Critical Product Lines*. PhD thesis, University of York, United Kingdom, 2009.

[66] C. Haddon-Cave. The Nimrod Review: An Independent Review Into the Broader Issues Surrounding the Loss of the RAF Nimrod MR2 Aircraft XV230 in Afghanistan in 2006. Technical Report 1025 2008-09, The Stationery Office (TSO), London, UK, 2009.

[67] T. Halpin. Fact-Oriented Modeling: Past, Present and Future. In J. Krogstie, A. L. Opdahl, and S. Brinkkemper, editors, *Conceptual Modelling in Information Systems Engineering*, pages 19–38. Springer, 2007.

[68] R. S. Hanmer. *Patterns for Fault Tolerant Software*. Wiley, 2007.

[69] J. Hartmanis. Some Observations About the Nature of Computer Science. In *Foundations of Software Technology and Theoretical Computer Science*, volume 761 of *LNCS*, pages 1–12. Springer, 1993.

[70] A. A. Hauge and K. Stølen. SACS – A Pattern Language for Safe Adaptive Control Software. In *Proceedings of the 18th Conference on Pattern Languages of Programs (PLoP'11)*. ACM, 2011.

[71] A. A. Hauge and K. Stølen. A Pattern-based Method for Safe Control Systems Exemplified within Nuclear Power Production. In *Proceedings of the International Conference on Computer Safety, Reliability and Security (SAFECOMP'12)*, volume 7612 of *LNCS*, pages 13–24. Springer, 2012.

[72] A. A. Hauge and K. Stølen. Developing Safe Control Systems using Patterns for Assurance. In *Proceedings of the Third International Conference on Performance, Safety and Robustness in Complex Systems and Applications (PESARO'13)*, pages 1–8. IARIA, 2013.

[73] A. A. Hauge and K. Stølen. Syntax & Semantics of the SaCS Pattern Language. Technical Report HWR-1052, Institute for energy technology, OECD Halden Reactor Project, Halden, Norway, 2013.

[74] A. A. Hauge and K. Stølen. A Pattern Based Method for Safe Control Conceptualisation – Exemplified Within Nuclear Power Production. Technical Report HWR-1029 rev 2, Institute for energy technology, OECD Halden Reactor Project, Halden, Norway, 2014.

[75] A. A. Hauge and K. Stølen. A Pattern Based Method for Safe Control Conceptualisation – Exemplified Within Railway Signalling. Technical Report HWR-1037 rev 2, Institute for energy technology, OECD Halden Reactor Project, Halden, Norway, 2014.

[76] A. A. Hauge and K. Stølen. An Analytic Evaluation of the SaCS Pattern Language – Including Explanations of Major Design Choices. In *Proceedings of the International Conference on Pervasive Patterns and Applications (PATTERNS'14)*, pages 79–88. IARIA, 2014.

[77] C. G. Healey and J. Enns. Attention and Visual Memory in Visualization and Computer Graphics. *Visualization and Computer Graphics*, 18(7):1170–1188, 2012.

[78] R. Heckel and S. Sauer. Strengthening UML Collaboration Diagrams by State Transformations. In *4th International Conference on Fundamental Approaches to Software Engineering (FASE'01), held as Part of the Joint European Conferences on Theory and Practice of Software (ETAPS'01)*, volume 2029 of *LNCS*, pages 109–123. Springer, 2001.

[79] S. Henninger and V. Corrêa. Software Pattern Communities: Current Practices and Challenges. In *Proceedings of the 14th Conference on Pattern Languages of Programs (PLOP'07)*, pages 14:1–14:19. ACM, 2007.

[80] C. Hentrich and U. Zdun. A Pattern Language for Process Execution and Integration Design in Service-Oriented Architectures. In J. Noble and R. Johnson, editors, *Transactions on Pattern Languages of Programming I*, volume 5770 of *LNCS*, pages 136–191. Springer, 2009.

[81] A. R. Hevner, S. T. March, J. Park, and S. Ram. Design Science in Information Systems Research. *MIS Quarterly*, 28(1):75–105, 2004.

[82] Hillside Group. Design Patterns Library. Accessed 1 May, 2014, from `http://hillside.net/patterns/`.

[83] I. Hogganvik. *A Graphical Approach to Security Risk Analysis*. PhD thesis, Faculty of Mathematics and Natural Sciences, University of Oslo, 2007.

[84] E. Hull, K. Jackson, and J. Dick. *Requirements Engineering*. Springer, 3rd edition, 2010.

[85] R. Hull and R. King. Semantic Database Modeling: Survey, Applications, and Research Issues. *ACM Computing Surveys*, 19(3):201–260, 1987.

[86] IEC. Nuclear Power Plants – Instrumentation and Control Systems Important to Safety – General Requirements for Systems, IEC 61513, 2nd edition. International Electrotechnical Commission, 2001.

[87] IEC. Nuclear Power Plants – Instrumentation and Control Important to Safety – Software Aspects for Computer-based Systems Performing Category B or C Functions, IEC 62138, 1st edition. International Electrotechnical Commission, 2004.

[88] IEC. Analysis Techniques for System Reliability – Procedure for Failure Mode and Effects Analysis (FMEA), IEC 60812. International Electrotechnical Commission, 2006.

[89] IEC. Fault Tree Analysis (FTA), IEC 61025, 2nd edition. International Electrotechnical Commission, 2006.

[90] IEC. Nuclear Power Plants – Instrumentation and Control Systems Important to Safety – Software Aspects for Computer-based Systems Performing Category A Functions, IEC 60880, 2nd edition. International Electrotechnical Commission, 2006.

[91] IEC. Nuclear Power Plants – Instrumentation and Control Important to Safety – Classification of Instrumentation and Control Functions, IEC 61226, 3rd edition. International Electrotechnical Commission, 2009.

[92] IEC. Event Tree Analysis (ETA) Techniques, IEC 62502, 1st edition. International Electrotechnical Commission, 2010.

[93] IEC. Functional Safety of Electrical / Electronic / Programmble Electronic Safety-related Systems, IEC 61508, 2nd edition. International Electrotechnical Commission, 2010.

[94] IEEE. IEEE Standard Glossary of Software Engineering Terminology, IEEE Std 610.12-1990. Institute of Electrical and Electronics Engineers, 1990.

[95] ISO. Systems and Software Engineering – Vocabulary, ISO 24765. International Organization for Standardization, 2010.

[96] ISO. Road Vehicles – Functional Safety, ISO 26262. International Organization for Standardization, 2011.

[97] ISO/IEC. Information Technology - Syntactic Metalanguage - Extended BNF. International Organization for Standardization, 1996.

[98] M. Jackson. *Problem Frames: Analysing and Structuring Software Development Problems*. Addison-Wesley, 2001.

[99] Jernbaneverket (JBV). Teknisk Regelverk, JD550 Prosjektering. Accessed May 1, 2014, from `https://trv.jbv.no/wiki/Signal/Prosjektering`.

[100] J. Katz. *Designing Information: Human Factors and Common Sense in Information Design*. Wiley, 1st edition, 2012.

[101] T. Kelly and R. Weaver. The Goal Structuring Notation – A Safety Argument Notation. In *Proceedings of Workshop on Assurance Cases: Best Practices, Possible Obstacles, and Future Opportunities. Supplemental Volume of the 2004 International Conference on Dependable Systems and Networks (DSN'04)*. IEEE Computer Society, 2004.

[102] T. P. Kelly. *Arguing Safety – A Systematic Approach to Managing Safety Cases*. PhD thesis, University of York, United Kingdom, 1998.

[103] T. P. Kelly. Can Process-Based and Product-Based Approaches to Software Safety Certification be Reconciled? In F. Redmill and T. Anderson, editors, *Improvements in System Safety*, pages 3–12. Springer, 2008.

[104] J. Kim, J. Hahn, and H. Hahn. How Do We Understand a System with (So) Many Diagrams? Cognitive Integration Processes in Diagrammatic Reasoning. *Information Systems Research*, 11(3):284–303, 2000.

[105] M. Kircher and P. Jain. *Pattern-Oriented Software Architecture Volume 3: Patterns for Resource Management*. Wiley, 2004.

[106] J. C. Knight. Safety Critical Systems: Challenges and Directions. In *Proceedings of the 24th International Conference on Software Engineering (ICSE'02)*, pages 547–550. ACM, 2002.

[107] J. C. Knight and N. G. Leveson. An Experimental Evaluation of the Assumption of Independence in Multiversion Programming. *IEEE Transactions on Software Engineering*, 12(1):96–109, 1986.

[108] J. Krogstie. *Model-based Development and Evolution of Information Systems: A Quality Approach*. Springer, 2012.

[109] D. R. Kuhn, D. R. Wallace, and A. M. J. Gallo. Software Fault Interactions and Implications for Software Testing. *IEEE Transactions on Software Engineering*, 30(6):418–421, 2004.

[110] Z. Kurd. *Artificial Neural Networks in Safety-critical Applications*. PhD thesis, University of York, United Kingdom, 2005.

[111] J. H. Larkin and H. A. Simon. Why a Diagram is (Sometimes) Worth Ten Thousand Words. *Cognitive Science*, 11(1):65–100, 1987.

[112] S. LeFraniere. Design Flaws Cited in Deadly Train Crash in China. The New York Times, December 28, 2011, Retrieved May 1, 2014, from `http://www.nytimes.com/2011/12/29/world/asia/design-flaws-cited-in-china-train-crash.html`, 2011.

[113] N. G. Leveson. *Safeware: System Safety and Computers*. Addison-Wesley, 1st edition, 1995.

[114] N. G. Leveson. The Use of Safety Cases in Certification and Regulation. *Journal of System Safety*, 47(6), 2011.

[115] W. Lidwell, K. Holden, and J. Butler. *Universal Principles of Design*. Rockport Publishers, 2nd edition, 2010.

[116] O. S. Ligaarden. *A Framework for Analyzing and Monitoring the Impact of Dependencies on Quality*. PhD thesis, Faculty of Mathematics and Natural Sciences, University of Oslo, 2013.

[117] J. L. Lions. Ariane 5, Fligh 501 Failure, Report by the Inquiry Board. Retrieved May 1, 2014, from `http://esamultimedia.esa.int/docs/esa-x-1819eng.pdf`, 1996.

[118] S. T. March and G. F. Smith. Design and Natural Science Research on Information Technology. *Decision Support Systems*, 15(4):251–266, 1995.

[119] J. A. McDermid. Software Safety: Where's the Evidence? In *Proceedings of the Sixth Australian workshop on Safety critical systems and software (SCS'01)*, volume 3, pages 1–6. Australian Computer Society, 2001.

[120] J. E. McGrath. *Groups: interaction and performance*. Prentice-Hall, 1984.

[121] J. Mendling, G. Neumann, and W. van der Aalst. On the Correlation Between Process Model Metrics and Errors. In *Proceedings of 26th International Conference on Conceptual Modeling*, volume 83, pages 173–178, 2007.

[122] G. A. Miller. The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information. *Psychological Review*, 63(2):81–97, 1956.

[123] MoD. Defence Standard 00-56, Safety Management Requirements for Defence Systems, Part 1, Issue 4, 2007.

[124] D. L. Moody. The "Physics" of Notations: Towards a Scientific Basis for Constructing Visual Notations in Software Engineering. *IEEE Transactions on Software Engineering*, 35(6):756–779, 2009.

[125] D. L. Moody, G. Sindre, T. Brasethvik, and A. Slvberg. Evaluating the Quality of Process Models: Empirical Testing of a Quality Framework. In *Proceedings of the 21st International Conference on Conceptual Modeling*, volume 2503 of *LNCS*, pages 380–396. Springer, 2013.

[126] A. Q. Nauman and A. Perini. Engineering Adaptive Requirements. In *Proceedings of the ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems (SEAMS'09)*, pages 126–131. IEEE Computer Society, 2009.

[127] Y. V. Nesterov, M. A. Pikin, and E. Y. Romanovskaya. Development of Technological Algorithms for Automated Process Control Systems of Power Units. *Thermal Engineering*, 56(10):827–831, 2009.

[128] J. Nilsen and J. Levy. Measuring Usability: Performance vs. Preference. *Communications of the ACM*, 37(4):66–75, 1994.

[129] J. Noble. Classifying Relationships Between Object-oriented Design Patterns. In *Proceedings of the 1998 Australian Software Engineering Conference (ASWEC'98)*, pages 98–107. IEEE Computer Society, 1998.

[130] J. Noble and R. Johnson, editors. *Transactions on Pattern Languages of Programming I*, volume 5770 of *LNCS*. Springer, 2009.

[131] J. Noble, R. Johnson, P. Avgeriou, N. B. Harrison, and U. Zdun, editors. *Transactions on Pattern Languages of Programming II*, volume 6510 of *LNCS*. Springer, 2011.

[132] J. Noble, R. Johnson, U. Zdun, and E. Wallingford, editors. *Transactions on Pattern Languages of Programming III*, volume 7840 of *LNCS*. Springer, 2013.

[133] D. A. Norman. *The Design of Everyday Things*. Basic Books, 2002.

[134] A. G. Nysetvold and J. Krogstie. Assessing Business Process Modeling Languages Using a Generic Quality Framework. In *Proceedings of the 17th Conference on Advanced Information Systems Engineering (CAiSE'05) Workshops*, pages 545–556. Idea Group, 2005.

[135] A. Omerovic. *PREDIQT: A Method for Model-based Prediction of Impacts of Architectural Design Changes on System Quality*. PhD thesis, Faculty of Mathematics and Natural Sciences, University of Oslo, 2012.

[136] OMG. Business Process Model and Notation (BPMN), Version 2.0. Object Management Group, 2011.

[137] OMG. Unified Modeling Language Specification, Version 2.4.1. Object Management Group, 2012.

[138] A. Paivio. *Mental Representations: A Dual Coding Approach*. Oxford University Press, 1986.

[139] PriceWaterhouseCoopers. Rapport til Jernbaneverket – Revisjon av Anskaffelses- og Utviklingsprosessen for Merkur Sikringsanlegg. Retrieved May 1, 2014, from `http://brage.bibsys.no/jbv/bitstream/URN:NBN:no-bibsys_brage_39459/1/101929_ocr_red.pdf`, 2009.

[140] A. Ratzka. User Interface Patterns for Multimodal Interaction. In J. Noble, R. Johnson, U. Zdun, and E. Wallingford, editors, *Transactions on Pattern Languages of Programming III*, volume 7840 of *LNCS*, pages 111–167. Springer, 2013.

[141] A. Refsdal. *Specifying Computer Systems with Probabilistic Sequence Diagrams*. PhD thesis, Faculty of Mathematics and Natural Sciences, University of Oslo, 2008.

[142] J. R. Rumbaugh, M. R. Blaha, W. Lorensen, F. Eddy, and W. Premerlani. *Object-Oriented Modeling and Design*. Prentice-Hall, 1990.

[143] A. Rüping. Transform! Patterns for Data Migration. In J. Noble, R. Johnson, U. Zdun, and E. Wallingford, editors, *Transactions on Pattern Languages of Programming III*, volume 7840 of *LNCS*, pages 1–23. Springer, 2013.

[144] SAE. Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment, ARP 4761. Society of Automotive Engineers, 1996.

[145] M. Salehie and L. Tahvildari. Self-adaptive Software: Landscape and Research Challenges. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 4(2):1–42, 2009.

[146] D. Schmidt, M. Stal, H. Rohnert, and F. Buschmann. *Pattern-Oriented Software Architecture Volume 2: Patterns for Concurrent and Networked Objects*. Wiley, 2000.

[147] L. Sha. Using Simplicity to Control Complexity. *IEEE Software*, 18:20–28, 2001.

[148] J. Siddle. "Choose Your Own Architecture" - Interactive Pattern Storytelling. In J. Noble, R. Johnson, P. Avgeriou, N. Harrison, and U. Zdun, editors, *Transactions on Pattern Languages of Programming II*, volume 6510 of *LNCS*, pages 16–33. Springer, 2011.

[149] M. Sierhuis, J. M. Bradshaw, A. Acquisti, R. van Hoof, R. Jeffers, and A. Uszok. Human-agent teamwork and adjustable autonomy in practice. In *Proceedings of the Seventh International Symposium on Artificial Intelligence, Robotics and Automation in Space (i-SAIRAS'03)*. Retrieved May 1, 2014, from `http://robotics.estec.esa.int/i-SAIRAS/isairas2003/data/table.html`, 2003.

[150] G. Sindre. Boilerplates for Application Interoperability Requirements. In *Proceedings of 19th Norsk konferanse for organisasjoners bruk av IT (NOKOBIT'12)*. Tapir, 2012.

[151] J. M. Smith. *Elemental Design Patterns*. Addison-Wesley, 2012.

[152] F. Soares, J. Burken, and T. Marwala. Neural Network Applications in Advanced Aircraft Flight Control System, a Hybrid System, a Flight Test Demonstration. In *Neural Information Processing*, volume 4234 of *LNCS*, pages 684–691. Springer, 2006.

[153] I. Solheim and K. Stølen. Technology Research Explained. Technical Report A313, SINTEF ICT, Oslo, Norway, 2007.

[154] J. Spriggs. *GSN – The Goal Structuring Notation: A Structured Approach to Presenting Arguments*. Springer, 2012.

[155] Statens Jernbanetilsyn (SJT). Orstad Stasjon Ganddal Godsterminal – Avslag på søknad om tillatels til å ta i bruk signalanlegget. Retrieved 1 May, 2014, from `http://www.sjt.no/Global/Nyheter/Avslag-Merkur-Ganddal.pdf`, 2008.

[156] N. Storey. *Safety-critical Computer Systems*. Prentice Hall, 1996.

[157] N. Storey. Design for Safety. In *Towards System Safety: Proceedings of the 7th Safety-Critical Systems Symposium*, pages 1–25. Springer, 1999.

[158] G. S. Tallant, P. Bose, J. M. Buffington, V. W. Crum, R. A. Hull, T. Johnson, B. Krogh, and R. Prasanth. Validation & Verification of Intelligent and Adaptive Control Systems. In *2005 IEEE Aerospace Conference*, volume 1, pages 1–11. IEEE, 2005.

[159] B. J. Taylor. *Methods and Procedures for the Verification and Validation of Artificial Neural Networks*. Springer, 2005.

[160] The members of the Task Force on Safety Critical Software. Licensing of Safety Critical Software for Nuclear Reactors: Common Position of Seven European Nuclear Regulators and Authorised Technical Support Organisations. Retrieved 1 May, 2014, from `http://www.belv.be/images/pdf/12-10%20common%20position%20-%202013%20revision%20(secure).pdf`, 2013.

[161] W. F. Tichy. Should Computer Scientists Experiment More? *Computer*, 31(5):32–40, 1998.

[162] S. E. Toulmin. *The Uses of Argument*. Cambridge University Press, 1st edition, 1958.

[163] E. R. Tufte. *Envisioning Information*. Graphics Press, 1990.

[164] E. R. Tufte. *The Visual Display of Quantitative Information*. Graphics Press, 2nd edition, 2001.

[165] B. Tyler, F. Crawley, and M. Preston. *HAZOP: Guide to Best Practice*. Institution of Chemical Engineers (IChemE), 2nd edition, 2008.

[166] D. Vainsencher and A. P. Black. A Pattern Language for Extensible Program Representation. In J. Noble and R. Johnson, editors, *Transactions on Pattern Languages of Programming I*, volume 5770 of *LNCS*, pages 1–47. Springer, 2009.

[167] W. M. P. van der Aalst, A. P. Barros, A. H. M. ter Hofstede, and B. Kiepuszewski. Advanced Workflow Patterns. In *Proceedings of the 7th International Conference on Cooperative Information Systems (CoopIS'02)*, volume 1901 of *LNCS*, pages 18–29. Springer, 2000.

[168] W. M. P. van der Aalst and A. H. M. ter Hofstede. Workflow Patterns: On the Expressive Power of (Petri-net-based) Workflow Languages. In *Proceedings of the 4th International Workshop on Practical Use of Coloured Petri Nets and the CPN Tools*, pages 1–20, 2002.

[169] J. M. Vlissides. Notation, Notation, Notation. *C++ Report*, 10(4):48–51, 1998.

[170] C. Ware. *Visual Thinking for Design*. Morgan Kaufmann, 2008.

[171] M. Wertheimer. Laws of Organization in Perceptual Forms. In W. D. Ellis, editor, *A sourcebook of Gestalt Psychology*, pages 71–88. Routledge and Kegan Paul, 1938.

[172] J. Whittle, P. Sawyer, N. Bencomo, B. H. C. Cheng, and J.-M. Bruel. Relax: Incorporating Uncertainty Into the Specification of Self-adaptive Systems. In *Proceedings of 17th IEEE International Requirements Engineering Conference (RE'09)*, pages 79–88. IEEE Computer Society, 2009.

[173] T. Winograd and F. Flores. *Understanding Computers and Cognition: A New Foundation for Design*. Addison-Wesley, 1987.

[174] S. Withall. *Software Requirement Patterns (Best Practices)*. Microsoft Press, 1st edition, 2007.

[175] W. Wong, A. Demel, V. Debroy, and M. Siok. Safe Software: Does It Cost More to Develop? In *Fifth International Conference on Secure Software Integration and Reliability Improvement (SSIRI'11)*, pages 198–207. IEEE Computer Society, 2011.

[176] R. K. Yin. *Case Study Research: Design and Methods*. SAGE Publications, 4th edition, 2008.

[177] E. S. K. Yu and J. Mylopoulos. Using Goals, Rules, and Methods to Support Reasoning in Business Process Reengineering. In *Proceedings of the 27th Annual Hawaii International Conference on System Sciences*, volume 4, pages 234–243. IEEE Computer Society, 1994.

[178] W. Zimmer. Relationships between Design Patterns. In *Pattern Languages of Program Design*, pages 345–364. Addison-Wesley, 1994.

[179] B. Zimmermann, C. Rensing, and R. Steinmetz. Experiences in Using Patterns to Support Process Experts in Process Description and Wizard Creation. In J. Noble, R. Johnson, P. Avgeriou, N. Harrison, and U. Zdun, editors, *Transactions on Pattern Languages of Programming II*, volume 6510 of *LNCS*, pages 34–61. Springer, 2011.

Part II

Research Papers

Chapter 9

Paper 1: Syntax & Semantics of the SaCS Pattern Language

HWR-1052

OECD HALDEN REACTOR PROJECT

Syntax & Semantics of the SaCS Pattern Language

by

André A. Hauge[1,3] and Ketil Stølen[2,3]

[1]*Institute for energy technology, OECD Halden Reactor Project, Department of Software Engineering*

[2]*SINTEF ICT, Department of Networked Systems and Services*

[3] *University of Oslo, Department of Informatics*

andre.hauge@hrp.no, ketil.stolen@sintef.no

2013-11-11

FOREWORD

The experimental operation of the Halden Boiling Water Reactor and associated research programmes are sponsored through an international agreement by:

- the Institutt for energiteknikk (IFE), Norway,
- the Belgian Nuclear Research Centre SCK•CEN, acting also on behalf of other public or private organisations in Belgium,
- the Risø DTU National Laboratory for Sustainable Energy, Technical University of Denmark,
- the Finnish Ministry of Employment and the Economy (TYÖ),
- the Electricité de France (EDF),
- the Gesellschaft für Anlagen- und Reaktorsicherheit (GRS) mbH, representing a German group of companies working in agreement with the German Federal Ministry of Economics and Technology,
- the Japan Nuclear Energy Safety Organization (JNES),
- the Korean Atomic Energy Research Institute (KAERI), acting also on behalf of other public or private organisations in Korea,
- the Spanish Centro de Investigaciones Energéticas, Medioambientales y Tecnológicas (CIEMAT), representing a group of national and industry organisations in Spain,
- the Swedish Radiation Safety Authority (SSM), representing public and privat nuclear organisations in Sweden,
- the Swiss Federal Nuclear Safety Inspectorate ENSI, representing also the Swiss nuclear utilities (Swissnuclear) and the Paul Scherrer Institute,
- the National Nuclear Laboratory (NNL), representing a group of nuclear licensing and industry organisations in the United Kingdom, and
- the United States Nuclear Regulatory Commission (USNRC),

and as associated parties:

- Japan Atomic Energy Agency (JAEA),
- the Central Research Institute of Electric Power Industry (CRIEPI), representing a group of nuclear research and industry organisations in Japan
- the Mitsubishi Nuclear Fuel Co., Ltd. (MNF)
- the Czech Nuclear Research Institute (NRI),
- the French Institut de Radioprotection et de Sûreté Nucléaire (IRSN),
- the Ulba Metallurgical Plant JSC in Kazakhstan,
- the Hungarian Academy of Sciences, KFKI Atomic Energy Research Institute,
- the JSC "TVEL" and NRC "Kurchatov Institute", Russia,
- All-Russian Research Institute for Nuclear Power Plants Operation (VNIIAES), Russia,
- the Slovakian VUJE - Nuclear Power Plant Research Institute, and
- EU JRC Institute for Transuranium Elements, Karlsruhe,

and associated parties from USA:

- the Westinghouse Electric Power Company, LLC (WEC),
- the Electric Power Research Institute (EPRI),
- the Global Nuclear Fuel (GNF) – Americas, LLC and GE-Hitachi Nuclear Energy, LLC, and
- the US Department of Energy (DOE)

The right to utilise information originating from the research work of the Halden Project is limited to persons and undertakings specifically given this right by one of these Project member organisations.

Recipients are invited to use information contained in this report to the discretion normally applied to research and development programmes. Recipients are urged to contact the Project for further and more recent information on programme items of special interest.

Institutt for energiteknikk
OECD HALDEN REACTOR PROJECT

Title

Syntax & Semantics of the SaCS Pattern Language

Author:

André Alexandersen Hauge and Ketil Stølen

Document ID:
HWR-1052

Keywords:
pattern language; syntax; semantics;

Abstract:

HWR-1052 describes the syntax and semantics of the SaCS pattern language. SaCS is a pattern-based method that defines the systematic application of SaCS patterns for design conceptualisation of safety critical systems. The method is supported by the SaCS pattern language that consists of a set of patterns and a notation for describing the combined application of patterns.

We present a textual syntax and a structured semantics for the different kinds of SaCS patterns. The SaCS pattern language offers six kinds of patterns, referred to as basic patterns, organised in a library of patterns. The pattern language also defines a notation that allows the user to specify a composition of SaCS patterns. The result of combining patterns into a pattern composition is referred to as a composite SaCS pattern and represents the seventh kind of pattern supported in the language.

The SaCS pattern language is applied on two cases, presented in HWR-1029 and HWR-1037, respectively.

Issue Date:		Name	Signature	Date
2013-11-11	Prepared by:	André A. Hauge	Sign.	2013-09-09
Confidential grade:	Reviewed by:	Terje Sivertsen	Sign.	2013-11-11
HRP Only	Approved by:	Terje Johnsen	Sign.	2013-11-11

MAIL
P.O. BOX 173
NO-1751 HALDEN
Norway

TELEPHONE
+47 69 21 22 00

TELEFAX
Administration + 47 69 21 22 01
Nuclear Safety + 47 69 21 22 01
Purchasing Office + 47 69 21 24 40

TELEFAX
IND/OC div. + 47 69 21 24 90
VISIT/RID/COSS div. + 47 69 21 24 60
Reactor Plant + 47 69 21 24 70

TABLE OF CONTENTS

1. INTRODUCTION

The SaCS pattern language consists of a set of predefined patterns and different kinds of operators for combining patterns. The intention is to offer guidance on development of conceptual safety designs in a format that is easy to comprehend and apply. By a conceptual safety design we mean an early stage specification of system requirements, system design, and safety case for a safety critical system. The intended users of the SaCS pattern language are system developers, safety engineers, and HW/SW engineers.

A specification of a conceptual safety design is the accumulated result of combining a number of intermediate development results, e.g., a specification of functional requirements and a result of risk assessment. In order to provide confidence that a specific system design is suitable for a given application, it is not necessarily sufficient to present the system design itself – the process of developing the system design should also be presented. If the applied process for developing a system design is adequate, it provides additional assurance for the result being based on a proper set of activities and increases confidence in the result being correct and sufficient.

The SaCS pattern language provides, in addition to development guidance, a means to visualise and document the sequential application of patterns. The visualisation intends to capture the development process in a manner that is easy to comprehend. Composite patterns are user defined and describes how a set of patterns are combined and applied as support for conceptual safety design. A composite pattern may identify every pattern applied during safety design conceptualisation, the order in which the different patterns shall be applied, the input and output parameters of each pattern, the relationships between patterns, and optionally how patterns are instantiated. Thus, a composite pattern may serve as a means to model a reusable combination of patterns that may be applied in different contexts, or document the process of applying patterns as development support.

We distinguish between two main groups of patterns, basic patterns and composite patterns, the former presented as a combination of text and illustrations in a defined sequence of named sections and the latter represented graphically as a combination of the former. The basic patterns provide guidance on different aspects of development, e.g., requirement elicitation, safety assessment, system design, and safety demonstration challenges.

In order to ease the comprehension of SaCS patterns, this report offers a structured semantics. The structured semantics enables the user to extract the meaning of a SaCS pattern, which is expressed with graphics and text, and translate it into readable paragraphs in English prose based on a systematic translation of the elements that defines the pattern.

The remainder of this report is structured as follows: Section 2 describes the approach to our structured semantics. Section 3 describes the syntax of basic SaCS patterns. Section 4 describes the structured semantics of basic SaCS patterns. Section 5 describes the graphical syntax of composite SaCS patterns. Section 6 describes the textual syntax of composite SaCS patterns. Section 7 describes a structured semantics of composite SaCS patterns. Section 8 exemplifies the translation of basic SaCS patterns. Section 9 exemplifies the translation of composite SaCS patterns. Section 10 concludes.

2. THE APPROACH TO A STRUCTURED SEMANTICS

The main intention of a pattern in SaCS is to provide guidance on a solution to a recurring challenge and aid the communication between different users on how a certain challenge should be approached. The semantics for a given language may be approached in different ways depending on considerations such as who are the intended users, what is the purpose, and what are the needs for precision in terms of providing mathematical and logical definitions versus a more informal natural language definition [10] of the semantics. The semantics of SaCS patterns as presented here is primarily intended to aid communication between users on the interpretation of defined patterns without requiring any specific training. Our structured semantics is based on a systematic translation of pattern fragments into sentences or paragraphs in English.

The method for providing the structured semantics of SaCS patterns is adapted from [1] and performed in two steps:

- The translation of a SaCS pattern into its textual syntax, and
- The translation of the textual syntax of a pattern into a meaningful text in English.

The textual syntax of SaCS patterns is defined by the use of Extended Backus-Naur Form (EBNF) [7]. The vertical bar "|" represents options, braces "{ }" means an ordered sequence of zero or more repetitions of the enclosed element, "{ }$^{-}$" means and ordered sequence of one or more repetitions of the enclosed element, square brackets "[ ]" denotes optional features, comma "," represents the concatenation symbol, and semicolon ";" marks the termination of a rule. In EBNF, parentheses cannot be placed next to identifiers but must be concatenated with them. We take advantage of the constricted use of parentheses in EBNF and extend the syntax description with terminals (denoted in a **PT Sans bold** font for better readability) placed adjacent to a parenthesis to symbolise a syntactic fragment for which there is associated a semantic rule.

The following exemplifies a fragment of a textual syntax described with the use of EBNF:

name = **name**(*identifier*);

The following exemplifies a semantic rule for the above textual syntax that may be used to translate a textual syntax into English.

⟦ **name**(*identifier*) ⟧ = *identifier* is a pattern.

The symbols ⟦ ⟧ denote a function that takes a syntactical element defined according to a specific textual syntax as input and provides its translation as a sentence or a paragraph in English as output.

In the following sections, the syntax and semantics of SaCS patterns are defined. We do not provide a very strict syntax of basic SaCS patterns, although some elements of a basic pattern are associated with a strict syntax, and consequently the semantics of basic patterns is detailed at a high level. The reason is to allow pattern authors a high degree of freedom in defining basic patterns. In addition, a basic pattern should be self-explanatory. The syntax and semantics of composite SaCS patterns are given at a detailed level.

Examples on the translation of basic patterns and composite patterns are given in Section 8 and 9, respectively.

3. SYNTAX OF BASIC SACS PATTERNS

This section defines the textual syntax of basic SaCS patterns. There are six different kinds of basic patterns in SaCS, three kinds for patterns reflecting on issues associated with the development process (process assurance patterns) and three kinds for patterns reflecting upon issues of the product that is under development (product assurance patterns).

basic pattern = *process assurance requirement pattern* |
process assurance solution pattern |
process assurance safety case pattern |
product assurance requirement pattern |
product assurance solution pattern |
product assurance safety case pattern;

The syntax for the different kinds of basic patterns is as follows:

process assurance requirement pattern =
name, process requirement pattern signature, intent,
applicability, problem, process solution, instantiation rule,
[*related patterns*], [*known uses*];

process assurance solution pattern =
name, process solution pattern signature, intent,
applicability, problem, method solution, instantiation rule,
[*related patterns*], [*known uses*];

process assurance safety case pattern =
name, process safety case pattern signature, intent,
applicability, problem, argument structure solution, instantiation rule,
[*related patterns*], [*known uses*];

product assurance requirement pattern =
name, product requirement pattern signature, intent,
applicability, problem, problem frame solution, instantiation rule,
[*related patterns*], [*known uses*];

product assurance solution pattern =
name, product solution pattern signature, intent,
applicability, problem, design solution, *instantiation rule,*
[*related patterns*], [*known uses*];

product assurance safety case pattern =
name, product safety case pattern signature, intent,
applicability, problem, argument structure solution, instantiation rule,
[*related patterns*], [*known uses*];

In the following definitions, the terms *identifier, prose,* and *documentation* are not defined. The term *identifier* is assumed to be an alphanumeric string. The term *prose* is assumed to represent a sequence of words in English that is complete in itself. The term *documentation* is not defined but is assumed to be a combination of prose and some drawing, table or diagram. As mentioned, a terminal symbol (written in **PT Sans bold** font to increase readability) is used to define different syntactical elements.

name = **name**(*identifier*);

Immediately following the section on the naming of a pattern there is section containing an illustration of the signature of the pattern. Each basic pattern has a section that defines how the pattern may be referenced graphically in a composite pattern. The pattern signatures are defined slightly differently depending the kind of pattern defined.

pattern signature = *process requirement pattern signature* |
process solution pattern signature |
process safety case pattern signature |
product requirement pattern signature |
product solution pattern signature |
product safety case pattern signature;

process requirement pattern signature =
procreqpatternsignature(*identifier*, [*input*], *output*);
process solution pattern signature =
procsolpatternsignature(*identifier*, [*input*], *output*);
process safety case pattern signature =
procsafcasepatternsignature(*identifier*, [*input*], *output*);
product requirement pattern signature =
prodreqpatternsignature(*identifier*, [*input*], *output*);
product solution pattern signature =
prodsolpatternsignature(*identifier*, [*input*], *output*);
product safety case pattern signature =
prodsafcasepatternsignature(*identifier*, [*input*], *output*);

Each pattern signature identifies the respective input and output parameters of the pattern that is defined.

input = **input**({ *parameter* }$^{-}$);
output = **output**({ *parameter* }$^{-}$);

A parameter may be one of four different kinds.

parameter = *requirement parameter* | *design parameter* |
safety case parameter | *documentation parameter;*

requirement parameter = **requirementparameter**(*identifier*);
design parameter = **designparameter**(*identifier*);
safety case parameter = **safetycaseparameter**(*identifier*);
documentation parameter = **documentationparameter**(*identifier*);

Figure 1 illustrates the translation of the different kinds of pattern signatures as defined by their graphical syntax into their corresponding textual syntax. In Figure 1, a term is given to the left followed by an example of the graphical syntax of the named element. The right pointing white arrow with a black stroke should be interpreted “translates to” and separates the graphical syntax (or example of a graphical element) from its textual syntax (described in EBNF).

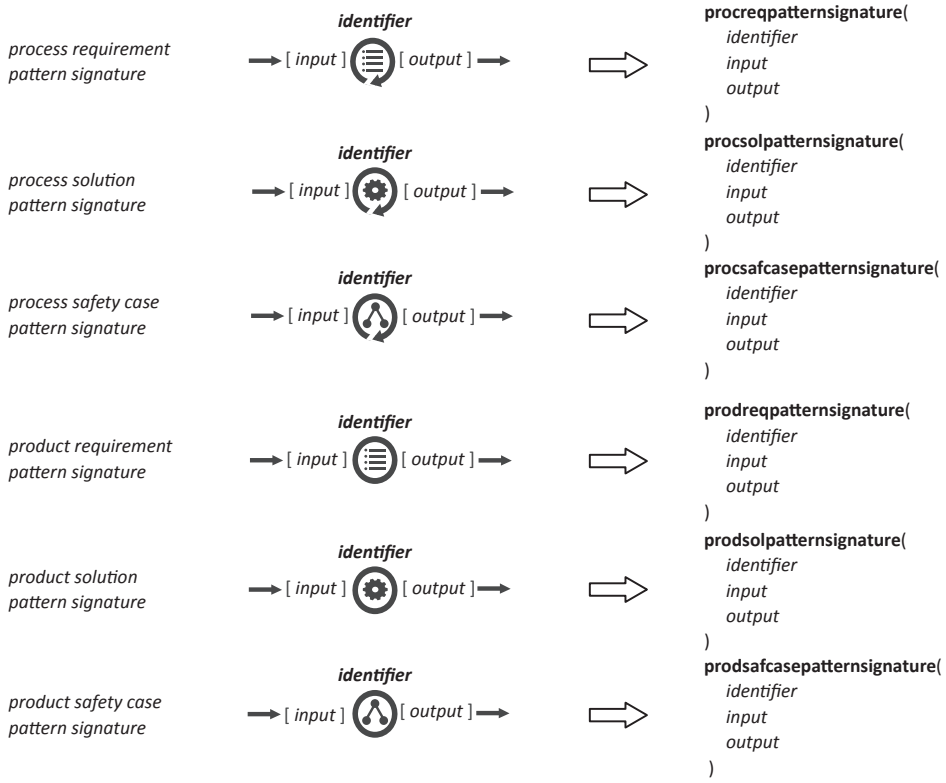


Figure 1 Pattern Signatures

The *input* and *output* terms in Figure 1 represents parameters separated by comma.

Figure 2 illustrates the graphical syntax and the corresponding textual syntax of the different kinds of parameters.

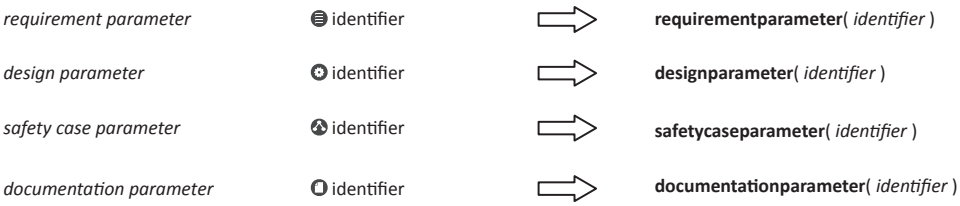


Figure 2 Different kinds of parameters

After the definition of the pattern signature, the following sections are presented in a pattern definition:

intent = **intent**(*documentation*);
applicability = **applicability**(*prose*);
problem = **problem**(*documentation*);

Once the problem addressed by a pattern is described, a solution to that problem is defined. Depending on the kind of *basic pattern* defined, the respective solutions are described according to slightly different formats.

process solution = **processolution**(*uml activity diagram, documentation*);
design solution = **designsolution**(*uml diagram, documentation*);
method solution = **methodsolution**(*arbitrary diagram, documentation*);
argument structure solution = **argstructuresolution**(*gsn diagram, documentation*);
problem frame solution = **problemframesolution**(*problem frames diagram, documentation*);

In the following, we loosely define the syntax for the different kinds of diagrams that may be used to detail the solution part of a pattern. Annotations are added to, e.g., a UML [11] diagram or a GSN [2][9] diagram in order to denote how the input and output parameters of a pattern relate to entities in the diagrams that illustrate a solution in a pattern.

diagram = uml activity diagram | uml diagram | arbitrary diagram |
gsn diagram | problem frames diagram;

In the following, the terms *uml, gsn, pfd,* and *illustration* are only defined by a *reference* that may be assumed to represent an alphanumeric string that identifies the diagram. The referenced diagram however is defined according to the syntax of UML [11], GSN [2], PFD (Problem Frames Diagram) [3][8], or some arbitrary format, respectively.

uml activity diagram = **umlactivitydiagram**(*uml*,
{ *input to activity* }, { *output from activity* }$^{-}$);

uml = **uml**(*reference*);
input to activity = **inputtoactivity**(*inputparameter, activity*);
output from activity = **outputfromactivity**(*outputparameter, activity*);

inputparameter = **inputparameter**(*identifier*);
outputparameter = **outputparameter**(*identifier*);
activity = **activity**(*identifier*);

uml diagram = **umldiagram**(*uml,*
{ *input to entity* }, { *output from entity* });

input to entity = **inputtoentity**(*inputparameter,* [*entity*]);
output from entity = **outputfromentity**(*outputparameter,* [*entity*]);
entity = **entity**(*identifier*);

arbitrary diagram = **arbitrarydiagram**(*illustration*
{ *input to entity* }, { *output from entity* });
illustration = **illustration**(*reference*);

gsn diagram = **gsndiagram**(*gsn,*
{ *goal* }$^{-}$, { *context* }, {*undeveloped goal* }, { *solution* });

gsn = **gsn**(*reference*);
goal = **goal**(*identifier*);
context = **context**(*identifier*);
undeveloped goal = **undevelopedgoal**(*identifier*);
solution = **solution**(*identifier*);

problem frames diagram = **problemframesdiagram**(*pfd,*
{ *input to machine* }$^{-}$, { *input to problem domain* },
{ *input to description* }, { *input to requirement* },
{ *output from requirement* }$^{-}$);

pfd = **pfd**(*reference*);
input to machine = **inputtomachine**(*inputparameter, machine*);
input to problem domain = **inputtoproblemdomain**(*inputparameter, problem domain*);
input to description = **inputtodescription**(*inputparameter, description*);
input to requirement = **inputtorequirement**(*inputparameter, requirement*);
output from requirement = **outputfromrequirement**(*outputparameter, requirement*);

machine = **machine**(*identifier*);
problemdomain = **problemdomain**(*identifier*);
description = **description**(*identifier*);
requirement = **requirement**(*identifier*);

The diagrams used for illustrating the different concepts presented in SaCS patterns are annotated versions of diagrams defined in languages like UML [11], GSN [2], and PFD [8]. We do not define the syntax and semantics of every kind of diagram or illustration that may be part of a composite SaCS pattern but assume the following:

- *uml activity* diagram is a UML [11] activity diagram with the addition of SaCS specific annotations. The annotations identify how each input and output parameter of a pattern is represented as either an input or an output to the different activities identified in an activity diagram.
- *uml diagram* is a UML diagram with the addition of SaCS specific annotations. The annotations identify how the input and output parameters of a pattern relate to items in a UML diagram, e.g., a component, a lifeline, or a class.
- *arbitrary* diagram is a diagram in a format that either requires no explanation in order to be comprehended, or that is accompanied with a legend or a description, or that is defined in a notation that has a defined syntax and semantics. An example of a type of illustration that has its own syntax and semantics but addressed by the generic semantic rules of an arbitrary diagram is the fault tree diagram [6] used in the pattern named FTA presented in Section 8.3. The SaCS specific annotations indicate a relationship between the input and output parameters of a pattern and entities illustrated in the diagram.
- *gsn diagram* is a diagram defined according to the Goal Structured Notation [2] with the addition of SaCS specific annotations. The annotations identify how the input and output parameters of a pattern relate to elements in a GSN diagram, e.g., goal, undeveloped goal, solution, and context.
- *problem frames* diagram is a diagram described according to the syntax of the problem frames notation [8] with the addition of SaCS specific annotations. The annotations identify how the input and output parameters of a pattern relate to elements in a problem frames diagram, e.g., machine, problem domain, description, and requirement.

In Figure 3, only the SaCS specific annotations that are used to indicate the relationship between parameters and entities in a specific diagram type are exemplified. Input and output parameters of a pattern are identified by their name inside a dotted drawn rectangle that are superimposed on a dotted drawn frame that surrounds the original illustration. An arrow pointing from a parameter and inwards indicates an input parameter; an arrow pointing from within the dotted frame and towards the box placed on the frame indicates an output parameter.

In the case of the *uml diagram*, and the *arbitrary diagram* in Figure 3, a diamond is used to represent any shape of the original illustration that may be connected to either an input parameter or an output parameter and thus does not represent any specific shape.

In the case of the *gsn diagram* in Figure 3, inputs and outputs are symbolised by placing GSN specific graphical elements on the dotted frame.

In order to assure consistency between the specification of a pattern signature and the specification of the solution part of a pattern, we define the following constraints with respect to the definition of the annotations of diagrams:

- An input parameter that is indicated in a *diagram* shall be explicitly denoted as an input parameter in the respective *pattern signature* of a pattern.
- An output parameter that is indicated in a *diagram* shall be explicitly denoted as an output parameter in the respective *pattern signature* of a pattern.

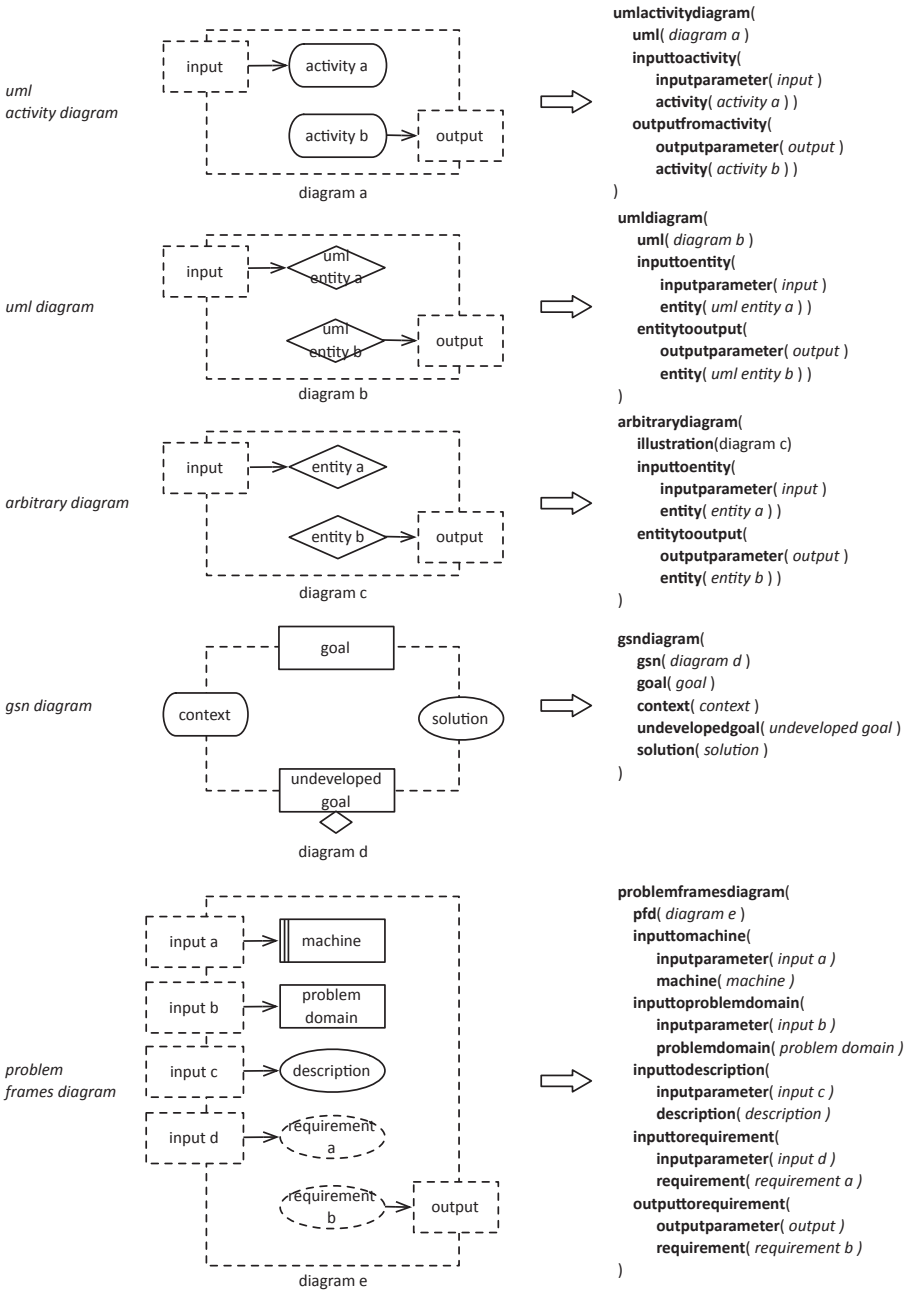


Figure 3 Annotations to diagrams for indicating relationships to input and output parameters

There is a relationship between the definitions of input and output parameters in a pattern signature and the definitions of input and output parameters in the respective diagrams used to define a solution in a pattern. As an example there is a relationship between the definition of a *process requirement pattern signature* and the *process solution* in the definition of a *process assurance requirement pattern.* This

relationship is not defined in the syntax but captured in the following constraints on the definition of basic patterns:

- The identifier of an input parameter given in the *pattern signature* section of a pattern should be explicitly denoted with the same identifier and an indication on what part of the solution it is associated with in the diagram given in the respective *solution* part of a pattern.
- The identifier of an output parameter given in the *pattern signature* section of a pattern should be explicitly denoted with the same identifier and an indication on what part of the solution it is associated with in the diagram given in the respective *solution* part of a pattern.

Once the solution part of a pattern is described, the pattern definition contains a section defining its *instantiation rule* followed by the definition of *related patterns* and *known uses*. The related patterns and known uses sections are optional. The term *pattern identifier* is assumed to represent an alphanumeric string.

instantiation rule = **instantiationrule**({ *parameter, pattern identifier*, { *rule* }$^{-}$ }$^{-}$);
rule = **rule**(*prose*);
related patterns = **relatedpatterns**(*pattern identifier*, *prose*);
known uses = **knownuses**(*prose*);

4. SEMANTICS OF BASIC SACS PATTERNS

This section describes the semantics of basic SaCS patterns. A basic pattern should be expressed in such a way that its semantics is understood from its definition. A pattern definition consists of prose and illustrations structured and grouped under a sequence of headings. The names given to headings indicate the content of the subsequent part of the pattern definition and it should be possible for a user to derive the semantics of the pattern from its different parts. Nevertheless, we provide semantic rules that may be systematically applied in order to generate a further explanation in English of the meaning of the different elements of a pattern definition.

A basic pattern is translated into a set of paragraphs in English where each fragment within the pattern definition is translated into either a sentence within a paragraph or a paragraph. The semantics is defined by a function ⟦ ⟧ that takes fragments of a pattern definition as its input and returns as output its translation in English. The semantics of a basic pattern is the semantics of each of the elements that defines the patterns.

⟦ **name**(*identifier*) ⟧ =
identifier is a generalised solution to the challenges described in the following.

In the following we define the semantics for the different kinds of pattern signatures. In the definitions, let *in* and *out* range over *parameter*. The semantics of the different types of parameters is defined immediately below the semantics for the different pattern signatures.

⟦ **procreqpatternsignature**(*identifier,* $in_1, \ldots, in_n, out_1, \ldots, out_n$) ⟧ =
identifier belongs to a category of patterns that addresses challenges appearing in a development context by defining its solution as a process and by defining requirements for documenting the results of the process. The pattern requires the input: ⟦ in_1 ⟧,…, ⟦ in_n ⟧
The pattern delivers the output*:* ⟦ out_1 ⟧,…, ⟦ out_n ⟧

⟦ **procsolpatternsignature**(*identifier,* $in_1, \ldots, in_n, out_1, \ldots, out_n$) ⟧ =
identifier belongs to a category of patterns that defines solutions to recurring development

challenges in the form of a method. The pattern requires the input: ⟦ in_1 ⟧,…, ⟦ in_n ⟧ The pattern delivers the output: ⟦ out_1 ⟧,…, ⟦ out_n ⟧

⟦ **procsafcasepatternsignature**(*identifier,* in_1,…, in_n, out_1,…, out_n) ⟧ =
identifier belongs to a category of patterns that addresses challenges of demonstrating that safety objectives are met and defines its solution as a structure of arguments. The pattern requires the input: ⟦ in_1 ⟧,…, ⟦ in_n ⟧ The pattern delivers the output: ⟦ out_1 ⟧,…, ⟦ out_n ⟧

⟦ **prodreqpatternsignature**(*identifier,* in_1,…, in_n, out_1,…, out_n) ⟧ =
identifier belongs to a category of patterns that defines a set of abstract requirements and a problem frames analysis solution as a means to address the challenges of eliciting requirements for a particular context. The pattern requires the input: ⟦ in_1 ⟧,…, ⟦ in_n ⟧ The pattern delivers the output: ⟦ out_1 ⟧,…, ⟦ out_n ⟧

⟦ **prodsolpatternsignature**(*identifier,* out_1,…, out_n) ⟧ =
identifier belongs to a category of patterns that defines a design solution as a means to handle a commonly occurring development challenge. The pattern delivers the output: ⟦ out_1 ⟧,…, ⟦ out_n ⟧.

⟦ **prodsafcasepatternsignature**(*identifier,* in_1,…, in_n, out_1,…, out_n) ⟧ =
identifier belongs to a category of patterns that addresses challenges of demonstrating that safety objectives are met and defines its solution as a structure of arguments. The pattern requires the input: ⟦ in_1 ⟧,…, ⟦ in_n ⟧ The pattern delivers the output: ⟦ out_1 ⟧,…, ⟦ out_n ⟧

The following defines the semantics for the different kinds of parameters:

⟦ **requirementparameter**(*identifier*) ⟧ = A set of requirements *identifier*.
⟦ **designparameter**(*identifier*) ⟧ = A specification of the design of *identifier*.
⟦ **safetycaseparameter**(*identifier*) ⟧ = A safety case element *identifier*.
⟦ **documentationparameter**(*identifier*) ⟧ = A description *identifier*.

The following defines the semantics of the *intent, applicability*, and *problem* sections of a pattern.

⟦ **intent**(*documentation*) ⟧ = The intent of the pattern is to: *documentation*
⟦ **applicability**(*prose*) ⟧ = The scenarios for which the pattern typically may be applied are as follows: *prose*
⟦ **problem**(*documentation*) ⟧ = The main challenges addressed by the pattern are described in the following: *documentation*

The solution sections of basic patterns are defined slightly different depending on the kind of basic pattern that is expressed. The following defines the semantics of the different kinds of solution descriptions:

⟦ **processolution**(*uml activity diagram, documentation*) ⟧ =
The process solution is defined by ⟦ *uml activity diagram* ⟧ *documentation*
⟦ **designsolution**(*uml diagram, documentation*) ⟧ =
The solution, defined by ⟦ *uml diagram* ⟧ *documentation*
⟦ **methodsolution**(*arbitrary diagram, documentation*) ⟧ =
The method solution is exemplified by ⟦ *arbitrary diagram* ⟧ *documentation*
⟦ **argstructuresolution**(*gsn diagram, documentation*) ⟧ =
The safety case solution is defined by ⟦ *gsn diagram* ⟧ *documentation*
⟦ **problemframesolution**(*problem frames diagram, documentation*) ⟧ =

The solution is described as a requirement analysis, defined by
⟦ *problem frames diagram* ⟧ *documentation*

The following defines the semantics of the different kinds of annotated diagrams. In the semantics, we do not define every mapping rule but the most important ones. In addition, we do not translate the terms referred to as *uml*, *gsn*, *pdf*, and *illustration* as these represent parts of an annotated diagram in SaCS that are defined with a syntax and semantics defined elsewhere, e.g., as presented in UML [11], GSN [2], and PFD [8]. As an example, a translation of a diagram as in "diagram a" in Figure 3, only the SaCS specific annotations will be translated. The remaining UML specific parts of the diagram will only be referred in the translation as ⟦ **uml**(*diagram a*) ⟧, indicating that the meaning of the referred diagram parts that are not translated should be interpreted according the semantics of UML. In the semantics, let *id*(_) be a function that takes a *uml, gsn, pfd,* or *illustration* element as input and returns the associated *identifier*.

⟦ **umlactivitydiagram**(*uml, input to activity, output from activity*) ⟧ =
⟦ *uml* ⟧ In *id*(*uml*), the following relationships between parameters of the pattern and activities are defined: ⟦ *input to activity* ⟧ ⟦ *output from activity* ⟧

⟦ **inputtoactivity**(*input parameter, activity*) ⟧ =
The instantiation of ⟦ *input parameter* ⟧ is used for performing the ⟦ *activity* ⟧.
⟦ **outputfromactivity**(*output parameter, activity*) ⟧ =
Performing the ⟦ *activity* ⟧ produces the ⟦ *output parameter* ⟧.
⟦ **inputparameter**(*identifier*) ⟧ = input *identifier*
⟦ **outputparameter**(*identifier*) ⟧ = output *identifier*
⟦ **activity**(*identifier*) ⟧ = activity *identifier*

⟦ **umldiagram**(*uml, input to entity, entity to output*) ⟧ =
⟦ *uml* ⟧ In *id*(*uml*), the following relationships between parameters of the pattern and UML entities are defined: ⟦ *input to entity* ⟧ ⟦ *entity to output* ⟧

⟦ **inputtoentity**(*input parameter, entity*) ⟧ =
The instantiation of ⟦ *input parameter* ⟧ is associated with ⟦ *entity* ⟧.
⟦ **inputtoentity**(*input parameter*) ⟧ = ⟦ *input parameter* ⟧ represents a required input.
⟦ **outputfromentity**(*outputparameter, entity*) ⟧ = ⟦ *entity* ⟧ *is associated with* ⟦ *output parameter* ⟧
⟦ **outputfromentity**(*outputparameter*) ⟧ = ⟦ *output parameter* ⟧ represents an instantiation result.

⟦ **entity**(*identifier*) ⟧ = *identifier*

⟦ **arbitrarydiagram**(*illustration, input to entity, entity to output*) ⟧ =
⟦ *illustration* ⟧. In *id*(*illustration*), the following relationship between parameters of the pattern and the elements in the illustration are defined: ⟦ *input to entity* ⟧ ⟦ *entity to output* ⟧

⟦ **gsndiagram**(*gsn, goal, context, undeveloped goal, solution*) ⟧ =
⟦ *gsn* ⟧. In *id*(*gsn*), the following elements represent the parameters of the pattern:
⟦ *goal* ⟧ ⟦ *context* ⟧⟦ *undeveloped goal* ⟧ ⟦ *solution* ⟧

⟦ **goal**(*identifier*) ⟧ =
identifier represents an output of the pattern once instantiated.
⟦ **context**(*identifier*) ⟧ =
identifier represents an input that describes the context of the associated element in the diagram.
⟦ **undevelopedgoal**(*identifier*) ⟧ =

identifier represents an output that needs to be instantiated and further developed.

⟦ **solution**(*identifier*) ⟧ =
identifier represents an input required to be provided for supporting the claim described by the associated element in the diagram.

⟦ **problemframesdiagram**(*pfd, input to machine, input to problem domain, input to description, input to requirement, output to requirement*) ⟧ =
⟦ *pfd* ⟧ In *id*(*pfd*), the following relationships between parameters of the pattern and entities in the diagram are defined: ⟦ *input to machine* ⟧ ⟦ *input to problem domain* ⟧ ⟦ *input to description* ⟧ ⟦ *input to requirement* ⟧ ⟦ *output from requirement* ⟧

⟦ **inputtomachine**(*input parameter, machine*) ⟧ =
The instantiation of ⟦ *input parameter* ⟧ is associated with the ⟦ *machine* ⟧.
⟦ **inputtoproblemdomain**(*input parameter, problem domain*) ⟧ =
The instantiation of ⟦ *input parameter* ⟧ is associated with the ⟦ *problem domain* ⟧.
⟦ **inputtodescription**(*input parameter, description*) ⟧ =
The instantiation of ⟦ *input parameter* ⟧ is associated with the ⟦ *description* ⟧.
⟦ **inputtorequirement**(*input parameter, requirement*) ⟧ =
The instantiation of ⟦ *input parameter* ⟧ is associated with the ⟦ *requirement* ⟧.
⟦ **outputfromrequirement**(*output parameter, requirement*) ⟧ =
The instantiation of the ⟦ *requirement* ⟧ is associated with ⟦ *output parameter* ⟧.

⟦ **machine**(*identifier*) ⟧ = machine named *identifier*
⟦ **problemdomain**(*identifier*) ⟧ = problem domain named *identifier*
⟦ **description**(*identifier*) ⟧ = description named *identifier*
⟦ **requirement**(*identifier*) ⟧ = requirement named *identifier*

The following describes the semantics for *instantiation rule*, *related patterns*, and *known uses*.

⟦ **instantiationrule**(*parameter, pattern identifier, rule*$_1$*,…, rule*$_n$) ⟧ =
The pattern *pattern identifier* is instantiated by instantiating the output *parameter* according to the following rules: *rule*$_1$*,…, rule*$_n$
⟦ r**elatedpatterns**(*pattern identifier*, *prose*) ⟧ =
The following describes patterns that are closely related to *pattern identifier*: *prose*
⟦ **knownuses**(*prose*) ⟧ =
The following describes known uses of the pattern: *prose*

5. GRAPHICAL SYNTAX OF COMPOSITE SACS PATTERNS

Figure 4 exemplifies a composite SaCS pattern; the filled arrows and associated text in the figure are added for highlighting the different kinds of graphical elements in SaCS. The original composite pattern without the added annotations is presented in Section 9.1 along with its semantics. In the following we will explain the different graphical elements that appears in Figure 4 and more for specifying composite patterns.

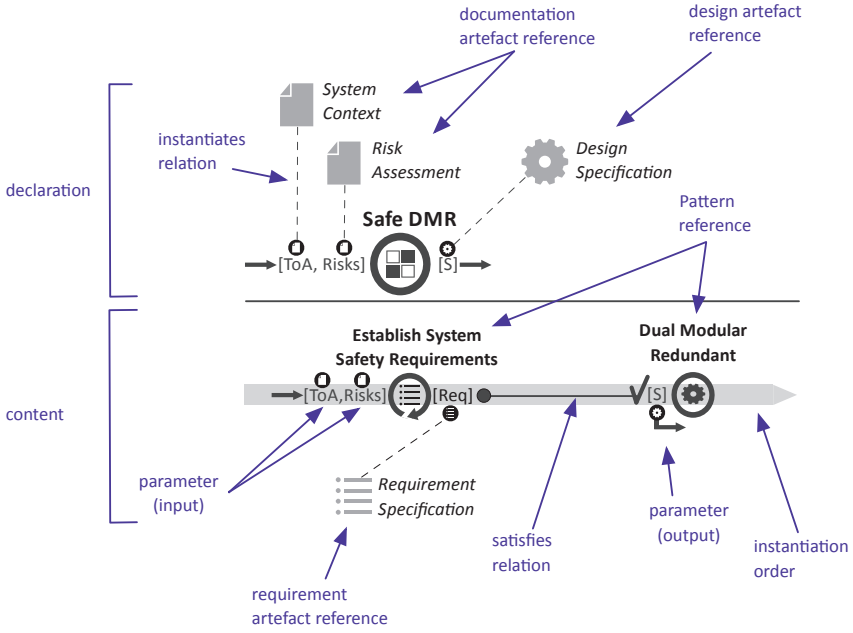


Figure 4 Example Composite Pattern with explanations

Figure 5 gives an example of how the different graphical elements in SaCS are described in the coming sections. In Figure 5, a term representing a naming of a graphical element or a group of elements is given to the left followed by an example of the graphical syntax of the named element. The right pointing arrow should be interpreted “translates to” and separates the graphical syntax (or example of a graphical element) from the textual syntax (described in Extended Backus-Naur Form). In the definitions, the term *identifier* is not defined but is assumed to be an alphanumeric string. A terminal symbol (written in **PT Sans bold** font to increase readability) is used to define different types of syntactical elements.

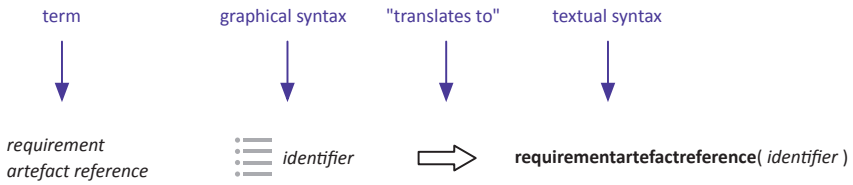


Figure 5 Example – Detailing the syntax of elements

5.1 Artefact references

Figure 6 illustrates the different artefact references that are available in SaCS. When a SaCS pattern is instantiated some artefacts may be required as inputs in order to instantiate the pattern, artefacts may also be provided as a result of pattern instantiation. The parameters of a pattern indicate what kinds of artefacts are required or what artefacts that are provided upon pattern instantiation.

The following icons are used to identify the different types of artefacts that may be referred to within a composite SaCS pattern.

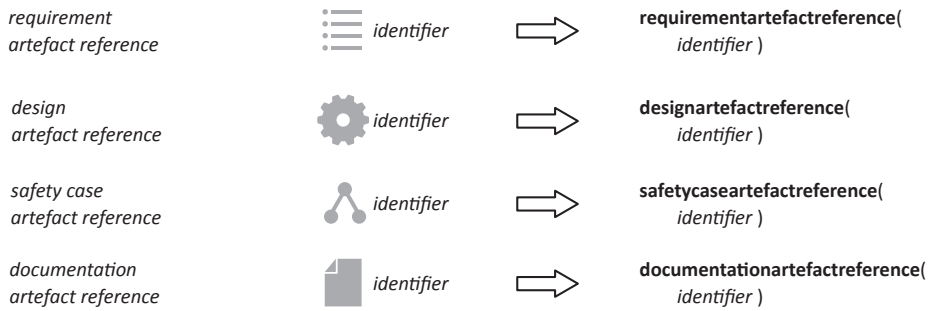


Figure 6 Icons identifying different types of artefacts

5.2 Parameters

Figure 7 illustrates the different kinds of parameters that are available in SaCS. In a composite pattern it is optional to include an icon symbolising the type of a parameter. The icons are simply smaller versions of the icons for symbolising artefact references but in white and placed inside a black circle. An icon is placed adjacent to an identifier for a parameter in order to symbolise its type.

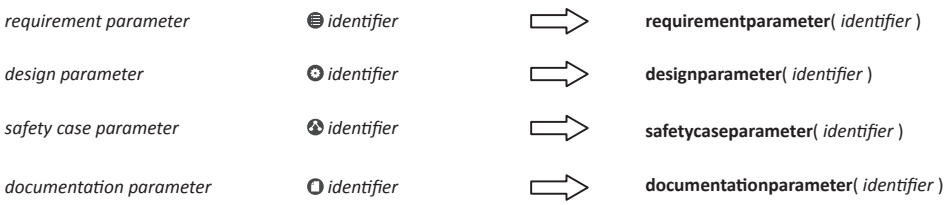


Figure 7 Icons for identifying different types of parameters

Figure 8 illustrates the different annotations that may be added to a parameter when defining a composite pattern.

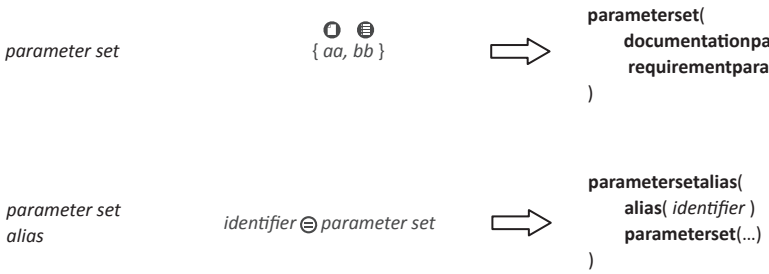


Figure 8 Alias and set annotations

The *parameter set* annotation in Figure 8 is used to denote that the parameters listed inside the curly brackets are elements of a set; curly brackets may be omitted when the set contains only one element.

The *parameter set alias* operator in Figure 8 is right associative; an identifier given to the left of the operator represents an alias for any parameter indicated to the right of the operator.

Figure 9 illustrates the different annotations that may be added to a parameter in order to define whether a parameter is either a local parameter or a public parameter and if the parameter is an input or an output. In Figure 9 we have assumed that the parameter that is annotated is a *documentation parameter*.

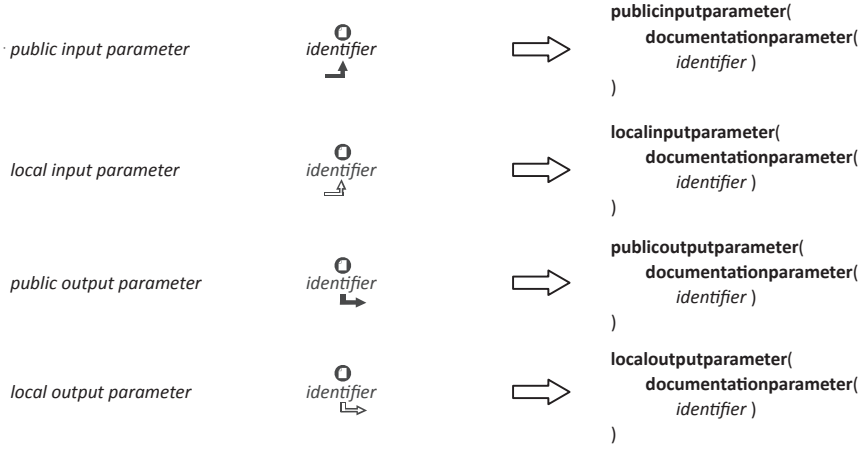


Figure 9 Input and output annotations

A parameter that is defined as *local* is only available within the definition of the composite pattern. A parameter that is defined as *public* is available within the definition of the composite pattern and also available externally to a user of the pattern.

Every parameter associated with a referenced pattern within a definition of a composite pattern is assumed to be local unless otherwise specified, thus the use of the white arrow with black stroke pointing either from or toward a parameter as illustrated in Figure 9 may be omitted.

5.3 Pattern references

Figure 10, Figure 11, and Figure 12 illustrate the different kinds of pattern references available in SaCS. There are seven different kinds of pattern references, six for referencing basic SaCS patterns (Figure 10 and Figure 11) and the seventh for referencing composite patterns (Figure 12).

Figure 10 illustrates the three different types of process assurance patterns references that may be used within a composite SaCS pattern.

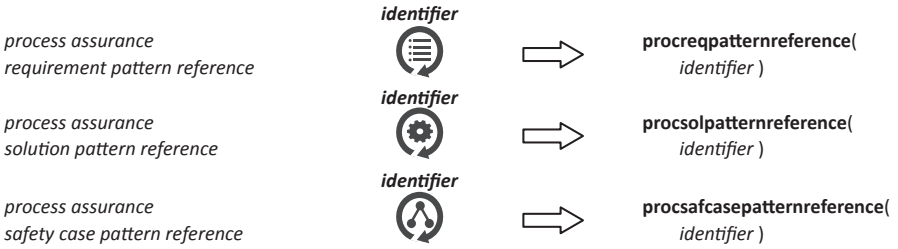


Figure 10 Icons for referencing different process assurance patterns

Figure 11 illustrates the three different types of product assurance patterns references that may be used within a composite SaCS pattern.

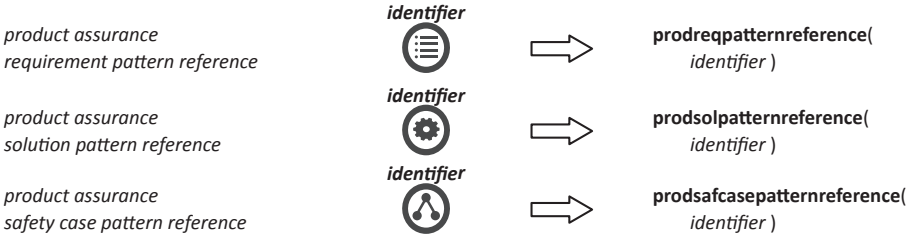


Figure 11 Icons for referencing different product assurance patterns

Figure 12 illustrates a composite pattern reference.

Figure 12 Icon for referencing composite patterns

Figure 13 exemplifies the use of the *pattern group reference* where a group of patterns is identified by the individual identifiers of each member of the group.

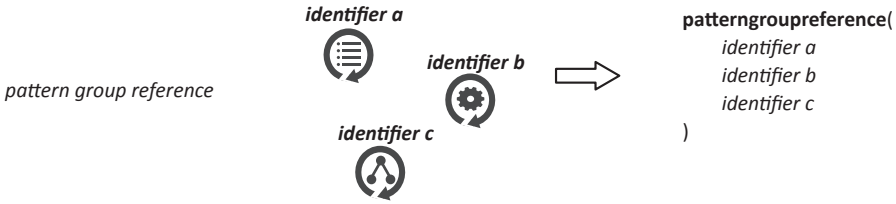


Figure 13 A pattern group reference

5.4 Relations

Figure 14 illustrates the different relations that are defined in SaCS. The *instantiates* relation is used to model a relationship between an artefact and a parameter, the remaining relations model a relationship between patterns in the form of a relationship between the respective parameters of the related patterns.

In Figure 14 we have assumed the existence of two composite patterns, named *A and B,* respectively, that are combined with a relation operator. We have assumed in each scenario that pattern *A* has a parameter *a*, and that pattern *B* has a parameter *b*.

Input and output parameters are always listed inside square brackets and placed adjacent to the belonging pattern. A comma is used as a separator when there is more than one parameter. The parameters of a pattern are always assumed to be local (*local input parameter* or *local output parameter*) in a relation unless otherwise specified. In a relation, there is no need to visualise whether a parameter is an input or an output as this is implied by the different relation types (see Section 6.4).

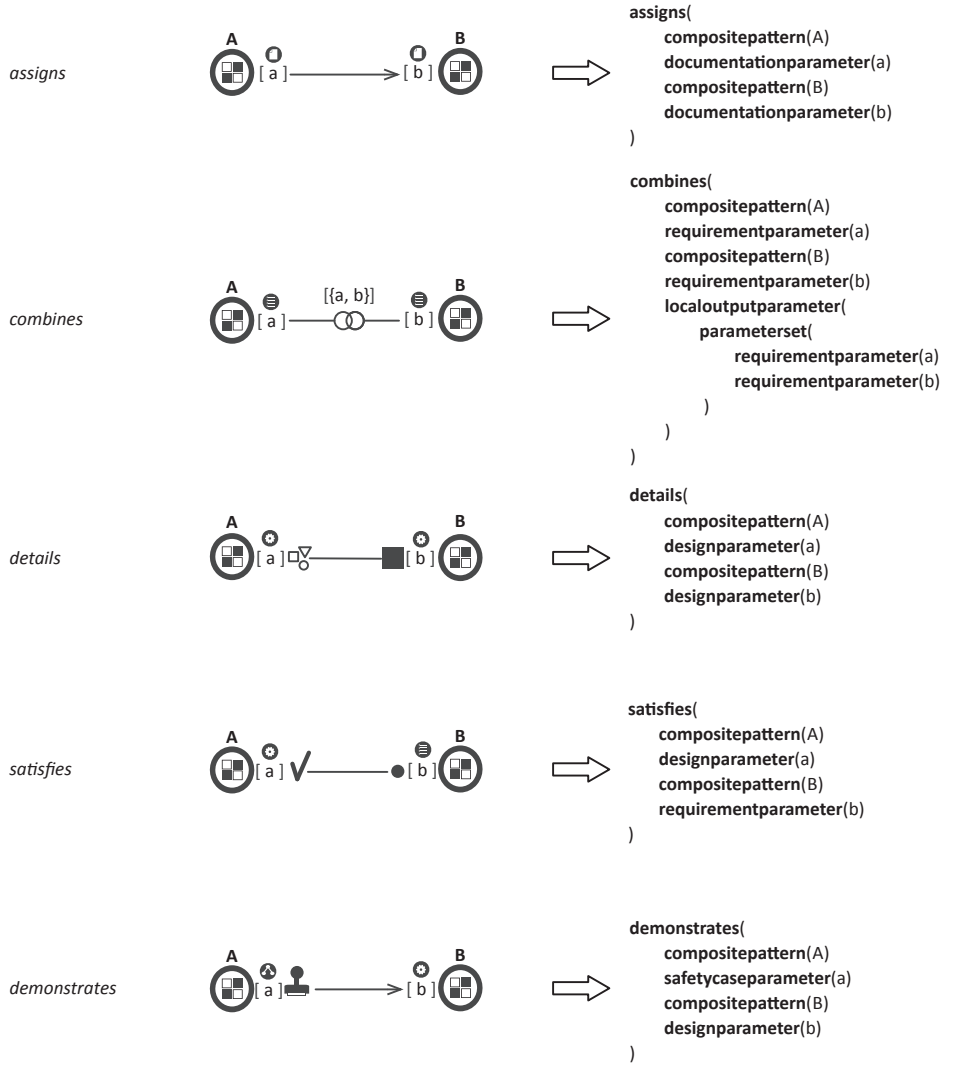


Figure 14 Graphical elements for symbolising relations

In a relation between patterns as illustrated in Figure 14, the relation operates on the parameters of the respective patterns. A relation between patterns may operate on multiple parameters; by a list matching of the respective parameter lists belonging to the patterns that are related.

Figure 15 exemplifies the translation of a relation where patterns have multiple parameters. In the relation illustrated topmost in Figure 15, *a* shall be assigned to *b*, and *aa* shall be assigned to *bb*. In the

relation illustrated at the bottom of Figure 15, *a* shall be assigned to *b*, *aa* shall be assigned to *bb*, and *aaa* shall be assigned to *bbb*. In addition, the icons for symbolising the type of a parameter are omitted in the illustration at the bottom of Figure 15 in order to simplify the illustration.

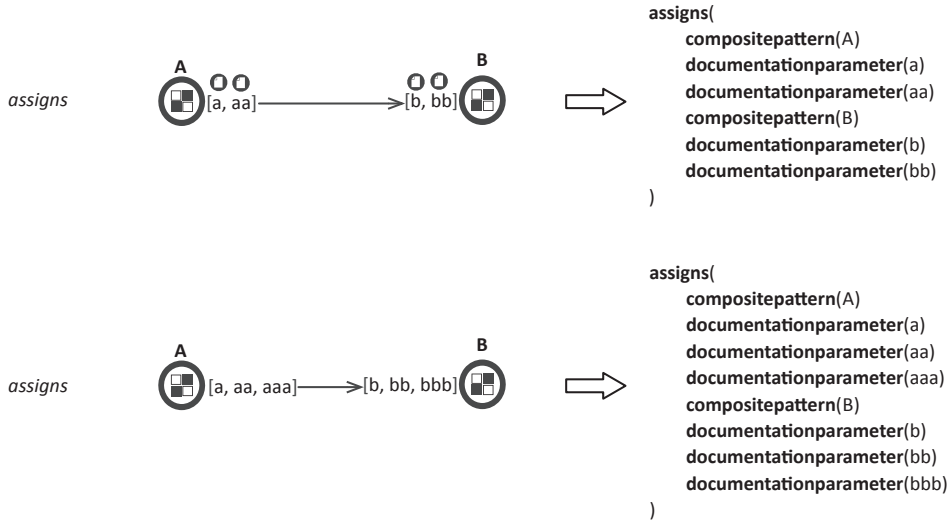


Figure 15 Parameter matching in relations

The *instantiates* relation is used to associate an artefact and a parameter and indicates that the artefact instantiates the parameter.

The *assigns* relation is used to denote that an output parameter of a pattern is assigned to an input parameter of a related pattern. The arrow always points towards the input parameter.

The *combines* relation is used to denote that the outputs of two or more patterns are combined in a union, the result is a set that comprises all outputs. A list (indicated by [_]) placed adjacent to the icon symbolising the relation denotes the result of combining.

The *details* relation is used to denote that an output of a pattern is detailed by the output of a related pattern. The black box is associated with the output that is detailed; the set of smaller icons is associated with the output that details.

The *satisfies* relation is used to denote that an output of type *requirement parameter* of a pattern is satisfied by the output of type *design parameter* of a related pattern. The bullet is associated with the output of type *requirement parameter* that describes what shall be satisfied; the checkmark is associated with the output of type *design parameter*. The relation indicates that the delivered design shall satisfy the requirements.

The *demonstrates* relation is used to denote that an output of type *safety case parameter* demonstrates safety of the output of type *design parameter* of a related pattern. The stamp icon is associated with the output that provides a safety demonstration; the arrow points towards the design output that is demonstrated safe.

5.5 Instantiation order

Figure 16 illustrates the additional guidance that may be given in a composite pattern on visualising the intended instantiation order of patterns. The intended instantiation order of patterns is visualised by superimposing pattern references on top of a grey arrow. The direction of the arrow indicates the pattern instantiation order, patterns placed closer to the starting point of the arrow are instantiated prior to patterns placed close to the tip of the arrow. Patterns may have no specific order; this is visualised as a parallel instantiation.

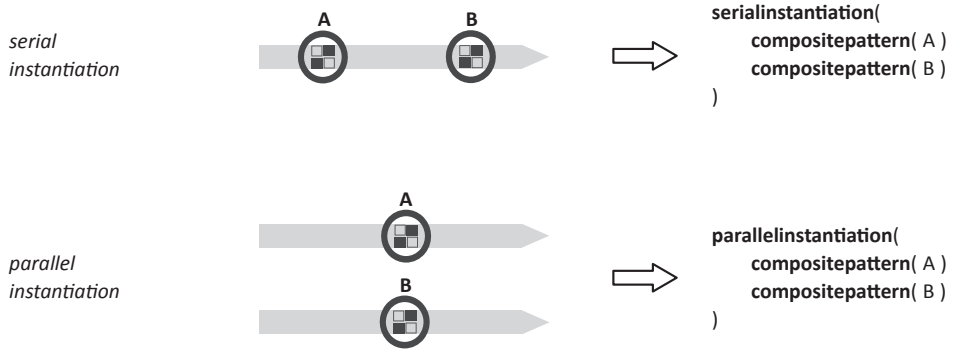


Figure 16 Illustrating the pattern instantiation order

In the example of the *serial instantiation*, two composite pattern references named *A* and *B* are superimposed in a sequence on a grey arrow indicating that *A* should be instantiated before *B*.

In the example of the *parallel instantiation*, each of the two composite pattern references *A* and *B* are superimposed on top of a grey arrow where the two arrows are separate, indicating that *A* and *B* may be instantiated in parallel.

5.6 Composite SaCS patterns

Figure 17 illustrates the declaration of a composite SaCS pattern. A composite pattern consists of a declaration followed by a specification of its content (see Figure 4). The declaration is recognised by an icon similar (but larger) to the one in a *composite pattern reference*. A line at the bottom of the declaration denotes the end of the declaration, the content of the composite is visualised below the line.

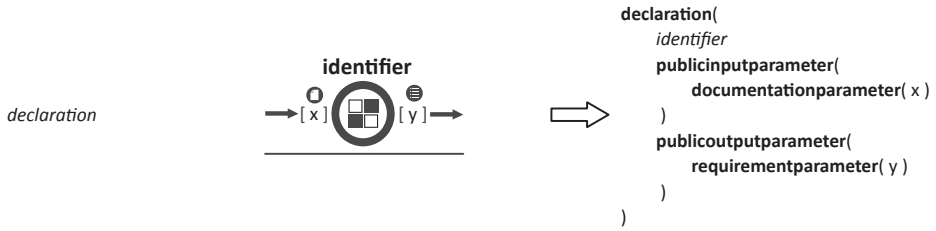


Figure 17 Example of a composite pattern declaration

The content of a composite pattern consists of a description of a combination of patterns by the use of the different graphical elements described in this section as well as the preceding sections (e.g., pattern references, parameters, artefacts references, and relations).

Figure 18 exemplifies how patterns that act within a composite pattern may be declared. In Figure 18 we have assumed the existence of a composite pattern named *P* in different scenarios where *P* is associated with different kinds of parameters (that are either input or output, and either local or public) named *x* and y.

The parameters of a pattern are visualised by enclosing parameter identifiers within square brackets that are placed adjacent to the respective pattern. The square brackets symbolise a list of parameters. The parameters are separated by comma.

In Figure 18 we use a short hand notation (see Figure 9) for illustrating that the parameters of a pattern are defined as local or public and if the parameters are defined as input or output. An arrow without a bend points either towards (indicating inputs) or from (indicating output) a list of parameters (the square brackets indicate a list, the identifiers inside the square brackets represent names of parameters).

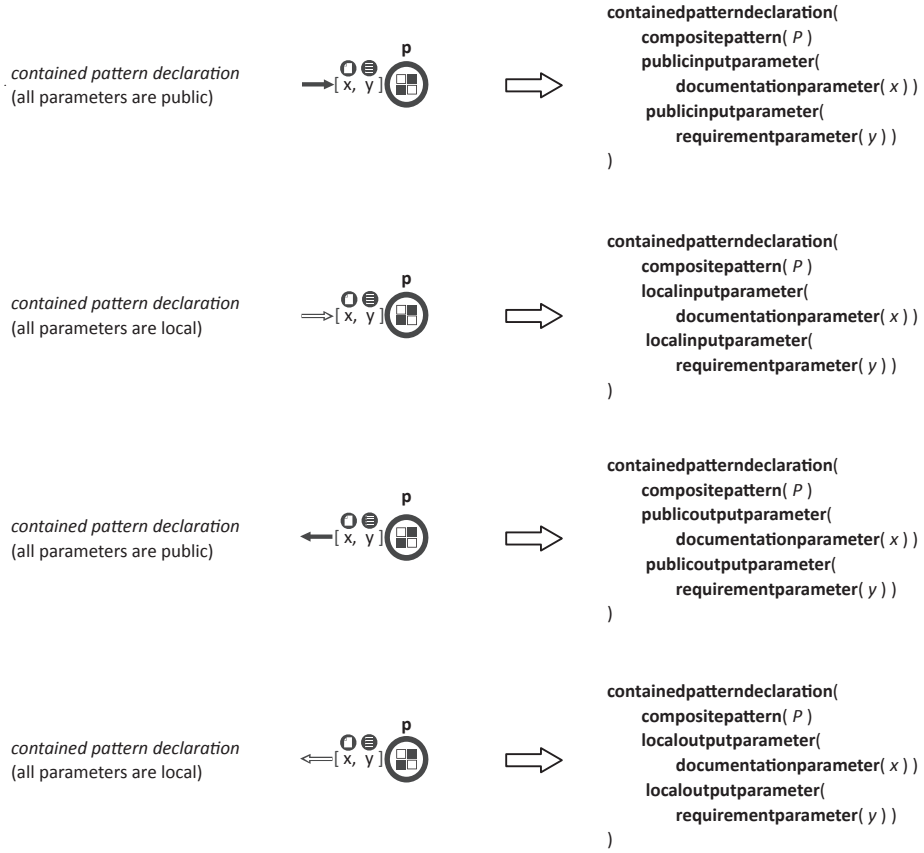


Figure 18 Contained pattern declaration examples

A black arrow symbolises that the parameters in the list are public and thus available as parameters of the composite that is defined. The white arrow with black stroke symbolises that the parameters in the list are only locally available within the composite that is defined.

6. TEXTUAL SYNTAX OF COMPOSITE SACS PATTERNS

This section defines the general textual syntax of composite SaCS patterns. The syntax is described by the use of EBNF (Extended Backus-Naur Form). In the definitions, the term *identifier* is not defined but is assumed to be an alphanumeric string. A terminal symbol (written in **PT Sans bold** font to increase readability) is used to define different types of syntactical elements.

6.1 Artefact references

A pattern in SaCS has parameters of different types, representing either the input to be provided when instantiating a pattern or an output provided upon pattern instantiation. An instantiation of a parameter is represented as an artefact. It is assumed that an identifier may uniquely identify a specific artefact, the concrete artefact may then be found by using the identifier given in a diagram as a key. A reference to different types of development artefacts may be given as follows:

requirement artefact reference = **requirementartefactreference**(*identifier*);
design artefact reference = **designartefactreference**(*identifier*);
safety case artefact reference = **safetycaseartefactreference**(*identifier*);
documentation artefact reference = **documentationartefactreference**(*identifier*);

The following term is used to denote a classifier for the different types of artefact references and is defined as follows:

artefact reference = requirement artefact reference | *design artefact reference*
safety case artefact reference | *documentation artefact reference;*

6.2 Parameters

A pattern in SaCS has parameters of different types. The identifier and type of a parameter is found in each pattern definition. The syntax for referring to the different types of parameters of a pattern are as follows:

requirement parameter = **requirementparameter**(*identifier*);
design parameter = **designparameter**(*identifier*);
safety case parameter = **safetycaseparameter**(*identifier*);
documentation parameter = **documentationparameter**(*identifier*);

The following term is used to denote a classifier for the different basic types of parameters that are used in the definitions of SaCS patterns.

basic parameter = design parameter | *requirement parameter* |
safety case parameter | *documentation parameter;*

When defining a composite SaCS pattern, additional annotations may be used to denote, e.g., an alias for a parameter or define a set of parameters. In addition, a parameter may be referred to by its alias.

parameter set alias = **parametersetalias**(*alias*, [*parameter set*]);
parameter set = **parameterset**({ *basic parameter* | *alias* }$^{-}$);
alias = **alias**(*identifier*);

annotated parameter = parameter set | *parameter set alias* | *alias;*

A parameter is either an input or an output parameter. However, an input parameter may also act as either a local parameter or as a public parameter. A local parameter is only used within the definition of a composite pattern and is not externally accessible while a public parameter is externally accessible.

public input parameter = **publicinputparameter**(*basic parameter* | *annotated parameter*) ;
local input parameter = **localinputparameter**(*basic parameter* | *annotated parameter*);
public output parameter = **publicoutputparameter**(*basic parameter* | *annotated parameter*);
local output parameter = **localoutputparameter**(*basic parameter* | *annotated parameter*);

input parameter = public input parameter | local input parameter;
output parameter = public output parameter | local output parameter;

parameter = input parameter | output parameter;

6.3 Pattern references

The definitions of the syntax for the different types of patterns in SaCS are as follows:

process assurance requirement pattern reference = **procreqpatternreference**(*identifier*);
process assurance solution pattern reference = **procsolpatternreference**(*identifier*);
process assurance safety case pattern reference = **procsafcasepatternreference**(*identifier*);

product assurance requirement pattern reference = **prodreqpatternreference**(*identifier*);
product assurance solution pattern reference = **prodsolpatternreference**(*identifier*);
product assurance safety case pattern reference = **prodsafcasepatternreference**(*identifier*);

composite pattern reference = **compositepatternreference**(*identifier*);

In some cases it is necessary to reference several patterns. In these cases, the group itself is unnamed and instead each individual pattern in the group is referenced by its identifier. A pattern group reference is defined as:

pattern group reference = **patterngroupreference**(*identifier,* { *indentifier* }$^{-}$);

The following restrictions apply to the specification of a *group pattern reference*:

- A *group* pattern *reference* may only be used in a *pattern relation* when referencing the associated patterns of an *output parameter* provided from applying the *combines* relation. A *combines* relation denotes that parameters from two or more patterns are combined, the *output parameter* provided as a result of combining does not have a single pattern reference associated with it. Thus, each *pattern reference* in the *combines* relation is referenced.

The following term defines a classifier for the different types of pattern references:

pattern reference = process assurance requirement pattern reference |
process assurance solution pattern reference |
process assurance safety case pattern reference |
product assurance requirement pattern reference |
product assurance solution pattern reference |
product assurance safety case pattern reference |
composite pattern reference |
pattern group reference;

6.4 Relations

In SaCS a relation may be defined between an artefact and a parameter associated with a pattern, or between patterns (more specifically between the parameters of the respective patterns that are related).

The following defines the syntax of the instantiates relation that is used to express that an artefact is an instantiation of a parameter:

instantiates = **instantiates**(*artefact reference, parameter*);

The following restrictions apply to the specification of an *instantiates* relation:

- The type of the artefact and the type of the parameter that the artefact is said to instantiate shall be the same.

A relation between two patterns operates on the parameters of the respective referenced patterns that are related. A *pattern input* is defined as a pattern reference followed by one or more input parameters. A *pattern output* is defined as a pattern reference followed by one or more output parameters.

pattern input = *pattern reference*, { *input parameter* }$^{-}$;
pattern output = *pattern reference* ,{ *output parameter* }$^{-}$;

In a relation between patterns, the parameters of the pattern that represents a source in the relation are matched (by list matching) with the parameters of the pattern that represents a target in the relation.

source = pattern output;
target = pattern input | *pattern output;*

The syntax of the different relations in SaCS are defined as:

assigns = **assigns**(*source, target*);
combines = **combines**(*source,* { *source* }$^{-}$*, output parameter*);
details = **details**(*source, target*);
satisfies = **satisfies**(*source, target*);
demonstrates = **demonstrates**(*source, target*);

The following term defines a classifier for the different types of pattern relations:

pattern relation = assigns | *combines* | *details* | *satisfies* | *demonstrates;*

The following restrictions apply to the specification of an *assigns, satisfies, demonstrates, details* and a *combines* relation:

- The source and *target* of a relation cannot be the same element.
- A target in an *assigns* relation is a *pattern input*; in the remaining relations both *source* and *target* are *pattern output*.
- The parameters of a *source* that are matched to the parameters of a *target* in a *details* relation must be of the same type.
- Each parameter of a *source* in a *satisfies* relation is a *design parameter*; each parameter of a *target* in a satisfies relation is a *requirement parameter*.
- Each parameter of a *source* in a *demonstrates* relation is a *safety case parameter*; each parameter of a *target* in a *demonstrates* relation is a *design parameter*.
- Every parameter within every *source* in a *combines* relation must be of the same type and are combined into a *parameter set* that represents the result of combining.

- The *output* parameter in a *combines* relation represents a *parameter set* or a *parameter set alias* that is defined with either local or public accessibility (defined as either a *local output parameter* or a *public output parameter*).

6.5 Instantiation order

A composite pattern may define the intended instantiation order of contained patterns. The instantiation order of patterns in a set of patterns is described by partial orders among the patterns. The following gives the syntax for describing a serial and parallel instantiation order between a pair of patterns.

instantiation order = *serial instantiation* | *parallel instantiation;*

serial instantiation = **serialinstantiation**(*pattern reference*, *pattern reference*);
parallel instantiation = **parallelinstantiation**(*pattern reference*, *pattern reference*);

The following constraints are associated with the definition of an instantiation order:

- A *serial instantiation* order indicates a serial sequence of instantiating patterns, the first pattern that is listed in the syntax is assumed to precede the second pattern that is listed.
- A *parallel instantiation* order indicates that patterns is not required to be performed in any sequence thus it does not matter in which order patterns are listed in the syntax.

6.6 Composite SaCS patterns

A composite pattern diagram may be said to be composed of a declaration and its content. In the declaration, an identifier gives the name of the pattern that is defined in addition to a list of the inputs and outputs of the composite and information on its instantiation. The content part of the pattern consists of a set of pattern references, relations, as well as the intended instantiation order of the contained patterns.

composite pattern = *declaration, content*;

declaration = **declaration**(*identifier*, { *input parameter* },
{ *output parameter* }, { *instantiates* });
content = **content**(*identifier,* { *contained pattern declaration* }$^{-}$,
{ *pattern relation* }, { *pattern instantiation order* });

contained pattern declaration = **containedpatterndeclaration**(
pattern reference, { *input parameter* },
{ *output parameter* }, { *instantiates* });

The following restrictions apply to the definition of *input parameter* and *output parameter* in the *declaration* part of a composite:

- Every *input parameter* to a composite is a *public input parameter* and is uniquely defined in the declaration.
- Every *output parameter* of a composite is a *public output parameter* and is uniquely defined in the declaration.
- A one-to-many relationship exists between a parameter defined *as input parameter* in the declaration of a composite and any similarly named *public input parameter* in the content part of the composite. The modelling of the instantiation of an input parameter in the declaration of a composite implies that every similarly named public input parameter contained in the composite is similarly instantiated.

- A one-to-one relationship is implied between a parameter defined as *output parameter* in the declaration of a composite and a similarly named *public output parameter* in the content part of the composite.

7. SEMANTICS OF COMPOSITE SACS PATTERNS

This section describes the semantics of composite SaCS patterns. The semantics is a systematic translation of a SaCS composite pattern diagram into English. A composite pattern diagram is translated into a paragraph in English where each fragment within the diagram is translated into a sentence or a paragraph.

The semantics is defined by functions ⟦ _ ⟧ that takes diagrams and fragments of diagrams expressed by their textual syntax as inputs and return the associated English translation.

7.1 Artefact reference

In the definitions of the semantics of the different types of artefacts, let *id* range over *identifier*.

⟦ **requirementartefactreference**(*id*) ⟧ = the requirements *id*
⟦ **designartefactreference**(*id*) ⟧ = the design *id*
⟦ **safetycaseartefactreference**(*id*) ⟧ = the safety case *id*
⟦ **documentationartefactreference**(*id*) ⟧ = the description *id*

7.2 Parameters

In the semantics of the different kinds of basic parameters in SaCS, let *id* range over *identifier*.

⟦ **requirementparameter**(*id*) ⟧ = a set of requirements *id*
⟦ **designparameter**(*id*) ⟧ = a design specification *id*
⟦ **safetycaseparameter**(*id*) ⟧ = a safety case specification *id*
⟦ **documentationparameter**(*id*) ⟧ = a description *id*

In the semantics of *parameter set alias* let *al* range over *alias, ps* range over *parameter set*, and *id* range over *identifier*.

⟦ **parametersetalias**(*al*) ⟧ = the set named ⟦ *al* ⟧
⟦ **parametersetalias**(*al*, *ps*) ⟧ = ⟦ *al* ⟧ (alias for ⟦ *ps* ⟧)
⟦ **alias**(*id*) ⟧ = *id*

In the semantics of parameter set, let *prm* range over *basic parameter* and *alias,* let *id*(_) be a function that takes a *basic parameter* or an *alias* as input and deliver the associated identifier as output.

⟦ **parameterset**(*prm*) ⟧ = *id*(*prm*)
⟦ **parameterset**($prm_1, \ldots, prm_n$) ⟧ = the set consisting of *id*(prm_1),…, and *id*(prm_n)

In the semantics for input and output parameters, let *p* range over *basic parameter* and *annotated parameter*. Let *id*(_) be a function that takes a *basic parameter* or an *annotated parameter* as input and returns its associated identifier. In the case where an input is defined as a *parameter set alias*, the function *id*(_) returns the semantics of the *parameter set alias*.

⟦ **publicinputparameter**(*p*) ⟧ = *id*(*p*) (public)
⟦ **localinputparameter**(*p*) ⟧ = *id*(*p*) (local)

⟦ **publicoutputparameter**(*p*) ⟧ = *id*(*p*) (public)
⟦ **localoutputparameter**(*p*) ⟧ = *id*(*p*) (local)

7.3 Pattern references

In the definitions of the semantics of the different types of pattern references, let *id* range over identifier.

⟦ **procreqpatternreference**(*id*) ⟧ = the process pattern *id*
⟦ **procsolpatternreference**(*id*) ⟧ = the method pattern *id*
⟦ **procsafcasepatternreference**(*id*) ⟧ = the safety case pattern *id*

⟦ **prodreqpatternreference**(*id*) ⟧ = the requirements pattern *id*
⟦ **prodsolpatternreference**(*id*) ⟧ = the design pattern *id*
⟦ **prodssafcasepatternreference**(*id*) ⟧ = the safety case pattern *id*

⟦ **compositepatternreference**(*id*) ⟧ = the composite pattern *id*

⟦ **patterngroupreference**($id_1, \ldots, id_n$) ⟧ = the patterns $id_1, \ldots,$ *and* id_n

7.4 Relations

In the definition of the semantics for the instantiates relations let *ar* range over *artefact reference* (semantics of artefact references described in Section 7.1) and *p* range over *parameter* (semantics of parameters described in Section 7.2).

⟦ **instantiates**(*ar, p*) ⟧ = ⟦ *ar* ⟧ represents *id(p)*

In the definition of the semantics for the assigns relation let *pr* range over *pattern reference* and *p* range over *parameter*.

⟦ **assigns** (pr_1, p_1, pr_2, p_2) ⟧ =
$id(p_1)$ is assigned to $id(p_2)$ where:
$id(p_1)$ is the output of ⟦ pr_1 ⟧.
$id(p_2)$ is input to ⟦ pr_2 ⟧.

⟦ **assigns** (pr_1, $p_{11}, \ldots, p_{1n}$, pr_2, $p_{21}, \ldots, p_{2n}$) ⟧ =
$id(p_{11})$ is assigned to $id(p_{21}), \ldots,$
$id(p_{1n})$ is assigned to $id(p_{2n})$ where:
$id(p_{11}), \ldots, id(p_{1n})$ are outputs from ⟦ pr_1 ⟧.
$id(p_{21}), \ldots, id(p_{2n})$ are inputs to ⟦ pr_2 ⟧.

In the semantics of the combines relation let *pr* range over *pattern reference*, *p* and *op* range over *output parameter*.

⟦ **combines**(pr_1, p_1, pr_2, $p_2, \ldots,$ pr_n, p_n, *op*) ⟧ =
$id(p_1), \ldots,$ and $id(p_n)$ are combined in a union where:
$id(p_1)$ is the output from ⟦ pr_1 ⟧.
$id(p_2)$ is the output from ⟦ pr_2 ⟧.
. . .
$id(p_n)$ is the output from ⟦ pr_n ⟧.
The result of combining is represented by ⟦ *op* ⟧.

⟦ **combines**(pr_1, p_{11},…, p_{1n}, pr_2, p_{21},…, p_{2n},…, pr_m, p_{m1},…, p_{mn}, *op*) ⟧ =
id(p_{11}),…, *id*(p_{1n}), *id*(p_{21}) ,…, *id*(p_{2n}),…, and *id*(p_{mn}) are combined in a union where:
id(p_{11}),…, *id*(p_{1n}) are the outputs from ⟦ pr_1 ⟧.
id(p_{21}),…, *id*(p_{2n}) are the outputs from ⟦ pr_2 ⟧.
. . .
id(p_{m1}),…, *id*(p_{mn}) are the outputs from ⟦ pr_m ⟧.
The result of combining is represented by ⟦ *op* ⟧.

In the semantics of the details relation let *pr* range over *pattern reference* and *p* range over *parameter*.

⟦ **details**(pr_1, p_1, pr_2, p_2) ⟧ =
id(p_1) is required to detail *id*(pr_2) where:
id(p_1) is the output from ⟦ pr_1 ⟧.
id(p_2) is the output from ⟦ pr_2 ⟧.

⟦ **details**(pr_1, p_{11},…, p_{1n}, pr_2, p_{21},…, p_{2n}) ⟧ =
id(p_{11}) is required to detail *id*(p_{21}),…, *id*(p_{1n}) is required to satisfy *id*(p_{2n}) where:
id(p_{11}),…, *id*(p_{1n}) are the outputs from ⟦ pr_1 ⟧.
id(p_{21}),…, *id*(p_{2n}) are the outputs from ⟦ pr_2 ⟧.

In the semantics of the satisfies relation let *pr* range over *pattern reference* and *p* range over *parameter*.

⟦ **satisfies**(pr_1, p_1, pr_2, p_2) ⟧ =
id(p_1) is required to satisfy the requirements defined by *id*(pr_2) where:
id(p_1) is the output from ⟦ pr_1 ⟧. *id*(p_2) is the output from ⟦ pr_2 ⟧.
⟦ **satisfies**(pr_1, p_{11},…, p_{1n}, pr_2, p_{21},…, p_{2n}) ⟧ =
id(p_{11}) is required to satisfy the requirements defined by *id*(p_{21}),…,
id(p_{1n}) is required to satisfy the requirements defined by *id*(p_{2n}) where:
id(p_{11}),…, *id*(p_{1n}) are the outputs from ⟦ pr_1 ⟧.
id(p_{21}),…, *id*(p_{2n}) are the outputs from ⟦ pr_2 ⟧.

In the semantics of the *demonstrates* relation let *pr* range over *pattern reference* and *p* range over *parameter*.

⟦ **demonstrates**(pr_1, p_1, pr_2, p_2) ⟧ =
id(p_1) is required to demonstrate safety of *id*(pr_2) where:
id(p_1) is the output from ⟦ pr_1 ⟧. *id*(p_2) is the output from ⟦ pr_2 ⟧.
⟦ **demonstrates**(pr_1, p_{11},…, p_{1n}, pr_2, p_{21},…, p_{2n}) ⟧ =
id(p_{11}) is required to demonstrate safety of *id*(p_{21}),…,
id(p_{1n}) is required to demonstrate safety of *id*(p_{2n}) where:
id(p_{11}),…, *id*(p_{1n}) are the outputs from ⟦ pr_1 ⟧.
id(p_{21}),…, *id*(p_{2n}) are the outputs from ⟦ pr_2 ⟧.

7.5 Instantiation order

In the semantics of instantiation orders let *pr* range over *pattern reference* and let *id*(_) be a function that takes a pattern reference as input and returns the identifier of the pattern.

⟦ **serialinstantiation**(pr_1, pr_2) ⟧ = The pattern *id*(pr_1) should be instantiated before the pattern *id*(pr_2).
⟦ **parallelinstantiation**(pr_1, pr_2) ⟧ = The patterns *id*(pr_1) and *id*(pr_2) may be instantiated in parallel.

7.6 Composite SaCS patterns

The semantics of a composite pattern is defined by the semantic rules associated with the definition of the declaration and the content part of the composite. In the semantics, let *ip* range over *input parameter*, let *op* range over *output parameter*, and let *inst* range over the relation *instantiates*. Let *param*(_) be a function that takes an *input parameter* or an *output parameter* as input and returns the associated *basic parameter* or *annotated parameter* as output.

⟦ **declaration**(*identifier, $ip_{1,..}$, ip_n, $op_{1,..}$, op_n, $inst_{1,..}$, $inst_n$*) =
identifier requires the input: ⟦ *param*(ip_1) ⟧, …, ⟦ *param*(ip_n) ⟧
An assignment of an input of *identifier* is assigned to every correspondingly named input with public accessibility of contained patterns. An input with public accessibility is when introduced in the following demarked “(public)”, otherwise demarked “(local)”.
identifier delivers the output: ⟦ *param*(op_1) ⟧, …, ⟦ *param*(op_n) ⟧
An output of *identifier* is delivered when the correspondingly named output, with public accessibility, is delivered from a contained pattern. An output with public accessibility is when introduced in the following demarked “(public)”, otherwise demarked “(local)”.
identifier is instantiated such that: ⟦ $inst_1$ ⟧, …, ⟦ $inst_n$ ⟧

In the semantics of the content part of a composite pattern, let *cpd* range over *contained pattern declaration*, *r* range over *pattern relation, and io* range over *instantiation order*.

⟦ **content**(*identifier, $cpd_{1,..}$, cpd_n, $r_{1,..}$, r_n, $io_{1,..}$, io_n*) ⟧ =
identifier contains the patterns *id*(cpd_1),…, and *id*(cpd_n). ⟦ cpd_1 ⟧, …, ⟦ cpd_n ⟧. ⟦ r_1 ⟧, …, ⟦ r_n ⟧. *identifier* defines the following order for instantiating patterns: ⟦ io_1 ⟧, …, ⟦ io_n ⟧.

In the semantics of *contained pattern declaration*, let *pr* range over pattern reference, *ip* range over *input parameter*, *op* range over *output parameter*, and *inst* range over the relation *instantiates*.

⟦ **containedpatterndeclaration**(*pr, op*) ⟧ =
⟦ *op* ⟧ is the output of ⟦ *pr* ⟧.
⟦ **containedpatterndeclaration**(*pr, $op_{1,..}$, op_n*) ⟧ =
⟦ op_1 ⟧,…, ⟦ op_n ⟧ are the outputs of ⟦ *pr* ⟧.
⟦ **containedpatterndeclaration**(*pr, ip, op*) ⟧ =
⟦ *op* ⟧ is the output of ⟦ *pr* ⟧ when applied to ⟦ *ip* ⟧.
⟦ **containedpatterndeclaration**(*pr, $ip_{1,..}$, ip_n, , op*) ⟧ =
⟦ *op* ⟧ is the output of ⟦ *pr* ⟧ when applied to ⟦ ip_1 ⟧,.., and ⟦ ip_n ⟧.
⟦ **containedpatterndeclaration**(*pr, $ip_{1,..}$, ip_n, $op_{1,..}$, op_n*) ⟧ =
⟦ op_1 ⟧,…, ⟦ op_n ⟧ are the outputs of ⟦ *pr* ⟧ when applied to ⟦ ip_1 ⟧,…, and ⟦ ip_n ⟧
⟦ **containedpatterndeclaration**(*pr, $ip_{1,..}$, ip_n, , $op_{1,..}$, op_n, $inst_{1,..}$, $inst_n$*) ⟧ =
⟦ **containedpatterndeclaration**(*pr, $ip_{1,..}$, ip_n, , $op_{1,..}$, op_n*) ⟧
id(pr) is instantiated such that: ⟦ $inst_1$ ⟧ … ⟦ $inst_n$ ⟧.

8. EXAMPLE TRANSLATIONS OF BASIC PATTERNS

In this section we exemplify the translation of basic SaCS patterns, one from each of the six different kinds of basic patterns according to the categorisation described in [4] and [5]. Similar kinds of basic patterns are expressed according to the same format. As the rules for translating patterns are generic with minor differences between patterns from different categories we find it sufficient to exemplify the translation on six of the twenty-six basic SaCS patterns that is defined such that a pattern from each of the six categories is represented.

In each example translation we detail in separate sub-sections:

- the original pattern definition;
- the pattern expressed by its textual syntax;
- the semantics of the pattern as provided by the semantic rules given in Section 4.

A pattern definition is translated into its textual syntax and further into its semantics section by section from start to end.

We have formatted the presentation of the textual syntax slightly in order to increase readability. The sentences and paragraphs found in a pattern definition are written in *italics*. Whenever the symbols "…" are found in the specification of a pattern by its textual syntax or in the translation, the text is shortened in order to avoid unnecessary repetition of text. The full text is provided in the original pattern definition.

In the translations, text in italics is extractions from the pattern definition and non-italics text is generated from the semantic rules. The translated text is slightly formatted for increased readability.

8.1 Example 1: Establish System Safety Requirements

8.1.1 Pattern definition

Name: Establish System Safety Requirements

Pattern Signature: *Establish System Safety Requirements* is defined with the signature illustrated in Figure 19.

In Figure 19, the following abbreviations are used for denoting the parameters of the pattern:

- *ToA* is short for Target of Assessment.
- *Reg* is short for Regulations.
- *Risks* is not abbreviated but represents the documentation of the risks associated with the application of *ToA* in its intended context.
- *Req* is short for Requirements.

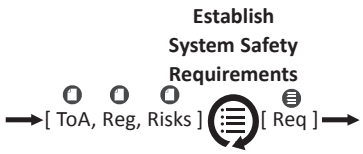


Figure 19 Establish System Safety Requirements Pattern Signature

Intent: Support the specification of system safety requirements *Req* on the basis of a risk-based approach. The safety requirements describe the required measures to be satisfied by the system *ToA* such that if satisfied assure the necessary safety integrity. The general approach for defining safety requirements is to define these on the basis of the result of a risk assessment *Risks*, especially the

mitigations identified as means to reduce risk to an acceptable level. The pattern describes the general process of addressing relevant aspects of the system that must be resolved in order to assure safety and how to capture this in the form of requirements.

Applicability: The *Establish System Safety Requirements* pattern is intended for the following situations:

- When the system under construction may negatively affect the overall system safety.
- When there are identified measures that suitably mitigate identified risks and thus provide input to the specification of safety requirements.

Problem: The main aspects relevant to establishing the safety requirements are:

- *Characteristics*: To define requirements that require certain system characteristics to be satisfied such that the occurrence of unwanted events are minimised or avoided.
- *Functions*: To define requirements that require certain safety functions to be satisfied in order assure safe operations.
- *Constraints*: To define requirements that require certain functional constrains to be satisfied in order to delimit potentially hazardous operations.
- *Environment*: To define requirements that require certain operational environment aspects to be fulfilled in order to assure conditions for safe operations.
- *Compliance*: To define the requirements that are required to be satisfied in order to comply with laws, regulation and standards, as a minimum the mandatory requirements related to assurance of safety. These requirements may involve requirements for e.g. applying some specific development process, perform certain activities, or make use of specific technique.

Process Solution: Figure 20 illustrates the *Establish System Safety Requirements* process annotated in a UML activity diagram.

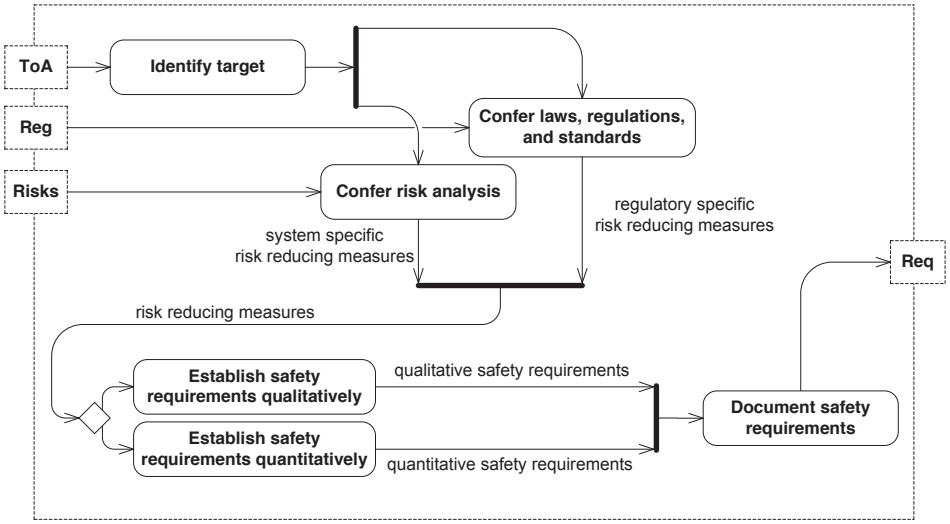


Figure 20 Establish System Safety Requirements – Process Flow

The input parameters associated with the activity diagram may be interpreted as follows:

- *ToA (Target of Assessment*): represents the target system for which safety requirements should be established.

- *Reg (Regulations*): represents any source of information describing mandatory or recommended practices (e.g. as provided in laws, regulations or standards) valuable for identifying risk reducing measures.
- *Risks*: represents risks associated with the target system.

The main activities serve the following purpose:

- *Identify target*: the intent of the activity is to identify *ToA*. The description of the target should as a minimum include a definition of the system and its boundaries, its operational profile, functional requirements, and safety integrity requirements.
- *Confer laws, regulations, and standards*: the intent of the activity is to capture all relevant data (data on requirements for risk reducing measures) from relevant sources (normative references) in order to outline the set of risk reducing measures that shall be met by compliance. Each source is inspected in order to identify, as a minimum, the mandatory risk reducing measures that shall be met in order to be compliant.
- *Confer risk analysis*: the intent of the activity is to capture all relevant data on risk analysis of the system that is under construction in order to outline the system specific risk reducing measure that shall be met.
- *Establish safety requirements qualitatively*: the intent of the activity is to define safety requirements on the basis of those identified risk-reducing measures required applied, and which may be demonstrated fulfilled qualitatively.
- *Establish safety requirements quantitatively*: the intent of the activity is to define safety requirements on the basis of those identified risk-reducing measures required applied, and which may be demonstrated fulfilled quantitatively.
- *Document safety requirements*: the intent of the activity is to detail all relevant information with respect to the specification of requirements in a system safety requirements specification. For each requirement defined in the requirement specification, information detailing what influenced its definition should be provided, e.g., the associated risks that are addressed, assumptions, calculations, and justifications.

Instantiation Rule: An artefact *Req* (see Figure 20 and Figure 19) is the result of a process that instantiates the *Establish System Safety Requirements* pattern if:

- *Req* is a set of requirements.
- *Req* is defined as a result of applying a process as illustrated in Figure 20 and described by the guidance provided in Section “Process Solution”. The process is initiated by an activity on describing the target *ToA*. Once a description of the target system and its operational context is provided, the next activities shall identify the risk reducing measures to be applied to the target by conferring relevant laws, regulations and standards as well as the result of target specific risk analysis for guidance. Once all relevant risk-reducing measures are identified, these shall be used as a basis to define the requirements to be met by the target system or by the process applied for developing the target. The requirements are defined quantitatively or qualitatively depending on the nature of the risk reducing measure that is addressed. The requirements are documented in a requirement specification *Req*.
- Every requirement of *Req* is traceable to the risks (identified by the instantiation of *Risks*), and/or regulatory requirements (identified by the instantiation of *Reg*) it addresses.
- Every requirement of *Req* is justified such that any assumptions, calculations, and assessments that support the specification of the requirement as a safety requirement are provided.

Related Patterns: The *Establish System Safety Requirements* pattern is related to other patterns in the following manner:

- May succeed the *Risk Analysis* pattern that supports identifying risks. The *Establish System Safety Requirements* may be successively applied as support for defining the requirements to be fulfilled in order to reduce risk to an acceptable risk level.
- May be used in order to detail requirements for the design provided upon instantiation of a design pattern.

8.1.2 Textual syntax

name(*Establish System Safety Requirements*)
procreqpatternsignature(
Establish System Safety Requirements
input(
documentationparameter(*ToA*)
documentationparameter(*Reg*)
documentationparameter(*Risks*)
)
output(**requirementparameter**(*Req*))
)
intent(*Support the specification of system safety requirements on the basis of a…*)
applicability(*The Establish System Safety Requirements pattern* is intended for…)
problem(*The main aspects relevant to establishing the safety requirements are:…*)
processolution(
umlactivitydiagram(
uml(*Establish System Safety Requirements – Process Flow*)
inputtoactivity(
inputparameter(*ToA*)
activity(*Identify target*)
)
inputtoactivity(
inputparameter(*Reg*)
activity(*Confer laws, regulations, and standards*)
)
inputtoactivity(
inputparameter(*Risks*)
activity(*Confer risk analysis*)
)
outputfromactivity(
outputparameter(*Req*)
activity(*Document safety requirements*)
)
)
The input parameters associated with the activity…
The main activities serve the following purpose…
)
instantiationrule(
requirementparameter(*Req*)
Establish System Safety Requirements
rule(*Req is a set of requirements*)
rule(*Req* is defined as a result of applying a process as illustrated in Figure 20 and…)
rule(*Every requirement of Req is traceable to…*)

rule(*Every requirement of Req is justified...*)

)

relatedpatterns(

Establish System Safety Requirements

The Establish System Safety Requirements pattern is related to…

)

8.1.3 Translation

Establish System Safety Requirements is a generalised solution to the challenges described in the following. *Establish System Safety Requirements* belongs to a category of patterns that addresses challenges appearing in a development context by defining its solution as a process and by defining requirements for documenting the results of the process.

The pattern requires the input:

- A description *ToA*.
- A description *Req*.
- A description *Risks*.

The pattern delivers the output:

- A set of requirements *Req*.

The intent of the pattern is to: *Support the specification of system safety requirements on the basis of a…*

The scenarios for which the pattern typically may be applied are as follows: *The Establish System Safety Requirements pattern is intended for…*

The main challenges addressed by the pattern are described in the following:

The main aspects relevant to establishing the safety requirements are:…

The process solution is defined by ⟦ **uml**(*Establish System Safety Requirements – Process Flow*) ⟧. In *Establish System Safety Requirements – Process Flow,* the following relationships between parameters of the pattern and activities are defined:

- The instantiation of input *ToA* is used for performing the activity *Identify target.*
- The instantiation of input *Reg* is used for performing the activity *Confer laws, regulations, and standards*.
- The instantiation of input *Risks* is used for performing the activity *Confer risk analysis*.
- Performing the activity *Document safety requirements* produces the output *Req*.

The input parameters associated with the activity…

The main activities serve the following purpose…

The pattern *Establish System Safety Requirements* is instantiated by generating the output *Req* in accordance with the following rules:

- *Req is a set of requirements.*
- *Req is defined as a result of applying a process as illustrated in* Figure 20 *and…*
- *Every requirement of Req is traceable to...*
- *Every requirement of Req is justified…*

The following describes patterns that are closely related to *Establish System Safety Requirements*:

- *The Establish System Safety Requirements pattern is related to…*

8.2 Example 2: Station Interlocking Requirements

8.2.1 Pattern definition

Name: Station Interlocking Requirements

Pattern Signature: *Station Interlocking Requirements* is defined with the signature illustrated in Figure 21.

In Figure 21, the following abbreviations are used for denoting the parameters of the pattern:

- *Mch* is short for Machine.
- *Req* is short for Requirements.

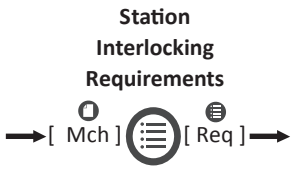


Figure 21 Station Interlocking – Pattern Signature

Intent: Support eliciting functional requirements *Req* for an interlock system *Mch* that shall control the appliances of a train station with two or more tracks at the station area. A typical example of a station with two tracks is given as an example in Figure 22.

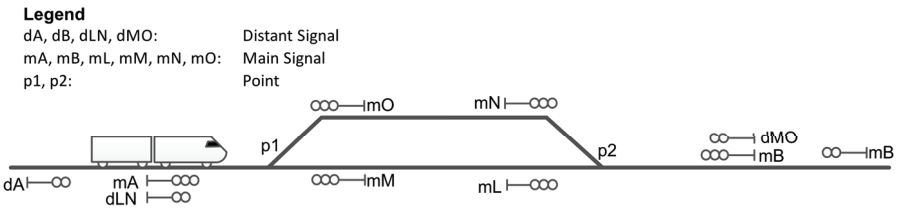


Figure 22 Example – Signal layout at a two track station

Figure 23 illustrates a typical sectioning of the railway tracks of a two-track station.

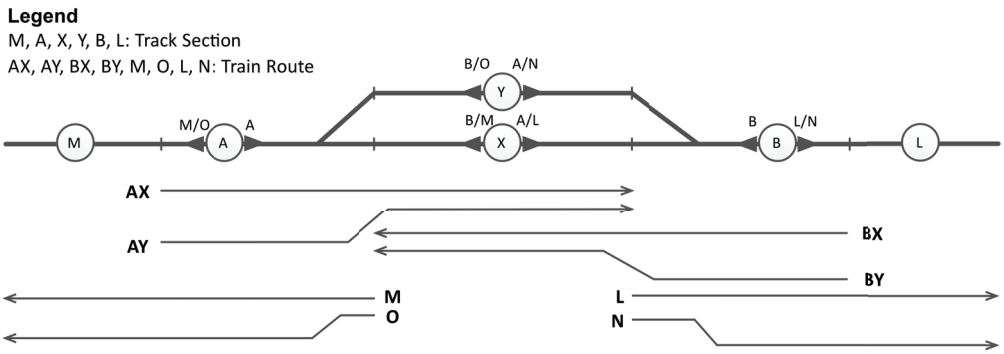


Figure 23 Example – Track sections and train routes at a two track station

Applicability: The *Station Interlocking Requirements* pattern is intended for the following situations:

- When building or upgrading an interlocking system that shall control the appliances of a station with two or more tracks in order to derive the functional requirements for the system.

Problem: The main aspects relevant for establishing functional requirements for the interlock system are:

- *States of appliances*: What are the possible states of the different appliances like distant signals, main signals and points.
- *System states*: What are the possible system states.
- *Transitions*: What are the expected transitions between system states.
- *Safety*: Is there some system states or transitions that shall never be allowed and that may be hazardous (e.g. simultaneous incoming or outgoing traffic or drive through).

Problem Frame Analysis Solution: Figure 24 illustrates the *Station Interlocking Requirements* problem frames diagram.

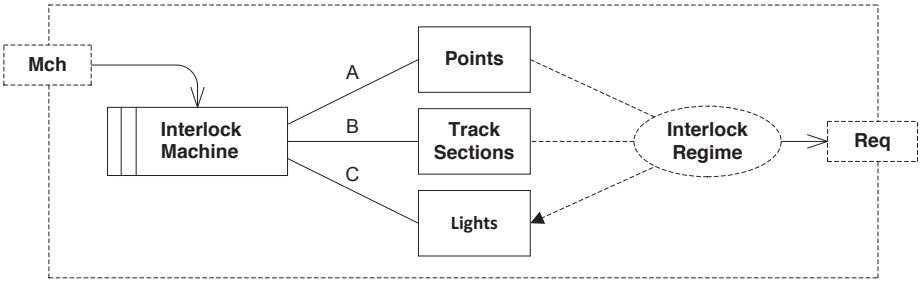


Figure 24 Station Interlocking Requirements – Problem Frame Diagram

The input parameters associated with the problem frame diagram shall be interpreted as:

- *Mch*: represents the machine to be constructed or an existing system that is modernised and that is responsible for controlling train movements in a safe manner.

The output expected by instantiation of the pattern is identified as:

- *Req*: represents the set of requirements derived on the basis instantiating the pattern according to the defined instantiation rule.

The problem domains that are represented in the problem frame diagram shall be interpreted as:

- *Points*: represents any point part of the track layout (e.g. the two points exemplified in Figure 22).
- *Track Sections*: represents the different sections of the tracks (e.g. as exemplified in Figure 23).
- *Lights*: represents the distant signals and the main signals (e.g. as exemplified in Figure 22).

The *Interlock Machine* shall assure safe allocation of train routes for trains. There are eight possible train routes in the configuration exemplified in Figure 24. A train route may be in two states, we name these states *Locked* (allocated to a train) and *Not Locked*. The *Interlock Machine* interacts with the appliances represented as problem domains in Figure 24 through the following interfaces in order to secure train routes:

- *A*: represents the interaction between the *Interlock Machine* and the *Points*. The points may be in two positions; we name these positions *Aligned* and *Diverging*.
- *B*: represents the interaction between the *Interlock Machine* and the *Track Sections*. A track section may be in two states, we name these states *Vacant* and *Occupied*.
- *C*: represents the interaction between the *Interlock Machine* and the *Lights*. We assume here that the lights may be in the following states depending on type of signal:

- *A distant signal may be in the states: Expect Proceed, Expect Proceed Slow, or Expect Stop.*
- *A main signal may be in the states: Proceed, Proceed Slow, or Stop.* An outbound main signal on a diverging track (e.g. mO or mN in Figure 22) may only be in the states *Proceed Slow and Stop* because the train when moving shall only proceed with a limited speed (thus the *Proceed* signal is not provided).

Requirements for the *Interlock Machine* may be elicited by instantiating the abstract requirements given in Table 1.

Table 1 Station Interlocking Requirements – Abstract Requirements

ID	Requirement	Note on instantiation
R.1	*Mch* shall manage train movements along the following train routes: *train routes*	Identify every train route that is required to be controlled (e.g. AX, AY,M,… as in Figure 23) and that is part of all *train routes*
R.2	*Mch* may set *train route* as locked when: *conditions*	For each *train route (e.g.* AX in Figure 23*)*: define each *condition* (e.g. A in state Not Locked, X in state Not Locled)*,* part of *conditions*, which must be satisfied in order to lock a train route (assign a train route to a train).
R.3	*Mch* shall when *train route* becomes locked signal: *signalling*	For each train route define the signalling to be performed once the train route is locked
R.4	*Mch* shall detect *train route* as commenced when: *conditions*	For each *train route* (e.g. AX in Figure 23*)*: define each *condition* (*e.g. A* goes to state *Occupied),* part of *conditions* which must be satisfied in order to detect train route as commenced
R.5	*Mch* shall when *train route* is detected as commenced signal: *signals*	For each train route (e.g. AX in Figure 23*):* define the signalling to be performed once the train route is detected as commenced (e.g. mA goes to state Stop)
R.6	*Mch* may only unlock *train route* when: *conditions*	For each *train route* (e.g. AX in Figure 23*)*: define each *condition* (e.g. *A* goes to state *Vacant*)*,* part of *conditions*, which must be satisfied in order to unlock a locked train route

Instantiation Rule: An artefact *Req* (see Figure 24 and Figure 21) is the result of instantiating the *Station Interlocking Requirements* pattern if:

- *Req* is a set of requirements.
- *Req* is the a result of an analysis, e.g., by the use the problem frames analysis approach as outlined in Figure 24 and the guidance provided in Section “Process Solution”, of how a system *Mch* shall operate in order to assure safe train movements at train station. In order to perform the analysis, the system *Mch* must be identified and described, relevant problem domains that interact with *Mch* or in other ways are important for the interlocking scenario addressed must be identified and their possible states defined. The objective of the analysis is to define an interlocking regime that provides the rules, in the form of requirements *Req* for *Mch*, that assures the required functionality is provided and such that safety is assured.
- Every requirement of *Req* is an instance of an abstract requirement. Abstract requirements are defined in column “Requirement” of Table 1.
- An abstract requirement is instantiated by applying the guidance provided in column “Notes on instantiation” described in Table 1.
- Every requirement of *Req* is traceable to a unique abstract requirement.

- Every requirement of *Req* describes a property, behaviour or a constraint of the system (identified by the instantiation of *Mch*).

8.2.2 Textual syntax

name(*Station Interlocking Requirements*)
prodreqpatternsignature(
Station Interlocking Requirements
input(**documentationparameter**(*Mch*))
output(**requirementparameter**(*Req*))
)
intent(*Support eliciting functional requirements for an interlock system that shall…*)
applicability(*The Station Interlocking pattern is intended for the following situations …*)
problem(*The main aspects relevant for establishing functional requirements…*)
problemframesolution(
problemframesdiagram(
pfd(*Station Interlocking Requirements – Problem Frame Diagram*)
inputtomachine(
inputparameter(*Mch*)
machine(*Interlock Machine*)
)
outputfromrequirement(
outputparameter(*Req*)
requirement(*Interlock Regime*)
)
)
The input parameters associated with the problem frame…
)
instantiationrule(
requirementparameter(*Req*)
Station Interlocking Requirements
rule(*Req is a set of requirements*)
rule(*Req is the a result of an analysis, e.g., by the use the problem frames analysis approach…*)
rule(*Every requirement of Req is an instance of an abstract requirement...*)
rule(*An abstract requirement is instantiated by applying the guidance…*)
rule(*Every requirement of Req is traceable to a unique abstract requirement*.)
rule(*Every requirement of Req describes a property, behaviour or a constraint*...)
)

8.2.3 Translation

Station Interlocking Requirements is a generalised solution to the challenges described in the following. *Station Interlocking Requirements* belongs to a category of patterns that defines a set of abstract requirements and a problem frames analysis solution as a means to address the challenges of eliciting requirements for a particular context.

The pattern requires the input:

- A description *Mch.*

The pattern delivers the output:

- A set of requirements *Req*.

The intent of the pattern is to: *Support eliciting functional requirements for an interlock system that shall…*

The scenarios for which the pattern typically may be applied are as follows: *The Station Interlocking Requirements pattern is intended for the following situations …*

The main challenges addressed by the pattern are described in the following: *The main aspects relevant for establishing functional requirements…*

The solution is described as a requirement analysis, defined by ⟦ **pfd**(*Station Interlocking Requirements – Problem Frame Diagram*) ⟧. In *Station Interlocking Requirements – Problem Frame Diagram,* the following relationships between parameters of the pattern and entities in the diagram are defined:

- The instantiation of input *Mch* is associated with the machine named *Interlock Machine*
- The instantiation of the requirement named *Interlock Regime* is associated with output *Req*

The input parameters associated with the problem frame…

The pattern *Station Interlocking* is instantiated by instantiating the output *Req* according to the following rules:

- *Req is a set of requirements.*
- *Req is the a result of an analysis, e.g., by the use the problem frames analysis approach…*
- *Every requirement of Req is an instance of an abstract requirement...*
- *An abstract requirement is instantiated by applying the guidance…*
- *Every requirement of Req is traceable to a unique abstract requirement.*
- *Every requirement of Req describes a property, behaviour or a constraint...*

8.3 Example 3: FTA

8.3.1 Pattern definition

Name: FTA (Fault Tree Analysis)

Pattern Signature: *FTA* is defined with the signature illustrated in Figure 25.

In Figure 25, the following abbreviations are used for denoting the parameters of the pattern:

- *ToA* is short for Target of Assessment.
- *UE* is short for Unwanted Event.
- *FT* is short for Fault Tree

Figure 25 Pattern Signature – FTA

Intent: Support a systematic and deductive assessment of the potential causes of identified unwanted events *UE* of a target *ToA*. The pattern describes the fault tree analysis method and provides guidance on its application. A fault tree *FT* is expressed as a tree where the root of the tree denotes the unwanted event *UE* (e.g. representing a system hazard) that is analysed and leaf nodes denotes basic events (e.g. system component failure modes) that may lead to the unwanted event. The primary goal is to identify minimal cut sets by the use of the tree structure. A cut set expresses a set of basic events such that if these are present at the same time, the unwanted event will occur. A minimal cut set expresses the minimal set of basic events that may lead to an unwanted event.

Applicability: The *FTA* pattern is intended for the following situations:

- As a means to deduce potential causes of unwanted events.
- As a means to identify combinations of faults that may lead to unwanted events.
- As a means to analyse system concepts, simple systems or complex systems.

Problem: The main challenges associated with methods for assessment of the potential causes of an unwanted event are:

- *Efficiency*: To support efficient application with minimal use of resources.
- *Logical*: To support a logical specification of the dependencies between an unwanted event that is investigated and the potential causes that is identified through the assessment.
- *Communication*: To support communication of results on a form that allows different actors to understand findings with minimal effort.

Method Solution: Figure 26 exemplifies a fault tree and outlines the relationships between a fault tree and the inputs and outputs of the FTA pattern.

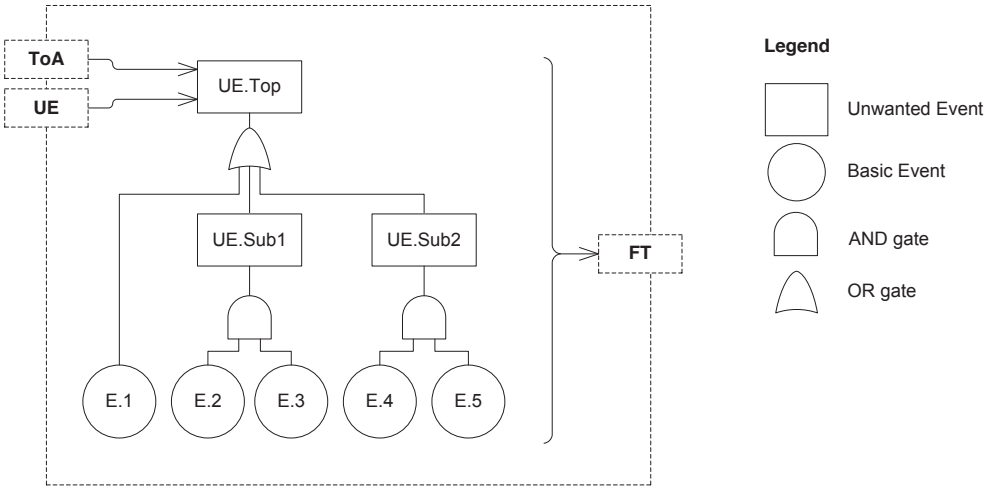


Figure 26 FTA example diagram

The following steps may apply the FTA method systematically:

- Step 1 – Define the top event: The intent is to define the top event that represent the root of a fault tree and the event that is analysed (UE.Top in Figure 26).
- Step 2 – Construct fault tree: The intent is to construct the fault tree by deductively constructing the tree in a top-down manner by the use of graphical elements symbolising events and gates. An unwanted event should state what is the fault/failure under consideration. Gates (e.g. the or gate in Figure 26) are used to define the combination of antecedents (e.g. basic event E.1, UE.Sub1, or UE.Sub2) that imply the consequent (e.g. UE.Top). Decomposition of an unwanted event (e.g. UE.Top) should not be described by connecting a series of gates down to basic events, intermediate unwanted events should be defined (e.g. UE.Sub1, and UE.Sub2). Basic events answer the how questions by defining how a system component or element may experience a fault/failure.
- Step 3 – Find minimal cut sets: The intent is to deduce from the tree structure the combination of basic events that may lead to the top event and arrange these in order. The set with lowest order

(contains least number of basic events) is most significant. In Figure 26, the event UE.Top occurs if event E.1 occurs (minimal cut set of order 1). The top event also occurs if events E.2 and E.3 occur or if events E.4 and E.5 occur (cut sets of order 2). There are many cut sets of order 3, 4, and 5 but these are less significant.

- Step 4 – Qualitative Analysis: The intent is to analyse the fault tree qualitatively in order identify week points of the system with respect to the top event. This may be performed on the basis of cut sets. If there exists cut-sets of order 1 then the system is vulnerable to single point of failure. If there exists none cut sets of order 1 but the events in the cut sets of order 2 have identical characteristics, then the system may be susceptible to common cause failures.
- Step 5 – Quantitative Analysis: The intent is to calculate the probability of a top event by combining probabilities of basic events (given that probabilities of basic events may be provided). The probability of UE.Top in Figure 26 may be calculated by a combination of the following general rules:
  - The probability of an event A ''and'' an event B is expressed: P(A*B) = P(A) * P(B)
  - The probability of an event A ''or'' an event B is expressed: P(A+B) = P(A) + P(B) – (P(A) * P(B))

Instantiation Rule: A documentation artefact *FT* (see Figure 26 and Figure 25) instantiates the *FTA* pattern if:

- *FT* is a set of fault trees.
- Every fault tree of *FT* is instantiated by applying the guidance provided in Section "Method Solution" for all unwanted events (identified by the instantiation of *UE*). For a specific target *ToA*, this means that every unwanted event *UE* that represents a suitable top event in the analysis is systemically assessed according to the process outlined through Steps 1 to Step 5.
- Every fault tree of *FT* is traceable to a unique unwanted event (identified by the instantiation of *UE*).
- Every fault tree of *FT* describes the relation between an unwanted event, represented as a top node in the tree structure, and potential initiating events (e.g. HW/SW failures) associated with a target system (identified by the instantiation of *ToA*).

Related Patterns: The *FTA* pattern is related to other patterns in the following manner:

- May support *Hazard Analysis* pattern by providing a method for performing identification of potential causes of hazards.
- May support *Risk Analysis* pattern by providing a method for performing qualitative and quantitative analysis of hazards.

Known Uses: Fault tree analysis is a commonly applied method within safety critical domains and is described in several standards and guidelines, e.g. nuclear domain [NUREG–0492], aerospace [ARP4761], and the more general applicable standard [IEC61025].

[NUREG–0492] NRC, Fault Tree Handbook, U.S. Nuclear Regulatory Commission (1981)

[ARP4761] SAE, Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment, Society of Automotive Engineers (1996)

[IEC61025] IEC, Fault Tree Analysis (FTA), Edition 2.0, International Electrotechnical Commission (2006)

8.3.2 Textual syntax

name(*FTA*)
procsolpatternsignature(

FTA
input(
documentationparameter(*ToA*)
documentationparameter(*UE*)
)
output(**documentationparameter**(*FT*))
)
intent(*Support a systematic and deductive assessment of the potential causes…*)
applicability(The *FTA* pattern is intended for the following situations…)
problem(*The main challenges associated with methods for assessment of…*)
methodsolution(
arbitrarydiagram(
illustration(*FTA example diagram*)
inputtoentity(
inputparameter(*ToA*)
entity(*UE.Top*)
)
inputtoentity(
inputparameter(*UE*)
entity(*UE.Top*)
)
outputfromentity(
outputparameter(*FT*)
)
)
The following steps may apply the FTA method systematically…
)
instantiationrule(
documentationparameter(*FT*)
FTA
rule(*FT is a set of fault trees.*)
rule(*Every fault tree of FT is instantiated by applying the guidance...*)
rule(*Every fault tree of FT is traceable to a unique…*)
rule(*Every fault tree of FT describes the relation between…*)
)
relatedpatterns(
FTA
The FTA pattern is related to…
)
knownuses(*Fault tree analysis is a commonly applied method within…*)

8.3.3 Translation

FTA is a generalised solution to the challenges described in the following. *FTA* belongs to a category of patterns that defines solutions to recurring development challenges in the form of a method.

The pattern requires the input:

- A description *ToA*.
- A description *UE.*

The pattern delivers the output:

- A description *FT*.

The intent of the pattern is to: *Support a systematic and deductive assessment of the potential causes …*

The scenarios for which the pattern typically may be applied are as follows: *The FTA pattern is intended for the following situations…*

The main challenges addressed by the pattern are described in the following: *The main challenges associated with methods for assessment of…*

The method solution is exemplified by ⟦ *FTA example diagram* ⟧. In *FTA example diagram*, the following relationship between parameters of the pattern and the elements in the illustration are defined:

- The instantiation of input *ToA* is associated with *UE.Top*
- The instantiation of input *UE* is associated with *UE.Top*
- Output *FT* represents an instantiation result.

The following steps may apply the FTA method systematically…

The pattern *FTA is* instantiated by instantiating the output *FT* according to the following rules:

- *FT is a set of fault trees*
- *Every fault tree of FT is instantiated by applying the guidance…*
- *Every fault tree of FT is traceable to a unique…*
- *Every fault tree of FT describes the relation between…*

The following describes patterns that are closely related to *name*: *The FTA pattern is related to…*

The following describes known uses of the pattern: *Fault tree analysis is a commonly applied method within…*

8.4 Example 4: Dual Modular Redundant

8.4.1 Pattern definition

Name: Dual Modular Redundant

Pattern Signature: The *Dual Modular Redundant* is defined with the signature illustrated in Figure 27.

In Figure 27, the following abbreviations are used for denoting the parameters of the pattern:

- *Cmd* is short for Command.
- *C1* is short for Controller 1.
- *C2* is short for Controller 2.
- *V* is short for Voter.
- *S* is short for System.

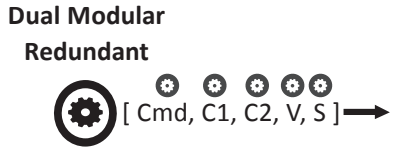


Figure 27 Dual Modular Redundant – Pattern Signature

Intent: Describe the design of a system *S* that offers a simple and cost effective protection against random hardware failure by the use of two redundant controllers, *C1* and *C2*, which operate in parallel. Command signals to the two controllers and status indications from these are handled by a component *Cmd* that also provides the interface between the components performing control operations and the overall system. A voter *V* is used to implement functionality for reacting upon discrepancies between the parallel operating controllers.

Applicability: The *Dual Modular Redundant* pattern is intended for the following situations:

- When the reliability of the individual controllers in the dual controller setup and supporting system parts is high enough, and there is satisfactory protection against common cause failure, to provide within bound system reliability and a safe application.
- When the safety management and quality management are performed at a level protecting against systematic errors at a satisfactory level.

Problem: The main aspects to be considered when providing protection against failure are:

- *Random hardware failure*: hardware is susceptible to wear and tear and failure may manifest at random, protection should be incorporated against random hardware failure.
- *Separation of functions*: it must be ensured that the potential failure of non-critical functions do not negatively affect critical functions thus there should be a clear separation between functional parts of different safety integrity.
- *Reliability of components*: it must be ensured that reliability of individual components is satisfactory in order to meet reliability targets.
- *Common cause failure*: it must be ensured that the system is adequately protected against common cause failure threats.
- *Fail-safe behaviour*: it should be ensured that any potential failure of the system should not negatively affect safety.

Design Solution: Figure 28 illustrates the main structure of components and their interfaces, annotated in UML notation, which play a part in obtaining a *Dual Modular Redundant* design.

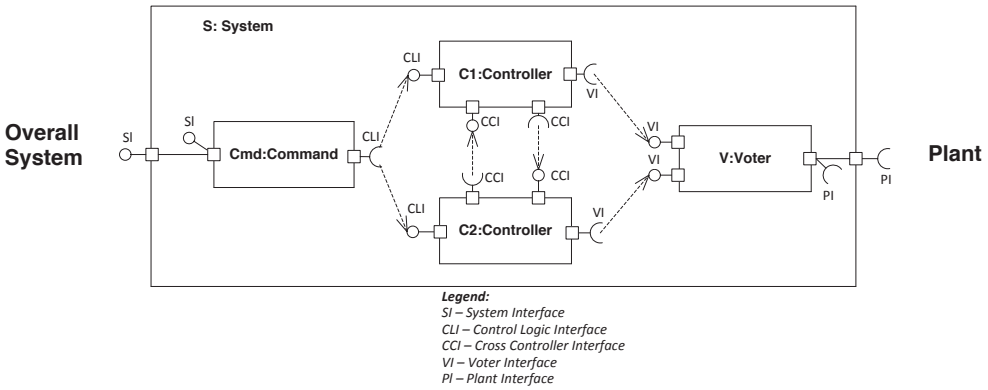


Figure 28 Dual Modular Redundant Design

The component *Cmd* is responsible for communicating with an overall system and the parts performing critical operations, *C1* and *C2*. *C1* and *C2* are two identical systems that operate in parallel and perform critical operations (e.g. interlocking operations). A voter interacts with the plant/EUC (Equipment Under Control) and carry out the control operations from *C1* and *C2*. The voter will implement mechanisms for 1-out-of-2 or 2-out-of-2 voting functionality.

An important feature with respect to utilising the design for safety critical systems is the physical and functional separation of non-critical functionality (resided in *Cmd*) from the critical functionality (resided in *C1* and *C2*). The redundant controllers provide protection against random hardware failure that may affect control operations and the voter assures that no potential erroneous signalling due to random hardware failure is performed. Protection against systematic failure is handled by adequate development processes (e.g., safety management activities and quality management activities) and is not addressed here.

The roles and responsibilities represented in the pattern are:

- *S: System* – represents the system defined by the *Dual Modular Redundant*. The system (e.g. a railway interlocking system) interacts with an overall system or human operator (e.g. a train leader through a centralised train control system) and the plant that is controlled (e.g. distant light signals, main light signals, and track switches).
- *C1: Controller* – responsible for providing control signals (e.g. interlocking control signals) based on plant states and commands from the overall system or some operator.
- *C2: Controller* – responsible for providing control signals (e.g. interlocking control signals) based on plant states and commands from the overall system or some operator.
- *Cmd: Command* – responsible for the interaction with an overall system such that commands (e.g. from an operator) may be communicated to the control system. Cmd may also be responsible for secondary functions such as data logging.
- *V: Voter* – responsible for activating plant control in accordance with the commands given by the dual controllers and take proper safe actions in the case of disagreement (that is an indication of a component failure) between the controllers.

Instantiation Rule: An artefact *S* (see Figure 28 and Figure 27) instantiates the *Dual Modular Redundant* pattern if:

- *S* is a specification of the hardware design and/or the software design of a control system.
- *S* specifies a system containing at least two redundant controller parts that operate in parallel and that interface with a voting mechanism.
- *S* specifies a system part that provides an interface with an overall system and the dual controllers such that orders may be given to, and indications obtained from, the controllers.
- *S* specifies a system part that is responsible for providing a voting mechanism such that protection against random failure is detected and may be mitigated.

Related Patterns: The pattern relates to the other patterns in the following manner:

- Any product requirement pattern that might be used to derive the functional requirements for the system.
- Any process requirement pattern that might be used to assess and derive safety functional requirements for the system.
- Any safety case pattern that might be used as a means to argue that the system is adequately safe for its purpose.

Known Uses: In Norway, a commonly used interlocking system type known as NSB-94 uses a dual modular redundant design as described here. Critical functions, such as the interlocking functionality, are implemented in terms of two identical PLCs (Programmable Logic Controllers) that operate in parallel. A voting mechanism ensures that no random hardware failure may lead to potentially harmful signalling, e.g., no train receives a go (green light signal) if there is any disagreement between the controllers (2-out-of-2 voting to give a go). Given that one controller commands a stop signal (red light signal) and the other controller commands a go signal for a given light signal then a stop is signalled (1-out-of-2 voting to give a stop).

8.4.2 Textual syntax

name(*Dual Modular Redundant*)
prodsolpatternsignature(
Dual Modular Redundant
output(
designparameter(*Cmd*)
designparameter(*C1*)
designparameter(*C2*)
designparameter(*V*)
designparameter(*S*)
)
)
intent(*Describe the design of a system S that offers a simple and cost effective…*)
applicability(*The Dual Modular Redundant pattern is intended for the following situations…*)
problem(*The main aspects to be considered when providing protection against failure are…*)
designsolution(
umldiagram(**uml**(*Dual Modular Redundant Design*))
The component Cmd is responsible for communicating with an overall system…
)
instantiationrule(
designparameter(*S*)
Dual Modular Redundant
rule(*S is a specification of the hardware design and/or the software design of…*)
rule(*S specifies a system containing at least two redundant controller parts that …..*)
rule(*S specifies a system part that provides an interface with...*)
rule(*S specifies a system part that is responsible for providing a...*)
)
relatedpatterns(
Dual Modular Redundant
The pattern relates to the other patterns in the following manner…
)
knownuses(*In Norway, a commonly used interlocking system type known as…*)

8.4.3 Translation

Dual Modular Redundant is a generalised solution to the challenges described in the following. *Dual Modular Redundant* belongs to a category of patterns that defines a design solution as a means to handle a commonly occurring development challenge. The pattern delivers the output*:*

- A specification of the design of *Cmd*
- A specification of the design of *C1*
- A specification of the design of *C2*
- A specification of the design of *V*
- A specification of the design of *S*

The intent of the pattern is to: *Describe the design of a system S that offers a simple and cost effective…*

The scenarios for which the pattern typically may be applied are as follows: *The Dual Modular Redundant pattern is intended for the following situations…*

The main challenges addressed by the pattern are described in the following: *The main aspects to be considered when providing protection against failure are…*

The solution is defined by ⟦ *Dual Modular Redundant Design* ⟧.

The component Cmd is responsible for communicating with an overall system and…

The pattern *Dual Modular Redundant* is instantiated by instantiating the output *S* according to the following rules:

- *S is a specification of the hardware design and/or the software design of…*
- *S specifies a system containing at least two redundant controller parts that...*
- *S specifies a system part that provides an interface with...*
- *S specifies a system part that is responsible for providing a...*

The following describes patterns that are closely related to *Dual Modular Redundant*: *The pattern relates to the other patterns in the following manner…*

The following describes known uses of the pattern: *In Norway, a commonly used interlocking system type known as…*

8.5 Example 5: Overall Safety

8.5.1 Pattern definition

Name: Overall Safety

Pattern Signature: *Overall Safety* is defined with the signature illustrated in Figure 29.

In Figure 29, the following abbreviations are used for denoting the parameters of the pattern:

- *ToD* is short for Target of Demonstration.
- *QualMng* is short for Quality Management.
- *SafMng* is short for Safety Management.
- *TechSaf* is short for Technical Safety.
- *Case* short for Safety Case

Figure 29 Overall Safety – Pattern Signature

Intent: Provide a structure for the specification of an overall strategy (or plan) for the safety demonstration *Case* of a target system *ToD*. The overall strategy is a means to document the set of practices that jointly provide confidence in the procurement of a safe system. This entails addressing managerial and technical aspects relevant with respect to the procurement of the system under constructions such that an effective top-down deduction of the safety demonstration may be provided. The focus of the pattern is on how to combine strategies on different safety concerns into an overall strategy such that it may be claimed and proven with confidence that the system is sufficiently safe.

Applicability: The *Overall Safety* pattern is intended for the following situations:

- At the initial stages of development in order to outline the main set of demonstration strategies and how these may be combined in order to demonstrate system safety.

- Prior to project initiation as a means for stakeholders (vendor, customer, licensee) to reflect upon main demonstration challenges and their possible solution.
- As a means to specify the top-level strategies in a safety case demonstration.

Problem: The main aspects to be considered when providing a solution for the demonstration of safety are:

- *Quality management*: it is necessary to demonstrate that the quality of the system is, and shall continue to be, controlled by an effective quality management system. Quality management is a means to minimise the occurrence of human errors at each stage in the development life cycle, and thus to reduce the risk of systematic faults in the system.
- *Safety management*: it is necessary to demonstrate that the safety management is controlled by, and shall continue to be, controlled by and effective means. Safety management is a means to reduce the occurrence of safety-related human errors throughout the life cycle, and thus minimise the residual risk of safety-related systematic faults.
- *Technical safety*: it is necessary to demonstrate the safety of the technical design.
- *Strategy identification*: in order to plan and effectively apply a suitable demonstration strategy at the different system/subsystem, it is important to identify as early as possible the main lines of demonstration. This will reduce the cost related to performing safety assessment by avoiding those activities that does not support an effective safety demonstration. In addition, if the choice of demonstration strategies does not provide the necessary confidence, then there is a risk of not getting safety approval. The effect of not getting a safety approval may induce a great cost related to e.g. performing additional work in order to get approval, or loss of returns if the project is stopped.
- *Effectiveness*: it is important that the demonstration strategy is effective in providing the necessary safety demonstration. To develop safety related and safety critical systems is costly, mainly due to the activities required to assure and demonstrate that the system is sufficiently safe. It is important for a vendor to choose strategies that minimise cost but provide required results.
- *Confidence*: it is important to choose a demonstration strategy that has the potential to provide the necessary confidence for the procurement of a safe system. At the end, an approval body decides upon acceptance or not. In a development project, the effectiveness of different demonstration strategies must be weighed against the estimated cost and the ability to fulfil the primary goal, to provide the necessary confidence.
- *Acceptance*: it is important that the choice of demonstration strategies are acceptable for all involved parties such that they represent an effective means for demonstrating safety and may be applied in a manner that provides confidence that the system is safe.

Argument Structure Solution: Figure 30 and Table 2 jointly represent the demonstration solution and should be read together. Figure 30 illustrates the decomposition of the safety argument in a tree structure annotated in GSN notation. Table 2 details the tree structure by defining node types and the expected content of the nodes.

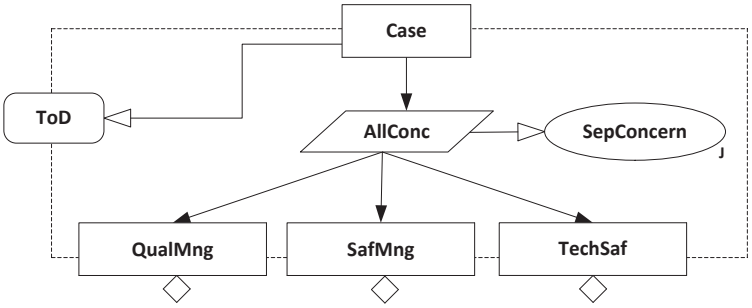


Figure 30 Overall Safety – Argument Structure

Associated with the contextual elements of the argument structure, the following is assumed:

- *ToD*: identifies the target system under consideration;

Table 2 Overall Safety – Argument Structure Details

Node	Node Type	Node Content Description
Case	Goal	*ToD* is safe
ToD	Context	Definition of parameter *ToD* (Target of Demonstration)
AllConc	Strategy	*ToD* is demonstrated safe by demonstrating that QM (Quality Management), SM (Safety Management), and TS (Technical Safety) is sufficiently addressed
SepConcern	Justification	Justification for the appropriateness of the separation of the demonstration concerning QM, SM, and TS
QualMng	Goal	*ToD* is safe with respect to proper QM (Quality Management)
SafMng	Goal	*ToD* is safe with respect to the SM (Safety Management) concern
TechSaf	Goal	*ToD* is safe with respect to the TS (Technical Safety) concern

Instantiation Rule: An artefact *Case* (see Figure 30 and Figure 29) instantiates the *Overall Safety* pattern if:

- *Case* represents the conclusion in a top-down decomposable safety argumentation on the safety of a target system (the target is identified by the instantiation of *ToD*).
- *Case* represents the root node in a tree like presentation of the safety argument as indicated in Figure 30**.**
- Every element of the decomposable safety argument where *Case* is the root node, is traceable to a unique abstract safety case element as indicated with GSN notation in Figure 30. An abstract safety case element is instantiated by adapting the descriptions in column “Node Content Description” in Table 2 to the context that is addressed in order to define a structure as given in Figure 30.
- *Case* expresses the decomposition of a main claim, that a target system is sufficiently safe for its intended purpose by a strategy of claiming sufficient quality management (the claim expressed in the node *QualMng*), safety management (the claim expressed in the node *SafMng*), and technical safety (the claim expressed in the node *TechSaf*). Note: No guidance is given in this pattern on how to decompose the safety argument parts represented by the claims *QualMng*,

SafMng and *TechSaf* down to its supporting evidences, suitable patterns supporting such a decomposition should be conferred.

Related Patterns: The *Overall Safety* pattern is related to other patterns in the following manner:

- May be supported by other safety case patterns for detailing those parts that are not fully developed, e.g. the pattern *Technical Safety* may be used to detail a demonstration on the issue of demonstrating that a system is technical safe and thus detail further the decomposition of the node *TechSaf* presented in Figure 30.
- May be used in order to establish a safety demonstration for a design derived from a design pattern being sufficiently safe for its intended purpose

8.5.2 Textual syntax

name(*Overall Safety*)
procrsafcasepatternsignature(
Overall Safety
input(**documentationparameter**(*ToD*))
output(
safetycaseparameter(*QualMng*)
safetycaseparameter(*SafMng*)
safetycaseparameter(*TechSaf*)
safetycaseparameter(*Case*)
)
)
intent(*Provide a structure for the specification of an overall strategy…*)
applicability(*The Overall Safety pattern is intended for the following situations…*)
problem(*The main aspects to be considered when providing a solution for the demonstration of…*)
argstructuresolution(
gsndiagram(
gsn(*Overall Safety – Argument Structure*)
goal(*Case*)
context(*ToD*)
undevelopedgoal(*QualMng*)
undevelopedgoal(*SafMng*)
undevelopedgoal(*TechSaf*)
)
Associated with the contextual elements of the argument structure, the following is assumed…
)
instantiationrule(
requirementparameter(*Case*)
Overall Safety
rule(*Case represents the conclusion in a top-down decomposable safety argumentation...*)
rule(*Case represents the root node in a tree like presentation of the safety...*)
rule(*Every element of the decomposable safety argument where Case is the root note...*)
rule(*Case expresses the decomposition of a main claim, that a target system is…*)
)
relatedpatterns(
Overall Safety
The Overall Safety pattern is related to…
)

8.5.3 Translation

Overall Safety is a generalised solution to the challenges described in the following. *Overall Safety* belongs to a category of pattern that addresses challenges of demonstrating that safety objectives are met and defines its solution as a structure of arguments.

The pattern requires the input:

- A description *ToD.*

The pattern delivers the output:

- A safety case element *QualMng*
- A safety case element *SafeMng*
- A safety case element *TechSaf*
- A safety case element *Case*.

The intent of the pattern is to: *Provide a structure for the specification of an overall strategy…*

The scenarios for which the pattern typically may be applied are as follows: *The Overall Safety pattern is intended for the following situations…*

The main challenges addressed by the pattern are described in the following: *The main aspects to be considered when providing a solution for the demonstration of…*

The safety case solution is defined by ⟦ *Overall Safety – Argument Structure* ⟧. In the diagram *Overall Safety – Argument Structure,* the following elements represent the parameters of the pattern:

- *Case* represents an output of the pattern once instantiated.
- *ToD* represents an input that describes the context of the associated element in the diagram.
- *QualMng* represents an output that needs to be instantiated and further developed.
- *SafMng* represents an output that needs to be instantiated and further developed.
- *TechSaf* represents an output that needs to be instantiated and further developed.

Associated with the contextual elements of the argument structure, the following is assumed…

The pattern *Overall Safety* is instantiated by instantiating the output *Case* according to the following rules:

- *Case represents the conclusion in a top-down decomposable safety argumentation...*
- *Case represents the root node in a tree like presentation of the safety...*
- *Every element of the decomposable safety argument where Case is the root note...*
- *Case expresses the decomposition of a main claim, that a target system is…*

The following describes patterns that are closely related to *Overall Safety***:** *The Overall Safety pattern is related to…*

8.6 Example 6: Safety Requirements Satisfied

8.6.1 Pattern definition

Name: Safety Requirements Satisfied

Pattern Signature: *Safety Requirements Satisfied* is defined with the signature illustrated in Figure 31.

In Figure 31, the following abbreviations are used for denoting the parameters of the pattern:

- *ToD* is short for Target of Demonstration.
- *Req* is short for Requirements.
- *ReqSat* is short for Requirements Satisfied.
- *Case is short for Safety Case.*

Figure 31 Safety Requirements Satisfied - Pattern Signature

Intent: Provide a structure for the specification of a safety demonstration *Case* showing that a system *ToD* is safe by arguing that all safety requirements *Req* are satisfied.

Example #1: a system S shall satisfy the requirements identified as R1, R2, SR1 and SR2 where SR1 and SR2 are safety requirements and R1 and R2 are functional requirements. In order to demonstrate that S is sufficiently safe for its intended purpose it is enough to demonstrate that SR1 and SR2 are satisfied. In order for system S to provide the intended function then R1 and R2 must be satisfied but as these requirements do no impact on safety they are not addressed in the safety case.

Applicability: The *Safety Requirements Satisfied* pattern is intended for the following situations:

- When it is required to provide an explicit demonstration of the ability of a system to uphold safety invariants expressed as safety requirements.

Problem: The main aspects to be considered when providing a solution for the demonstration of safety are:

- *Completeness of specification*: it is necessary to provide confidence that the set of safety requirements is complete or contain all relevant requirements.
- *Correct demonstration approach*: it is necessary to provide confidence that for each requirement, the chosen approach for demonstration is suitable.
- *Confirming evidences*: it is necessary to provide confidence that for each chosen demonstration approach, confirming evidences for a safety claim is available or may be provided.

Argument Structure Solution: Figure 32 and Table 3 jointly represent the demonstration solution and should be read together. Figure 32 illustrates the decomposition of the safety argument in a tree structure annotated in GSN notation. Table 3 details the tree structure by defining node types and the expected content of the nodes.

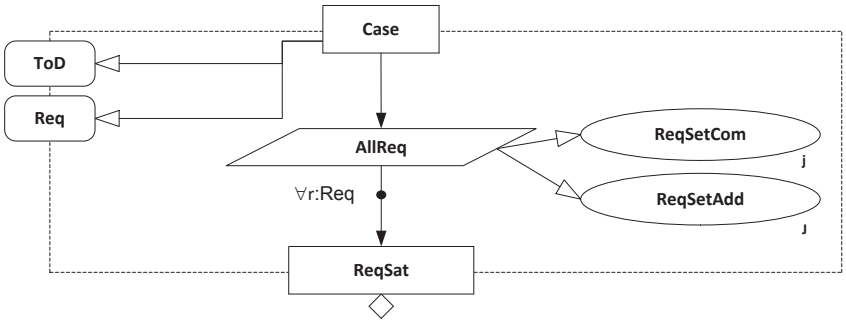


Figure 32 Safety Requirements Satisfied – Argument Structure

Associated with the contextual elements of the argument structure, the following is assumed:

- *ToD*: identifies the target system under consideration.
- *Req*: represents a set of safety requirements associated with the target system *ToD*.

Table 3 Safety Requirements Satisfied - Argument Structure Details

Node	Node Type	Node Content Description
Case	Goal	*ToD* satisfies *Req*
ToD	Context	Definition of *ToD*
Req	Context	Definition or *Req*
AllReq	Strategy	*ToD* is acceptably safe as all safety requirements are addressed and found satisfied
ReqSetCom	Justification	Justification for the requirements set *Req* being complete or contain all safety requirements that are relevant for *ToD*
ReqSetAdd	Justification	Justification for all elements of the set *Req* are accounted for. Elements may be addressed individually or in groups, any grouping of requirements needs to be justified.
ReqSat	Goal	The requirement *r,* element of *Req*, is satisfied

Instantiation Rule: A demonstration artefact *Case* (see Figure 32 and Figure 31) instantiates the *Safety Requirements Satisfied* pattern if:

- *Case* is a safety demonstration of a target system *ToD*.
- Every element of *Case* is traceable to a unique abstract safety case element as indicated with GSN notation in Figure 32. An abstract safety case element is instantiated by applying the descriptions in column “Node Content Description” in Table 3 in order to define a structure as given in Figure 32.
- *Case* expresses the decomposition of a main claim, that a target system *ToD* is safe, by a strategy of demonstrating that all safety requirements *Req* associated with the system *ToD* are satisfied.

Related Patterns: The pattern is related to other patterns in the following manner:

- May be used for detailing demonstration concerns addressed in the *Technical Safety* pattern on demonstrating that risk is explicitly addressed by demonstrating that the safety requirements, defined on the basis of risk assessment, are satisfied.
- May be used together with the *Establish System Safety Requirements* pattern that may be applied in order to derive safety requirements, *Safety Requirements Satisfied* is then used to develop a demonstration that argues that identified requirements are satisfied.

8.6.2 Textual syntax

name(*Safety Requirements Satisfied*)
prodsafcasepatternsignature(
Safety Requirements Satisfied
input(
documentationparameter(*ToD*)
requirementparameter(*Req*)

)
output(
safetycaseparameter(*ReqSat*)
safetycaseparameter(*Case*)
)
)
intent(*Provide a structure for the specification of a safety demonstration Case showing…*)
applicability(*The Safety Requirements Satisfied pattern is intended for the following situations…*)
problem(*The main aspects to be considered when providing a solution for the demonstration of …*)
argstructuresolution(
gsndiagram(
gsn(*Overall Safety – Argument Structure*)
goal(*Case*)
context(*ToD*)
context(*Req*)
undevelopedgoal(*ReqSat*)
)
Associated with the contextual elements of the argument structure, the following is assumed…
)
instantiationrule(
requirementparameter(*Case*)
Safety Requirements Satisfied
rule(*Case is a safety demonstration of a target system ToD.*)
rule(*Every element of Case is traceable to a unique abstract safety case element...*)
rule(*Case expresses the decomposition of a main claim…*)
)
relatedpatterns(
Safety Requirements Satisfied
The pattern is related other patterns in the following manner…
)

8.6.3 Translation

Safety Requirements Satisfied is a generalised solution to the challenges described in the following. *Safety Requirements Satisfied* belongs to a category of patterns that addresses challenges of demonstrating that safety objectives are met and defines its solution as a structure of arguments.

The pattern requires the input:

- A description *ToD*.
- A set of requirements *Req*.

The pattern delivers the output:

- A safety case element *ReqSat*
- A safety case element *Case*.

The intent of the pattern is to: *Provide a structure for the specification of a safety demonstration Case showing…*

The scenarios for which the pattern typically may be applied are as follows: *The Safety Requirements Satisfied pattern is intended for the following situations…*

The main challenges addressed by the pattern are described in the following: *The main aspects to be considered when providing a solution for the demonstration of…*

The safety case solution is defined by ⟦ **gsn**(*Overall Safety – Argument Structure*) ⟧. In *Overall Safety – Argument Structure,* the following elements represent the parameters of the pattern:

- *Case* represents an output of the pattern once instantiated.
- *ToD* represents an input that describes the context of the associated element in the diagram.
- *Req* represents an input that describes the context of the associated element in the diagram.
- *ReqSat* represents an output that needs to be instantiated and further developed.

Associated with the contextual elements of the argument structure, the following is assumed…

The pattern *Safety Requirements Satisfied* is instantiated by instantiating the output *Case* according to the following rules:

- *Case is a safety demonstration of a target system ToD*.
- *Every element of Case is traceable to a unique abstract safety case element...*
- *Case expresses the decomposition of a main claim…*

The following describes patterns that are closely related to *Safety Requirements Satisfied*: *The pattern is related to other patterns in the following manner…*

9. EXAMPLE TRANSLATIONS OF COMPOSITE PATTERNS

In this section we exemplify the translation of a chosen set of composite patterns into a text in English that provide a semantic for the patterns. Each composite pattern and associated translation is presented in separate sections where individual subsections provide a pattern definition, the representation of the pattern in its textual syntax, and finally the result of translating the textual syntax into English.

The composite patterns used in this section for exemplifying the translation of patterns into their semantics are chosen in the following manner:

- Example 1, described in Section 9.1 exemplifies the translation of a small composite pattern consisting of a combination of the basic patterns *Establish System Safety Requirements* presented in Section 8.1 and *Dual Modular Redundant* presented in Section 8.4. The composite contains all the main elements of the SaCS pattern language with no simplification of the visual presentation, e.g., the icons used for denoting the type of parameters is not omitted in order to simplify the visualisation although this may be done.
- Example 2, 3, 4, 5, and 6 described in Section 9.2, 9.3, 9.4, 9.5, and 9.6, respectively, represents the composite patterns that were applied for addressing the development challenges in the case reported in HWR-1037 [5]. As the patterns are taken from a case, we find them being representative of what may be expected in terms of complexity and expressiveness from the application of SaCS. The patterns in the different examples represent a modularly defined composition of patterns that is bound together in the composite presented in Example 6.
- Example 7, described in Section 9.7, presents a composite pattern where the relationships defined between the patterns within the composites presented in Example 2, 3, 4, 5, and 6 are contained within one pattern definition instead of a modular one. The identifiers used for the artefacts presented in Example 7 differ from those in Example 2, 3, 4, 5, and 6, as it is a constructed example.
- Example 1 to Example 7 are selected as every pattern type, parameter type, artefact type, relation, and annotations that are available in the SaCS pattern language are covered. The

examples also cover the simplifications that can be made to the visual presentation of a patterns as well as the possibility to modularise a pattern specification.

In the presentation of the textual syntax and the translation results we have slightly formatted the text to increase readability. Text in *italics* is used for identifiers that are found in the graphical presentations.

9.1 Example 1: Safe DMR

9.1.1 Pattern definition

Figure 33 illustrates a composite pattern named *Safe DMR*. We read the figure from the left corner of the figure towards the right repeatedly from the top and downwards like reading a book. The first graphical elements identify the inputs to the composite, the name of the composite and the outputs of the composite. The next set of graphical elements defines the constituents of the composite, e.g. patterns, relations and instantiation orders.

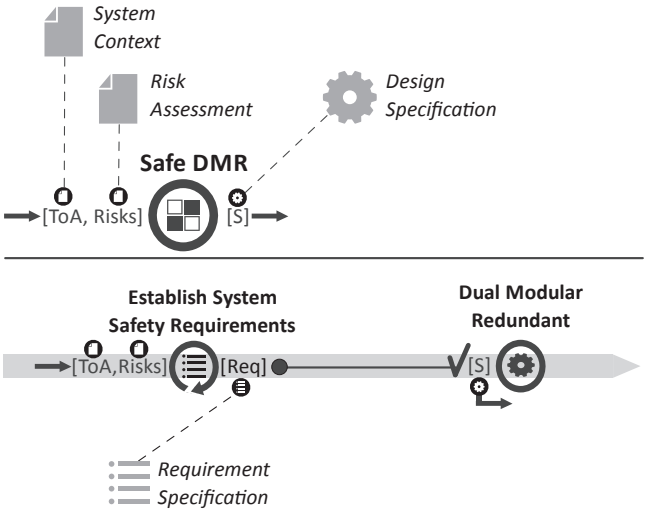


Figure 33 Safe DMR – Composite Pattern

A composite pattern may model the intended or actual application of a combination of patterns. We assume that the composite depicted in Figure 33 models the intended application of patterns in some context, thus the instantiation of parameters are visualised.

Assume that the composite pattern *Safe DMR* models the application of patterns in a context where the following documentation represents parameter instantiations:

- *System Context Description* is a document describing the intended context and the function of the system under development
- *Risk Assessment* describes the result of a risk assessment of the system at hand and presents an overview of every relevant hazard with respect to the operation of the system in its intended context.
- *Requirement Specification* is a specification of the safety requirements associated with the system at hand. The requirements are defined as a result of applying the Safe DMR pattern with the documentations *System Context Description* and *Risk Assessment* as input.
- *Design Specifications* is a design specification that details the technical design of a system in accordance with the requirements identified in the *Requirement Specification*.

Figure 33 is similar to the one presented in Figure 4 but where the explanatory annotations are removed.

9.1.2 Textual syntax

The translation of the composite pattern in Figure 33 into its textual syntax gives the following:

```
declaration(
        Safe DMR
        publicinputparameter( documentationparameter( ToA ) )
        publicinputparameter( documentationparameter( Risks ) )
        publicoutputparameter( designparameter( S ) )
        instantiates(
            documentationartefact( System Context Description )
            documentationparameter( ToA )
        )
        instantiates(
            documentationartefact( Risk Assessment )
            documentationparameter( Risks )
        )
        instantiates(
            designartefact( Design Specification )
            designparameter( S )
        )
)
content(
        Safe DMR
        containedpatterndeclaration(
            procreqpatternreference( Establish System Safety Requirement )
            publicinputparameter( documentationparameter( ToA ) )
            publicinputparameter( documentationparameter( Risks ) )
            localoutputparameter( requirementparameter( Req ) )
            instantiates(
                requirementartefact( Requirement Specification )
                documentationparameter( Req )
            )
        )
        containedpatterndeclaration(
            prodsolpatternreference( Dual Modular Redundant)
            publicoutputparameter( designparameter( S ) )
        )
        satisfies(
            prodsolpatternreference( Dual Modular Redundant )
            designparameter( S )
            procreqpatternreference( Establish System Safety Requirements )
            requirementparameter( Req )
        )
        serialinstantiation(
            procreqpatternreference( Establish System Safety Requirements )
            prodsolpatternreference( Dual Modular Redundant )
```

)

)

9.1.3 Translation

The translation of the textual syntax of the composite pattern illustrated in Figure 33 results in the following English text:

Safe DMR requires the input:

- a description *ToA*
- a description *Risks*

An assignment of an input of *Safe DMR* is assigned to every correspondingly named input with public accessibility of contained patterns. An input defined with public accessibility is when introduced in the following demarked “(public)”, otherwise demarked “(local)”.

Safe DMR delivers the output:

- a design specification *S*

An output of *Safe DMR* is delivered when the correspondingly named output, with public accessibility, is delivered from a contained pattern. An output with public accessibility is when introduced in the following demarked “(public)”, otherwise demarked “(local)”.

Safe DMR is instantiated such that:

- the description *System Context* represents *ToA*
- the description *Risk Assessment* represents *Risks*
- the design *Design Specification* represents *S*

Safe DMR contains the patterns *Establish System Safety Requirement* and *Dual Modular Redundant*.

Req (local) is the output of the process pattern *Establish System Safety Requirements* when applied to *ToA* (public) and *Risks* (public). *Establish System Safety Requirements* is instantiated such that:

- the requirements *Requirement Specification* represents *Req*

S (public) is the output of the design pattern *Dual Modular Redundant*.

S is required to satisfy the requirements defined by *Req* where:

- *S* is the output from the design pattern *Dual Modular Redundant*.
- *Req* is the output from the process pattern *Establish System Safety Requirements*.

Safe DMR defines the following order for instantiating patterns:

- The *pattern Establish System Safety Requirements* should be instantiated before the pattern *Dual Modular Redundant*.

9.2 Example 2: Functional Requirements

9.2.1 Pattern definition

In the composite pattern illustrated in Figure 34, the icons denoting the type of a parameter is not used in order to simplify the visual presentation. In the translation of the pattern into its textual syntax, the parameter types are indicated although not visualised in Figure 34 as these are known from the specifications of the respective referenced patterns detailed in [5].

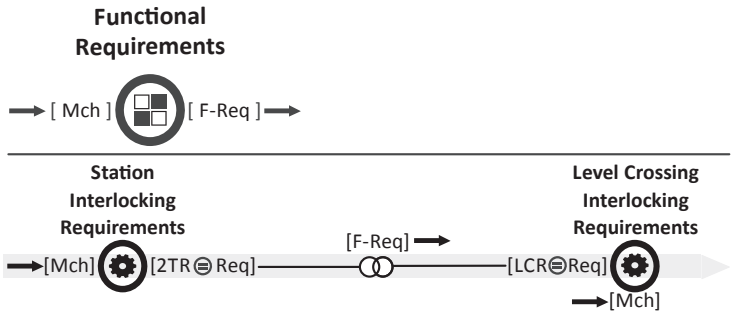


Figure 34 Functional Requirements – Composite Pattern (from [5])

9.2.2 Textual syntax

The translation of the composite pattern in Figure 34 into its textual syntax gives the following:

declaration(
    Functional Requirements
    publicinputparameter(**documentationparameter**(*Mch*))
    publicoutputparameter(**requirementparameter**(*F-Req*))
)
content(
    Functional Requirements
    containedpatterndeclaration(
        prodreqpatternreference(*Station Interlocking Requirements*)
        publicinputparameter(**documentationparameter**(*Mch*))
        localoutputparameter(
            parametersetalias(
                alias(*2TR*)
                requirementparameter(*Req*)
)
)
)
    containedpatterndeclaration(
        prodreqpatternreference(*Level Crossing Interlocking Requirements*)
        publicinputparameter(**documentationparameter**(*Mch*))
        localoutputparameter(
            parametersetalias(
                alias(*LCR*)
                requirementparameter(*Req*)
)
)
)
    combines(
        prodreqpatternreference(*Station Interlocking Requirements*)
        parametersetalias(**alias**(*2TR*))
        procreqpatternreference(*Level Crossing Interlocking Requirements*)

```
            parametersetalias( alias( LCR ) )
            publicoutputparameter( parametersetalias( alias( F-Req ) ) )
        )
        serialinstantiation(
            prodreqpatternreference( Station Interlocking Requirements )
            prodsolpatternreference( Level Crossing Interlocking Requirements)
        )
)
```

9.2.3 Translation

The translation of the textual syntax of the composite pattern illustrated in Figure 34 results in the following English text:

Functional Requirements requires the input

- a description *Mch*

An assignment of an input of *Functional Requirements* is assigned to every correspondingly named input with public accessibility of contained patterns. An input defined with public accessibility is when introduced in the following demarked “(public)”, otherwise demarked “(local)”.

Functional Requirements delivers the output

- a set of requirements *F-Req*

An output of *Functional Requirements* is delivered when the correspondingly named output, with public accessibility, is delivered from a contained pattern. An output with public accessibility is when introduced in the following demarked “(public)”, otherwise demarked “(local)”.

Functional Requirements contains the patterns *Station Interlocking Requirements* and *Level Crossing Interlocking Requirements.*

2TR (alias for *Req*) (local) is the output of the requirements pattern *Station Interlocking Requirements* when applied to *Mch* (public).

LCR (alias for *Req*) (local) is the output of the requirements pattern *Level Crossing Interlocking Requirements* when applied to *Mch* (public).

2TR and *LCR* are combined in a union where:

- *2TR* is the output from the requirements pattern *Station Interlocking Requirements.*
- *LCR* is the output from the requirements pattern *Level Crossing Interlocking Requirements.*
- The result of combining is represented by the set named *F-Req* (public).

Functional Requirements defines the following order for instantiating patterns:

- The pattern *Station Interlocking Requirements* should be instantiated before the pattern *Level Crossing Interlocking Requirements*.

9.3 Example 3: Safety Requirements

9.3.1 Pattern definition

Figure 35 illustrates a composite pattern named Safety Requirements. In the translation of the pattern into its textual syntax, the parameter types are indicated although not visualised in Figure 35 as these are known from the specifications of the respective referenced patterns detailed in [5].

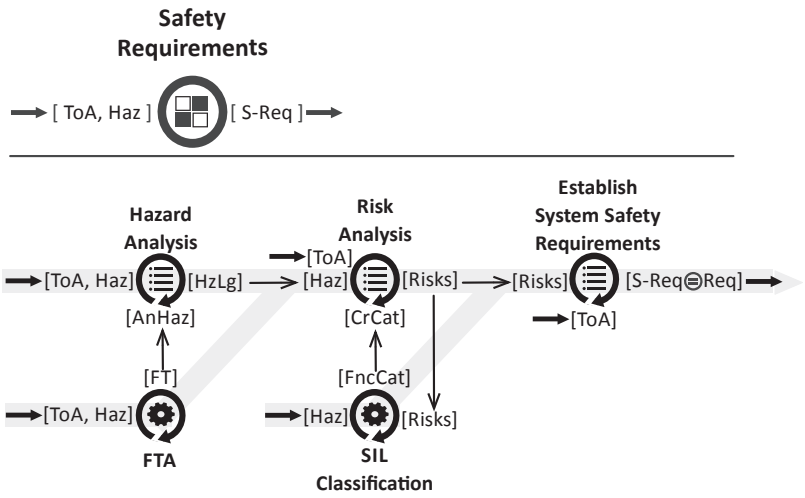


Figure 35 Safety Requirements – Composite Pattern (from [5])

9.3.2 Textual syntax

The translation of the composite pattern in Figure 35 into its textual syntax gives the following:

declaration(
Safety Requirements
publicinputparameter(**documentationparameter**(*ToA*))
publicinputparameter(**documentationparameter**(*Haz*))
publicoutputparameter(**requirementparameter**(*S-Req*))
)
content(
Safety Requirements
containedpatterndeclaration(
procreqpatternreference(*Hazard Analysis*)
publicinputparameter(**documentationparameter**(*ToA*))
publicinputparameter(**documentationparameter**(*Haz*))
localinputparameter(**documentationparameter**(*AnHaz*))
localoutputparameter(**documentationparameter**(*HzLg*))
)
containedpatterndeclaration(
procreqpatternreference(*Risk Analysis*)
publicinputparameter(**documentationparameter**(*ToA*))
localinputparameter(**documentationparameter**(*Haz*))
localinputparameter(**documentationparameter**(*CrcCat*))
localoutputparameter(**documentationparameter**(*Risks*))
)
containedpatterndeclaration(
procreqpatternreference(Establish System Safety Requirements)
publicinputparameter(**documentationparameter**(*ToA*))
localinputparameter(**documentationparameter**(*Risks*))
publicoutputparameter(

```
            parametersetalias(
                  alias( S-Req )
                  requirementparameter( Req )
            )
      )
)
containedpatterndeclaration(
      procsolpatternreference( FTA )
      publicinputparameter( documentationparameter( ToA ) )
      publicinputparameter( documentationparameter( Haz ) )
      localoutputparameter( documentationparameter( FT ) )
)
containedpatterndeclaration(
      procsolpatternreference( SIL Classification )
      publicinputparameter( documentationparameter( Haz ) )
      localinputparameter( documentationparameter( Risks ) )
      localoutputparameter( documentationparameter( FncCat ) )
)
assigns(
      procreqpatternreference( Hazard Analysis )
      documentationparameter( HzLg )
      procreqpatternreference( Risk Analysis )
      documentationparameter( Haz )
)
assigns(
      procrepatternreference( Risk Analysis )
      documentationparameter( Risks )
      procreqpatternreference( Establish System Safety Requirements )
      documentationparameter( Risks )
)
assigns(
      procsolpatternreference( FTA )
      documentationparameter( FT )
      procreqpatternreference( Hazard Analysis )
      documentationparameter( AnHaz )
)
assigns(
      procsolpatternreference( SIL Classification )
      documentationparameter( FncCat )
      procreqpatternreference( Risk Analysis )
      documentationparameter( CrCat )
)
serialinstantiation(
      procreqpatternreference( Hazard Analysis )
      procreqpatternreference( Risk Analysis )
)
parallelinstantiation(
      procreqpatternreference( Hazard Analysis )
      procsolpatternreference( FTA )
```

```
        )
        serialinstantiation(
            procreqpatternreference( Risk Analysis )
            procreqpatternreference( Establish System Safety Requirements )
        )
        parallelinstantiation(
            procreqpatternreference( Risk Analysis )
            procsolpatternreference( SIL Classification )
        )
)
```

9.3.3 Translation

The translation of the textual syntax of the composite pattern illustrated in Figure 35 results in the following English text:

Safety Requirements requires the input:

- a description *ToA*
- a description *Haz*

An assignment of an input of *Safety Requirements* is assigned to every correspondingly named input with public accessibility of contained patterns. An input defined with public accessibility is when introduced in the following demarked “(public)”, otherwise demarked “(local)”.

Safety Requirements delivers the output:

- a set of requirements *S-Req*

An output of *Safety Requirements* is delivered when the correspondingly named output, with public accessibility, is delivered from a contained pattern. An output with public accessibility is when introduced in the following demarked “(public)”, otherwise demarked “(local)”.

Safety Requirements contains the patterns *Hazard Analysis, Risk Analysis, Establish System Safety Requirements, FTA,* and *SIL Classification*.

HzLg (local) is the output of the process pattern *Hazard Analysis* when applied to *ToA* (public)*, Haz* (public)*,* and *AnHaz* (local)*.*

Risks (local) is the output of the process pattern *Risk Analysis* when applied to *ToA* (public)*, Haz* (local)*,* and *CrCat* (local).

S-Req (alias for *Req*) (public) is the output of the process pattern *Establish System Safety Requirements* when applied to *ToA* (public) and *Risks* (local).

FT (local) is the output of the method pattern *FTA* when applied to *ToA* (public) and *Haz* (public)*.*

FncCat (local) is the output of *SIL Classification* when applied to *Haz* (public) and *Risks* (local).

HzLg is assigned to *Haz* where:

- *HzLg* is the output of the process pattern *Hazard Analysis.*
- *Haz* is input to the process pattern *Risk Analysis.*

Risks is assigned to *Risks* where:

- *Risks* is the output of the process pattern *Risk Analysis.*

- *Risks* is input to the process pattern *Establish System Safety Requirements.*

FT is assigned to *AnHaz* where:

- *FT* is the output of the method pattern *FTA.*
- *AnHaz* is input to the process pattern *Hazard Analysis.*

FncCat is assigned to CrCat where:

- *FncCat* is the output of the method pattern *SIL Classification.*
- *CrCat* is input to the process pattern *Risk Analysis.*

Safety Requirements defines the following order for instantiating patterns:

- The pattern *Hazard Analysis* should be instantiated before the pattern *Risk Analysis.*
- The patterns *Hazard Analysis* and *FTA* may be instantiated in parallel.
- The pattern *Risk Analysis* should be instantiated before the pattern *Establish System Safety Requirements*.
- The patterns *Risk Analysis* and *SIL Classification* may be instantiated in parallel.

9.4 Example 4: Requirements

9.4.1 Pattern definition

In the composite pattern illustrated in Figure 36, icons for denoting the type of a parameter is not used in order to simplify the visual presentation. In the translation of the pattern into its textual syntax, the parameter types are indicated as these are known and may be found in the respective definitions of the patterns referred to within a composite pattern. The definitions of the referenced patterns are provided in Section 9.2 and 9.3.

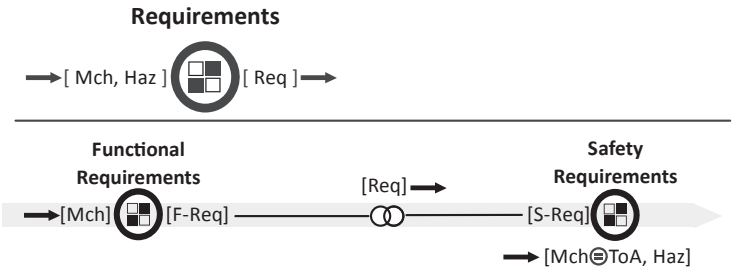


Figure 36 Requirements – Composite Pattern (from [5])

9.4.2 Textual syntax

The translation of the composite pattern in Figure 36 into its textual syntax gives the following:

declaration(
    Requirements
    publicinputparameter(**documentationparameter**(*Mch*))
    publicinputparameter(**documentationparameter**(*Haz*))
    publicoutputparameter(**requirementparameter**(*Req*))
)
content(

```
    Requirements
    containedpatterndeclaration(
        compositepatternreference( Functional Requirements )
        publicinputparameter( documentationparameter( Mch ) )
        localoutputparameter( requirementparameter( F-Req ) )
    )
    containedpatterndeclaration(
        compositepatternreference( Safety Requirements )
        publicinputparameter(
            parametersetalias(
                    alias( Mch )
                    documentationparameter( ToA )
            )
        )
        publicinputparameter( documentationparameter( Haz ) )
        localoutputparameter( requirementparameter( S-Req ) )
    )
    combines(
        compositepatternreference( Functional Requirements )
        requirementparameter( F-Req )
        compositepatternreference( Safety Requirements )
        requirementparameter( S-Req  )
        publicoutputparameter(
            parametersetalias(
                    alias( Req )
            )
        )
    )
    serialinstantiation(
        compositepatternreference( Functional Safety )
        compositepatternreference( Safety Requirements )
    )
)
```

9.4.3 Translation

The translation of the textual syntax of the composite pattern illustrated in Figure 36 results in the following English text:

Requirements requires the input

- a description *Mch*
- a description *Haz*

An assignment of an input of *Requirements* is assigned to every correspondingly named input with public accessibility of contained patterns. An input defined with public accessibility is when introduced in the following demarked “(public)”, otherwise demarked “(local)”.

Requirements delivers the output

- a set of requirements *Req*

An output of *Requirements* is delivered when the correspondingly named output, with public accessibility, is delivered from a contained pattern. An output with public accessibility is when introduced in the following demarked “(public)”, otherwise demarked “(local)”.

Requirements contains the patterns *Functional Requirements* and *Safety Requirements.*

F-Req (local) is the output of the composite pattern *Functional Requirements* when applied to *Mch* (public).

S-Req (local) is the output of the composite pattern *Safety Requirements* when applied to *Mch* (alias for *ToA*) (public) and *Haz* (public).

F-Req and *S-Req* are combined in a union where:

- *F-Req* is the output from the composite pattern *Functional Requirements.*
- *S-Req* is the output from the composite pattern *Safety Requirements.*
- The result of combining is represented by the set named *Req* (public).

Requirements defines the following order for instantiating patterns:

The pattern *Functional Requirements* should be instantiated before the pattern *Safety Requirements*.

9.5 Example 5: Safety Case

9.5.1 Pattern definition

Figure 37 illustrates a composite pattern named Safety Case. In the translation of the pattern into its textual syntax, the parameter types are indicated although not visualised in Figure 37 as these are known from the specifications of the respective referenced patterns detailed in [5].

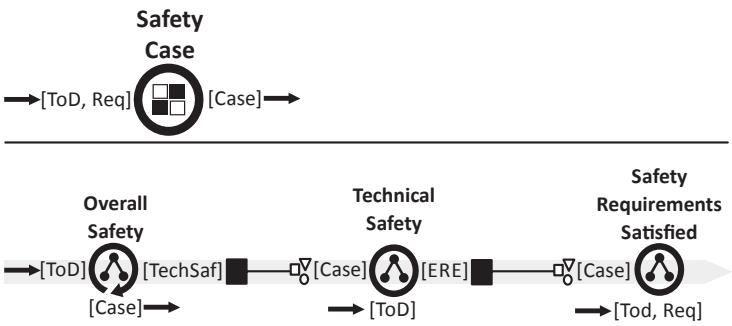


Figure 37 Safety Case – Composite Pattern (from [5])

9.5.2 Textual syntax

The translation of the composite pattern in Figure 37 into its textual syntax gives the following:

declaration(
Safety Case
publicinputparameter(**documentationparameter**(*ToD*))
publicinputparameter(**requirementparameter**(*Req*))
publicoutputparameter(**safetycaseparameter**(*Case*))

```
)
content(
        Safety Case
        containedpatterndeclaration(
            procsafcasepatternreference( Overall Safety )
            publicinputparameter( documentationparameter( ToD ) )
            publicoutputparameter( safetycaseparameter( Case ) )
            localoutputparameter( safetycaseparameter( TechSaf ) )
        )
        containedpatterndeclaration(
            prodsafcasepatternreference( Technical Safety )
            publicinputparameter( documentationparameter( ToD ) )
            localoutputparameter( safetycaseparameter( Case ) )
            localoutputparameter( safetycaseparameter( ERE ) )
        )
        containedpatterndeclaration(
            prodsafcasepatternreference( Safety Requirements Satisfied )
            publicinputparameter( documentationparameter( ToD ) )
            publicinputparameter( requirementparameter( Req ) )
            localoutputparameter( safetycaseparameter( Case ) )
        )
        details(
            prodsafcasepatternreference( Technical Safety )
            safetycaseparameter( Case )
            procsafcasepatternreference( Overall Safety )
            requirementparameter( TechSaf )
        )
        details(
            prodsafcasepatternreference( Safety Requirements Satisfied )
            safetycaseparameter( Case )
            prodsafcasepatternreference( Technical Safety )
            requirementparameter( ERE )
        )
        serialinstantiation(
            procsafcasepatternreference( Overall Safety )
            prodsafcasepatternreference( Technical Safety )
        )
        serialinstantiation(
            prodsafcasepatternreference( Technical Safety )
            prodsafcasepatternreference( Safety Requirements Satisfied )
        )
)
```

9.5.3 Translation

The translation of the textual syntax of the composite pattern illustrated in Figure 37 results in the following English text:

Safety Case requires the input:

- a description *ToD*

- a set of requirements *Req*

An assignment of an input of *Safety Case* is assigned to every correspondingly named input with public accessibility of contained patterns. An input defined with public accessibility is when introduced in the following demarked “(public)”, otherwise demarked “(local)”.

Safety Case delivers the output:

- a safety case specification *Case*

An output of *Safety Case* is delivered when the correspondingly named output, with public accessibility, is delivered from a contained pattern. An output with public accessibility is when introduced in the following demarked “(public)”, otherwise demarked “(local)”.

Safety Case contains the patterns *Overall Safety, Technical Safety, and Safety Requirements Satisfied*.

TechSaf (local) and *Case* (public) are the outputs of the safety case pattern *Overall Safety* when applied to *ToD* (public).

Case (local) and *ERE* (local) are the outputs of the safety case pattern *Technical Safety* when applied to *ToD* (public).

Case (local) is the output of the safety case pattern *Safety Requirements Satisfied* when applied to *ToD* (public) and *Req* (public).

Case is required to detail *TechSaf* where:

- *Case* is the output of the safety case pattern *Technical Safety.*
- *TechSaf* is the output of the safety case pattern *Overall Safety.*

Case is required to detail *ERE* where:

- *Case* is the output of the safety case pattern *Safety Requirements Satisfied.*
- *ERE* is the output of the safety case pattern *Technical Safety.*

Safety Requirements defines the following order for instantiating patterns:

- The pattern *Overall Safety* should be instantiated before the pattern *Technical Safety.*
- The pattern *Technical Safety* should be instantiated before the pattern *Safety Requirements Satisfied*.

9.6 Example 6: Pattern Solution

9.6.1 Pattern definition

Figure 38 illustrates a composite pattern named Pattern Solution. In the translation of the pattern into its textual syntax, the parameter types are indicated although not visualised in Figure 38 as these are known from the specifications of the respective referenced patterns. The referenced patterns are taken from [5] and in addition detailed in Section 8.4, 9.4, and 9.5.

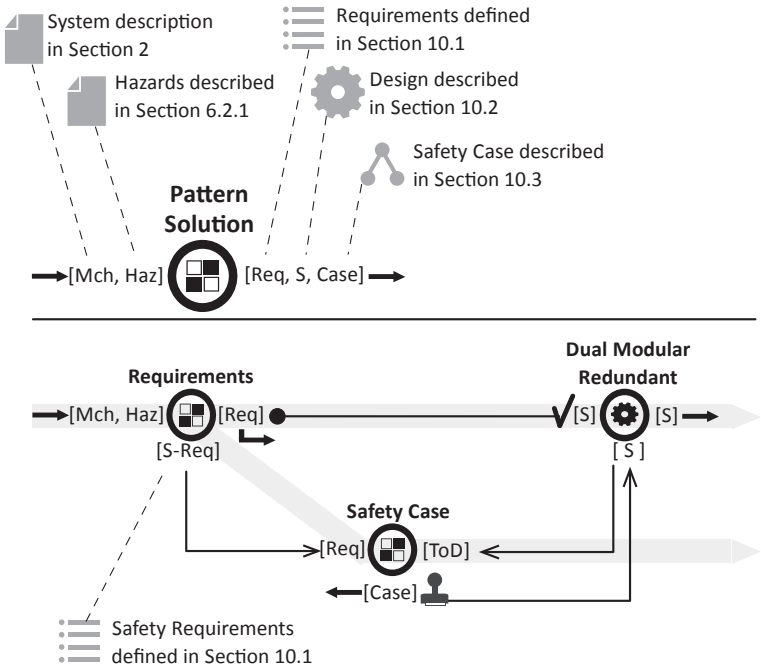


Figure 38 Pattern Solution – Composite Pattern (from [5])

9.6.2 Textual Syntax

The translation of the composite pattern in Figure 38 into its textual syntax gives the following:

declaration(
Pattern Solution
publicinputparameter(**documentationparameter**(*Mch*))
publicinputparameter(**requirementparameter**(*Haz*))
publicoutputparameter(**requirementparameter**(*Req*))
publicoutputparameter(**designparameter**(*S*))
publicoutputparameter(**safetycaseparameter**(*Case*))
instantiates(
documentationartefact(*System description in Section 2*)
documentationparameter(*Mch*))
instantiates(
documentationartefact(*Hazards described in Section 6.2.1*)
documentationparameter(*Haz*))
instantiates(
requirementartefact(*Requirements defined in Section 10.1*)
requirementparameter(*Req*))
instantiates(
designartefact(*Design described in Section 10.2*)
designparameter(*S*))
instantiates(
safetycaseartefact(*Safety Case described in Section 10.3*)
safetycaseparameter(*Case*))

```
)
content(
        Pattern Solution
        containedpatterndeclaration(
            compositepatternreference( Requirements )
            publicinputparameter( documentationparameter( Mch ) )
            publicinputparameter( documentationparameter( Haz ) )
            publicoutputparameter( requirementparameter( Req ) )
            localoutputparameter( requirementparameter( S-Req ) )
            instantiates(
                requirementartefact( Safety Requirements defined in Section 10.1)
                requirementparameter( S-Req ) )
        )
        containedpatterndeclaration(
            compositepatternreference(Safety Case )
            localinputparameter( documentationparameter( ToD ) )
            localinputparameter( requirementparameter( Req ) )
            publicoutputparameter( safetycaseparameter( Case ) )
        )
        containedpatterndeclaration(
            prodsolpatternreference( Dual Modular Redundant )
            publicoutputparameter( designparameter( S ) )
        )
        assigns(
            compositepatternreference( Requirements )
            requirementparameter( S-Req )
            compositepatternreference( Safety Case )
            requirementparameter( Req )
        )
        satisfies(
            designpatternreference( Dual Modular Redundant )
            designparameter( S )
            compositepatternreference( Requirements )
            requirementparameter( Req )
        )
        assigns(
            designpatternreference( Dual Modular Redundant )
            designparameter( S )
            compositepatternreference( Safety Case )
            documentationparameter( ToD )
        )
        demonstrates(
            compositepatternreference( Safety Case )
            safetycaseparameter( Case )
            designpatternreference( Dual Modular Redundant )
            designparameter( S )
        )
        serialinstantiation(
            compositepatternreference( Requirements )
```

```
        designpatternreference( Dual Modular Redundant )
    )
    serialinstantiation(
        compositepatternreference( Requirements )
        compositepatternreference( Safety Case )
    )
    parallelinstantiation(
        compositepatternreference( Safety Case )
        designpatternreference( Dual Modular Redundant )
    )
)
```

9.6.3 Translation

The translation of the textual syntax of the composite pattern illustrated in Figure 38 results in the following English text:

Pattern Solution requires the input:

- a description *Mch*
- a description *Haz*

An assignment of an input of *Pattern Solution* is assigned to every correspondingly named input with public accessibility of contained patterns. An input defined with public accessibility is when introduced in the following demarked “(public)”, otherwise demarked “(local)”.

Pattern Solution delivers the output:

- a set of requirements *Req*
- a design specification *S*
- a safety case specification *Case*

An output of *Pattern Solution* is delivered when the correspondingly named output, with public accessibility, is delivered from a contained pattern. An output with public accessibility is when introduced in the following demarked “(public)”, otherwise demarked “(local)”.

Pattern Solution is instantiated such that:

- the description *System description in Section 2* represents *Mch*
- the description *Hazards described in Section 6.2.1* represents *Haz*
- the requirements *Requirements defined in Section 10.1* represents *Req*
- the design *Design described in Section 10.2* represents *S*
- the safety case *Safety case described in Section 10.3* represents *Case*

Pattern Solution contains the patterns *Requirements, Safety Case, and Dual Modular Redundant*.

Req (public) and *S-Req* (local) are the outputs of the composite pattern *Requirements* when applied to *Mch* (public) and *Haz* (public). *Requirements* is instantiated such that:

- the requirements *Safety Requirements defined in Section 10.1* represents *S-Req*

Case (public) is the output of the composite pattern *Safety Case* when applied to *ToD* (local) and *Req* (local).

S (public) is the output of the design pattern *Dual Modular Redundant.*

S-Req is assigned to *Req* where:

- *S-Req* is the output of the composite pattern *Requirements.*
- *Req* is input to the composite pattern *Safety Case.*

S is required to satisfy the requirements defined by *Req* where:

- *S* is the output of the design pattern *Dual Modular Redundant.*
- *Req* is the output of the composite pattern *Requirements.*

S is assigned to *ToD* where:

- *S* is the output of the design pattern *Dual Modular Redundant.*
- *ToD* is input to the composite pattern *Safety Case.*

Case is required to demonstrate safety of *S* where:

- *Case* is the output of the composite pattern *Safety Case.*
- *S* is the output of the design pattern *Dual Modular Redundant.*

Pattern Solution defines the following order for instantiating patterns:

- The pattern *Requirements* should be instantiated before the pattern *Dual Modular Redundant.*
- The pattern *Requirements* should be instantiated before the pattern *Safety Case.*
- The patterns *Safety Case* and *Dual Modular Redundant* may be instantiated in parallel.

9.7 Example 7: Two Track Station Interlocking

9.7.1 Pattern definition

Figure 39 illustrates a composite pattern named *Two Track Station Interlocking*. The pattern presents the same relationships defined between the patterns within the composites presented in Example 2, 3, 4, 5, and 6 but then contained within one pattern definition instead of a modular one. The composite is constructed in order to exemplify a non-modular presentation of the same information as presented in the mentioned examples, besides different identifiers being used for artefact references.

Assume that the composite pattern *Two Track Station Interlocking* in Figure 39 models the application of patterns in a context where the following documentation represents parameter instantiations:

- *Ref. System Description* is a document describing the intended context and function of the system under development
- *Ref. Hazard Description* is a document describing the result of a risk assessment of the system under development.
- *Ref. Two Track Station Requirement Specification* is a specification of the requirements.
- *Ref. Level Crossing Requirements Specification* is a specification of requirements.
- *Ref. Safety Requirements Specification* is a specification of requirements.
- *Ref. Design Specification* is a specification of a design.
- *Ref. Safety Demonstration* is a safety case description.

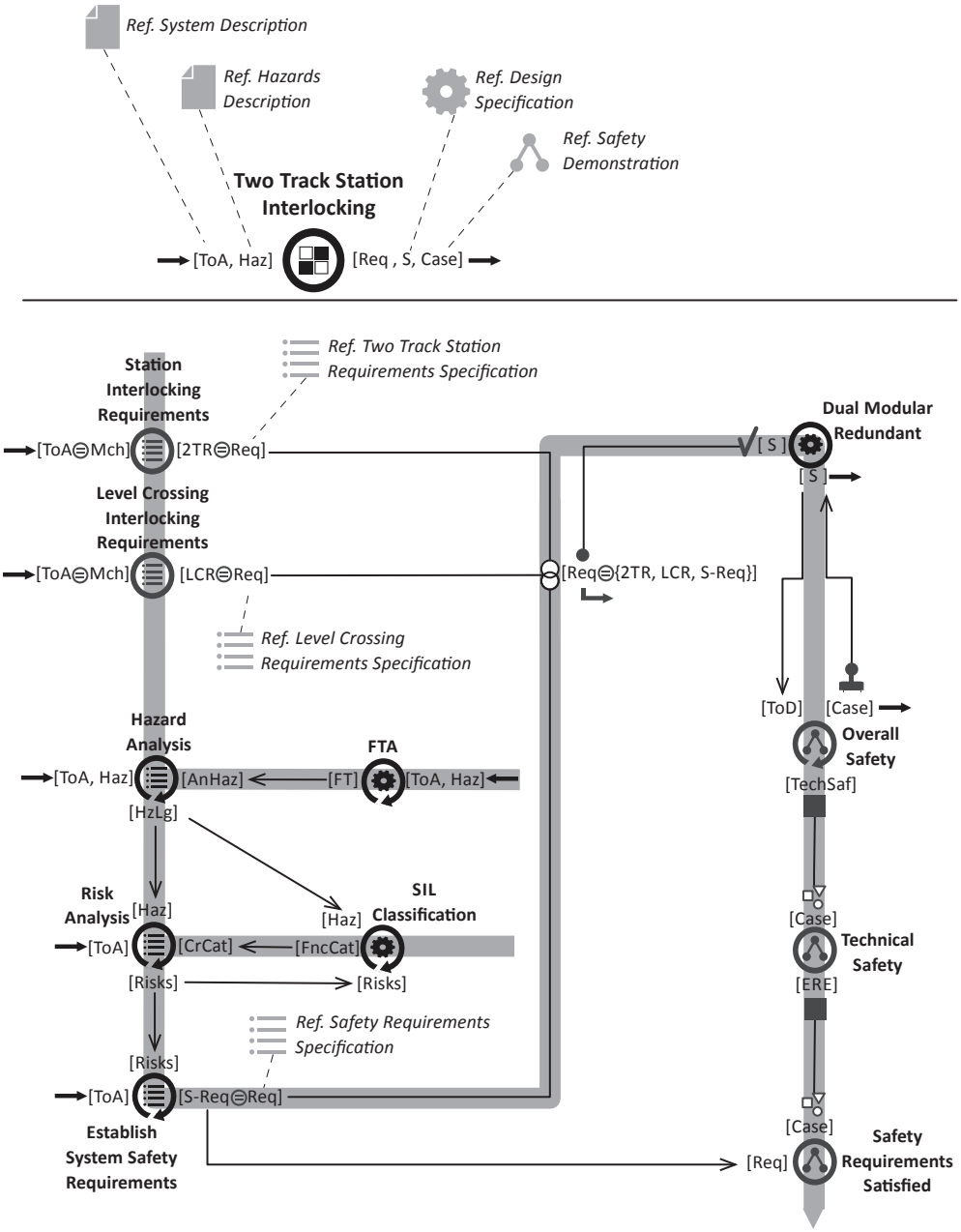


Figure 39 Two Track Station Interlocking – Composite Pattern

9.7.2 Textual syntax

The translation of the composite pattern in Figure 39 into its textual syntax gives the following:

declaration(

Two Track Station Interlocking

```
        publicinputparameter( documentationparameter( ToA ) )
        publicinputparameter( documentationparameter( Haz ) )
        publicoutputparameter( requirementparameter( Req ) )
        publicoutputparameter( designparameter( S ) )
        publicoutputparameter( safetycaseparameter( Case ) )
        instantiates(
            documentationartefact( Ref. System Description )
            documentationparameter( ToA ) )
        instantiates(
            documentationartefact( Ref. Hazards Description )
            documentationparameter( Haz ) )
        instantiates(
            designartefact( Ref. Design Specification )
            designparameter( S ) )
        instantiates(
            safetycaseartefact( Ref. Safety Demonstration )
            safetycaseparameter( Case ) )
)
content(
        Two Track Station Interlocking
        containedpatterndeclaration(
            prodreqpatternreference( Station Interlocking Requirements )
            publicinputparameter(
                parametersetalias(
                        alias( ToA )
                        documentationparameter( Mch )
                )
            )
            localoutputparameter(
                parametersetalias(
                        alias( 2TR )
                        requirementparameter( Req )
                )
            )
            instantiates(
                requirementartefact( Ref. Two Track Station Requirements Specification )
                requirementparameter( Req )
            )
        )
        containedpatterndeclaration(
            prodreqpatternreference( Level Crossing Interlocking Requirements)
            publicinputparameter(
                parametersetalias(
                        alias( ToA )
                        documentationparameter( Mch )
                )
            )
            localoutputparameter(
                parametersetalias(
```

```
                alias( LCR )
                requirementparameter( Req )
            )
        )
        instantiates(
            requirementartefact( Ref. Level Crossing Requirements Specification )
            requirementparameter( Req )
        )
)
containedpatterndeclaration(
        procreqpatternreference( Hazard Analysis )
        publicinputparameter( documentationparameter( ToA ) )
        publicinputparameter( documentationparameter( Haz ) )
        localinputparameter( documentationparameter( AnHaz ) )
        localoutputparameter( documentationparameter( HzLg ) )
)
containedpatterndeclaration(
        procsolpatternreference( FTA )
        publicinputparameter( documentationparameter( ToA ) )
        publicinputparameter( documentationparameter( Haz ) )
        localoutputparameter( documentationparameter( FT ) )
)
containedpatterndeclaration(
        procreqpatternreference( Risk Analysis )
        publicinputparameter( documentationparameter( ToA ) )
        localinputparameter( documentationparameter( Haz ) )
        localinputparameter( documentationparameter( CrCat ) )
        localoutputparameter( documentationparameter( Risks ) )
)
containedpatterndeclaration(
        procsolpatternreference( SIL Classification )
        localinputparameter( documentationparameter( Haz ) )
        localinputparameter( documentationparameter( Risks ) )
        localoutputparameter( documentationparameter( FncCat ) )
)
containedpatterndeclaration(
        procreqpatternreference( Establish System Safety Requirements )
        publicinputparameter( documentationparameter( ToA ) )
        localinputparameter( documentationparameter( Risks ) )
        localoutputparameter(
            parametersetalias(
                alias( S-Req )
                requirementparameter( Req )
            )
        )
        instantiates(
            requirementartefact( Ref. Safety Requirements Specification )
            requirementparameter( Req )
        )
```

```
)
containedpatterndeclaration(
      prodsolpatternreference( Dual Modular Redundant )
      publicoutputparameter( designparameter( S ) )
)
containedpatterndeclaration(
      procsafcasepatternreference( Overall Safety )
      publicinputparameter( documentationparameter( ToD ) )
      publicoutputparameter( safetycaseparameter( Case ) )
      localoutputparameter( safetycaseparameter( TechSaf ) )
)
containedpatterndeclaration(
      prodsafcasepatternreference( Technical Safety )
      localoutputparameter( safetycaseparameter( Case ) )
      localoutputparameter( safetycaseparameter( ERE ) )
)
containedpatterndeclaration(
      prodsafcasepatternreference( Safety Requirements Satisfied )
      localoutputparameter( safetycaseparameter( Case ) )
      localinputparameter( requirementparameter( Req ) )
)
combines(
      prodreqpatternreference( Station Interlocking Requirements )
      parametersetalias( alias( 2TR ) )
      prodreqpatternreference( Level Crossing Interlocking Requirements)
      parametersetalias( alias( LCR ) )
      procreqpatternreference( Establish System Safety Requirements )
      parametersetalias( alias( S-Req ) )
      publicoutputparameter(
          parametersetalias(
                alias( Req )
                parameterset(
                      alias( 2TR )
                      alias( LCR )
                      alias( S-Req )
                )
          )
      )
)
assigns(
      procreqpatternreference( Hazard Analysis )
      documentationparameter( HzLg )
      procreqpatternreference( Risk Analysis )
      documentationparameter( Haz )
)
assigns(
      procreqpatternreference( Hazard Analysis )
      documentationparameter( HzLg )
      procsolpatternreference( SIL Classification )
```

documentationparameter(*Haz*)
)
assigns(
procsolpatternreference(*FTA*)
documentationparameter(*FT*)
procreqpatternreference(*Hazard Analysis*)
documentationparameter(*AnHaz*)
)
assigns(
procreqpatternreference(*Risk Analysis*)
documentationparameter(*Risks*)
procreqpatternreference(*Establish System Safety Requirements*)
documentationparameter(*Risks*)
)
assigns(
procreqpatternreference(*Risk Analysis*)
documentationparameter(*Risks*)
procsolpatternreference(*SIL Classification*)
documentationparameter(*Risks*)
)
assigns(
procsolpatternreference(*SIL Classification*)
documentationparameter(*FncCat*)
procreqpatternreference(*Risk Analysis*)
documentationparameter(*CrCat*)
)
satisfies(
prodsolpatternreference(*Dual Modular Redundant*)
designparameter(*S*)
patterngroupreference(
Station Interlocking Requirements
Level Crossing Interlocking Requirements
Establish System Safety Requirements
)
parametersetalias(**alias**(*Req*))
)
assigns(
prodsolpatternreference(*Dual Modular Redundant*)
designparameter(*S*)
procsafcasepatternreference(*Overall Safety*)
documentationparameter(*ToD*)
)
demonstrates(
procsafcasepatternreference(*Overall Safety*)
safetycaseparameter(*Case*)
prodsolpatternreference(*Dual Modular Redundant*)
designparameter(*S*)
)
details(

prodsafcasepatternreference(*Technical Safety*)
safetycaseparameter(*Case*)
procsafcasepatternreference(*Overall Safety*)
safetycaseparameter(*TechSaf*)
)
details(
prodsafcasepatternreference(*Safety Requirements Satisfied*)
safetycaseparameter(*Case*)
prodsafcasepatternreference(*Technical Safety*)
safetycaseparameter(*ERE*)
)
assigns(
procreqpatternreference(*Establish System Safety Requirements*)
parametersetalias(**alias**(*S-Req*))
prodsafcasepatternreference(*Safety Requirements Satisfied*)
requirementparameter(*Req*)
)
serialinstantiation(
prodreqpatternreference(*Station Interlocking Requirements*)
prodreqpatternreference(*Level Crossing Interlocking Requirements*)
)
serialinstantiation(
prodreqpatternreference(*Level Crossing Interlocking Requirements*)
procreqpatternreference(*Hazard Analysis*)
)
parallelinstantiation(
procreqpatternreference(*Hazard Analysis*)
procsolpatternreference(*FTA*)
)
serialinstantiation(
procreqpatternreference(*Hazard Analysis*)
procreqpatternreference(*Risk Analysis*)
)
parallelinstantiation(
procreqpatternreference(*Risk Analysis*)
procsolpatternreference(*SIL Classification*)
)
serialinstantiation(
procreqpatternreference(*Risk Analysis*)
procreqpatternreference(*Establish System Safety Requirements*)
)
serialinstantiation(
procreqpatternreference(*Establish System Safety Requirements*)
prodsolpatternreference(*Dual Modular Redundant*)
)
serialinstantiation(
prodsopatternreference(*Dual Modular Redundant*)
procsafcasepatternreference(*Overall Safety*)
)

serialinstantiation(
procrsafecasepatternreference(*Overall Safety*)
prodsafcasepatternreference(*Technical Safety*)
)
serialinstantiation(
prodsafcasepatternreference(*Technical Safety*)
prodsafcasepatternreference(*Safety Requirements Satisfied*)
)
)

9.7.3 Translation

The translation of the textual syntax of the composite pattern illustrated in Figure 39 results in the following English text:

Two Track Station Interlocking requires the input:

- a description *ToA*
- a description *Haz*

An assignment of an input of *Two Track Station Interlocking* is assigned to every correspondingly named input with public accessibility of contained patterns. An input defined with public accessibility is when introduced in the following demarked “(public)”, otherwise demarked “(local)”.

Two Track Station Interlocking delivers the output:

- a set of requirements *Req*
- a design specification *S*
- a safety case specification *Case*

An output of *Two Track Station Interlocking* is delivered when the correspondingly named output, with public accessibility, is delivered from a contained pattern. An output with public accessibility is when introduced in the following demarked “(public)”, otherwise demarked “(local)”.

Two Track Station Interlocking is instantiated such that:

- the description *Ref. System Description* represents *ToA*
- the description *Ref. Hazards Description* represents *Haz*
- the design *Ref. Design Specification* represents *S*
- the safety case *Ref. Safety Demonstration* represents *Case*

Two Track Station Interlocking contains the patterns *Station Interlocking Requirements, Level Crossing Interlocking Requirements, Hazard Analysis, FTA, Risk Analysis, SIL Classification, Establish System Safety Requirements, Dual Modular Redundant, Overall Safety, Technical Safety,* and *Safety Requirements Satisfied.*

2TR (alias for *Req*) (local) is the output of the requirement pattern *Station Interlocking Requirements* when applied to *ToA* (alias for *Mch*) (public). *Station Interlocking Requirements* is instantiated such that:

- the requirements *Ref. Two Track Station Requirements Specification* represents *Req*

LCR (alias for *Req*) (local) is the output of the requirement pattern *Level Crossing Interlocking Requirements* when applied to *ToA* (alias for *Mch*) (public). *Level Crossing Interlocking Requirements* is instantiated such that:

- the requirements *Ref. Level Crossing Requirements Specification* represents *Req*

HzLg is the output of the process pattern *Hazard Analysis* when applied to *ToA* (public)*, Haz* (public)*, and AnHaz* (local).

FT (local) is the output of the method pattern *FTA* when applied to *ToA* (public)*, and Haz* (public).

Risks (local) is the output of the process pattern *Risk Analysis* when applied to *ToA* (public)*, Haz* (local)*, and CrCat* (local).

FncCat (local) is the output of the method pattern *SIL Classification* when applied to *Haz* (local)*, and Risks* (local).

S-Req (alias for *Req*) (local) is the output of the process pattern *Establish System Safety Requirements* when applied to *ToA* (public), and *Risks* (local)*. Establish System Safety Requirements* is instantiated such that:

- the requirements *Ref. Safety Requirements Specification* represents *Req*

S (public) is the output of the design pattern *Dual Modular Redundant.*

Case (public)*, and TechSaf* (local) are the outputs of the safety case pattern *Overall Safety* when applied to *ToD* (public).

ERE (local)*,* and *Case* (local) are the outputs of the safety case pattern *Technical Safety.*

Case (local) is the output of the safety case pattern *Safety Requirements Satisfied* when applied to *Req* (local).

2TR, *LCR*, and *S-Req* are combined in a union where:

- *2TR* is the output from the requirement pattern *Station Interlocking Requirements.*
- *LCR* is the output from the requirement pattern *Level Crossing Interlocking Requirements*.
- *S-Req* is the output from the process pattern *Establish System Safety Requirements.*
- The result of combining is represented by *Req* (alias for the set consisting of *2TR*, *LCR*, and *S-Req*) (public).

HzLg is assigned to *Haz* where:

- *HzLg* is the output from the process pattern *Hazard Analysis*.
- *Haz* is input to the process pattern *Risk Analysis*.

HzLg is assigned to *Haz* where:

- *HzLg* is the output from the process pattern *Hazard Analysis*.
- *Haz* is input to the method pattern *SIL Classification*.

FT is assigned to *AnHaz* where:

- *FT* is the output from the method pattern *FTA*.
- *AnHaz* is input to the process pattern *Hazard Analysis*.

Risks is assigned to *Risks* where:

- *Risks* is the output from the process pattern *Risk Analysis*.
- *Risks* is input to the process pattern *Establish System Safety Requirements*.

Risks is assigned to *Risks* where:

- *Risks* is the output from the process pattern *Risk Analysis*.
- *Risks* is input to the method pattern *SIL Classification*.

FncCat is assigned to *CrCat* where:

- *FncCat* is the output from the method pattern *SIL Classification*.
- *CrCat* is input to the process pattern *Risk Analysis*.

S is required to satisfy the requirements defined by *Req* where:

- *S* is the output from the design pattern *Dual Modular Redundant*.
- *Req* is the output from the patterns *Station Interlocking Requirements*, *Level Crossing Interlocking Requirements, and Establish System Safety Requirements*.

S is assigned to *ToD* where:

- *S* is the output from the design pattern *Dual Modular Redundant*.
- *ToD* is input to the safety case pattern *Overall Safety*.

Case is required to demonstrate safety of *S* where:

- *Case* is the output from the safety case pattern *Overall Safety*.
- *S* is the output from the design pattern *Dual Modular Redundant*.

Case is required to detail *TechSaf* where:

- *Case* is the output from the safety case pattern *Technical Safety.*
- *TechSaf* is the output from the safety case pattern *Overall Safety*.

Case is required to detail *ERE* where:

- *Case* is the output from the safety case pattern *Safety Requirements Satisfied.*
- *ERE* is the output from the safety case pattern *Technical Safety*.

S-Req is assigned to *Req* where:

- *S-Req* is the output from the process pattern *Establish System Safety Requirements.*
- *Req* is input to the safety case pattern *Safety Requirements Satisfied*.

Two Track Station Interlocking defines the following order for instantiating patterns:

- The pattern *Station Interlocking Requirements* should be instantiated before the pattern *Level Crossing Interlocking Requirements.*
- The pattern *Level Crossing Interlocking Requirements* should be instantiated before the pattern *Hazard Analysis.*
- The patterns *Hazard Analysis* and *FTA* may be instantiated in parallel.
- The pattern *Hazard Analysis* should be instantiated before the pattern *Risk Analysis.*
- The patterns *Risk Analysis* and *SIL Classification* may be instantiated in parallel.
- The pattern *Risk Analysis* should be instantiated before the pattern *Establish System Safety Requirements.*
- The pattern Establish System Safety Requirements should be instantiated before the pattern Dual Modular Redundant.
- The pattern *Dual Modular Redundant* should be instantiated before the pattern *Overall Safety.*
- The pattern *Overall Safety* should be instantiated before the pattern *Technical Safety.*
- The pattern *Technical Safety* should be instantiated before the pattern *Safety Requirements* Satisfied.

10. CONCLUSIONS

The SaCS pattern language has been designed as support for developing conceptual safety designs. The support is offered by a set of patterns acting as guidance on different aspects of conceptual safety design and by a notation for modelling how a combined set of patterns are applied for conceptualisation. The patterns of the language are divided into two main groups, basic patterns and composite patterns. The available basic patterns in SaCS, fully detailed in [4] and [5], are defined in a format that allows the essence of a problem and its solution to be comprehended without any additional explanation. A composite pattern is defined graphically in order to facilitate a simple presentation of a potentially complex combination of patterns. A composite may also specify the application of patterns for documentation purposes.

This report details the syntax of SaCS patterns and further details a structured semantics for the patterns. A pattern in SaCS has input and output parameters and may be instantiated in different contexts. The semantics of a pattern facilitates a common understanding of the pattern and its application. The translation of patterns into their semantics is divided into two steps: step one is to translate a pattern definition into its textual syntax, and step two is to translate a textual syntax into English. The first step is for basic patterns described in Section 3, and for composite pattern described in Section 5 and further detailed in Section 6. The second step is performed by matching an expression defined according to a textual syntax with an appropriate semantic rule to form an understandable text in English. The semantic rules are described for basic patterns in Section 4, and for composite patterns in Section 7. Examples of the systematic translation of several basic patterns and several composite patterns are provided in Section 8 and in Section 9, respectively. We find the translation relative straightforward to perform, meaning that little knowledge beyond the definition of the syntax and the structured semantics of patterns as presented in this report is required in order to perform the translation. The translation procedure should be possible to automate.

The intention of translating patterns into English is to facilitate a common understanding of the meaning patterns in a commonly used spoken and written language. The words and sentences in the semantic rules are words and sentences in English. The rules are defined to facilitate a translation of a textual syntax, fragment by fragment, into an understandable text in English. The fragmented translations are systematically combined into a description of the pattern as a whole. The resulting translations are expressed as a text in English that may be understood.

We have also applied SaCS in two cases, fully detailed in HWR-1029 [4] and HWR-1037 [5], in order to test the applicability of the SaCS method and the SaCS pattern language for conceptual safety design. In both cases the pattern language provided the needed expressiveness in terms of providing a suitable set of patterns and a notation for modelling the application of patterns.

For the reasons mentioned above we argue that the SaCS pattern language facilitates the specification of patterns and their application as guidance for conceptual safety design. Further, we argue that the translation of SaCS patterns into their semantics is straightforward.

11. REFERENCES

[1] H. E. I. Dahl, I. Hogganvik, K. Stølen, Structured Semantics for the CORAS Security Risk Modelling Language, STF07 A970, SNTEF ICT, 2007

[2] GSN Working Group, GSN Community Standard, version 1.0, York, England, 2011.

[3] J. G. Hall, L. Rapanotti, and M. Jackson, Problem Frame Semantics for Software Development, Software and Systems Modeling, Volume 4, Number 2, pp. 189-198, 2005

[4] A. A. Hauge and K. Stølen, A Pattern-based Method for Safe Control Systems – Exemplified Within Nuclear Power Production, HWR-1029, OECD Halden Reactor Project, 2013.

[5] A. A. Hauge and K. Stølen, A Pattern-based Method for Safe Control Systems – Exemplified Within Railway Signalling, HWR-1037, OECD Halden Reactor Project, 2013.

[6] International Electrotechnical Commission (IEC), Fault Tree Analysis (FTA), IEC61025, Edition 2.0, 2006

[7] ISO/IEC 14977:1996(E). Information technology – Syntactic metalanguage – Extended BNF, first edition, 1996.

[8] M. Jackson, Problem Frames: Analysing and Structuring Software Development Problems, Addison-Wesley Longman Publishing Co., 2001.

[9] T. Kelly and R. Weaver, The Goal Structuring Notation – A Safety Argument Notation, In *Proceedings of the dependable systems and networks 2004 workshop on assurance cases*, 2004

[10] M. S. Lund, A. Refsdal, and K. Stølen, Semantics for UML models for dynamic behaviour: A survey of different approaches. In book titled Model-Based Engineering of Embedded Real-Time Systems, LNCS 6100, pp. 77-103, Springer, 2010.

[11] OMG, Unified Modeling Language: Superstructure, version 2.4.1, OMG Document: formal/2012-05-07, Object Management Group, 2012

Chapter 10

Paper 2: A Pattern-based Method for Safe Control Conceptualisation Exemplified Within Nuclear Power Production

Instead of the paper we have included the full technical report, HWR-1029 rev 2, OECD Halden Reactor Project, Institute for energy technology, Halden, Norway, 2014.

OECD HALDEN REACTOR PROJECT

A Pattern-based Method for Safe Control Conceptualisation – Exemplified Within Nuclear Power Production

by

André A. Hauge[1,2] and Ketil Stølen[2,3]

[1]Institute for energy technology, OECD Halden Reactor Project,

[2] Department of Informatics, University of Oslo, Norway

[3] Department of Networked Systems and Services, SINTEF ICT

andre.hauge@hrp.no, ketil.stolen@sintef.no

2014-06-04

FOREWORD

The experimental operation of the Halden Boiling Water Reactor and associated research programmes are sponsored through an international agreement by:

- the Institutt for energiteknikk (IFE), Norway,
- the Belgian Nuclear Research Centre SCK•CEN, acting also on behalf of other public or private organisations in Belgium,
- the Risø DTU National Laboratory for Sustainable Energy, Technical University of Denmark,
- the Finnish Ministry of Employment and the Economy (TYÖ),
- the Electricité de France (EDF),
- the Gesellschaft für Anlagen- und Reaktorsicherheit (GRS) mbH, representing a German group of companies working in agreement with the German Federal Ministry of Economics and Technology,
- the Japan Nuclear Energy Safety Organization (JNES),
- the Korean Atomic Energy Research Institute (KAERI), acting also on behalf of other public or private organisations in Korea,
- the Spanish Centro de Investigaciones Energéticas, Medioambientales y Tecnológicas (CIEMAT), representing a group of national and industry organisations in Spain,
- the Swedish Radiation Safety Authority (SSM), representing public and privat nuclear organisations in Sweden,
- the Swiss Federal Nuclear Safety Inspectorate ENSI, representing also the Swiss nuclear utilities (Swissnuclear) and the Paul Scherrer Institute,
- the National Nuclear Laboratory (NNL), representing a group of nuclear licensing and industry organisations in the United Kingdom, and
- the United States Nuclear Regulatory Commission (USNRC),

and as associated parties:

- Japan Atomic Energy Agency (JAEA),
- the Central Research Institute of Electric Power Industry (CRIEPI), representing a group of nuclear research and industry organisations in Japan
- the Mitsubishi Nuclear Fuel Co., Ltd. (MNF)
- the Czech Nuclear Research Institute (NRI),
- the French Institut de Radioprotection et de Sûreté Nucléaire (IRSN),
- the Ulba Metallurgical Plant JSC in Kazakhstan,
- the Hungarian Academy of Sciences, KFKI Atomic Energy Research Institute,
- the JSC "TVEL" and NRC "Kurchatov Institute", Russia,
- All-Russian Research Institute for Nuclear Power Plants Operation (VNIIAES), Russia,
- the Slovakian VUJE - Nuclear Power Plant Research Institute, and
- EU JRC Institute for Transuranium Elements, Karlsruhe,

and associated parties from USA:

- the Westinghouse Electric Power Company, LLC (WEC),
- the Electric Power Research Institute (EPRI),
- the Global Nuclear Fuel (GNF) – Americas, LLC and GE-Hitachi Nuclear Energy, LLC, and
- the US Department of Energy (DOE)

The right to utilise information originating from the research work of the Halden Project is limited to persons and undertakings specifically given this right by one of these Project member organisations.

Recipients are invited to use information contained in this report to the discretion normally applied to research and development programmes. Recipients are urged to contact the Project for further and more recent information on programme items of special interest.

Institutt for energiteknikk
OECD HALDEN REACTOR PROJECT

Title

A Pattern-based Method for Safe Control Conceptualisation – Exemplified Within Nuclear Power Production

Author:
André A. Hauge and Ketil Stølen

Document ID:		First issued:
HWR-1029 rev 2		February 2013

Keywords:
conceptual safety design; pattern language; safety

Abstract:

This HWR exemplifies the application of a pattern-based method, called Safe Control Systems (SaCS), on a case from the nuclear domain. The method is supported by a pattern language and provides guidance on development of conceptual safety designs. By a conceptual safety design we mean an early stage specification of system requirements, system design, and safety case for a safety critical system. The SaCS pattern language offers six different kinds of basic patterns as well as operators for combining basic patterns into composite patterns. A composite pattern may be instantiated into a conceptual safety design.

Issue Date:		Name	Signature	Date
2014-06-04	Prepared by:	André A. Hauge	Sign.	2014-04-20
Confidential grade:	Reviewed by:	Vikash Katta	Sign.	2014-05-19
HRP Only	Approved by:	Terje Johnsen	Sign.	2014-06-04

04

MAIL	TELEPHONE	TELEFAX		TELEFAX	
P.O. BOX 173	+47 69 21 22 00	Administration	+ 47 69 21 22 01	IND/OC div.	+ 47 69 21 24 90
NO-1751 HALDEN		Nuclear Safety	+ 47 69 21 22 01	VISIT/RID/COSS div.	+ 47 69 21 24 60
Norway		Purchasing Office	+ 47 69 21 24 40	Reactor Plant	+ 47 69 21 24 70

TABLE OF CONTENTS

1. INTRODUCTION

This report presents a pattern-based method, referred to as Safe Control Systems (SaCS), facilitating development of conceptual safety designs of safety critical systems. By a conceptual safety design we mean an early stage specification of system requirements, system design, and safety case for a safety critical system. A summary of HWR-1029 is published in [11]. Intended users of SaCS are system developers, safety engineers, as well as hardware and software engineers. SaCS has also been applied for the development of a conceptual safety design of an example railway interlocking system, documented in HWR-1037 [12].

The SaCS method interleaves three main activities each of which is divided into sub-activities:

S. *Pattern Selection* – The purpose of this step is to support the conception of a safety design with respect to a given development case by:
   a. Selecting SaCS patterns for requirement elicitation.
   b. Selecting SaCS patterns for establishing design basis.
   c. Selecting SaCS patterns for establishing safety case.

C. *Pattern Composition* – The purpose of this step is to specify the intended use of the selected patterns by:
   a. Specifying compositions of patterns.
   b. Specifying instantiations of patterns.

I. *Pattern Instantiation* – The purpose of this step is to instantiate the composite pattern specification by:
   a. Selecting pattern instantiation order.
   b. Conducting stepwise instantiation.

Established formats for expressing design solutions, e.g., as presented in [1], [3], [5], [8], and [10], do not provide sufficient guidance for their users on whether the solution presented is applicable within a safety critical system context. A safety critical system design may be regarded as being sufficiently safe for its intended purpose only when the necessary evidence providing safety assurance has been established. Safety assurance is achieved by evidence providing confidence in a system being developed according to a suitable process as well as evidence for the system possessing required quality characteristics. The SaCS method has been developed to improve state-of-the-art in the following sense:

- Safety focus: The SaCS method improves state-of-the-art by offering six kinds of basic patterns reflecting the practises used for development of safety critical systems. Patterns are categorised according to two perspectives on how to achieve safety assurance: Process Assurance and Product Assurance. Both perspectives details patterns according to three aspects: Requirement, Solution, and Safety Case. Each basic pattern contains an instantiation rule that may be used for assessing whether a result is an instantiation of a pattern. The patterns support the procurement and demonstration of a safe design concept.
- Pattern composition: The SaCS method improves state-of-the-art by providing a graphical notation for the specification of a pattern composition. The graphical notation is used to explicitly detail the composition and may be used to assess whether a conceptual safety design is an instantiation of a pattern composition.

The SaCS pattern language is inspired by classical pattern literature, e.g., [1], [5], and [8]. The patterns are defined based on safety domain needs as expressed in international safety standards and guidelines, e.g., [14], [16], [17], [18], [19], and [27]. The graphical notation is inspired by languages for system modelling and research within human cognitive processes and visualisation theory, e.g., [6], [25], [23], and [28].

This report describes the application of the SaCS method and its supporting pattern language on an example case from the nuclear domain.

The report is structured as follows: Section 1 introduces the SaCS method and its intended contribution to state-of-the-art. Section 2 outlines our hypothesis and main predictions. Section 3 describes the case on developing a load following reactor control design. Section 4 exemplifies how functional requirements are elicited. Section 5 exemplifies how a design basis is established. Section 6 exemplifies how safety requirements for an initial design are elicited. Section 7 describes the revision of the base design based on inputs from the step on establishing safety requirements. Section 8 exemplifies how a safety case for demonstrating that the design is safe is established. Section 9 exemplifies how fragments of the pattern solution are combined into an overall pattern solution for the case. Section 10 summarises results. Section 11 discusses the fulfilment of success criteria. Section 12 concludes.

Appendix A provides an overview of the SaCS method. Appendix B contains pattern definitions.

2. SUCCESS CRITERIA

The SaCS method intends to support users like system developers, safety engineers, hardware engineers, and software engineers in the early stages of system development. The primary objective is to facilitate development of conceptual safety designs of safety critical systems. A safety critical system [22] represents here a system that upon failure may result in loss of life, significant damage to environment, or unacceptable economic losses. A system may be regarded as sufficiently safe for its intended purpose if it is free form unacceptable failures. In order for stakeholders to take informed decisions upon the further development of a system design concept into its implementation, the safety objectives as well as information on the ability of a system design to enforce safety should also be provided. In order to capture this need, we provide a definition of a conceptual safety design as follows:

Definition *A conceptual safety design is a triple consisting of an early stage specification of:*

- *system requirements (R),*
- *system design (D),*
- *safety case (S).*

An implementation *I* satisfies a conceptual safety design *(R, D, S)* if there exists a safety case *SC* in accordance with *S* that demonstrates that *I* is safe with respect to the specifications of *R* and *D*. Although the objective of a safety case is to argue that an implementation is sufficiently safe to apply in its intended context, the SaCS method only addresses the activities leading to the specification of a conceptual safety design. In the same way as a system design is refined into its implementation, so is the safety case specification refined into a safety case.

The SaCS method provides a pattern-based method supporting the development of conceptual safety designs by systematically building a case specific composite pattern specification that is instantiated in order to provide a conceptual safety design. We define *instantiate* in the context of SaCS as follows:

Definition *A conceptual safety design instantiates a SaCS composite pattern if each element of the triple can be instantiated from the SaCS composite pattern according to the instantiation rules of the individual patterns within the composite and according to the rule for composition.*

In the context of SaCS, safety engineering concepts and best practices are expressed by patterns, e.g., design pattern, requirement pattern, safety case pattern. A result of instantiating a pattern, e.g., a design specification, a requirement specification, or a safety case specification, is the result of applying a pattern in a specific context. Each basic SaCS pattern describes its instantiation rule. The instantiation rule provides guidance on valid instantiations of the pattern. As a conceptual safety design according to the definition is a triple of different and related specifications, the SaCS method must in order to be effective for safety design conceptualisation facilitate the procurement of the triple. We claim that SaCS is effective for this task by facilitating the specification of a composite pattern that is easily instantiated into a conceptual safety design, and define our hypothesis as follows:

H: *The SaCS method facilitates effective and efficient development of conceptual safety designs that are:*

- *In accordance with safety objectives.*
- *At a sufficient level of detail.*
- *Easy to use.*

In order to test our hypothesis, we deduce the following predictions that should hold for the conceptual safety design that is produced when applying SaCS on the load following case described in Section 3:

P: *Application of the SaCS method on the load following case described in Section 3 results in a conceptual safety design that characterises the load following case and is easily instantiated from a composite SaCS pattern. Furthermore, the conceptual safety design:*

a) *Is in accordance with safety objectives – the conceptual safety design is defined in agreement with safety objectives.*

b) *Is at a sufficient level of detail – the conceptual safety design is expressed in a manner that is sufficiently detailed for an early stage specification and may be easily understood.*

c) *Is easy to use – the conceptual safety design may be easily extended, detailed or refined.*

3. THE CASE: LOAD FOLLOWING MODE CONTROL

In many countries, power-producing nuclear reactors are controlled in a base load mode that is at maximum effect. In countries where a large share of the consumed electricity comes from nuclear power plants (NPP), there is a need for controlling the electricity production in accordance with demand that is varying with time, this is called

operating in a load following mode [24]. In France, approximately 75 % of the total electricity production is generated by nuclear power [24]. In order to adjust production according to demand variances, load following mode control is applied.

There are two primary drivers that necessitate load following mode control: (1) the need to manoeuvre NPP power production in line with daily and seasonal variances in power demand; and (2) the need to mitigate power production fluctuation amongst power producing utilities in a power grid. For the latter driver, the intent is to assure that the total power production in the grid meets a given demand and that loss of production due to intermittent power producers (e.g. wind and solar power utilities) or sudden fall in production at a utility may be mitigated.

Load following may be performed differently depending on the type of reactor. In the case of a Pressurised Water Reactor (PWR), sketched out in Figure 1, electricity production is typically controlled using:

- *Control rods*: Control rods are inserted into the core, leading to the control rods absorbing neutrons and thereby reducing the fission process.
- *Coolant as moderator*: Boron acid is added in the primary cooling water, leading to the coolant absorbing neutrons and thereby reducing the fission process.

Control rods may be efficiently used to adjust core reactivity; significant change in effect may be experienced within minutes, as the core will react immediately upon insertion. The process can also be reversed in an equally efficient manner by retracting the control rods. When using boron acid as a moderator, there is a time delay of several hours to reach destined reactivity level. Reversing the process requires filtering out the boron from the moderator, which is a slow and costly process. The concentration of boron acid in the moderator is controlled via the letdown and makeup channels. Water is drained via the letdown channel, and purified water with or without boron acid is supplied via the makeup channel.

When using boron acid in the coolant as moderator, fuel is consumed evenly in the reactor as the coolant circulates in the core. When using the control rods as moderator, the fuel is consumed unevenly in the reactor as the control rods are inserted at specific sections of the core and normally they would not be fully inserted. When fuel rods are replaced, good fuel economy is that the fuel is fully used.

Figure 1 provides a simplified view of the process and the design of the system that controls the process in our case.

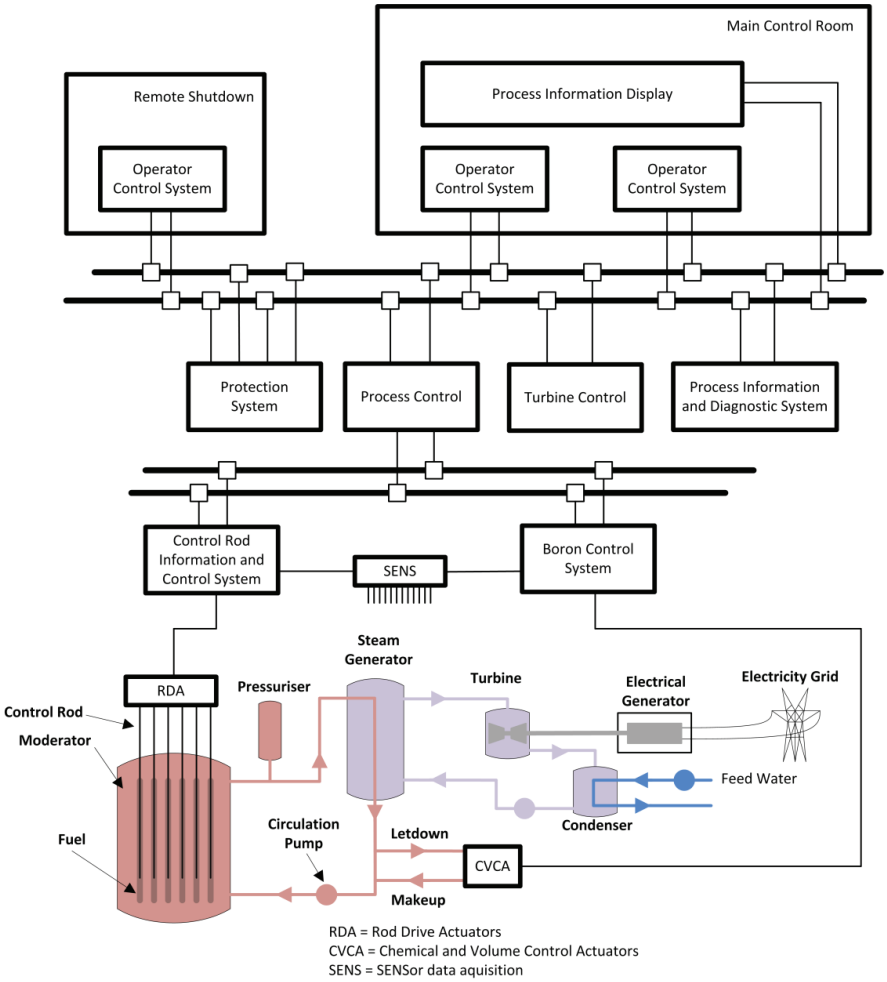


Figure 1 - The main parts of the PWR

The plant process in Figure 1 consists of different controlled loops where pressurised water is heated in the core and circulated in the primary loop. Thermal energy is transferred from the primary loop to a secondary loop in a steam generator. Steam provided by the steam generator drives a turbine. The turbine drives an electrical generator that produces electricity to the grid.

A system (system of systems) interacts with the process in order to control the thermal output of the reactor. We delimit the complexity of the illustration by only including the main systems and by focusing on the two subsystems of the “Process Control” system that are relevant for our case, identified as “Control Rod Information and Control

System" and "Boron Control System". These two systems interact with the plant process by sending actuation signals to the RDA (Rod Drive Actuators) and CVCA (Chemical Volume Control Actuators), respectively.

A successful introduction of an adaptable load following mode control requires satisfying the following goals:

G1 *Produce according to demand*: assure high manoeuvrability so that production may be easily scaled and assure precision by compensating for fuel burn up.

G2 *Cost optimisation:* assure optimal balance of control means (use of control rods versus boron acid to moderate process) with respect to cost (cost associated with the use of control rods versus boron acid to moderate process).

G3 *Fuel utilisation:* assure optimal fuel utilisation.

In the following sections, we will exemplify the application of the SaCS method for deriving an adaptable load following mode control system. The system under development is referred to as ALF (Adaptable Load Following). ALF is intended as an upgrade of an existing NPP control system. The architecture of the existing NPP system is illustrated in Figure 1. Adaptability [7] is introduced as a means to automatically calibrate the controller performing control rod control during operation in order to accommodate fuel burn up. The scope is limited to goal *G1* only.

4. ELICIT FUNCTIONAL REQUIREMENTS

4.1 Pattern Selection

The selection of SaCS basic patterns[1] is performed by the use of the pattern selection map illustrated in Figure 2[2]. Selection starts at the upper left corner and follows the direction of the arrows through the selection map. Pattern selection ends when all selection points have been explored. A diamond represents a choice between alternatives where more than one alternative may be chosen. The patterns with the symbol "*" next to their names in Figure 2 are used in this report. The different icons placed adjacent to pattern identifiers in Figure 2 symbolises different types of patterns. The different pattern types are indicated in Figure 3. The patterns are defined in Appendix B.

[1] SaCS basic patterns are predefined contrary to composite patterns that are user defined.

[2] Not all patterns in Figure 2 are yet available in the SaCS language but are indicated for illustration purpose. The patterns indicated in the map do not represent the final set; the set of basic patterns will be extended.

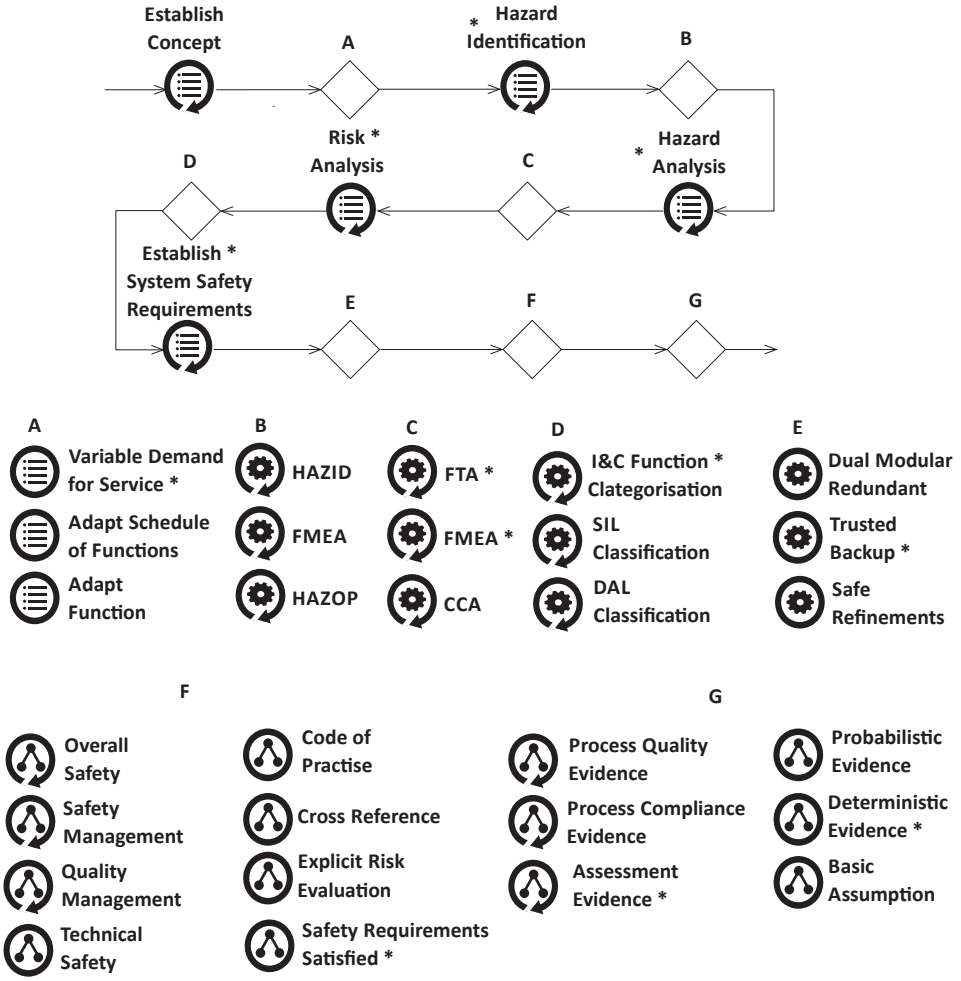


Figure 2 Pattern Selection Map

All the patterns in Figure 2 are known as *Basic Patterns.* Each of these patterns belongs to one of the two groups called *Process Assurance* patterns and *Product Assurance* patterns. Within each of these two groups of patterns we define three kinds of patterns: *Requirement*, *Solution*, and *Safety Case*. For example, the *Solution* type within *Process Assurance* is for patterns on methods supporting the process of developing the product. Whereas the *Solution* type within *Product Assurance* is for patterns on design of the product to be developed. Figure 3 illustrates the different icons that may be used for identifying the classification of a pattern.

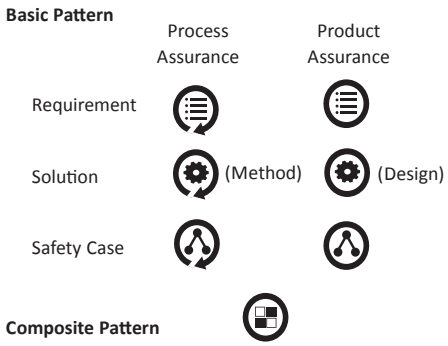


Figure 3 Icons for symbolising different types of patterns

In choice (A) of Figure 2, different product assurance requirement patterns are indicated. We assume in this report that the information provided in Section 2 sufficiently details the development objectives and context such that the pattern *Establish Context* may be passed. The pattern *Variable Demand for Service* in choice (A) captures the problem of specifying requirements for a system that shall adjust control of a NPP that experiences changes in demand for electricity as well as in the power production environment. The pattern is regarded as suitable for elicitation of requirements with respect to the objective defined by goal G1 described in Section 2.

4.2 Pattern Instantiation

The pattern *Variable Demand for Service* referred to in Figure 2 is a product assurance requirement pattern. The pattern is described in appendix B.1, and an excerpt from the pattern is given in Figure 4.

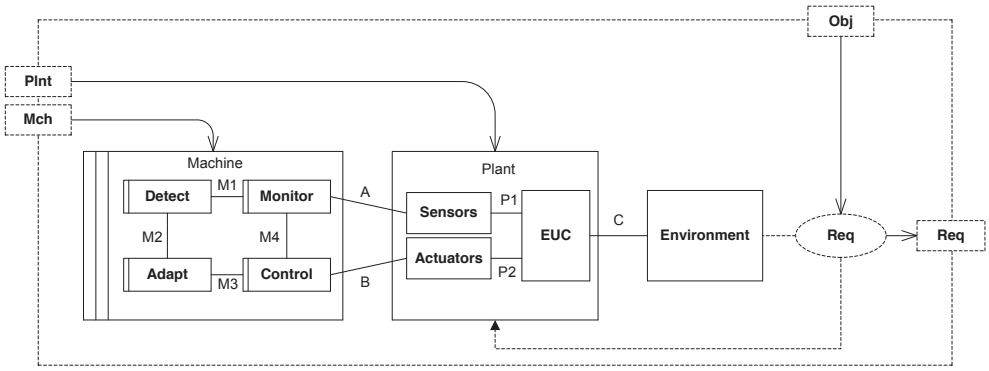


Figure 4 Excerpt - Variable Demand for Service

Figure 4 defines a SaCS adapted version (annotated with the dotted drawn enclosing frame, rectangles with arrows on the edges of the frame indicates input and output parameters) of the problem frames notation [20]. It provides the requirements analyst with a means for elaborating upon the problem of change in a NPP production environment in order to derive requirements (represented by *Req*) for a system under

construction (represented by the element *Machine*) that shall control a plant (represented by *Plant*) such that a given objective (represented by *Obj*) is fulfilled.

When *Variable Demand for Service* is instantiated, the *Req* parameter indicated in Figure 4 is instantiated with respect to the context given by the instantiation of the parameters *Obj* and *Plnt*. In Section 4.1, we selected the *Variable Demand for Service* pattern as support for eliciting requirements for a PWR system upgrade with respect to goal G1. Therefore, the parameter *Obj* is instantiated in the context of the defined goal identified by G1, and the parameters *Plnt* and *Mch* are instantiated in the context of the specification of the PWR system (see Figure 1) that represents the context of the ALF system upgrade. When instantiated, the parameter *Plnt* represents the nuclear power production process and the equipment that is controlled (e.g., the reactor, pressuriser, steam turbine, and generator) and *Mch* represents the instrumentation and control system that performs control. It is the *Process Control* system that performs control and the ALF system is intended to be a part of *Process Control*.

The *Variable Demand for Service* pattern described in appendix B.1 includes the definition of a set of abstract requirements. The abstract requirements are identified in the pattern description on the form “R.n” where n is a running number. We derive requirements for the ALF system by the use of the *Variable Demand for Service* pattern where the outputs of pattern instantiation are context specific requirements on the form “FR.n” where n is a running number. The outcome of pattern instantiation is collected in Table 1 where the column with heading “Req. ID” contains unique identifiers for each derived requirement, the column with heading “Ref. ID” contains a reference to the abstract requirement defined in *Variable Demand for Service* that served as its basis, and the column with heading “Requirement” contains the requirement text.

Table 1 Requirements for Variable Demand for Service

Req. ID	Ref. ID	Requirement
FR.1	R.1	ALF system shall monitor the demand for electricity required by the grid by acquiring the GDfE (Grid Demand for Electricity) signal from the PCBUS (Process Control BUS).
FR.2	R.1	ALF system shall monitor the reactor power output by acquiring the RPM (Reactor Power Measured) signal from SENS (SENsor data acquisition).
FR.3	R.1	ALF system shall monitor the control rod position by acquiring the CRP (Control Rod Pattern) signal from SENS (SENsor data acquisition).
FR.4	R.1	ALF system shall monitor the boron acid concentration by acquiring the BC (Boron Concentration) signal from SENS (SENsor data acquisition).
FR.5	R.2	ALF system shall acquire GDfE signal by polling the PCBUS.
FR.6	R.2	ALF system shall acquire validated RPM, CRP, BC signals by request to

		SENS.
FR.7	R.3	ALF system shall poll for GDfE signal at a rate 10 times per second.
FR.8	R.5	ALF system shall maintain a history over GDfE, CRP, BC and RPM.
FR.9	R.5	ALF system shall calculate a value DEVRP (DEViation from Required Power) that shall indicate non-optimal reactor control when operating in load following mode.
FR.10	R.6	ALF system shall calculate an indicator CALCI (CALibrate Control rod pattern Indicator) that shall identify the need for calibration of control rod patterns by the use of the history of DEVRP, CRP and BC.
FR.11	R.9	ALF system shall provide a mechanism for automatic calibration of control rod patterns based on CALCI indicator.
FR.12	R.10	ALF system shall activate calibration of the control rod patterns when the need to calibrate is indicated by the CALCI indicator.
FR.13	R.13	ALF system shall provide reactor control by providing actuation signals to RDA and CVCA.

4.3 Pattern Composition

Figure 5 illustrates a composite pattern named *Functional Requirements* defined according to the syntax of SaCS [13]. The composite contains only one pattern that is named *Variable Demand for Service*.

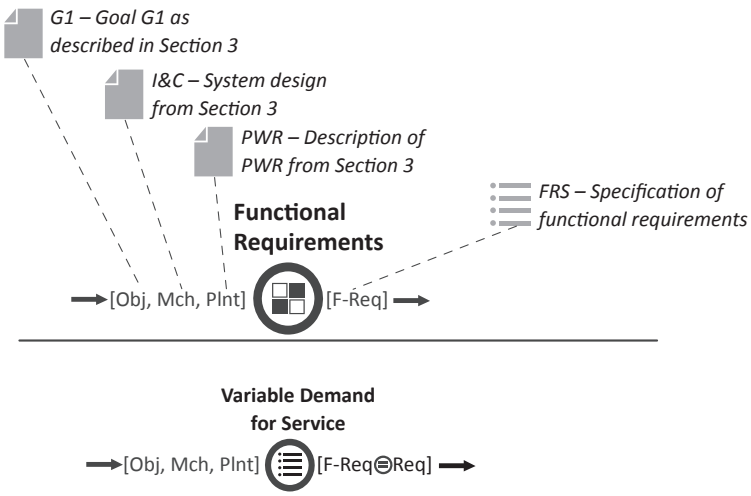


Figure 5 Definition of the composite "Functional Requirements"

A composite pattern describes an intended use of a set of patterns (containing at least one pattern). A specific pattern is referred to by the use of a *pattern reference*. A

pattern reference is illustrated by placing an identifier of a pattern that is referenced adjacent to an icon identifying the type of the pattern.

Figure 5 specifies a composite pattern named *Functional Requirements*. Input parameters to the composite are *Obj, Mch*, and *Plnt*. Output parameter of the composite is *F-Req*. The horizontal line below the composite pattern icon demarks the end of the declaration of the composite, and everything beneath the line defines the content of the composite. The composite contains one pattern named *Variable Demand for Service.* The input and output parameters of the composite is matched to the parameters that are defined as inputs (symbolised with a thick black arrow pointing towards a list of parameters) and outputs (symbolised with a thick black arrow pointing away from a list of parameters) in the content part of the composite pattern by a matching of parameter identifiers. An alias *F-Req* is defined for the parameter *Req* of the pattern *Variable Demand for Service*. Any instantiation of the input parameters to the composite implies that the correspondingly named input parameters of the contained patterns, here *Variable Demand for Service,* are similarly instantiated. Any instantiation of an output parameter of a contained pattern implies that the correspondingly named output parameter of the composite is similarly instantiated. An instantiation of a parameter is symbolised by drawing a dotted line between an artefact reference (consisting of an icon for symbolising the type of artefact and an identifier for providing a reference to a specific artefact) and a parameter.

5. ESTABLISH DESIGN BASIS

5.1 Pattern Selection

In choice (E) of Figure 2, a set of alternative design patterns can be selected. We go from eliciting functional requirements in Section 4, to considering design alternatives that may fulfil the required function here as an alternative to strictly following the order that is depicted in Figure 2, i.e., from choice (A) to choice (E) instead of from choice (A) to *Hazard Identification* to choice (B) and so on. All design patterns in choice (E) describes adaptable control concepts. The patterns differ in how adaptable control is approached and how negative effects due to potential erroneous adaptation are mitigated.

The *Trusted Backup* pattern, referred to in Figure 2 and defined in Appendix B.5, describes a system concept where an adaptable controller may operate freely in a delimited operational state space. Safety is assured by a redundant non-adaptable controller that operates in a broader state space and in parallel with the adaptable controller. A control delegator grants control privileges to the most suitable controller at any given time based on switching rules and information from safety monitoring.

The *Trusted Backup* is selected as a design basis for the ALF system based on an evaluation of the strengths and weaknesses of the different design patterns with respect to their ability to fulfil functional requirements, i.e., *FR.1* to *FR.13,* identified in Section 4.

5.2 Pattern Instantiation

Figure 6 is an excerpt from the *Trusted Backup* pattern. A part of the definition of the pattern is a UML [25] diagram and this is reproduced in Figure 6. The pattern definition also includes textual descriptions expressing the expected interactions between system parts. Requirements may be associated with the system (denoted S for short in Figure 6) or any part (denoted AC, AL, CL, TC, M, CD for short) described by the pattern. When the *Trusted Backup* pattern was instantiated, the requirements defined in Section 4 were associated with the component denoted as S in the pattern description.

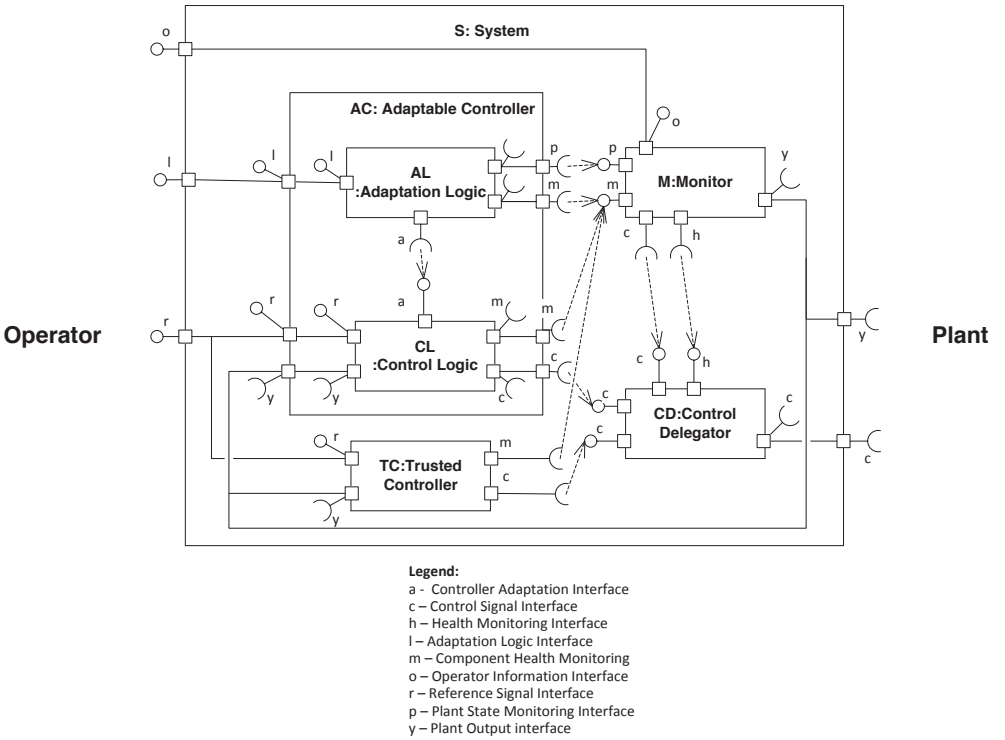


Figure 6 Excerpt – Trusted Backup

The instantiation of *Trusted Backup* resulted in a design specification identified as *ALF Dgn*. The *ALF Dgn* specification is here represented by the system illustrated in Figure 7, Figure 8 and Figure 9 as well as the belonging textual descriptions. The *ALF Dgn* design is an interpretation of the *Trusted Backup* pattern in the context described in Section 3.

The following is a mapping between parts of the *ALF Dgn* and parts described in the *Trusted Backup* pattern:

- *ALF* in Figure 7 is an instance of *S* in Figure 6.
- *ACRAC* in Figure 8 is an instance of *AC* in Figure 6.
- *CCRAC* in Figure 8 is an instance of *TC* in Figure 6.
- *CRA* in Figure 8 is an instance of *CD* in Figure 6.

- *CRC* in Figure 8 is intended to contain a part providing the monitoring functionality as represented by the *M* part of Figure 6 in addition to other functionality.
- The *iBUS* interface in Figure 7 represents the interface towards the reactor controller and relevant reactor process data identified as the interfaces *o, l and r* in Figure 6.
- The interfaces *iRDA* and *iCVCA* in Figure 7 represents the interface to plant actuators identified as interface *c* in Figure 6.
- The *iSENS* interface in Figure 7 represents the interface to plant sensors identified as interface *y* in Figure 6.

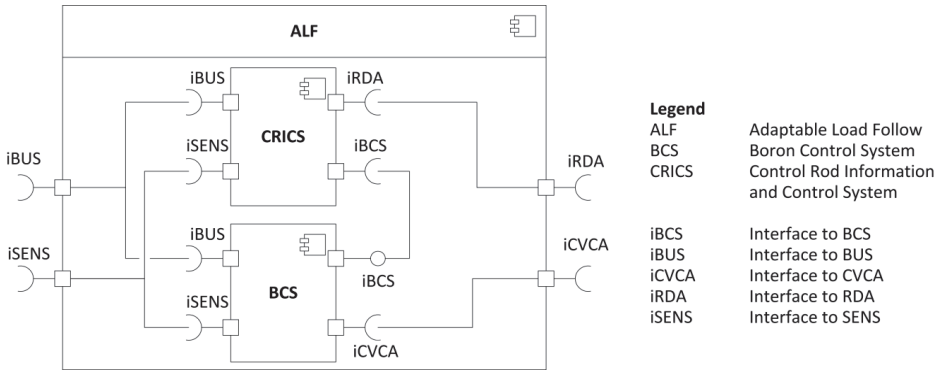


Figure 7 The ALF Design

The *ALF* system illustrated in Figure 7 consists of two interacting subsystems, *CRICS* (Control Rod Information and Control System) and *BCS* (Boron Control System). The *CRICS* system is detailed in Figure 8, and the *BCS* system is detailed in Figure 9.

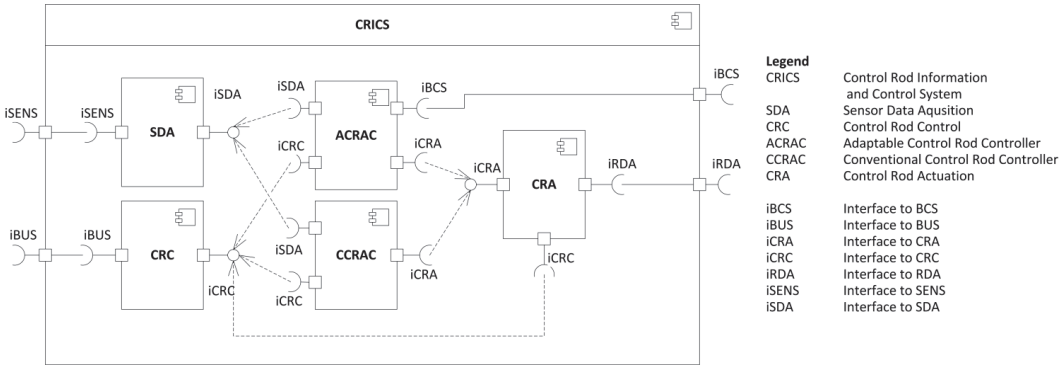


Figure 8 The CRICS system

The *CRICS* system illustrated in Figure 8 incorporates adaptability in the part identified as *ACRAC*. As stated in [24], fuel burn up has significant influence on the manoeuvrability of the reactor such that a higher boron acid concentration is required in the beginning of the fuel cycle than later in order to moderate reactivity. The concentration of boron acid is decreased with time. At the end of the fuel cycle the boron concentration is almost zero and the control rods are in the upper position, thus

the manoeuvrability of the reactor decreases. The *ACRAC* system is intended to automatically adjust and calibrate the control rod insertion patterns in the core and compensate for fuel burn up. The *ACRAC* system is also intended to account for the decreased need for boron acid in the moderator as the fuel reaches the end of the fuel cycle.

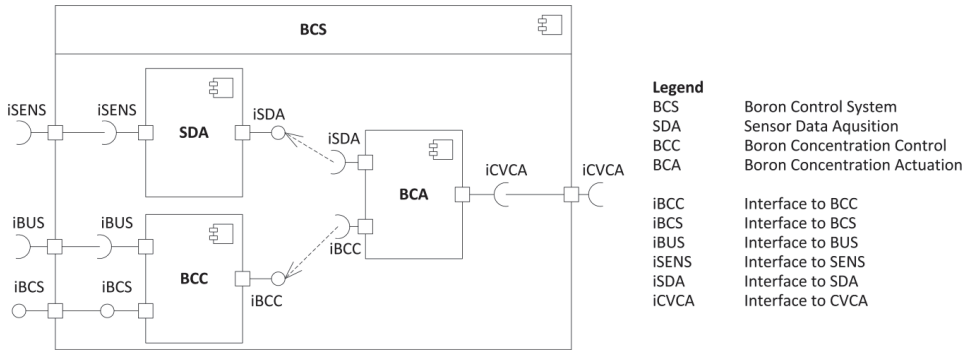


Figure 9 The BCS system

The *CRICS* system illustrated in Figure 8 is intended to work in the following manner:

- *CRC* monitors the process control bus at a frequency of 10 times per second through the interface *iBUS* for commands from plant operator or other systems that affect control rod control. *CRC* also sends control status information to the process control bus. Signals that are handled by *CRC* are:
  - *AcBL, DeAcBL* – activate or deactivate base-load generation (constant power).
  - *AcPFC, DeAcPFC* – activate or deactivate primary frequency control (short term or immediate change, seconds scale, of electricity generation to variation in demand detected by variations in frequency in the grid).
  - *AcSFC, DeAcSFC* – activate or deactivate secondary frequency control (adaptation of electricity generation over a longer time frame, minutes scale, by adapting electricity generation to a frequency deviation over a time period).
  - *AcLF, DeAcLF* – activate or deactivate load following (e.g., a variable load programme with one or several power changes over a period of 24 hours).
  - *AcAC, DeAcAC* - activate of deactivate adaptable control. When an *AcAC* or *DeACAC* is retrived, *CRC* forwards the signal to *CRA.*
  - *GDfE* – provides information on the grid demand for electricity.
- *SDA* is responsible for gathering validated sensor signals (through the interface *iSENS*). Signals that are acquired by *SDA* are:
  - *RPM* – provides a measure of reactor power.
  - *CRP* – provides data on control rod position in the core.
- *ACRAC* is responsible for providing control rod control and shall automatically calibrate control rod actuation patterns in order to compensate for fuel degradation. The *ACRAC* system shall:
  - Retrieve from CRC and SDA the signals GDfE, CRP and RPM.
  - Provide control rod actuation signals to CRA.

  - Calculate a value DEVRP that shall be used as an indicator for non-optimal reactor control when operating in load following mode.
  - Calculate a value CALCI that shall indicate if calibration of control rod patterns is required.
  - Calibrate control rod control patterns automatically when the value of CALCI gives a clear indication on suboptimal control.
  - Interact with the *BCC* system in order to request specific Boron acid concentration levels and thus seek an optimal balance of control rods and boron acid moderation of the process.
- *CCRAC* is responsible for providing control rod control. The *CCRAC* system shall:
  - Retrieve from *CRC* and *SDA* the signals *GDfE, CRP and RPM*.
  - Provide control rod actuation signals to *CRA*.
- *CRA* is responsible for deciding upon which of the two systems, *CCRAC* or *ACRAC*, that will perform control rod actuation by passing forward actuation signals from one of them to *RDA*. *CCRAC* and *ACRAC* are two diverse systems for control rod control where the former is assumed here to be an existing control rod control system and the latter is the adaptable control rod system that is part of the upgrade. Both systems operate in parallel and provide actuation signals to *RDA*. The *RDA* system is assumed to consist of the drive motors and other mechanisms for inserting and retracting control rods. When an *AcAC* signal is received from *CRC* then actuation signals from *ACRAC* will be allowed and preferred for actuation of *RDA*. When a *DeAcAC* signal is received from *CRC* only actuation signals from *CCRAC* actuation will be allowed for actuation of *RDA*.

The *BCS* system illustrated in Figure 9 is intended to work in the following manner:

- *BCC* monitors the process control bus at a frequency of 10 times per second through the interface *iBUS* for commands from plant operator or other systems that affect boron acid concentration control. *BCC* also sends control status information to the process control bus. Signals that are handled by *BCC* are:
  - *acracBC* – informs on the boron concentration level required by the *ACRAC* system.
  - opBC – informs on the boron concentration level required by an operator.
- *SDA* is responsible for gathering and validating sensor signal on boron concentration in the primary loop.
- *BCA* is responsible for providing actuation signals to the different subsystems of the *CVCA* such that a required boron concentration level required to be met may be achieved. The *CVCA* system is here assumed to consist of valves, piping, boron acid tank, water tanks, and all appliances necessary to supply water via the makeup channel and allow draining of water via the letdown channel.

The requirements *FR.1* to *FR.13* are regarded as accounted for by the specification of the *ALF* system and its different subsystems as given here.

5.3 Pattern Composition

Figure 10 illustrates a composite pattern named *Design*. The composite contains only one pattern that is named *Trusted Backup*. The description of the ALF system presented in Section 5.2 is here assumed to represent the instantiation of the

parameter *S* of the pattern *Trusted Backup.* The instantiation of *S* is symbolised by drawing a dotted line between the parameter *S* and the associated artefact reference (consisting of an icon for symbolising the type of artefact, here a design artefact, and an identifier that provides a reference to the artefact).

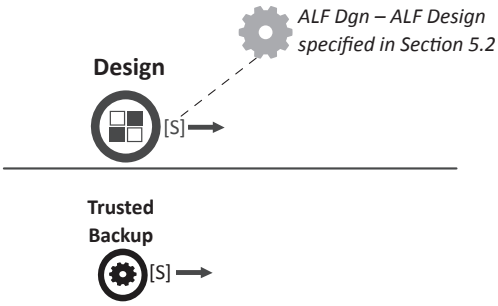


Figure 10 Design – Composite Pattern

Figure 11 illustrates a composite pattern named *Intermediate Solution #1*. *Intermediate Solution #1* is here a dummy used for explaining the intended use of the composite named *Design* together with the composite *Functional Requirements* that was defined in Section 4.3.

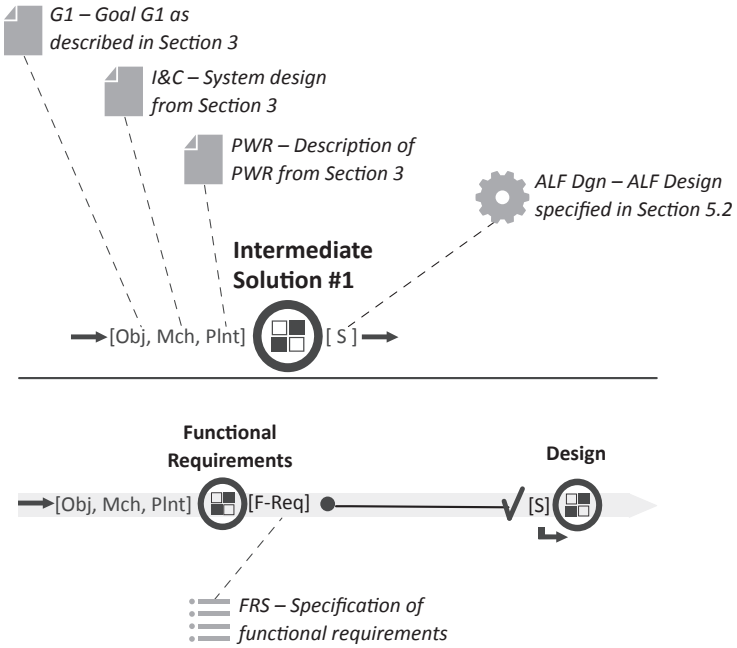


Figure 11 Fragment showing use of "Design" composite

The reference to a parameter *S* of the pattern named *Design* in Figure 11 is a reference to the parameter *S* of the pattern *Trusted Backup* as the *Design* composite (defined in Figure 10) contains *Trusted Backup* and because the parameter *S* of *Design* is matched to the parameter *S* of *Trusted Backup*.

In Figure 11, a *satisfies* relation (illustrated with a bullet connected to a checkmark with a solid line) is used to associate requirements (denoted F-Req) with a description of a system (denoted *S*). The *satisfies* relation indicates that any instantiation of *S* (that is a parameter of the pattern named *Design)* shall satisfy any instantiation of F-Req (that is a parameter of the composite pattern *Functional Requirements*). The composite named *Functional Requirements* is detailed in Figure 5.

6. ELICIT SAFETY REQUIREMENTS

6.1 Pattern Selection

Once a design is selected in choice (E) of Figure 2, traversal of the pattern selection map is restarted from the pattern *Hazard Identification* as the pattern that has to be selected prior to *Hazard Identification* is already considered in Section 4.1. The pattern *Hazard Identification* defines the process of identifying potential hazards and requirements for documenting the process results. We select the pattern as a means for identifying hazards associated with the ALF system.

In choice (B) of Figure 2, a set of method patterns supporting hazard identification is provided. The *FMEA* pattern is selected under the assumption that a Failure Modes and Effects Analysis (FMEA) is suitable for identifying potential failure modes of the ALF system and the hazards associates with these failure modes.

Once a hazard identification method is selected, further traversal of Figure 2 leads to the pattern *Hazard Analysis*, we select this pattern as support as it provides guidance on the process of deriving potential causes of hazards.

In choice (C) of Figure 2, process solution patterns supporting hazard analysis may be selected. The *FTA* pattern is selected as support for *Hazard Analysis* under the assumption that a top-down Fault Tree Analysis (FTA) assessment is a suitable complement to the bottom-up assessment provided by FMEA.

Traversal of Figure 2 from choice (C) leads to the pattern *Risk Analysis*. The pattern provides guidance on how to address identified hazards with respect to their potential severity and likelihood and how to establish a notion of risk. We select the pattern as a support for establishing risks associated with the ALF system.

In choice (D) of Figure 2, domain specific patterns capturing different methods for criticality classification are indicated. The *I&C Functions Classification* is selected as the ALF system is developed within a nuclear context and the pattern provides guidance on classification of nuclear Instrumentation and Control (I&C) systems.

Traversal of Figure 2 from choice (D) leads to the pattern *Establish System Safety Requirements*. The pattern describes the process of eliciting requirements on the basis of identified risks. We select this pattern as support for establishing safety requirements for the ALF system.

6.2 Pattern Instantiation

Safety requirements are defined on the basis of risk assessment. The process requirement patterns selected in Section 6.1 support a risk-based process of eliciting safety requirements and may be applied sequentially according to the following order:

1. *Hazard Identification*: is detailed in appendix B.7 and is applied as guidance on the process of identifying hazards. The *FMEA* pattern described in appendix B.2 is used as support on hazard identification by a method of analysing failure modes.
2. Hazard Analysis: is detailed in appendix B.8 and is applied as guidance on the process of identifying the potential causes of hazards. The *FTA* pattern described in appendix B.3 is used as support for hazard analysis by a method of constructing fault trees.
3. *Risk Analysis*: is detailed in appendix B.9 and is applied as guidance on the process of combining information on hazards, the likely severity of hazards as well as the likelihood of hazards occurring into a notion of risk. The *I&C Function Categorisation* pattern described in appendix B.4 is used as support for risk classification.
4. *Establish System Safety Requirements*: is detailed in appendix B.10 and is applied as guidance on the process of defining safety requirements on the basis of identified risks.

Assume that the *Hazard Identification* pattern is applied on the load following case such that the parameter:

- *ToA* is associated with *ALF Dgn* design (see Section 5).
- *IdHz* is represented by the table produced by applying the *FMEA* pattern (see Table 2 below).
- *HzLg* represents the hazard log provided when the pattern is instantiated.

Table 2 represents the result of applying a high level *FMEA* assessment on the *ACRAC* part of the *ALF* system. We assume that *FMEA* is applied on all parts of *ALF* but delimit our scope to the *ACRAC* part.

Table 2 ALF FMEA

ID	Item	Func.	Failure Mode	Failure Cause	Local Effect (ALF)	System Effect (PWR)
FM.1	ACRAC	Provides control rod control functionality and performs self-calibration of control rod patterns	Fails to provide control rod actuation	Adaptation mechanism interferes with control function. Erroneously adapted control function	No signal from ACRAC, detected by CRA, mitigated by CCRAC	None
FM.2			Provides	Erroneously	ACRAC	Increase in

			wrong control rod actuation	adapted control function	commands erroneously no control rod moderation	reactor power level towards an upper bound
FM.3			Provides sub-optimal control rod actuation – to high moderation	Erroneously adapted control function	ACRAC commands control rods inserted more than needed	Power output lower than power demand
FM.4			Provides sub-optimal control rod actuation – to low moderation	Erroneously adapted control function	ACRAC commands control rods inserted less than needed	Power output higher than power demand
FM.5			Fails to self-calibrate	Adaptation mechanism failure	ACRAC control rod patterns is unchanged	Decreased precision in power output moderation

We assume that the system effect associated with failure mode *FM.2* in Table 2 requires a further assessment. We regard the possible system effects of the other failure modes as tolerable and thus a more detailed assessment is at this stage of development not required. The hazard log *HzLg* will contain an overview of all identified hazards associated with the system. In the load following case we assume that there exists one hazard related to the effect of failure mode *FM.2* that are defined as *"H.1 Reactor power level more than upper bound"* documented in a manner like illustrated in Table 3.

Table 3 Hazard Log (Version 1)

ID	Description	Cause	Reference	Mitigation
H.1	Reactor power level more than upper bound	Erroneously adapted control function	ALF FMEA, FM.2	

Assume that the *Hazard Analysis* pattern is applied on the load following case such that the parameter:

- *ToA* is associated with *ALF Dgn* (see Section 5).
- *Haz* is associated with the hazard *H.1* as given in Table 3.
- *IdCsHz* is represented by the fault tree provided by applying the *FTA* pattern.
- *HzLg* represents the hazard log produced when the *Hazard Analysis* pattern is instantiated.

Table 4 represents a tabular representation of a fault tree resulting from a high level *FTA* assessment of the system. The hazard *H.1* has several potential causes, we only provide details with respect to the potential causes arising from errors within the *ACRAC* system, other potential causes are not detailed and are marked as out of scope.

Table 4 ALF FTA

<table>
<tr><td colspan="6">Hazard H.1: Reactor power level more than upper bound</td></tr>
<tr><td colspan="6">AND</td></tr>
<tr><td colspan="4">C.1: Process Control system fails to keep power level within bounds
AND</td><td>C2: Operator fails to manually keep power level within bounds</td><td>C.3: Protection system fails to perform reactor trip upon power level equal or more than upper bound</td></tr>
<tr><td colspan="3">C.1.1: Erroneously adapted ACRAC control function
OR</td><td>C.1.2: Erroneous CCRAC control function</td><td>C.1.3: Erroneous CRA granting of control privileges</td><td rowspan="2">Out of scope</td><td rowspan="2">Out of scope</td></tr>
<tr><td>C.1.1.1: Erroneous adaptation algorithm</td><td>C.1.1.2: Erroneous validation of correctness of control function adaptation</td><td>C.1.1.3: Erroneous configuration of control function with updated parameters</td><td>Out of scope</td><td>Out of scope</td></tr>
</table>

Table 4 identifies three potential failures that can lead to an erroneously adapted *ACRAC* control function, identified as C.1.1.1, C.1.1.2 and C.1.1.3. We assume that the hazard log produced when instantiating the *Hazard Analysis* pattern is used to maintain an overview of all hazards and their potential causes. Table 5 represents the updated hazard log.

Table 5 Hazard Log (Version 2)

ID	Description	Cause	Reference	Mitigation
H.1	Reactor power level more than upper bound	Erroneously adapted control function	ALF FMEA - FM.2 ALF FTA – C.1.1.1, C1.1.2, C.1.1.3	
		Operator fails to manually keep power level	ALF FTA – C.2	
		Protection system fails to perform reactor trip upon power level	ALF FTA – C.3	

Assume that the *Risk Analysis* pattern is applied on the load following case such that the parameter:

- *ToA* is associated with ALF Dgn design (see Section 5).
- *Haz* is represented by the hazard log produced when instantiating the *Hazard Analysis* pattern (see Table 5).
- *ClsSev* and *ClsLi* are not used as the notion of risk in this case is guided by the *I&C Functions Categorisation* pattern that provides guidance on the categorisation of nuclear power plants instrumentation and control systems according to the importance of their functions on plant safety.
- *ClsCr* is represented by the *I&C Functions Categorisation pattern.*
- *Risks* is represented by the documentation provided when the pattern is instantiated and contain an overview of all system hazards, potential contribution of subsystem failure to identified hazards, classification of subsystems, and possible mitigations.

We assume that the *ALF* system is addressed with respect to the criticality of the function it provides on overall system safety by the use of the *I&C Function Categorisation* pattern, categorising *ALF* as a category *C* system. We assume that the plant operators and the protection system are the primary means for assuring plant safety. Although the plant operators and the protection system represent barriers that may trip the reactor in the event of an adaptation failure within the *ACRAC* part of the *ALF* system, an unplanned reactor trip is an event that is surely unwanted. Thus, mitigations for avoiding such events are needed. We assume that among a set of possible mitigations to avoid erroneously adaptation of the *ACRAC* function, the most promising risk reducing measures are to: only allow incremental adaptation of the adaptable control function; disable the adaptable control function from performing

control while it is updated; and validate every proposed adaptation before the controller is updated. Table 6 represents the hazard log updated with possible mitigations.

Table 6 Hazard Log (Version 3)

Id	***Description***	***Cause***	***Reference***	***Mitigations***
H.1	*Reactor power level more than upper bound*	Erroneously adapted control function	*ALF FMEA - FM.2* *ALF FTA – C.1.1.1, C1.1.2, C.1.1.3*	Only allow incremental adaptation of the adaptable control function. Disable the adaptable control function from performing control while it is updated. Validate every proposed adaptation before effectuating the update.

Figure 12 represents an excerpt of the *Establish System Safety Requirements* pattern and describes a SaCS adapted version (adapted by a dotted drawn frame with dotted drawn rectangles on the edges representing input or output parameters) of a UML [25] activity diagrams. Figure 12 and the other elements of the pattern description provide the analyst with a means for elaborating upon the problem of establishing safety requirements based on inputs on the risks associated with a given target.

Assume that *Establish System Safety Requirements* is instantiated such that:

- *Risks* is associated with the risk of hazard *H.1* due to erroneous adapted control function (see Table 6).
- *ToA* is associated with the *ALF Dgn* design (see Section 5).
- *Reg* is not used here in order to reduce the exemplification of SaCS. The parameter could be associated with the standard IEC61513 [15] and IEC62138 [17]. The standard IEC61513 provides general requirements for systems important to safety and the standard IEC62138 provides guidance on software aspects for systems performing category C function.
- *Req* represents the requirements set provided when the pattern is instantiated.

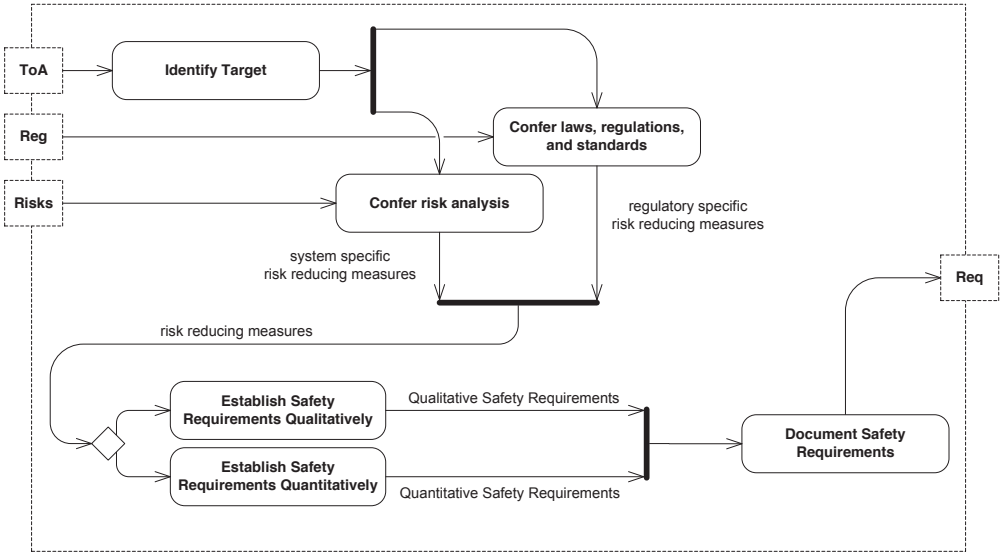


Figure 12 Excerpt – Establish System Safety Requirements

We assume that the instantiation of the *Establish System Safety* Requirements pattern produces a set of safety requirements contained in Table 7. Furthermore, we assume that satisfying these requirements is regarded as providing a sufficiently safe system.

Table 7 ALF Safety Requirements

Req	Reference	Description
SR.1	HazLog, H.1 ALF FTA, C1.1.1	ALF shall adapt the adaptable control function in increments.
SR.2	HazLog, H.1 ALF FTA, C1.1.2	ALF shall evaluate each increment of the adaptable control function for potential erroneous adaptations of the adaptable control function.
SR.3	HazLog, H.1 ALF FTA, C1.1.3	ALF shall disable adaptive control during the time period when the adaptable controller is modified.
SR.4	HazLog, H.1 ALF FTA, C1.1.3	ALF shall assure that configured parameters are correctly modified before enabling adaptable control.
SR.5	HazLog, H.1 ALF FTA, C1.1.2	The System shall assure no negative effects on system safety in the event of erroneous adaptation of the adaptable controller part of the ALF subsystem.

6.3 Pattern Composition

Figure 13 illustrates a composite pattern named *Safety Requirements*. The composite *Safety Requirements* illustrated in Figure 13 is defined such that the intended instantiation order of the patterns as presented in Section 6.2 is indicated. The arrows that connect different patterns (e.g., the arrow pointing from the parameter *HzLg* of the pattern *Hazard Identification* towards the parameter *Haz* of the pattern *Hazard Analysis*) symbolise *assigns* relations. The direction of the arrow indicates that a parameter associated with a source pattern must be instantiated before it may be used as an input to the instantiation of a parameter associated with a target pattern.

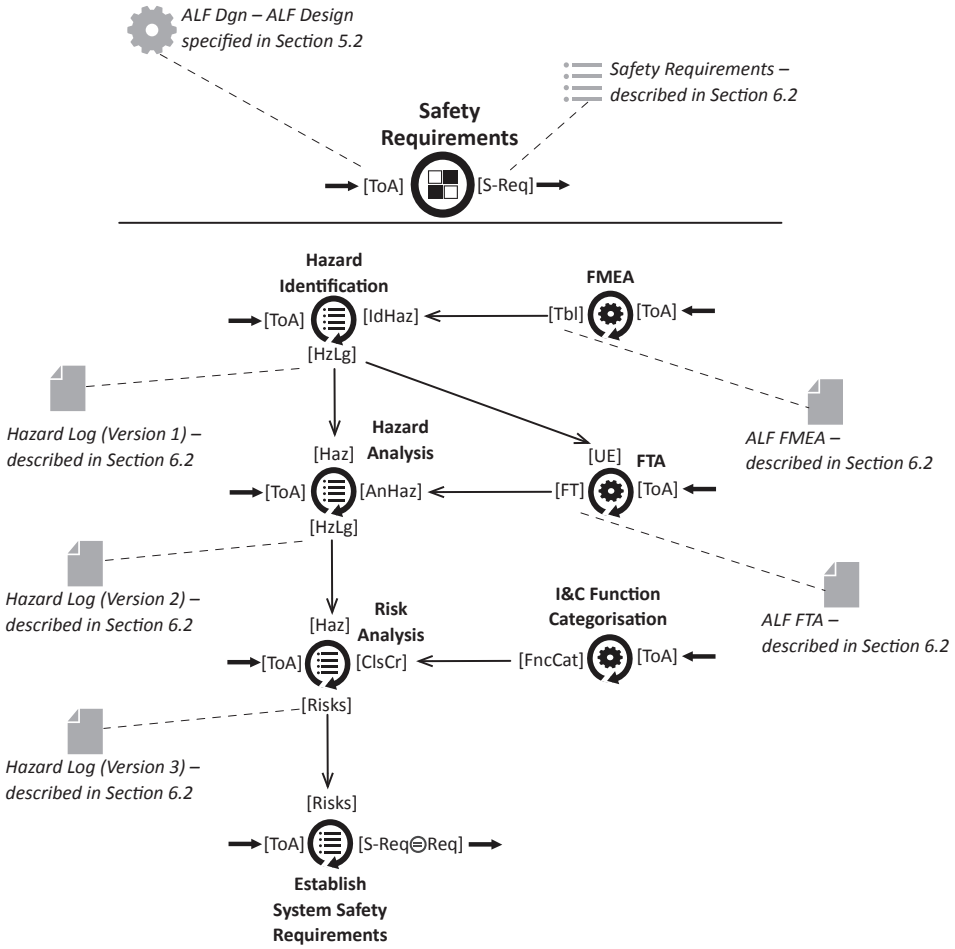


Figure 13 Fragment showing use of "Safety Requirements" composite

In Figure 13 we assume that the instantiation of the pattern *Hazard Identification* produces an output represented by the hazards described in Section 6.2. This is indicated in Figure 13 by connecting an artefact reference with the parameter it represents. An artefact reference is symbolised with a grey icon and an identifier for the artefact. The text "*Hazard Log (Version 1) – described in Section 6.2"* represents our identifier and is placed adjacent to an icon representing the type of artefact (here a documentation artefact). The artefact reference is connected to the parameter it instantiates with a dotted line. The dotted line represents an *instantiates* relation.

The input parameter *ToA* of *Hazard Analysis* is not connected to any pattern, the instantiation of this parameter is implied by the instantiation of the correspondingly named input parameter to the composite. According to the syntax of SaCS, every pattern contained in the composite *Safety Requirements* that has an input parameter with public accessibility (thick arrow) that is named *ToA* is automatically assigned to the instantiation of the input parameter *ToA* of the composite *Safety Requirements.* In a similar manner, the instantiation of the output parameter *S-Req* (defined as an alias for

the parameter *Req*) of the pattern *Establish System Safety Requirements* is an instantiation of the output parameter *S-Req* of the pattern *Safety Requirements.*

The *FMEA* method pattern is a support for the process requirement pattern *Hazard Identification*. The *FMEA* pattern describes a method for identification of hazards, and the artefact that is produced upon pattern instantiation is a table (represented by *Tbl*) intended to contain the results of the assessment described by the pattern. The *Hazard Identification* pattern may take identified hazards as input (represented by *IdHaz*). The *assigns* relation here combines the *FMEA* pattern and the *Hazard Identification* pattern. The *assigns* relation matches any instantiation of the parameter *Tbl* to *IdHaz*.

The pattern *I&C Functions Categorisation* reflects the method for risk classification used within a nuclear context. We assume here that the instantiation of this pattern provides a classification of the *ALF* system as a category *C* system. The classification of the *ALF* system as a category C system implies that every mandatory requirement associated with the development of category C systems as detailed in e.g. the standard IEC62138 [17] shall be met. We delimit the demonstration of the SaCS method to address the safety requirements *SR.1* to *SR.5* only.

7. REVISE DESIGN

The preliminary design described in Section 5.2 should be revisited in order to assure that it satisfies the safety requirements defined in Section 6. If the design describes a system that is not able to satisfy the safety requirements it should be revised.

We assume that revision is only needed with respect to a detailing of the *ACRAC* behaviour and amend the design specification with the UML sequence diagram presented in Figure 14.

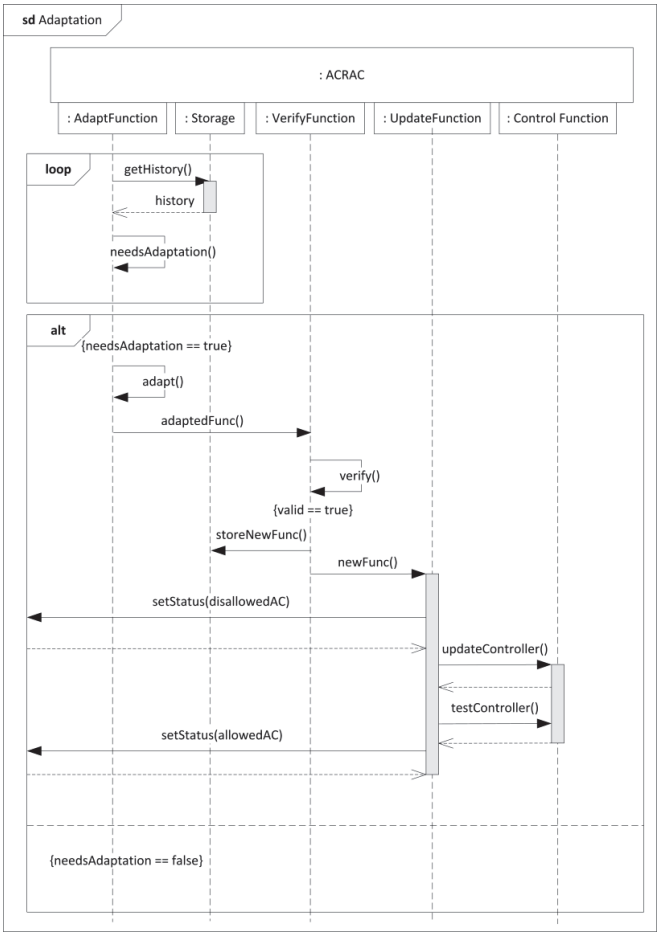


Figure 14 Sequence diagram illustrating the controller adaptation process

Figure 14 illustrates the main sequence for adapting the controller part of *ACRAC*. The inline decomposition of *ACRAC* reflects functional parts of the *ACRAC* controller.

8. ESTABLISH SAFETY CASE

8.1 Pattern Selection

Patterns supporting safety demonstration is provided from choice (F) and onwards in Figure 2. We select *Safety Requirements Satisfied* in choice (F) as support for deriving a safety case demonstrating that the safety requirements (identified in Section 6) are satisfied. We assume that the *Assessment Evidence* pattern in choice (G) is the most suitable for addressing the safety requirement *SR.5* as the satisfaction of the requirement may not directly be demonstrated with a reference to the design specification, but rather an assessment of the design. The *Deterministic Evidence* pattern in choice (G) is regarded as suitable for addressing the requirements *SR.1* to

SR.4 as these requirements should be able to be demonstrated met on the basis of the features of the design specification.

8.2 Pattern Instantiation

Given that the *Safety Requirements* composite illustrated in Figure 13 is instantiated and have produced the safety requirements *SR.1* to *SR.5* (see Section 6.2), then the *Safety Requirements Satisfied* pattern selected in Section 8.1 may be instantiated with the requirements as input. The *Safety Requirements Satisfied* is described in appendix B.16 and defines a demonstration strategy where safety is demonstrated by addressing all the safety requirements. The successful use of this pattern is based on the assumption that satisfying the safety requirements ensures a sufficiently safe system, this was assumed in Section 6.2. Figure 15 illustrates the result of applying *Safety Requirements Satisfied* pattern in the context of demonstrating that *ALF Dgn* is safe.

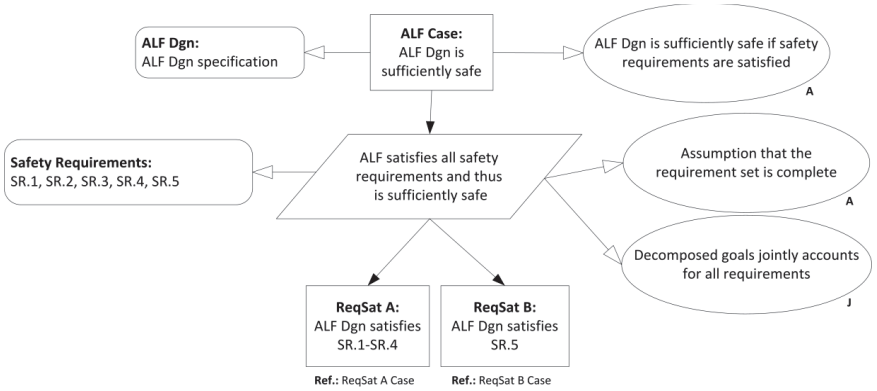


Figure 15 The "ALF Case" safety case

Figure 16 represents an excerpt from the pattern *Deterministic Evidence* and defines a parameterised argument structure annotated by a SaCS adapted version of the GSN notation [2][4][9][21][26]. The parameters of the argument structure shall be bound such that:

- *ToD* sets the target of the safety demonstration.
- *Cond* sets the condition that is claimed satisfactory assessed and satisfied.
- The evidence (e.g., reference to documentation) that may be conferred and that state that *Cond* is satisfied is set by *CondEv.*

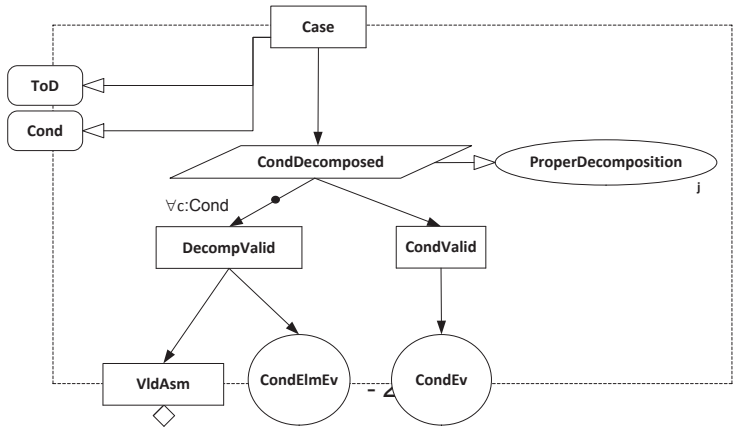


Figure 16 Excerpt - Deterministic Evidence

Figure 17 represents the result from instantiating the *Deterministic Evidence* pattern that is a safety case expressed in GSN notation. The pattern was applied such that the parameter *ToD* represented the *ALF Dgn* design, and the parameter *Cond* represented the safety requirements *SR.1* to *SR.4.*

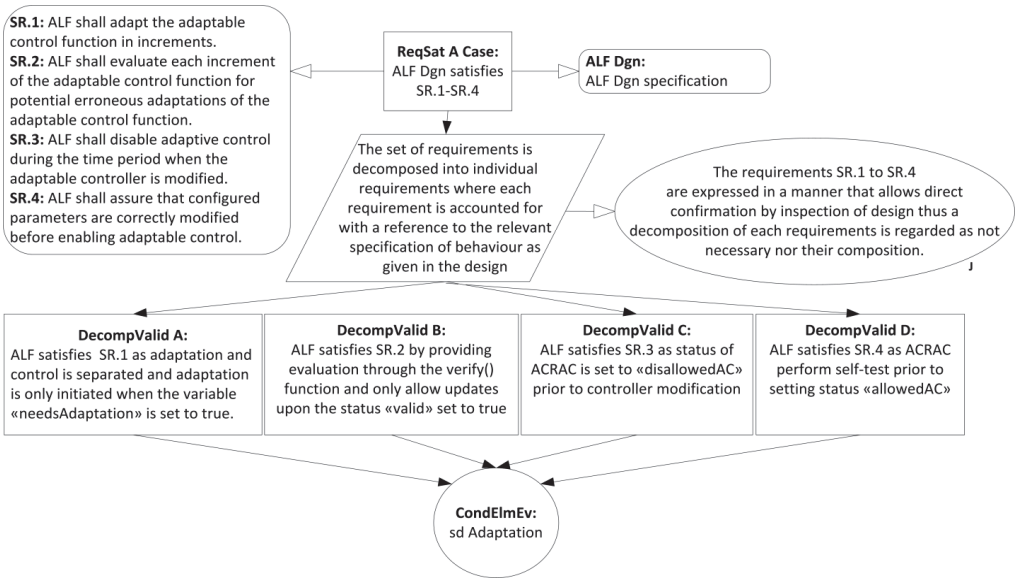


Figure 17 The "ReqSat A Case" safety case

The *Assessment Evidence* pattern is described in appendix B.17, an excerpt from the pattern is given in Figure 18. Figure 18 defines a parameterised argument structure annotated by a SaCS adapted version of the GSN notation. The argument structure decomposes an overall claim via sub-claims down to evidences. The parameters of the argument structure shall be bound such that:

- *ToD* sets the target of the safety demonstration.
- *Cond* sets the condition that is claimed satisfactory assessed and satisfied.
- *Crit* sets the criterion that shall be fulfilled in order for an assessment approach to provide satisfactory results.
- The rationale for the criteria being proper is given in the justification node identified as *ValidSetOfCriteria*.
- *Mtd* sets the method that is applied.
- The rationale for the method being suitable by satisfying criteria is given in the justification node identified as *MethodSuitable*.
- The evidence (e.g., reference to documentation) that may be conferred and that state that condition is satisfied is set by *CndSatEv*.

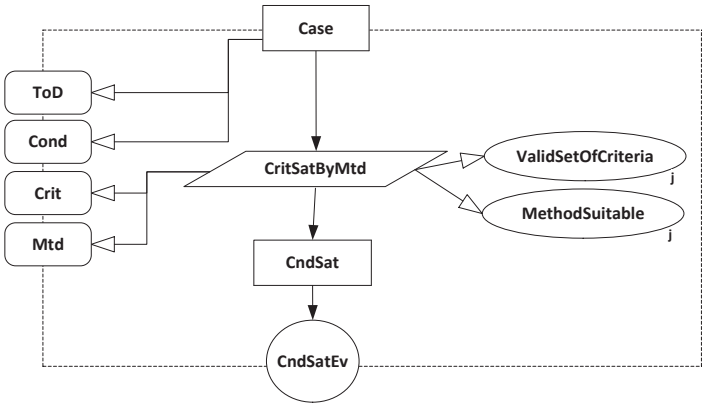


Figure 18 Excerpt – Assessment Evidence

In the instantiation of the *Assessment Evidence* pattern, the target of demonstration is the *ALF Dgn.* The condition to be addressed is the safety requirement *SR.5.* However, we also need to identify the method that shall be applied in order to perform the assessment. Further, the criteria that shall be applied in order to determine that a specific method is suitable must be identified. In addition, the chosen method must be applied in order to provide assessment results that confirm the main claim, i.e., *ALF Dgn* satisfies *SR.5*.

The instantiation of the *Assessment Evidence* pattern produces a safety demonstration that argues that the overall system (represented by the *ALF* system) satisfies the requirement *SR.5.* The instantiation result is presented in Figure 19 as an argument structure expressed with the GSN notation. We assume here for the sake of the argument that:

- The FMEA and FTA assessment methods, used in Section 6, represent the approaches that are recognised as suitable for assessing the system with respect to satisfaction of the identified safety requirements.
- The results from applying FMEA and FTA, identified as *ALF FMEA* and *ALF FTA,* respectively, represent valid evidences.

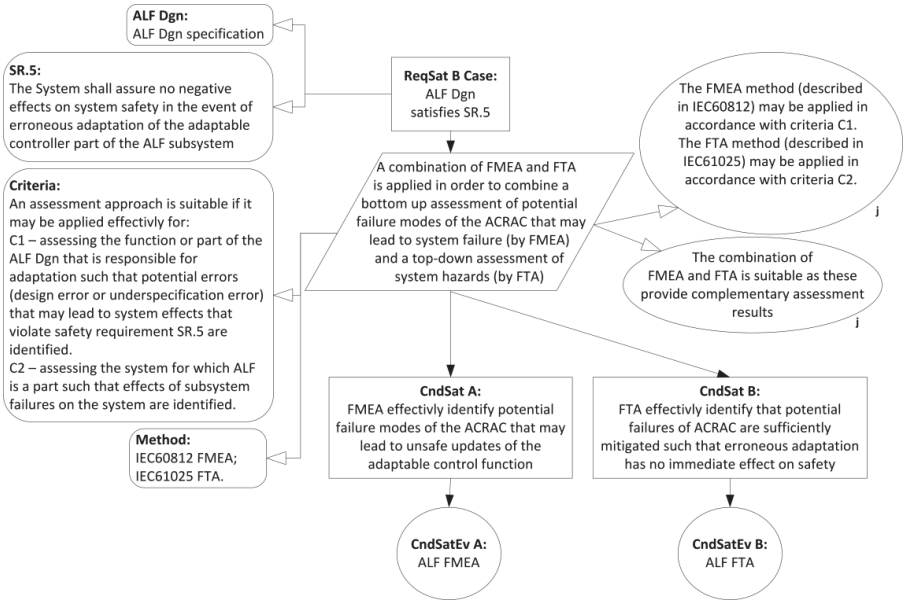


Figure 19 The "ReqSat B Case" safety case

8.3 Pattern Composition

Figure 20 specifies the composite *Safety Requirements*. The inputs to the instantiation of *Safety Requirements* may be seen from the assignment of the parameters *ToD* (short for Target of Demonstration) and *Req* (short for Requirements). The two *assigns* relations connecting artefact references with parameters indicate the assignments of the parameters. The instantiation of the *Safety Requirements Satisfied* pattern provides the main basis for demonstrating that the ALF system is suitably safe. Two different safety case patterns named *Deterministic Evidence* and *Assessment Evidence* are applied as support for arguing that the different safety requirements are sufficiently addressed. The *Assessment Evidence* pattern details a demonstration strategy that is applied for arguing that requirement *SR.5* is met. The *Deterministic Evidence* pattern details a demonstration strategy that is applied for arguing that the requirements *SR.1* to *SR.4* are met.

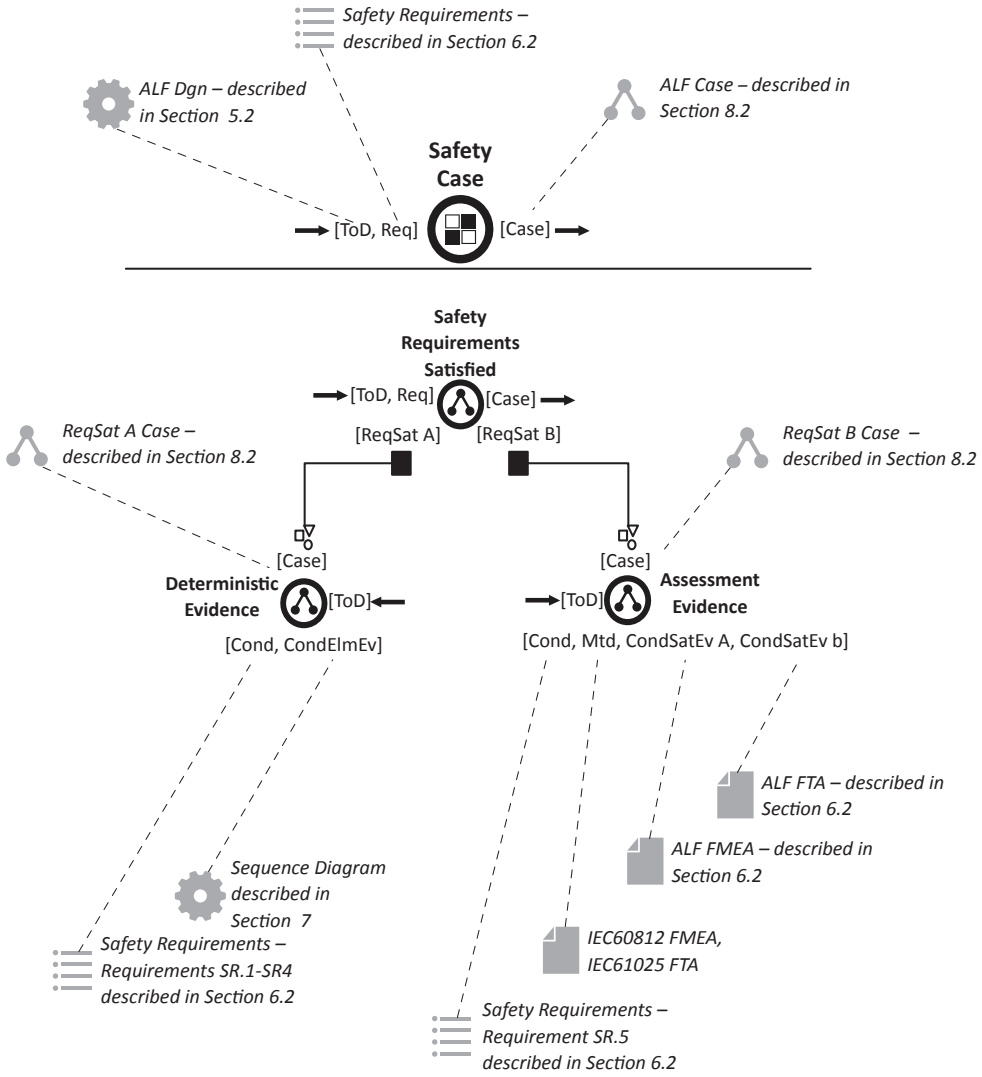


Figure 20 Composite pattern named "Safety Case"

Two *details* relations are used in Figure 20 to identify those parts of the pattern *Safety Requirements Satisfied* that are further detailed by the support of the patterns *Deterministic Evidence* and the pattern *Assessment Evidence*. A *details* relation is illustrated as a line connecting a black box with a set of smaller white shapes with a black stroke. The *details* relation between *Safety Requirements Satisfied* and *Deterministic Evidence* indicates that the instantiation of the parameter *Case* of *Deterministic Evidence* details the instantiation of the parameter *ReqSat A* of *Safety Requirements Satisfied.*

The artefact produced upon the instantiation of the *Assessment Evidence* pattern is named *ReqSat B Case* (See Figure 19). *ReqSat B Case* details the artefact *ReqSat B*

(See Figure 15 and Figure 20) produced upon the instantiation the *Safety Requirements Satisfied* pattern.

9. COMBINE FRAGMENTS

9.1 Pattern Composition

Figure 21 defines the composite *Pattern Solution* and illustrates how the different composite patterns defined in the previous sections are combined.

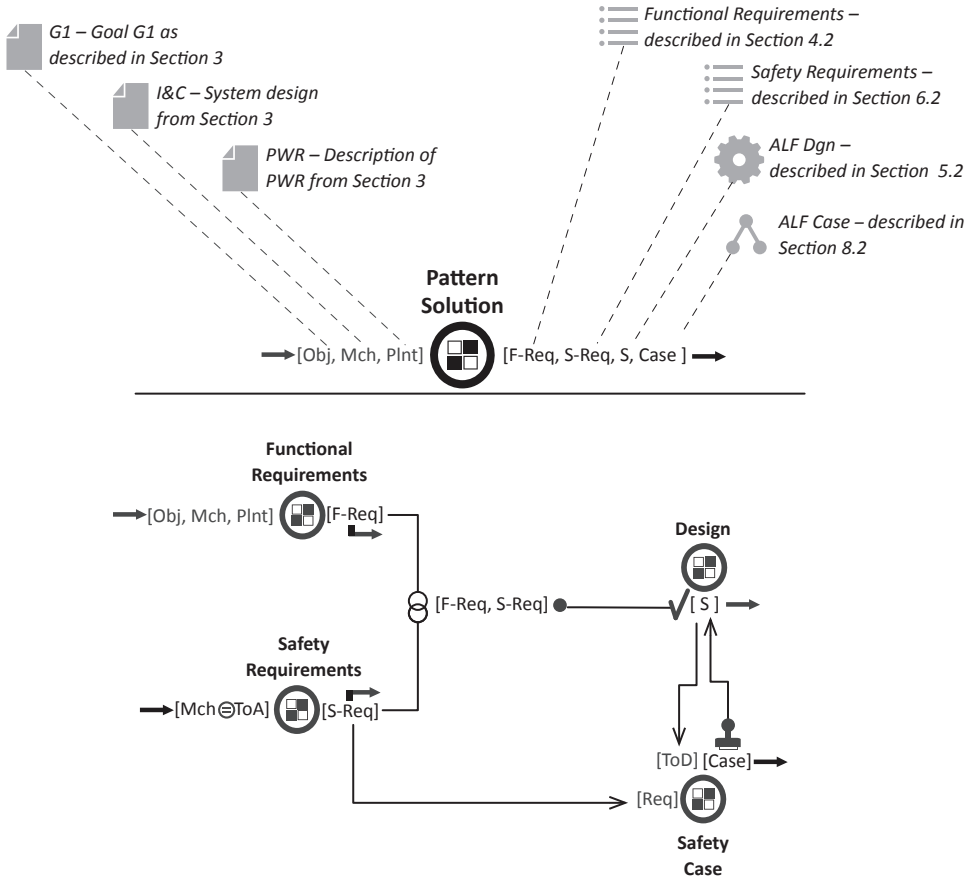


Figure 21 Composite pattern named "Pattern Solution"

Figure 21 documents the concepts and best practices applied in order to derive a conceptual safety design for the case by the identifying the patterns used as support. The different patterns references identifies that patterns applied. The relations describe how patterns are combined. The result of pattern instantiation is given by the different artefact references. The traceability between development artefacts and the patterns that was used as support is provided by the use of the *instantiates* relation.

Applying the SaCS method in iterations may refine the composite illustrated in Figure 21 as well as the conceptual safety design derived by instantiating it. As we have only addressed the goal G1 in this report, a second iteration over the SaCS method in the case should also address the goals G2 and G3 from Section 3.

10. SUMMARY OF RESULTS

This section summarises the result of applying the SaCS method on the case outlined in Section 3. The application of SaCS on the load following case was scoped such that only goal G1 described in Section 3 was addressed. Section 4 to Section 9 describes the application of SaCS in the case. The composite pattern in Figure 21 presented in Section 9 references the patterns applied in the case. The instantiation of the composite expressed in Figure 21 represents the conceptual safety design derived for the case.

A conceptual safety design, according to the definition given in Section 2, is a triple consisting of an early stage specification of system requirements, system design, and safety case of a safety critical system. A conceptual safety design for the load following case is represented by the specifications described in Section 10.1, 10.2 and 10.3. The specifications are extracted from the information produced upon the instantiation of the patterns as described in Section 4.2, 5.2, 6.2, 7 and 8.2.

10.1 Requirements Specification

The following abbreviations are used in the specification of the requirements that is contained in Table 8 and Table 9:

- ALF: Adaptable Load Following system
- GDfE: Grid Demand for Electricity
- PCBUS: Process Control BUS
- RPM: Reactor Power Measured
- SENS: SENsor data acquisition System
- CRP: Control Rod Pattern
- BC: Boron Concentration
- DEVRP: DEViation from Required Power
- CALCI: CALibrate Control rod pattern Indicator
- RDA: Rod Drive Actuators
- CVCA: Chemical and Volume Control Actuators

Table 8 Functional Requirements

Req	Description
FR.1	ALF system shall monitor the demand for electricity required by the grid by acquiring the GDfE signal from the PCBUS.
FR.2	ALF system shall monitor the reactor power output by acquiring the RPM signal from SENS.
FR.3	ALF system shall monitor the control rod position by acquiring the CRP signal from SENS.
FR.4	ALF system shall monitor the boron acid concentration by acquiring the BC signal from SENS.
FR.5	ALF system shall acquire GDfE signal by polling the PCBUS.
FR.6	ALF system shall acquire validated RPM, CRP, BC signals by request to

	SENS.
FR.7	ALF system shall poll for GDfE signal at a rate 10 times per second.
FR.8	ALF system shall maintain a history over GDfE, CRP, BC and RPM.
FR.9	ALF system shall calculate a value DEVRP that shall indicate non-optimal reactor control when operating in load following mode.
FR.10	ALF system shall calculate an indicator CALCI that shall identify the need for calibration of control rod patterns by the use of the history of DEVRP, CRP and BC.
FR.11	ALF system shall provide a mechanism for automatic calibration of control rod patterns based on CALCI indicator.
FR.12	ALF system shall activate calibration of the control rod patterns when the need to calibrate is indicated by the CALCI indicator.
FR.13	ALF system shall provide reactor control by providing actuation signals to RDA and CVCA.

Table 9 Safety Requirements

Req	Description
SR.1	ALF shall adapt the adaptable control function in increments.
SR.2	ALF shall evaluate each increment of the adaptable control function for potential erroneous adaptations of the adaptable control function.
SR.3	ALF shall disable adaptive control during the time period when the adaptable controller is modified.
SR.4	ALF shall assure that configured parameters are correctly modified before enabling adaptable control.
SR.5	The System shall assure no negative effects on system safety in the event of erroneous adaptation of the adaptable controller part of the ALF subsystem.

10.2 Design Specification

Figure 22 is a UML component diagram illustrating graphically the overall ALF design. ALF consist of two parts, CRICS (Control Rod Information and Control System) and BCS (Boron Control System).

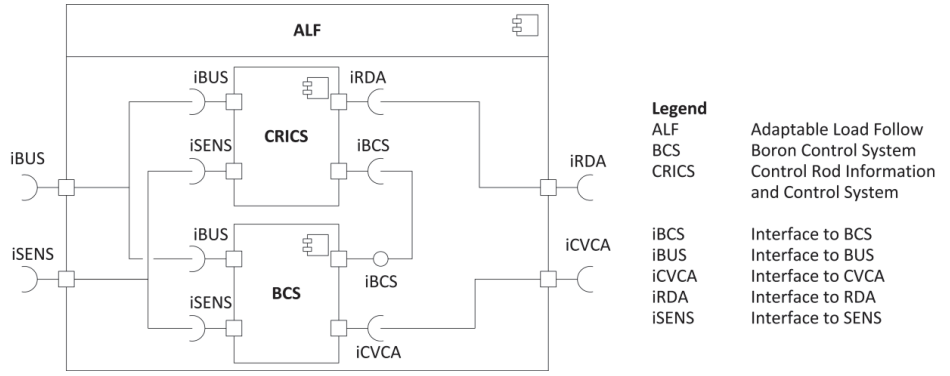


Figure 22 The "ALF" system

Figure 23 details the CRICS system and Figure 24 details the BCS system. Adaptability is incorporated in the CRICS system in the part identified as ACRAC. The ACRAC system is responsible for automatically calibrating the control rod insertion patterns as the reactor fuel is degraded during the fuel life cycle. The ACRAC system shall also account for the decreased need for boron acid in the moderator as the fuel reaches the end of the fuel cycle.

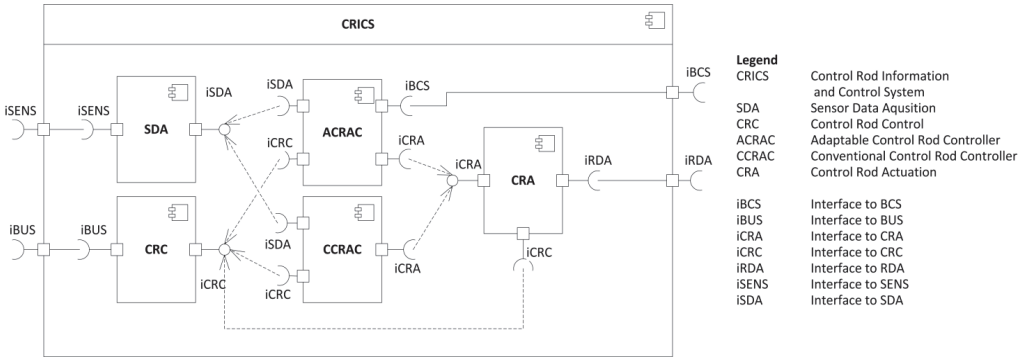


Figure 23 The "CRICS" System

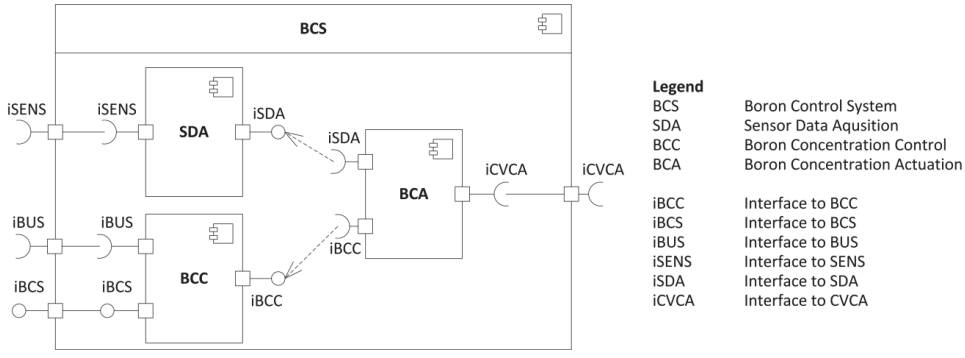


Figure 24 The "BCS" system

The CRICS system shall work in the following manner:

- CRC monitors the process control bus at a frequency of 10 times per second through the interface iBUS for commands from plant operator or other systems that affect control rod control. CRC also sends control status information to the process control bus. Signals that are handled by CRC are:
    - AcBL, DeAcBL – deactivate or activate base-load generation (constant power).
    - AcPFC, DeAcPFC – deactivate or activate primary frequency control (short term or immediate change of electricity generation to variation in demand detected by variations in frequency in the grid).
    - AcSFC, DeAcSFC – activate or deactivate secondary frequency control (adaptation of electricity generation over a longer time frame, minutes scale, by adapting electricity generation to a frequency deviation over a time period).
    - AcLF, DeAcLF – activate or deactivate load following (e.g. a variable load programme with one or several power changes over a period of 24 hours).
    - AcAC, DeAcAC – activate of deactivate adaptable control. When an AcAC or DeAcAC is retrieved, CRC forwards this to CRA.
    - GDfE – provide information on the grid demand for electricity.
- SDA is responsible for gathering validated sensor signals (through the interface iSENS). Signals that are acquired by SDA are:
    - RPM – That provides a measure of reactor power.
    - CRP – That provides data on control rod position in the core.
- ACRAC is responsible for providing control rod control and shall automatically calibrate control rod actuation patterns in order to compensate for fuel degradation. The ACRAC system shall:
    - Retrieve from CRC and SDA the signals GDfE, CRP and RPM.
    - Provide control rod actuation signals to CRA.
    - Calculate a value DEVRP that shall be used as an indicator for non-optimal reactor control when operating in load following mode.
    - Calculate a value CALCI that shall indicate if calibration of control rod patterns is required.
    - Perform calibration of control rod patterns automatically when CALCI provides a clear indication on suboptimal control.
    - Interact with the BCC system in order to request specific Boron acid concentration levels and thus seek an optimal balance of control rods and boron acid moderation of the process.
- CCRAC is responsible for providing control rod control. The CCRAC system shall:
    - Retrieve from CRC and SDA the signals GDfE, CRP and RPM.
    - Provide control rod actuation signals to CRA.
- CRA is responsible for deciding upon which of the two systems, CCRAC or ACRAC, that will perform control rod actuation by passing forward actuation signals from one of them to RDA. CCRAC and ACRAC are two diverse systems for control rod control where the former is assumed here to be an existing control rod control system and the latter is the adaptable control rod system that is part of the upgrade. Both systems operate in parallel and provide actuation signals to RDA. The RDA system is assumed to consist of the drive motors and other

mechanisms for inserting and retracting control rods. When an AcAC signal is received from CRC then actuation signals from ACRAC will be allowed and preferred for actuation of RDA. When a DeAcAC signal is received from CRC only actuation signals from CCRAC actuation will be allowed for actuation of RDA.

The BCS system is intended to work in the following manner:

- BCC monitors the process control bus at a frequency of 10 times per second through the interface iBUS for commands from plant operator or other systems that affect boron acid concentration control. BCC also sends control status information to the process control bus. Signals that are handled by BCC are:
    - acracBC – That informs on the boron concentration level required by the ACRAC system.
    - opBC – That informs on the boron concentration level required by an operator.
- SDA is responsible for gathering and validating sensor signal on boron concentration in the primary loop.
- BCA is responsible for providing actuation signals to the different subsystems of the CVCA such that a required boron concentration level may be achieved. The CVCA system is here assumed to consist of valves, piping, boron acid tank, water tanks and all appliances necessary to supply of water via the makeup channel and allow draining of water via the letdown channel.

Figure 25 is a UML sequence diagram illustrating how the ACRAC system shall adapt its control function. The inline decomposition of ACRAC reflects functional parts of the ACRAC controller.

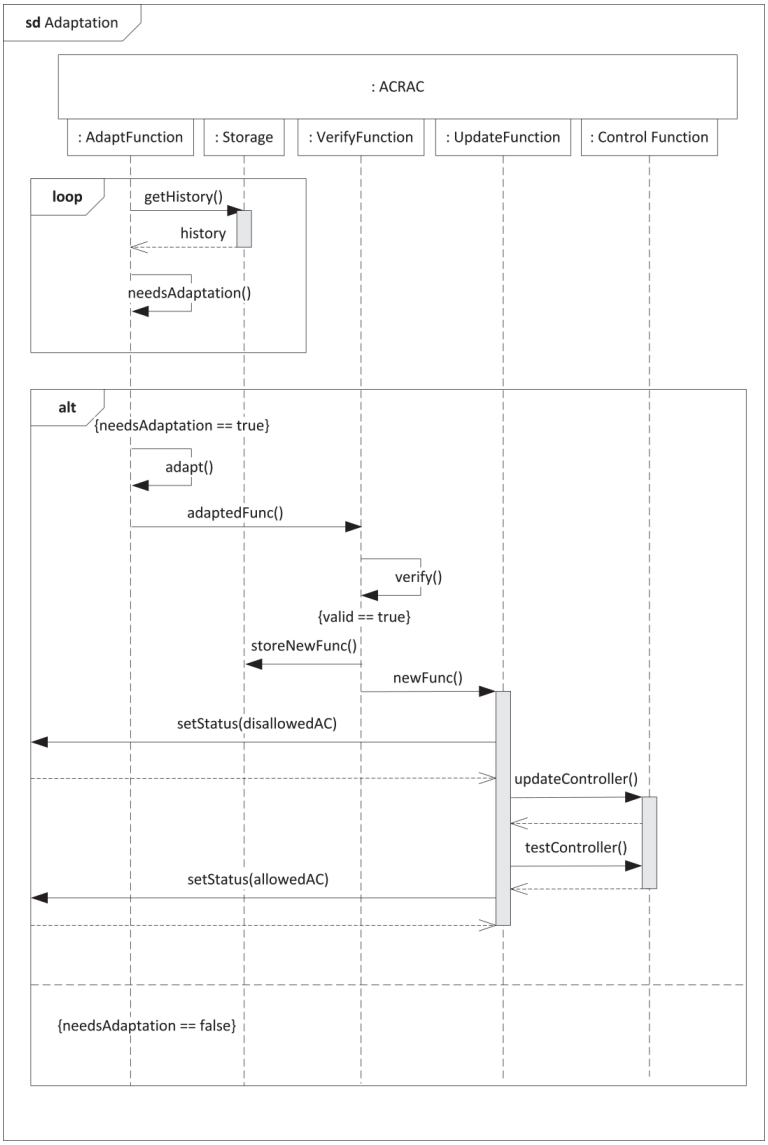


Figure 25 Sequence diagram "sd Adaptation" describing ACRAC behaviour

10.3 Safety Case Specification

Figure 26, Figure 27 and Figure 28 illustrates graphically the *ALF* safety case. The safety case only addresses technical safety aspects by addressing the safety requirements, issues like quality management or safety management is not addressed.

The safety case provided here illustrates the safety demonstration strategy as outlined at the initial stage of *ALF* development.

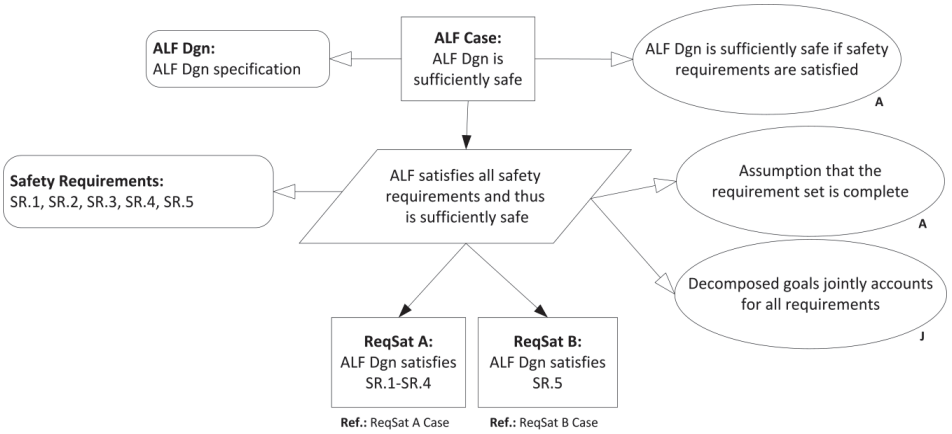


Figure 26 The "ALF Case" safety case

Figure 26 expresses that the *ALF* design is sufficiently safe for its purpose by a strategy of demonstrating that safety requirements are satisfied. The safety case described by Figure 26 contains references to decomposed parts of the case that are illustrated in Figure 27 and Figure 28 respectively. Figure 27 addresses the safety requirements SR.1 to SR.4 and demonstrates by providing references to relevant parts of the design specification that these requirements are satisfied. Figure 28 addresses the safety requirement SR.5 and demonstrates that the requirement is satisfied by providing reference to the relevant safety assessment as supporting evidence.

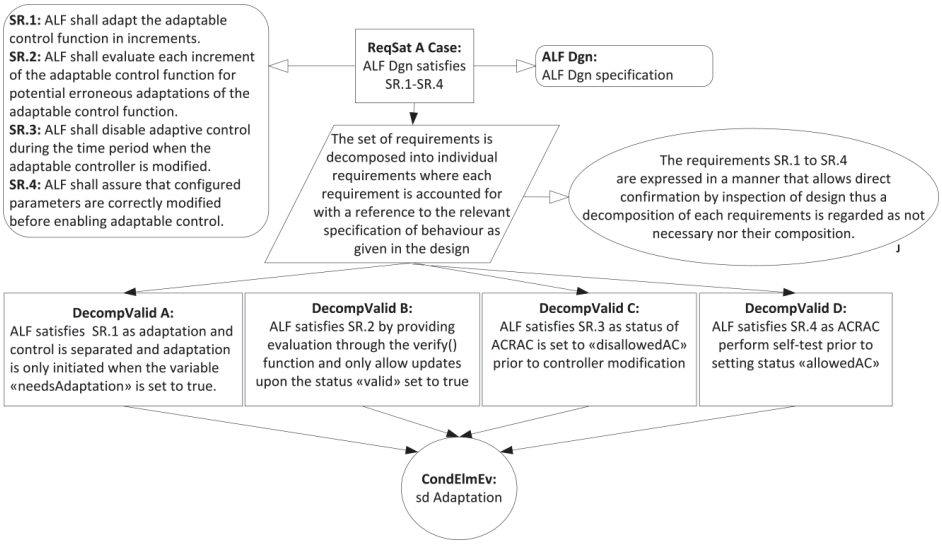


Figure 27 The "ReqSat A Case" safety case

The reference to "*sd Adaptation"* evidence in Figure 27 refers to the sequence diagram provided in Figure 25.

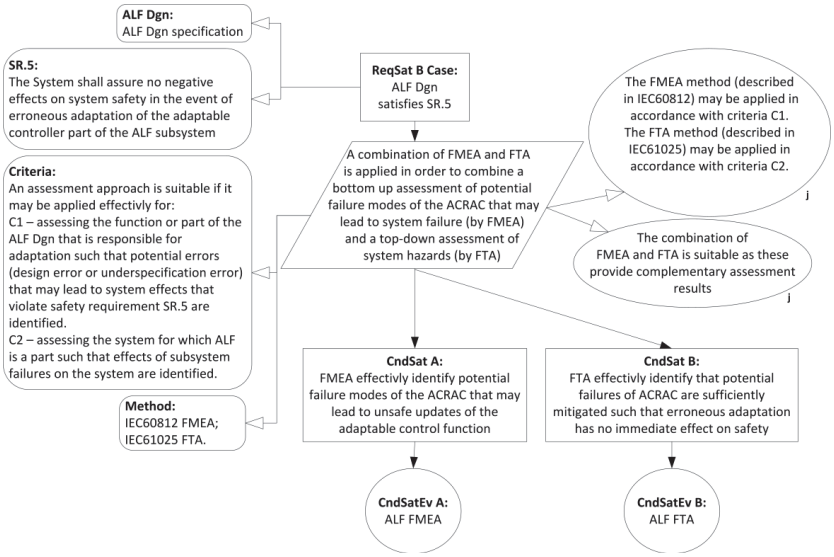


Figure 28 The "ReqSat B Case" safety case

The references to *ALF FMEA* and *ALF FTA* in Figure 27 are references to the results of the FMEA and FTA assessments provided in Section 6.2, given in Table 2 and Table 4, respectively.

11. DISCUSSION

The main intention with this report is to demonstrate the applicability of the SaCS method on a relevant case. Each step in the application of the SaCS method in the case is performed by the authors; thus, the following discussion represents a self-evaluation. However, each step of the SaCS method is documented in a manner that allows others to perform an independent evaluation of the feasibility of the SaCS method.

The suitability of the SaCS method and its supporting pattern language in providing a utility for a user is in this section discussed by elaborating upon the fulfilment of the success criteria defined in Section 2 that states:

P: *Application of the SaCS method on the load following case described in Section 3 results in a conceptual safety design that characterises the load following case and is easily instantiated from a composite SaCS pattern. Furthermore, the conceptual safety design:*

a) Is in accordance with safety objectives – the conceptual safety design is defined in agreement with safety objectives.

b) Is at a sufficient level of detail – the conceptual safety design is expressed in a manner that is sufficiently detailed for an early stage specification and may be easily understood.

c) *Is easy to use – the conceptual safety design may be easily extended, detailed or refined.*

Firstly, a conceptual safety design according to the definition given in Section 2 is a triplet consisting of specification of system requirements, system design, and safety case. The three different kinds of specifications that constitutes a conceptual safety design for the case are documented in separate sections, Section 10.1, 10.2, and 10.3, respectively. Thus, at least the conceptual safety design for the case consists of the required specification parts.

Secondly, we find it fair to argue that the conceptual safety design is easily instantiated from a composite SaCS pattern if each of its parts, described in Section 10.1, 10.2 and 10.3, are easily instantiated from the composite pattern for the case. Figure 21 represents the composite pattern for the case. A pattern needs to be interpreted by a user in its application context in order to be useful, which is a process that may be difficult to document. Although we have tried to provide a detailed description of the pattern instantiation process, we do not present how a pattern is interpreted in its context other than documenting the result of pattern instantiation. It is the pattern definitions, the descriptions of the application of patterns in the case, and the descriptions of the results of pattern instantiation that represents the documentation that allows others to perform an independent evaluation of the feasibility of the SaCS approach. In the following, we argue that each of the specifications described in Section 10.1, 10.2 and 10.3 are easily instantiated from the composite in Figure 21.

A composite pattern shall be instantiated according to the rules for composition. The rules for composition are to some extent given in Appendix A.3. The complete syntax and semantics of the SaCS pattern language is described in [13], which includes the rules for composition. In the process of instantiating a composite pattern, the first task is to decide the order in which contained patterns within the composite should be instantiated. The instantiation order of the patterns within a composite may be deduced from the relations that connect the contained patterns. A second task is to instantiate the contained patterns according to the decided order as well as according to their instantiation rule. The instantiation rule of a basic pattern is stated within a separate section of its definition. The definitions of the patterns used in this report are fully documented in Appendix B.

The conceptual safety design presented in Section 10 is derived as a solution for the problem context described in Section 3. The conceptual safety design is the outcome of the instantiation of the composite pattern defined in Figure 21. The composite in Figure 21 represents a specification of the combined use of the four composite patterns defined in Figure 5, Figure 10, Figure 13, and Figure 20. The four composites specify the use of the basic patterns defined in Appendix B according to the syntax of the SaCS pattern language defined in [13]. It may be deduced from Figure 21 that *Functional Requirements* and *Safety Requirements* can be instantiated in parallel. The *Design* pattern should be instantiated prior to *Safety Case*. The pattern instantiation order is deduced on the basis of the rules for composition as defined in [13], especially the rules for connecting patterns by the use of relations.

We argue that the requirements specification described in Section 10.1 is easily instantiated from the composite pattern described in Figure 21 through the instantiation of the composites named *Functional Requirements* and *Safety Requirements*. In Figure 21, it is specified that the requirements derived by the use of the *Functional Requirements* pattern is given in Section 4.2. It is also specified that the requirements derived by the use of the *Safety Requirements* pattern is given in Section 6.2. By

inspecting Section 4.2 and Section 6.2, the descriptions of the instantiation of *Functional Requirements* and *Safety Requirements* are found.

Figure 5 specifies the composite named *Functional Requirements* and how its only constituent pattern, named *Variable Demand for Service,* is instantiated as support for eliciting requirements. Basically, *Functional Requirements* is instantiated by the instantiation of *Variable Demand for Service*. Section 4.2 describes the instantiation of the *Variable Demand for Service,* while the pattern itself is defined in appendix B.1. The requirements are derived by instantiating *Variable Demand for Service* according to its instantiation rule. The instantiation rule of *Variable Demand for Service* is part of the pattern definition presented in appendix B.1. The instantiation rule expresses how a set of abstract requirements defined in the pattern may be used to define context specific requirements. Table 1 provides the traceability between the abstract requirements defined within the pattern *Variable Demand for Service* and the requirements defined for the case.

The composite *Safety Requirements* defined in Figure 13 expresses a combination of several basic patterns. The detailed description of the instantiation of the contained patterns of *Safety Requirements* is given in Section 6.2. In short, the safety requirements are the outcome of instantiating the pattern *Establish System Safety Requirements* contained within *Safety Requirements*. The other patterns within *Safety Requirements* are used as support for deriving the necessary information required for specifying the safety requirements, such as identifying the hazards associated with the system in question and the potential causes of these hazards. Each constituent pattern of *Safety Requirements* is detailed in Appendix B. The outcomes of instantiating the different patterns of *Safety Requirements* are documented in Section 6.2. Figure 13 identifies the relationship between different development artefacts and the patterns that used as support for their definition. The development artefacts referred to in Figure 13 are documented in Table 2 to Table 7.

We argue that the design specification described in Section 10.2 is easily instantiated from the composite pattern described in Figure 21 through the instantiation of the composite named *Design.* The *Design* composite is defined in Figure 10. It consists of only one pattern named *Trusted Backup.* The *Trusted Backup* pattern is defined in appendix B.5 and includes, as every other basic pattern, an instantiation rule providing guidance on how the pattern should be instantiated. The instantiation of the *Trusted Backup* pattern is described in Section 5.2. A detailed explanation of the relation between the abstract design presented in the pattern and the specification of the *ALF* system is also given in Section 5.2.

We argue that the safety case specification described in Section 10.3 is easily instantiated from the composite pattern identified in Figure 21 with the name *Safety Case.* The pattern *Safety Case* is defined in Figure 20 where a combined use of the basic patterns *Safety Requirements Satisfied*, *Assessment Evidence,* and *Deterministic Evidence* is specified. The definitions of the basic patterns are given in the appendices B.16, B.17, and B.20, respectively. The outcome of applying the patterns, which is a safety case specified according to the GSN notation [9], is derived by assigning values to the parameters of the pattern structure specified by the composite diagram in Figure 20 and conduct a stepwise instantiation of the patterns. Once the input parameters of each pattern are assigned their values, the guidance provided by the pattern descriptions may be systematically followed in order to derive results. Section 8.2

describes how each of the mentioned basic patterns is instantiated as well as how the outcomes are combined into a safety case.

As argued above, the conceptual safety design for the case is the result of the systematic application of SaCS patterns. We think that the application of the SaCS method in the case is described to a level of detail that allows the feasibility of the SaCS approach to be independently evaluated by others. In this report, the SaCS method is defined in Appendix A, each step of applying the SaCS method in the case is described in Section 4 to Section 9, each basic pattern that is used in order to derive results is defined in Appendix B, and the language for specifying SaCS patterns is outlined in Appendix A and fully detailed in [13].

Regarding a), we argue that the conceptual safety design is defined in agreement with safety objectives as follows.

Section 10.1, 10.2, and 10.3 describes the three different specifications that represent the conceptual safety design for the case. The conceptual safety design is defined in accordance with safety objectives in the following manner:

- Section 10.1 contains a description of the safety objectives. The safety objectives are represented by the safety requirements described in Table 9. The safety requirements are established by the application of the composite pattern named *Safety Requirements* as described earlier. The *Safety Requirements* composite describe the systematic application of patterns as support for establishing the risk associated with the use of the system under development as well how these risks are used to define safety requirements. The system under development is outlined in Section 3 along with a description of its intended context of operation.
- Section 10.2 contains description of a system design that accommodates the safety requirements presented in Table 9 as well as the functional requirements presented in Table 8. Although the system design is defined in accordance with the safety requirements, it is mainly the safety case defined in Section 10.3 that express to which extent the safety requirements are satisfied.
- Section 10.3 contains a description of a safety case arguing that the safety requirements are met. The safety case is specified by the use of the GSN notation. The semantic of GSN allows the relationship between the claims put forward in the safety case and the supporting evidences for claims to be easily identified. The safety case in Section 10.3 demonstrates that the system design presented in Section 10.2 is sufficiently safe for its intended purpose by a strategy of showing that the safety requirements presented in Section 10.1 are met. The safety case reference different parts of the system design as supporting evidence for claims. The assumption that the system is sufficiently safe given that the safety requirements are met is rationalised in Section 6.2.

The three specifications serve different purposes where the system requirements specification captures the objectives of the system under construction. The system design specification represents an early stage technical description of a system that fulfils objectives. The safety case expresses in what way the safety objectives are fulfilled.

Regarding b), we argue that the conceptual safety design is at a sufficient level of detail as follows.

According to the definition in Section 2, a conceptual safety design is an early stage specification. By an early stage specification it is here meant a description that shows the main features of a solution for a given problem. Although the specification is not expected to be complete, a conceptual safety design is of little use if it does not clearly convey a potential solution for the problem in question in a manner that may be easily understood. We find it reasonable to argue that the conceptual safety design is at a sufficient level of detail if it is expressed in a manner that clearly shows how the objectives of the case is intended to be solved. Furthermore, the conceptual safety design should be expressed in a format that may be easily understood by its intended users.

The objective in the case as it is expressed in Section 3 is to derive an adaptable load following mode control system concept that offers a high degree of reactor manoeuvrability as well as precision by automatically compensating for fuel burn up. An initial starting point in the application of the SaCS method is the development objective stated in Section 3. The outcome of applying the SaCS method is the conceptual safety design presented in Section 10. Through the process of applying the SaCS method, the overall development objective is refined into a requirements specification and further into a system design. The requirements specification is intended to detail what is required by the system under development in order to satisfy the overall objective. Each step in the application of the SaCS method is detailed in Section 4 to Section 9. Furthermore, Appendix B details every basic SaCS patterns applied in this report. In this sense, all the necessary information required in order to trace the end result step-by-step through the application of the SaCS method back to the initial development objective is available.

The conceptual design is a triplet where each part is described in the following formats:

- The requirements specification, described in Section 10.1, is defined textually in natural language. Requirements specifications are commonly expressed in natural language.
- The design specification, described in Section 10.2, is defined by a combination of UML diagrams and textual descriptions. UML is a commonly used modelling language for describing systems in terms of their structure and behaviour.
- The safety case specification, described in Section 10.3, is defined with GSN. The GSN notation facilitates structuring a safety argument with simple graphical constructs that requires little effort to be understood.

Although textual specifications of requirements, GSN, and UML models in general should be understandable for the intended users of SaCS, this may not be the case for the specifications provided here. We have not tested the specifications on potential users of SaCS in order to investigate if the specifications are easy to understand, but rather provide the specifications themselves.

Regarding c), we argue that the conceptual safety design is easy to use as it may be easily extended, detailed or refined as follows.

The system design in Section 10.2 is a refinement of the requirements in Section 10.1, the requirements are derived on the basis of systematic analysis of the development objectives and the context described in Section 3. The scope of investigating the applicability of the SaCS method was limited to only addressing goal G1. The conceptual safety design may be refined by repeating the process and account for all

the development objectives described in Section 3. The repetition of the SaCS method to address the other objectives does not necessarily imply that each part of the conceptual safety design is changed; a high degree of reuse is expected. The conceptual safety design presented in Section 10 is described at a high level of abstraction, and it may be difficult at this stage to evaluate if the design is easy to refine into an implementation. However, we find the specification informative enough for experts to judge whether the concept described is feasible and detailed enough to use as a starting point for further refinement.

The requirements specification in Section 10.1 consists of requirements that are uniquely identified. Any requirement may be detailed or rephrased; the modified requirement will then exist in another version. Requirements may be added to the specification easily by adding a unique requirement identifier and associated requirement text.

The design specification is described by UML. UML has a rich language for specifying different aspects of a system. The design specified in Section 10.2 may be easily reused, extended and detailed by using the UML language. Examples may be to detail the structure and behaviour of the ALF design by using UML diagrams like class diagrams, sequence diagrams, state machine diagrams and activity diagrams.

The safety case specification is described by GSN. GSN offers constructs for modularising a safety argument such that existing argument structures may be easily reused. The scope of the safety case in Section 10.3 may be extended easily by, e.g., adding nodes to the existing tree structure, adding sub trees to the structure, or make the existing safety case a sub tree in a larger argument structure.

12. CONCLUSIONS

In this report we have exemplified the application of the SaCS-method on the load following mode control application. We have tried to argue that the SaCS method facilitates effective and efficient development of conceptual safety designs on the basis of the demonstration of the application of the SaCS on the load following case. The application of SaCS is described in Section 4 to Section 9. The fulfilment of success criteria is discussed in Section 11.

The conceptual safety design summarised in Section 10 is instantiated from several basic SaCS patterns within a case specific composite SaCS pattern that is illustrated in Figure 21. Each basic pattern (defined in Appendix B) has clearly defined inputs and outputs and provides guidance on instantiation through defined instantiation rules (see sections identified with heading “Instantiation Rule” of each individual pattern). The combination of instantiation results from several patterns is defined by *relations* (the relations are defined in Appendix A.3). The conceptual safety design is built systematically according to the SaCS-method (described in Appendix A.1) in manageable steps (exemplified in Section 4 to Section 9) by instantiating pieces (basic patterns) of the whole (composite pattern) and merge results.

The conceptual safety design described in Section 10 is consistent with the definition given in Section 2. A conceptual design is a triple consisting of a specification of system

requirements, system design, and safety case, each of which is detailed in Section 10.1, Section 10.2, and Section 10.3, respectively.

13. REFERENCES

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

[2] R. Alexander, T. Kelly, and J. McDermid, Safety Cases for Advanced Control Software: Safety Case Patterns, Technical Report FA8655-07-1-3025, Department of Computer Science, University of York, 2008.

[3] A. Armoush, F. Salewski, and S. Kowalewski, Design Pattern Representation for Safety-Critical Embedded Systems, *Journal of Software Engineering and Applications*, Vol. 2, No. 1, 2009.

[4] P. Bishop, R. Bloomfield, and S. Guerra, The Future of Goal-based Assurance Cases, in *Proc. Workshop on Assurance Cases, 2004 International Conference on Dependable Systems and Networks*, pp. 390–395, 2004.

[5] F. Buschmann, K. Henney, and D. C. Schmidt, *Pattern-Oriented Software Architecture: On Patterns and Pattern Languages*. Vol. 5, Wiley, 2007.

[6] W. D. Ellis, *A Source Book of Gestalt Psychology*, The Gestalt Journal Press, 1997.

[7] D. B. Fogel (Ed.), *Evolutionary Computation: The Fossil Record,* Wiley-IEEE Press, 1998.

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

[9] GSN Working Group, GSN Community Standard, Version 1.0, GSNWG, York, England, 2011.

[10] R. Hanmer, *Patterns for Fault Tolerant Software,* Wiley, 2007.

[11] A. A. Hauge and K. Stølen, A Pattern-based Method for Safe Control Conceptualisation Exemplified Within Nuclear Power Production, In *Proc. 31st International Conference on Computer Safety, Reliability and Security (SafeComp´12)*, Lecture Notes in Computer Science, Vol. 7612, pp. 13-24, Springer-Verlag, 2012.

[12] A. A. Hauge and K. Stølen, A Pattern-based Method for Safe Control Conceptualisation – Exemplified Within Railway Signalling, HWR-1037 rev 2, OECD Halden Reactor Project, 2014.

[13] A. A. Hauge and K. Stølen, Syntax & Semantics of the SaCS Pattern Language, HWR-1052, OECD Halden Reactor Project, 2013.

[14] IAEA, Modern Instrumentation and Control for Nuclear Power Plants: A Guidebook, Technical Report Series No. 237, International Atomic Energy Agency, 1999.

[15] IEC, Power Plants – Instrumentation and Control Important to Safety – General Requirements for Systems. IEC 61513, International Electrotechnical Commission, 2011.

[16] IEC, Nuclear Power Plants – Instrumentation and Control Important to Safety – Classification of Instrumentation and Control Functions. IEC 61226, International Electrotechnical Commission, 2009.

[17] IEC, Nuclear Power Plants – Instrumentation and Control Important to Safety – Software Aspects for Computer-based Systems Performing Category B and C functions. IEC 62138, International Electrotechnical Commission, 2004.

[18] IEC, Railway Applications – Specification and Demonstration of Reliability, Availability, Maintainability, and Safety (RAMS). IEC 62278, International Electrotechnical Commission, 2002.

[19] IEC, Analysis techniques for system reliability – Procedure for failure mode and effects analysis (FMEA), Edition 2.0, International Electrotechnical Commission, 2006.

[20] M. Jackson, Problem Frames – Analysing and Structuring Software Development Problems, Addison-Wesley, 2001.

[21] T. Kelly and R.Weaver. The Goal Structuring Notation – A Safety Argument Notation, In Proceedings of Workshop on Assurance Cases: Best Practices, Possible Obstacles, and Future Opportunities. Supplemental Volume of the 2004 International Conference on Dependable Systems and Networks (DSN'04), 2004.

[22] J. C. Knight, Safety critical systems: challenges and directions, In Proceedings of 24th International Conference on Software Engineering (ICSE'02), pp. 547-550, ACM, 2002.

[23] W. Lidwell, K. Holden, and J. Butler, *Universal Principles of Design,* Rockport Publishers, Second edition, 2010.

[24] A. Lokhov, Technical and Economic Aspects of Load Following with Nuclear Power Plants, Nuclear Development Division, OECD NEA, 2011.

[25] Object Management Group, Unified Modelling Language Specification, Version 2.4.1, 2011.

[26] J. Spriggs, GSN – The Goal Structuring Notation: A Structured Approach to Presenting Arguments. Springer, 2012.

[27] M. Vuori, H. Virtanen, J. Koskinen, and M. Katara, Safety Process Patterns in the Context of IEC 61508-3, Report 15, Department of Software Systems, Tampere, University of Technology, 2011.

[28] M. Wertheimer, Laws of organization in perceptual forms. In W. D. Ellis (ed.) *A Sourcebook of Gestalt Psychology*, pp. 71-88, Routledge and Kegan Paul, 1938.

APPENDIX A OVERVIEW OF THE SACS PATTERN LANGUAGE

A.1 THE SACS METHOD

Figure 29 illustrates the activities of the SaCS method. The three main activities are: pattern selection; pattern composition; and pattern instantiation (denoted S, C and I for short). The main activities may be performed in parallel and their sub-activities may be interleaved. The sub-activities of S, C and I shall be performed in order (e.g., S.a is performed before S.b) but sub-activities of one main activity may be interleaved by sub-activities of another main activity (e.g., S.a is performed, I.a is performed, I.b is performed, and then S.b is performed).

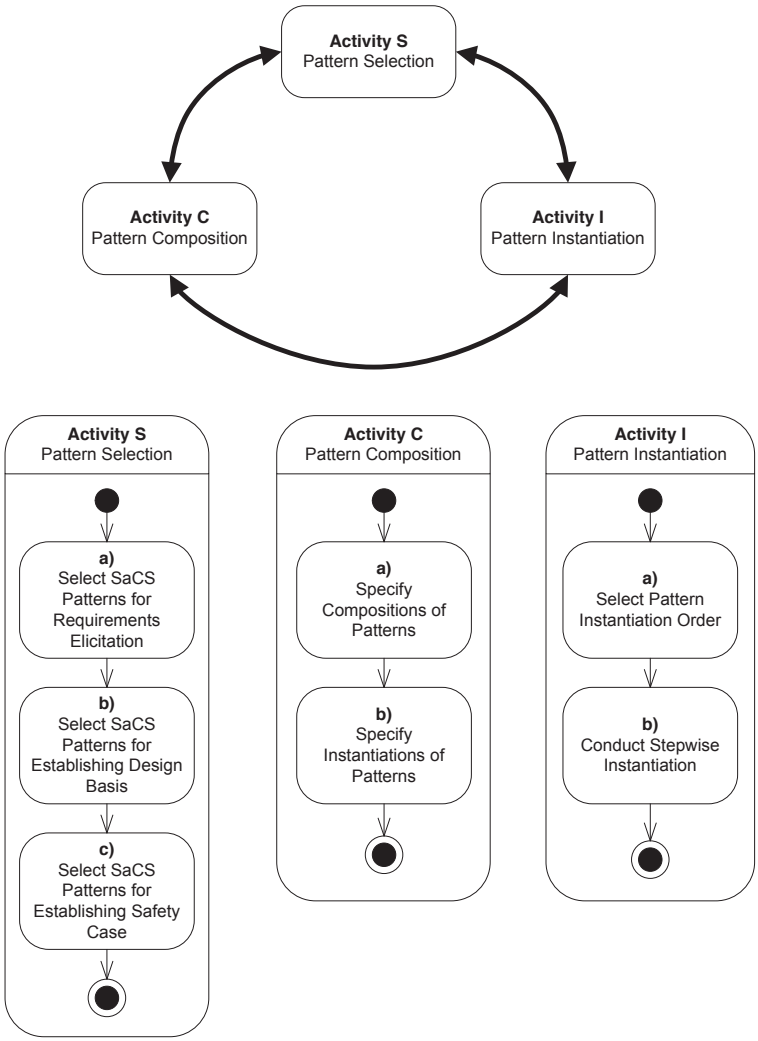


Figure 29 The SaCS Method

Pattern Selection – The purpose of this activity is to select the relevant SaCS patterns to support the conception of a safety design with respect to a given development case. In SaCS, a special focus is on offering patterns for capturing the requirements for the system design under development, patterns on design solutions, and patterns on safety demonstration. The sub-activities are:

a) Select SaCS Patterns for Requirement Elicitation: requirements may be elicited as a result of applying patterns reflecting a process assurance perspective or a product assurance perspective. Patterns supporting the process assurance perspective are process requirement patterns and process solution patterns that support elicitation of requirements by applying a specific process and by applying specific methods. Patterns supporting the product assurance perspective are product requirement patterns that supports elicitation of

requirements by focusing on system specific phenomena, e.g., analysis of the interaction between entities within a system design;

b) Select SaCS Patterns for Establishing Design Basis: a design basis may be established by the support of design patterns. The selection of an appropriate design pattern as support for system design should be performed by comparing the strengths and weaknesses of the concepts as expressed in the different patterns with the needs as expressed by the system requirements in order to evaluate if a pattern is able to offer a solution that satisfy the needs. The system requirements are defined on the basis instantiating the patterns selected in step a).
c) Select SaCS Patterns for Establishing Safety Case: a safety case represents a means to systematically argue that a specific system design is sufficiently safe for its intended purpose. Different safety case patterns express different strategies for arguing safety. In order to select a suitable safety case pattern that offers relevant guidance on safety demonstration, the demonstration challenges that occur in a given context should be assessed and matched with the appropriate patterns that express their solution.

Pattern Composition – The purpose of this activity is to specify the use of a set of patterns within a composite pattern specification by the activities:

a) Specify Compositions of Patterns: a composite pattern is a structure of patterns defined according to the syntax of SaCS [13] where operators are used to combine patterns. A composite pattern may be detailed as a combination of basic patterns, composite patterns, or as a combination of basic and composite patterns.
b) Specify Instantiations of Patterns: a composite pattern may be annotated according to the syntax of SaCS [13] to indicate a specific application of the pattern. The specification of the instantiation of input and output parameters of a pattern expresses the instantiation of the pattern.

Pattern Instantiation – The purpose of this activity is to systematically instantiate a set of patterns. A user may choose whether to perform pattern composition prior to pattern instantiation or vice versa. Pattern instantiation is performed by the activities:

a) Select Pattern Instantiation Order: If the set of patterns to be instantiated consists of only one basic pattern or a composite pattern consisting of one basic pattern there is no need to select instantiation order. Given a set with more than one pattern, the user must deduce the instantiation order by checking whether there are any dependencies between the patterns in the set, e.g., investigate if a pattern requires an input that is provided as a result of instantiating another pattern. Dependencies are modelled in a composite pattern by the use of relations. Guidance on the order of instantiating patterns may also be given within a composite by the annotations for indicating instantiation order.
b) Conduct Stepwise Instantiation: every basic SaCS pattern defines its parameters. Every composite pattern may be described as a structure of basic patterns combined with operators that acts on the parameters. Thus, when a composite pattern is instantiated, it is actually the basic patterns that are instantiated. The input parameters express the information required for pattern instantiation. The output parameters express the expected results of pattern instantiation. In order to instantiate a basic pattern, a user must confer the

respective pattern description and interpret the guidance provided with respect to the context it is applied.

A.2 The SaCS Pattern Language

There are six kinds of basic patterns available in SaCS. These are:

- *Process Assurance Requirement Pattern:* the pattern type is inspired by the recommended practises associated with the development of safety critical systems as given in safety standards, e.g., [EN50129] and [ED-12B], where confidence in achieving a safe product is to a large degree provided by following a recommended process.
- *Product Assurance Requirement Pattern:* the pattern type is inspired by the problem frames approach [Jackson 2001]. The problem frames approach is a means to specify the problem domains and interaction phenomena between problem domains in order to derive requirements for the system that is under development.
- *Process Assurance Solution (Method) Pattern*: the pattern type is defined as a means to document specific methods that represent common safety methods.
- *Product Assurance Solution (Design) Pattern*: the type is inspired by the well-known [Gamma et al. 1995] approach to describe design solutions.
- *Process Assurance Safety Case Pattern:* the type is defined as a means to document the structure of claims such that it may be logically deduced and concluded that a target system is sufficiently safe. The case depends primarily on process-oriented claims, arguments and evidences.
- *Product Assurance Safety Case Pattern*: the type is similar to the Process Safety Case type but the described safety case depends primarily on product-oriented claims, arguments and evidences.

In addition to the six kinds of basic patterns there is an additional type identified as *Composite Pattern*. The composite pattern type is provided as a means for a user to detail a specific integration of patterns. A composite pattern is specified by the use of a graphical notation where the basic building blocks are basic patterns. The SaCS graphical notation is outlined in appendix A.3.

Patterns are integrated in a composite pattern specification by the use of relations between the parameters of a source pattern and the parameters of a target pattern. The following parameter types are provided in SaCS:

- *Requirement*: represents a parameter that shall be recognised as a specification of requirements.
- *Design*: represents a parameter that shall be recognised as a specification of a design.
- *Safety Case*: represents a parameter that shall be recognised as a specification of a safety case.
- *Documentation*: represents a parameter that is not required to be strongly typed in SaCS (explicit data types are requirements, design and safety case) but represents data that is required to be documented.

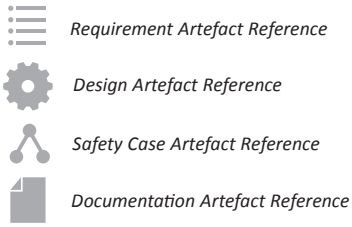
[EN50129] CENELEC, Railway Applications – Communication, Signalling and Processing Systems – Safety Related Electronic Systems for Signalling, EN50129, European Committee for Electrotechnical Standardization, 2003.

[ED-12B] EUROCAE, Software Considerations in Airborne Systems and Equipment Certification, ED-12B, European Organisation for Civil Aviation Equipment, 1992.

[Jackson 2001] M. Jackson, Problem Frames – Analysing and Structuring Software Development Problems, Addison-Wesley, 2001.

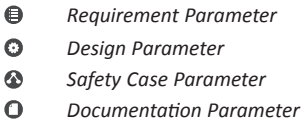
A.3 THE SACS GRAPHICAL NOTATION

The following icons are used to identify the different types of artefacts that may be referred to within a composite SaCS pattern.



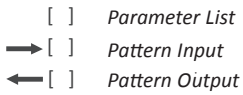
The parameters of a pattern indicate what artefacts are required to be provided or what artefacts are produced upon pattern instantiation. An artefact reference is used to give a reference to a concrete artefact that represents the instantiation of a parameter. In a diagram, an artefact is related to the parameter it instantiates by a dotted drawn line.

The following icons are used to identify the type of a parameter.



In a composite pattern, it is optional to include an icon symbolising the type of a parameter. The icons are simply smaller versions of the icons for symbolising artefacts, but represented in white and placed inside a black circle. An icon is placed adjacent to an identifier for a parameter in order to symbolise its type.

Parameters are always visualised inside a parameter list. A parameter list is visualised by square brackets, where the parameters are listed inside the square brackets. If the list contains more than one parameter, then parameters are separated by comma.



An arrow pointing towards a parameter list indicates that the parameters listed inside the square brackets are input parameters (i.e., input to the pattern to which the parameter list is placed adjacent to). An arrow pointing away from a parameter list indicates that the parameters inside the parameter list are output parameters (i.e., output of the pattern to which the parameter list is placed adjacent to).

A parameter may represent a compound entity consisting of several elements. It is possible to detail the elements of a parameter by a notation for indicating a set. It is also possible to create an alias for a parameter.

{ } *parameter set*
parameter alias

The *set* annotation is used to denote that the elements listed inside the curly bracket are elements of a set.

The *alias* operator is right-associative; an identifier to the left of the operator represents an alias for any parameter or set indicated to the right of the operator.

A reference to seven different types of patterns may be given within the definition of a composite SaCS pattern, where six of these are references to basic SaCS patterns and the seventh type may be used to reference composite SaCS patterns.

The following icons identify the three different types of process assurance patterns that may be referenced within a composite SaCS pattern.


Process Assurance
Requirement Pattern Reference


Process Assurance
Solution Pattern Reference


Process Assurance
Safety Case Pattern Reference

The following icons identify the three different types of product assurance patterns that may be referenced within a composite SaCS pattern.


Product Assurance
Requirement Pattern Reference

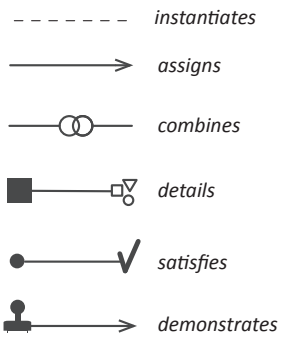

Product Assurance
Solution Pattern Reference


Product Assurance
Safety Case Pattern Reference

The following icon is used when referring to a composite pattern.


Composite Pattern Reference

Relations are used to model a relationship between an artefact and a parameter. Relations can also be used to model a relationship between two patterns in the form of a relationship between the respective parameters of the related patterns. The following icons identify the different types of relations in SaCS.



The *instantiates* relation is used to associate an artefact with a parameter indicating that the artefact instantiates the parameter.

The *assigns* relation is used to denote that one or more output parameters of a pattern is assigned to one or more input parameters of a related pattern. The arrow always points towards the input parameters.

The *combines* relation is used to denote that the outputs of the patterns that are related are combined, the result is a set consisting of the union of all outputs. Optionally a parameter list may be placed adjacent to the icon in the middle of the relation symbol to denote the result of combining.

The *details* relation is used to denote that an output of a pattern is detailed by the output of a related pattern. The black box is associated with the output that is detailed; the set of smaller icons is associated with the output that details.

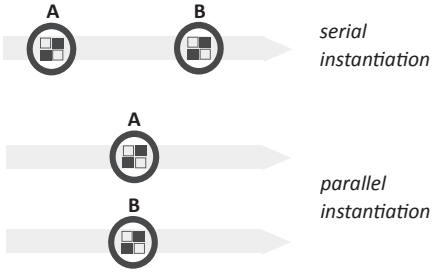
The *satisfies* relation is used to denote that an output of a pattern (typically an instantiation of a requirement parameter, which is a requirement artefact) is satisfied by the output of a related pattern (typically an instantiation of a design parameter, which is a design artefact). The bullet is associated with the output that describes what shall be satisfied (e.g., a requirement), while the checkmark is associated with the output that describes the fulfilment (e.g., describes a system design in accordance with requirements).

The *demonstrates* relation is used to denote that an output (typically an instantiation of a safety case parameter, which is a safety case artefact) demonstrates safe the output (typically an instantiation of a design parameter, which is a design artefact) of a related pattern. The stamp icon is associated with the output that provides a safety demonstration; the arrow points towards the output that is demonstrated safe.

A composite pattern consists of a declaration followed by its content. The declaration is recognised by an icon similar (but larger) to the one in a composite pattern reference. The declaration also contains a specification of the inputs and outputs of the composite. A line at the bottom of the declaration denotes the end of the declaration, and the content of the composite is visualised below the line.

The content of a composite pattern consists of a description of a combination of patterns by the use of the different graphical elements described in this appendix (e.g., pattern references, parameters, artefacts references, and relations).

Guidance may be given on the intended instantiation order of patterns. Superimposing pattern reference icons on top of a grey arrow indicates the recommended instantiation order.



The direction of the arrow indicates the pattern instantiation order, patterns placed closer to the starting point of the arrow is instantiated prior to patterns placed close to the tip of the arrow. Patterns may have no specific order, then this is visualised as a parallel instantiation.

In the example of the serial instantiation, two composite pattern references *A* and *B* are superimposed in a sequence on a grey arrow indicating that *A* should be instantiated before *B*.

In the example of the parallel instantiation, each of the two composite pattern references *A* and *B* are superimposed on separate arrows, indicating that *A* and *B* may be instantiated in parallel.

APPENDIX B THE PATTERNS

Table 10 provides an overview of the available basic patterns in SaCS. The last column identifies the appendix where each pattern is described.

The patterns in their current version represent best practices established as pattern descriptions inspired by the following safety standards and guidelines:

- Representing the railway domain:
  - EN50126: Railway Applications – The Specification and Demonstration of Reliability, Availability, Maintainability, and Safety (RAMS), European Committee for Electrotechnical Standardization, 1999.
  - EN50129: Railway Applications – Communication, Signalling and Processing Systems – Safety Related Electronic Systems for Signalling, European Committee for Electrotechnical Standardization, 2003.
  - EN50128: Railway Applications – Communication, Signalling and Processing Systems – Software for Railway Control and Protection Systems, European Committee for Electrotechnical Standardization, 2001.
  - ERA/GUI/01-2008/SAF: Guide for the application of the Commission Regulation on the adoption of a common safety method on risk evaluation and assessment as referred to in Article 6(3)(a) of the Railway Safety Directive, European Railway Agency, 2009.
- Representing the aviation domain:
  - ED-12B: Software Considerations in Airborne Systems and Equipment Certification, European Organisation for Civil Aviation Equipment,1992.
  - ED-79: Certification Considerations for Highly-Integrated or Complex Aircraft Systems, European Organisation for Civil Aviation Equipment, 1996.
  - ED-109: Guidelines for Communication, Navigation, Surveillance, and Air Traffic Management (CNS/ATM) Systems Software Integrity Assurance, European Organisation for Civil Aviation Equipment, 2002.
- Representing the nuclear domain:
  - IEC61226: Nuclear Power Plants – Instrumentation and Control Systems Important to Safety – Classification of Instrumentation and Control Functions, International Electrotechnical Commission, 2009.
  - IEC60880: Nuclear Power Plants – Instrumentation and Control Systems Important to Safety – Software Aspects for Computer-Based Systems Performing Category A Functions, International Electrotechnical Commission, 2006.
  - IEC62138: Nuclear Power Plants – Instrumentation and Control Systems Important for Safety – Software Aspects for Computer-Based Systems Performing Category B or C Functions, International Electrotechnical Commission, 2004.

Table 10 Summaries of Patterns

Product Assurance Solution (Design) Pattern		
Name	**Short Description**	**Ref.**
Trusted Backup	Specifies a design solution where an adaptable controller operates in a limited and safe part of a partitioned state space	B.5
Process Assurance Solution (Method) Pattern		
Name	**Short Description**	**Ref.**
FMEA	Captures the Failure Mode and Effects Analysis method	B.2
FTA	Captures the Fault Tree Analysis method	B.3
I&C Functions Categorisation	Captures the method for classification of nuclear power plant systems according to their importance to safety as defined in IEC61226	B.4
Product Assurance Requirement Pattern		
Name	**Short Description**	**Ref.**
Variable Demand for Service	Concerns the elicitation of requirements for a control system that shall control a nuclear power plant production of electricity to meet a variable demand	B.1
Process Assurance Requirement Patterns		
Name	**Short Description**	**Ref.**
Establish Concept	Concerns the process of establishing the purpose of the system and any constraints	B.6
Hazard Identification	Concerns the process of identifying any hazards applicable to the operation of the system	B.7
Hazard Analysis	Concerns the process of identifying potential causes of hazards	B.8
Risk Analysis	Concerns the process of assessing the severity of accidents should hazardous event occur and assessment of the likelihoods of such events	B.9

Establish System Safety Requirements	Concerns the process of specifying safety requirements on the basis of proposed risk mitigations	B.10
Process Assurance Safety Case Pattern		
Name	**Short Description**	**Ref.**
Overall Safety	Demonstration of system safety by diverse means applied on individual system parts	B.11
Process Quality Evidence	Demonstration of claim support obtained on the basis of a high quality process	B.18
Process Compliance Evidence	Demonstration of claim support obtained by demonstrating compliance to an acknowledged process	B.19
Product Assurance Safety Case Patterns		
Name	**Short Description**	**Ref.**
Technical Safety	Demonstration of a safe system by addressing the technical aspects	B.12
Code of Practice	Demonstration of system safety by means of applying well proven practices that implies required quality	B.13
Cross Reference	Demonstration of system safety by implied strategy to similar reference system that is already approved	B.14
Explicit Risk Evaluation	Demonstration of system safety by addressing all risk explicitly	B.15
Safety Requirements Satisfied	Demonstration of system safety by addressing all safety requirements	B.16
Assessment Evidence	Demonstration of claim support obtained on the basis of assessment	B.17
Deterministic Evidence	Demonstration of claim support obtained by a deterministic approach	B.20
Probabilistic Evidence	Demonstration of claim support obtained on the basis of a probabilistic approach	B.21

Basic Assumption Evidence	Demonstration of claim support based on rationale and/or indications such that no further evidence is required	B.22

B.1 VARIABLE DEMAND FOR SERVICE

Name: Variable Demand for Service

Pattern Signature: *Variable Demand for Service* is defined with the signature illustrated in Figure 30.

In Figure 30, the following abbreviations are used for denoting the parameters of the pattern:

- *Plnt* is short for Plant.
- *Mch* is short for Machine.
- *Obj* is short for Objective.
- *Req* is short for Requirements.

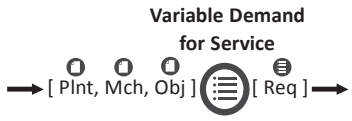


Figure 30 Variable Demand for Service – Pattern Signature

Intent: Support the specification of requirements *Req* for a control system that is required to adapt its control function in order to deliver service according to demand when there is a variable demand for the service and the conditions for delivering service changes. The plant *Plnt* that is controlled is a Nuclear Power Plant (NPP). The service provided by the NPP is electricity production. The control system under construction is expected to inflict NPP control in such a manner that objectives *Obj* of providing service as demanded is satisfied by an adaptable control function.

Applicability: The *Variable Demand for Service* pattern is suitable to apply in the following situations:

- When there is expected changes in the demand for service provided by a NPP where the change in demand is of an unpredictable nature;
- When the expected change in demand is intended to be accommodated with an adaptable control function in order to optimize electricity production according to demand at all times with minimal cost (e.g. production cost, waste production).

Problem: The main aspects relevant for establishing functional requirements for the adaptable control function are:

- *Nature of Change*: What changes in demand may be expected from the environment that shall be addressed by the adaptable control function.
- *Objective*: What is the objective(s) for introducing an adaptable control function and is an adaptable control function the most suitable means for satisfying the objective(s).
- *Conflicting Objectives*: Given multiple objectives, if there exist conflicts between the possible fulfilments of objectives, what characterises when one objective should be pursued before another.
- *Monitoring*: What characterises the change(s) with respect to available input data.

- *Detection*: What are the input patterns that indicate a non-optimal control function that may be used to trigger adaptation of the control function. If there exists no direct input for detecting change, a challenge is to define an aggregated value formed by basic input data such that a significant change is positively detected and false positives are eliminated.
- *Response:* How shall the system respond to change by adapting the control function. If the change impacts on the ability to satisfy several objectives, how shall it be evaluated which objective are the most significant. Given that the control function is adapted, a challenge is to assured that service is still provided although adaptation of the control function is performed.
- *Variability*: What part of the system is allowed to vary in order to provide the ability to adapt, e.g., only the control function.
- *Constraints:* What part of the control function is not allowed to vary and should always be maintained.
- *Timing*: When is adaptation of the function allowed: on demand, discrete steps, or continuous adaptation.
- *Timeliness*: What constraints are associated with the time that may be used for adapting the control function.

Problem Frame Analysis Solution: Figure 31 illustrates the *Variable Demand for Service* problem frames diagram.

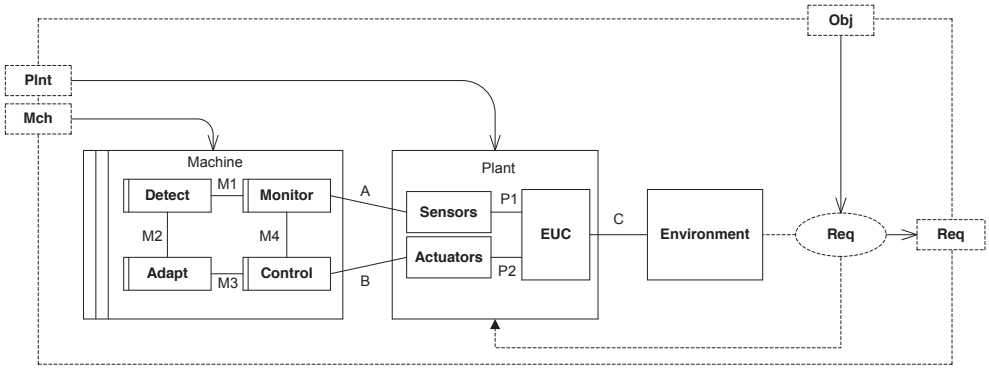


Figure 31 - Variable Demand for Service - Problem Frame Diagram

The input parameters associated with the problem frame diagram shall be interpreted as:

- *Obj:* represents the objective that is addressed and which the requirements of the requirement set *Req* should support;
- *Plnt:* represents the plant that is controlled, the plant is here decomposed into *Sensors* and *Actuators* and *EUC* (Equipment Under Control) where EUC represents the remaining physical entities besides sensors and actuators;
- *Mch*: represents the machine to be constructed or an existing system that is modernised and that controls plant.

The output expected by pattern instantiation is identified as:

- *Req:* represents the set of requirements derived on the basis of instantiating the pattern according to the defined instantiation rule.

The problem domains that are represented in the problem frame diagram shall be interpreted as:

- *Environment:* represents an entity (e.g. a power grid) demanding service (e.g. electricity) produced by *Plnt.* In the scenario addressed in this pattern, the environment always requires service but at a variable quantity over time. Further it is assumed that the quantity of service may not be predicted at any given time (e.g. electricity demand) thus rapid response to change in demand is required. The change in demand is expected to be accommodated by an altered interaction between the *Plnt* and the *Environment* (e.g., the electricity provided by *Plnt* to *Environment*). The *Machine* is expected to adapt its control of the *Plnt* such that the service provided by *Plnt* to *Environment* is optimised in order to fulfil objectives with minimal waste production.

The main challenges associated with the fulfilment of an objective *Obj* arise in the interaction between entities in the following interfaces:

- *A:* represents how the *Machine* becomes aware of the change in demand for service where:
  - *Sensors* provide signals on the status of the *Plnt*;
  - *Monitor* receives sensor signals such that the *Machine* may process these.
- *B*: represents how the *Machine* responds to an altered demand for service where:
  - *Control* sends control signals to the *Actuators* in order to affect *Plnt* behaviour;
  - *Actuators* act according to the behaviour demanded by control signals of the *Controller.*
- *C*: represents how *Plant and Environment* interact where:
  - *Plant* provides service (e.g., electricity) to *Environment*;
  - *Environment* provides a demand for service (e.g., demand for electricity) to *Plant*.

Requirements for the *Machine* may be elicited with respect to four functions. The identified functions denote neither physical parts nor any software solution, but denote designed problem domains that are addressed separately, these are:

- *Monitor:* represents the functionality for interacting with the sensors of a plant;
- *Detect:* represents the functionality for detecting the need to adapt the control function;
- *Adapt:* represents the functionality for adapting the control function (represented by *Control)* based on data provided by the function for detecting the need to adapt (represented by *Detect);*
- *Control:* represents the functionality for controlling a plant (represented by *Plant*) by sending commands to its actuators (represented by *Actuators*).

The four functions are addressed with respect to the challenges arising in the interface between functions denoted as*:*

- *M1*: representing the interface between *Monitor* and *Detect*.
- *M2*: representing the interface between *Detect* and *Adapt*.
- *M3*: representing the interface between *Adapt* and *Control*.
- *M4*: representing the interface between *Monitor* and *Control*.

There are two loops that are interesting:

- Control loop: The control loop is responsible for upholding the service. In Figure 31, the control loop is represented by the interactions:
  - Data on interface: *A* → Processing of data by: *Monitor* → Data on interface: *M4* → Processing of data by: Control → Data on interface: *B*.
- Adaptation loop: The adaptation loop is responsible for modifying the control function in order to improve service or reduce the cost associated with delivering the service. In Figure 31, the adaptation loop is represented by the following interactions:
  - Data on interface: *A* → Processing of data by: *Monitor* → Data on interface: *M1* → Processing of data by: *Detect* → Data on interface: *M2* → Processing of data by: *Adapt* → Data on interface: *M3* → Processing of data by: *Control* → Data on interface: *B*.

Table 11 provides support for elicitation of requirements with respect to the problem as outlined in Figure 31 and in the description of main interfaces and functions. A list of abstract requirements is given in Table 11, representing an abstract version of the requirement set identified as *Req.*

The following abbreviations are used in Table 11 and symbolises requirement attributes. A user shall give value to the attributes:

- act: one or more specific actuators
- actVal: one or more specific actuator values
- sig: one or more specific sensor signals
- sigVal: one or more specific sensor signal values
- mtdA: method for sensor signal acquisition (e.g., push or pull)
- mtdV: method for validation
- mtdP: method for defining input patterns
- mtdAd: method for adaptation of a control function
- ptrn: one or more sensor signal patterns
- rtd: represents a rate (e.g., x times pr. minute)
- tm: represents a period of time (e.g., x seconds)

Table 11 – Abstract Requirements

ID	Requirement	Note on instantiation
Function: Monitor **Interfaces:** A; M1		
R.1	*Mch* shall monitor *sig*	Detail each signal that is required to be monitored.
R.2	*Mch* shall acquire *sig* by *mtdA*	For each signal: detail how it shall be acquired.
R.3	*Mch* shall acquire *sig* at a rate *rtd*	For each signal: detail the rate at which it should be acquired.
R.4	*Mch* shall validate s*ig by mtdV*	For each signal: detail how it shall be validated.

Function: Detect **Interfaces:** M1; M2		
R.5	*Mch* shall detect *ptrn*	Detail each signal pattern (combination of signals) that shall be detected and which provides indication on non-optimal control function.
R.6	*Mch* shall acquire *ptrn* by *mtdP*	For each pattern: detail how it shall be acquired on the basis of a specific combination of signals.
R.7	*Mch* shall acquire *ptrn* at a rate *rtd*	For each pattern: detail the rate at which it should be acquired.
R.8	*Mch* shall validate s*ig by mtdV*	For each pattern: detail how it shall be validated.
Function: Adapt **Interfaces:** M2, M3		
R.9	*Mch* shall modify the control function with mtdAd when *ptrn* is detected	For each combination of mtdAd and ptrn, detail explicit requirements.
R.10	*Mch* shall adapt the control function at a rate no less than *rtd*	Detail how often adaptation shall be performed.
R.11	*Mch* shall perform an adaptation of the control function no longer than *rtd*	Detail constraints with respect to time for performing an adaptation of the control function.
R.12	*Mch* shall validate an adaptation by *mtdVal*	Detail how it shall be assured that the adaptation is correct.
Function: Control **Interfaces:** M3, M4, B		
R.13	*Mch* shall *control act*	Detail each actuator that shall be controlled.
R.14	*Mch* shall actuate *act* with actV*al* when *sig* is sigV*al* else actVal	Detail all combinations of values for input signals that lead to a specific actuation of actuators.
R.15	*Mch* shall actuate *act* within *tm* of detection of *sig* with *sigVal*	Detail time constraints for actuation upon detection.
R.16	*Mch* shall perform *act* within *time*	Detail time constraints for completing actuation.

Instantiation Rule: An artefact *Req* (see Figure 31 and Figure 30) instantiates the *Variable Demand for Service* pattern if:

- *Req* is a set of requirements.
- Every requirement of *Req* is an instance of an abstract requirement. Abstract requirements are defined in column “Requirement” of Table 11. An abstract requirement is instantiated by applying the guidance provided in the column “Notes on instantiation” described in Table 11.
- Every requirement of *Req* is traceable to a unique abstract requirement.
- Every requirement of *Req* describes a property, behaviour or a constraint of the system (identified by the instantiation of *Mch).*
- Every requirement of *Req* is relevant for the satisfaction of at least on objective (identified by the instantiation of *Obj).*

Related Patterns: The *Variable Demand for Service* pattern is related to other patterns in the following manner:

- May succeed *Establish Concept* given that its instantiation produces a description of goals and functions relevant to detail further by the use of *Variable Demand for Service.*

B.2 FMEA

Name: FMEA (Failure Modes and Effects Analysis)

Pattern Signature: *FMEA* is defined with the signature illustrated in Figure 32.

In Figure 32, the following abbreviations are used for denoting the parameters of the pattern:

- *ToA* is short for Target of Assessment.
- *Tbl* is short for Table

Figure 32 FMEA – Pattern Signature

Intent: Provide guidance on the application of the Failure Modes and Effects Analysis (FMEA) method. The intent of an FMEA is to identify failure modes that may lead to system failure or some target *ToA* under evaluation. The FMEA result is commonly summarised in a table *Tbl*. The pattern may be used as support for qualitative or quantitative analysis of an item(s), component(s), subsystem(s), or a system(s). FMEA's may be combined, e.g., individual assessment of components may represent a component level FMEA assessment that provide input to a subsystem level FMEA. In the case where FMEA's build upon each other, the identified effects of a component level FMEA may represent the failure modes in a subsystem level FMEA.

Applicability: The *FMEA* pattern is suitable to apply in the following situation:

- When a system, subsystem, component or item needs to be assessed with respect to:
  - Potential failure modes.
  - Likely cause(s) of failure modes.
  - Likely effect(s) of the occurrence of a failure mode.

Problem: The main challenges associated with methods for assessment of the potential failure modes are:

- *Efficiency*: To support efficient application such that results are provided with minimal use of resources.
- *Structured*: To support a structured assessment and of the characteristics of a target such that the potential for not delivering its function as intended is identified along with its likely causes and system effects.
- *Communication*: To support communication of results on a form that allows different actors to understand findings with minimal effort.

Method Solution: Figure 33 illustrates the *FMEA* method pattern diagram. The following steps may apply the FMEA method systematically:

- Step 1 – Identify Item (Item): What is the item that is analysed. Is the target of assessment *ToA* one specific item or may the target be decomposed to a set of items.

- Step 2 – Identify Function (Func): For each item in the table, describe its associated functions (Func).
- Step 3 – Define Failure Mode (F. Mode): For each identified function, describe the potential failure modes.
- Step 4 – Define Failure Cause (F. Cause): For each failure mode, describe the likely failure cause.
- Step 5 – Define Local Effect (L. Effect): for each failure mode, describe the local effect, meaning what is the effect on operation of the item that is analysed.
- Step 6 – Define System Effect (S. Effect): for each failure mode, describe the likely system effect, meaning what is the likely effect of the failure of the item in the context it is intended to be used.
- Step 7 – Define Severity (Sev.): Define the severity of effects identified in step 6.
- Step 8 – Identify Failure Rate (F. Rate): Identify the expected rate of occurrence associated with the failure mode. This may be given qualitatively or quantitatively depending on the nature of the item analysed and availability of data and methods for estimation of occurrences.
- Step 9 – Define Recommended Action (R. Act.): Define recommended action in order to avoid fault, detect and isolate the fault given an occurrence, or otherwise tolerate the fault.

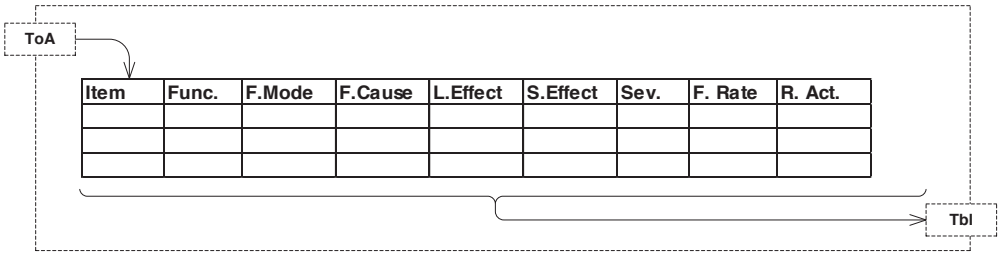


Figure 33 Method Pattern Diagram – FMEA example

Instantiation Rule: An artefact *Tbl* (see Figure 33 and Figure 32) instantiates the *FMEA* pattern if:

- *Tbl* is a tabular documentation of the assessment results from applying the steps described in Section “Method Solution”.
- Every row of *Tbl* contains a description of a potential failure mode of the target that is assessed (identified by the instantiation of *ToA*).
- Every row of *Tbl* contains a description of the likely effects of a failure mode on the target that is assessed (identified by the instantiation of *ToA*).

Related Patterns: The *FMEA* pattern is related to other patterns in the following manner:

- May support *Hazard Identification* pattern by providing a method for performing identification of potential causes of hazards, the information provided in the system effects column of the FMEA table identify system level hazards.
- May support *Hazard Analysis* pattern by providing a method for identification of failure modes and potential causes of hazards.
- May support *Risk Analysis* pattern by providing a method for performing qualitative and quantitative analysis of hazards.

Known Uses: failure modes and effect analysis is a commonly applied method within safety critical domains and is described in several standards and guidelines, e.g. nuclear domain [NUREG–0492], aerospace [ARP4761], and the more general applicable standard [IEC60812].

[NUREG–0492] NRC, Fault Tree Handbook, U.S. Nuclear Regulatory Commission, 1981.

[ARP4761] SAE, Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment, Society of Automotive Engineers, 1996.

[IEC60812] IEC, Analysis techniques for system reliability – Procedure for failure mode and effects analysis (FMEA), Edition 2.0, International Electrotechnical Commission, 2006.

B.3 FTA

Name: FTA (Fault Tree Analysis)

Pattern Signature: *FTA* is defined with the signature illustrated in Figure 34.

In Figure 34, the following abbreviations are used for denoting the parameters of the pattern:

- *ToA* is short for Target of Assessment.
- *UE* is short for Unwanted Event.
- *FT* is short for Fault Tree

Figure 34 FTA – Pattern Signature

Intent: Support a systematic and deductive assessment of the potential causes of identified unwanted events *UE* of a target *ToA*. The pattern describes the fault tree analysis method and provides guidance on its application. A fault tree *FT* is expressed as a tree where the root of the tree denotes the unwanted event *UE* (e.g., representing a system hazard) that is analysed. Leaf nodes denote basic events (e.g., system component failure modes) that may lead to the unwanted event. The primary goal is to identify minimal cut sets by the use of the tree structure. A cut set expresses a set of basic events such that if these are present at the same time, the unwanted event will occur. A minimal cut set expresses the minimal set of basic events that may lead to an unwanted event.

Applicability: The *FTA* pattern is intended for the following situations:

- As a means to deduce potential causes of unwanted events.
- As a means to identify combinations of faults that may lead to unwanted events.
- As a means to analyse system concepts, simple systems or complex systems.

Problem: The main challenges associated with methods for assessment of the potential causes of an unwanted event are:

- *Efficiency*: To support efficient application with minimal use of resources.
- *Logical*: To support a logical specification of the dependencies between an unwanted event that is investigated and the potential causes that is identified through the assessment.
- *Communication*: To support communication of results on a form that allows different actors to understand findings with minimal effort.

Method Solution: Figure 35 exemplifies a fault tree and outlines the relationships between a fault tree and the inputs and outputs of the FTA pattern.

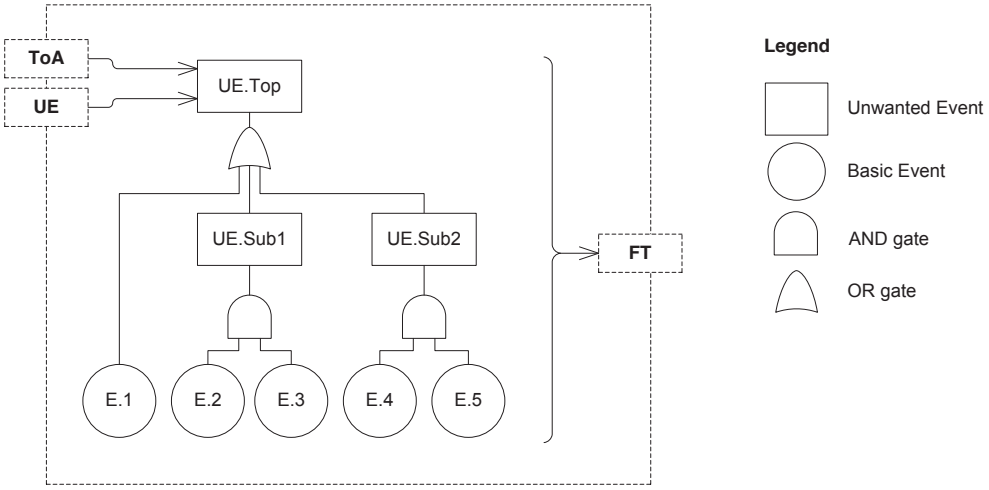


Figure 35 FTA example diagram

The following steps may apply the FTA method systematically:

- Step 1 – Define the top event: The intent is to define the top event that represent the root of a fault tree and the event that is analysed (UE.Top in Figure 35).
- Step 2 – Construct fault tree: The intent is to construct the fault tree by deductively constructing the tree in a top-down manner by the use of graphical elements symbolising events and gates. An unwanted event should state what is the fault/failure under consideration. Gates (e.g., the “OR” gate in Figure 35) are used to define the combination of antecedents (e.g., basic event E.1, UE.Sub1, or UE.Sub2) that imply the consequent (e.g., UE.Top). Decomposition of an unwanted event (e.g., UE.Top) should not be described by connecting a series of gates down to basic events, intermediate unwanted events should be defined (e.g., UE.Sub1 and UE.Sub2). Basic events answer “how” questions by defining how a system component or element may experience a fault/failure.
- Step 3 – Find minimal cut sets: The intent is to deduce from the tree structure the combination of basic events that may lead to the top event and arrange these in order. The set with lowest order (contains least number of basic events) is most significant. In Figure 35, the event UE.Top occurs if event E.1 occurs (minimal cut set of order 1). The top event also occurs if events E.2 and E.3 occur or if events E.4 and E.5 occur (cut sets of order 2). There are many cut sets of order 3, 4, and 5 but these are less significant.
- Step 4 – Qualitative Analysis: The intent is to analyse the fault tree qualitatively in order identify week points of the system with respect to the top event. This may be performed on the basis of cut sets. If there exists cut-sets of order 1 then the system is vulnerable to single point of failure. If there exists none cut sets of order 1 but the events in the cut sets of order 2 have identical characteristics, then the system may be susceptible to common cause failures.
- Step 5 – Quantitative Analysis: The intent is to calculate the probability of a top event by combining probabilities of basic events (given that probabilities of basic events may be provided). The probability of UE.Top in Figure 35 may be calculated by a combination of the following general rules:
  - The probability of an event A ‘’AND’’ an event B is expressed:

P(A*B) = P(A) * P(B)

- The probability of an event A ‘’OR’’ an event B is expressed:
  P(A+B) = P(A) + P(B) – (P(A) * P(B))

Instantiation Rule: A documentation artefact *FT* (see Figure 35 and Figure 34) instantiates the *FTA* pattern if:

- *FT* is a set of fault trees.
- Every fault tree of *FT* is instantiated by applying the guidance provided in Section “Method Solution” for all unwanted events (identified by the instantiation of *UE*). For a specific target *ToA*, this means that every unwanted event *UE* that represents a suitable top event in the analysis is systemically assessed according to the process outlined through Steps 1 to Step 5.
- Every fault tree of *FT* is traceable to a unique unwanted event (identified by the instantiation of *UE*).
- Every fault tree of *FT* describes the relation between an unwanted event, represented as a top node in the tree structure, and potential initiating events (e.g. HW/SW failures) associated with a target system (identified by the instantiation of *ToA*).

Related Patterns: The *FTA* pattern is related to other patterns in the following manner:

- May support *Hazard Analysis* pattern by providing a method for performing identification of potential causes of hazards.
- May support *Risk Analysis* pattern by providing a method for performing qualitative and quantitative analysis of hazards.

Known Uses: Fault tree analysis is a commonly applied method within safety critical domains and is described in several standards and guidelines, e.g. nuclear domain [NUREG–0492], aerospace [ARP4761], and the more general applicable standard [IEC61025].

[NUREG–0492] NRC, Fault Tree Handbook, U.S. Nuclear Regulatory Commission, 1981.

[ARP4761] SAE, Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment, Society of Automotive Engineers, 1996.

[IEC61025] IEC, Fault Tree Analysis (FTA), Edition 2.0, International Electrotechnical Commission, 2006.

B.4 I&C Functions Categorisation

Name: I&C Functions Categorisation

Pattern Signature: *I&C Functions Categorisation* is defined with the signature illustrated in Figure 36.

In Figure 36, the following abbreviations are used for denoting the parameters of the pattern:

- *ToA* is short for Target of Assessment.
- *FncCat* is short for Function Categorisation

Figure 36 I&C Functions Categorisation – Pattern Signature

Intent: Provide guidance on the classification of functions *FncCat* of a target *ToA* where the target is a nuclear power plants instrumentation and control system and the functions are classified according to their importance to safety. Functions important to safety are typically distributed over several system and subsystems. The intent of the categorisation is to: provide a means for assigning requirements for functions in a consistent manner; provide a means to differentiate requirements for development of functions according to their potential impact on safety. The classification of functions allows detailed requirements to be associated with different categories of functions. The classification represents a means to categorise a system with respect to the importance of its functions on safety. The result *FncCat* provided upon pattern instantiation may be used as input to activities on the refinement of the system design, e.g., as input to the modularisation of a system separating critical functions from non-critical functions into different sub-systems. The result may also be used in order to plan the optimal the allocation of resources, e.g., assigning more resources to the development of critical functions than none-critical functions.

Applicability: The *I&C Functions Categorisation* pattern is suitable to apply in the following situations:

- When developing an instrumentation and control system within the nuclear power plant domain.
- When the categorisation of functions shall be performed according to [IEC61226].

Problem: The main challenges associated with categorisation of the functions of critical systems are:

- *Criticality*: provide categorisation of systems based on an evaluation of the criticality of the functions it offers.
- *Conformity*: provide categorisation of systems such that systems of similar category may be associated with the same set of rules.

Method Solution: Figure 37 illustrates the *I&C Functions Categorisation* method pattern diagram. The following steps may apply the method systematically:

- Step 1 – Identify design basis: Collect data detailing the nature of the plant (e.g., reactor type), I&C design (e.g., detailing redundancy of mechanical and electrical systems and equipment), data on PIE[3] (Postulated Initiating Event) associated with the plant and their main mitigating functions and support functions.
- Step 2 – Define functions: Identify and define the functions that shall be performed by the system, e.g., reactor shutdown, post-accident monitoring, monitoring and controlling performance of individual systems or items or access control. Identify functions and requirements for functions in accordance with [IEC60964].
- Step 3 – Category Assignment: The intent is to categorise each function identified in Step 2. The applicable categories are denoted A, B, or C. Functions that may not be assigned to any of these three categories are denoted non-classified. The relationship between the characteristics of a function and category may be shortly summarised as:
    - A: Denote functions that play a principal role in the achievement and maintenance of plant safety. May also be assigned to functions whose failure could directly lead to accident conditions.
    - B: Denote functions that play a complementary role to category A functions in the achievement and maintenance of plant safety. May also be assigned to functions whose failure could initiate design basis event or worsen the severity of design basis event;
    - C: Denote functions that play an auxiliary or indirect role in the achievement or maintenance of plant safety. Category C includes functions that have some safety significance, but are not category A or B.
- Step 4 – Detail system requirements: Detail requirements for the system that shall provide the implementation of the functions identified in Step 3.
- Step 5 – Identify subsystems and items: Identify subsystems, components, items that shall fulfil the requirements identified in Step 4.
- Step 6 – Refinement of category assignment: Once the system design matures such that redundancy, diversity and other technical requirements of the functions are determined more exactly (e.g., on the basis of safety assessment) the classification list shall be refined and revised.
- Step 7 – List functions and assigned categories: The final list of functions with an associated category for each function shall be documented and maintained under configuration control.

[3] PIE - event identified during design as capable of leading to anticipated operational occurrences or accident conditions, ref [IEC61226]

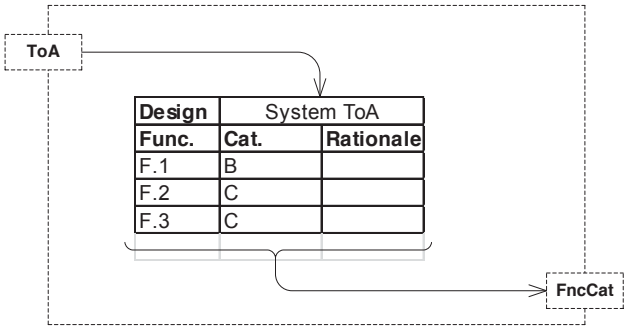


Figure 37 Method Pattern Diagram – I&C Functions Categorisation example

Instantiation Rule: A documentation artefact *FncCat* (see Figure 37 and Figure 36) instantiates the *I&C Functions Categorisation* pattern if:

- *FncCat* is a documentation of the results from a classification of the functions of a target system (identified by the instantiation of *ToA)*.
- *FncCat* represents the intermediate results from applying the steps 1 to 3 described in Section “Method Solution” or the full results from applying steps 1 to 7 in Section “Method Solution”.
- Every function described in *FncCat* is categorised as an A, B, or C function according to categorisation given in [IEC61226] or otherwise is not categorised.

Related Patterns: The *I&C Functions Categorisation* pattern is related to other patterns in the following manner:

- May support *Risk Analysis* pattern by providing a categorisation scheme for the categorisation of a system that is assessed with respect to risk.

Known Uses: The pattern represents the method for categorising systems within a nuclear context as defined in the standard [IEC61226].

[IEC61226] IEC, Nuclear Power Plants – Instrumentation and control important to safety – Classification of instrumentation and control functions, Edition 3.0, International Electrotechnical Commission, 2009.

[IEC60964] IEC, Nuclear Power Plants – Control Rooms – Design, Edition 2.0, International Electrotechnical Commission, 2009.

B.5 TRUSTED BACKUP

Name: Trusted Backup

Pattern Signature: The *Trusted Backup* is defined with the signature illustrated in Figure 38.

In Figure 38, the following abbreviations are used for denoting the parameters of the pattern:

- *S* is short for System (S is a design artefact that includes all the other design artefacts below).
- *AC* is short for Adaptable Controller.
- *AL* is short for Adaptation Logic.
- *CL* is short for Control Logic.
- *TC* is short for Trusted Controller.
- *CD* is short for Control Delegator.
- *M* is short for Monitor.

Figure 38 Trusted Backup – Pattern Signature

Intent: The intent of the pattern is to allow an adaptable controller *AC*, which might be of arbitrary assurance level, to control a safety related plant. The adaptable controller is here introduced in order to accommodate changes in the plant as a means to uphold service despite changes (e.g., adjust to plant degradation) or improve service over time (e.g., better fitting of control law). The intent is to divide the operational state space associated with some controlled plant into regions. An adaptable controller is in this control scheme not a trusted component, thus, it is only granted control privileges in those regions, called safety regions, where there are no negative effects of controller failure. In order to provide service in those regions, which are not safety regions, or take over control if the adaptable controller experiences failure, a redundant trusted controller *TC* is granted control privileges. The controllers are developed according to different principles in which the trusted controller guarantees safe control, but is less resilient to changes in the plant, whereas the adaptable controller accommodates changes and can optimise accordingly, but do not guarantee safe control. The system *S* controls the plant in a manner that emphasizes both optimal and safe control, a control delegator *CD* delegates control privileges to the most suitable controller depending on the state of the plant.

The pattern employs a strategy that facilitates the utilisation of adaptable control for optimising critical control functions, but where effort required for demonstrating that the system is safe is to a large degree minimised. The adaptable controller is intended to provide an increase in performance in changing environments compared to a conventional software system. Erroneous operation of the adaptable controller should

have no negative impact on safety, thus the adaptive controller may be of an arbitrary assurance level.

Applicability: The *Trusted Backup* pattern is suitable to apply in the following situations:

- Control scenarios where there is a need for a high degree of exploration of alternative adaptive control solutions, e.g., prototyping or upgrade scenarios. As the safety of the system is not ensured by the adaptive control component, the adaptive controller or the adaptation logic may be rapidly changed.
- When there is an existing trusted control system that lacks the ability to accommodate changes. Adding the adaptable controller, safety monitoring, and control delegation functionality may provide additional adaptable control functionality.

Problem: The main aspects to be considered when establishing an adaptable and critical system concept are:

- *Resilience and flexibility*: the benefit of an adaptable system is that it provides the ability to respond to internal (e.g., component failures) and external changes (e.g., changes to the operating environment) with resilience and flexibility. An adaptable system may be particularly useful to handle changes that occur in an unpredictable manner. If the change is not of an unpredictable nature, other techniques may be a better solution. It is essential that the nature of the change that shall be accommodated by adaptability be understood such that the adaptable feature is effective.
- *Determinism*: for each adaptation of a system, a new system is provided. In order to argue that a system is, and will continue to be, sufficiently safe for its purpose, it is essential that its behaviour may be determined. It is not possible to determine the behaviour of an adaptable system unless all possible inputs may be determined and all potential adaptations of the system may be determined. As it is not feasible to determine the adaptable system behaviour in all potential operational scenarios, determinism must be provided in those states that may impact on the safety property. It must be guaranteed that the system is sufficiently safe for its intended purpose.

Design Solution: Figure 39 illustrates the main structure of components and their interfaces, annotated in UML notation, which play a part in obtaining a *Trusted Backup* design.

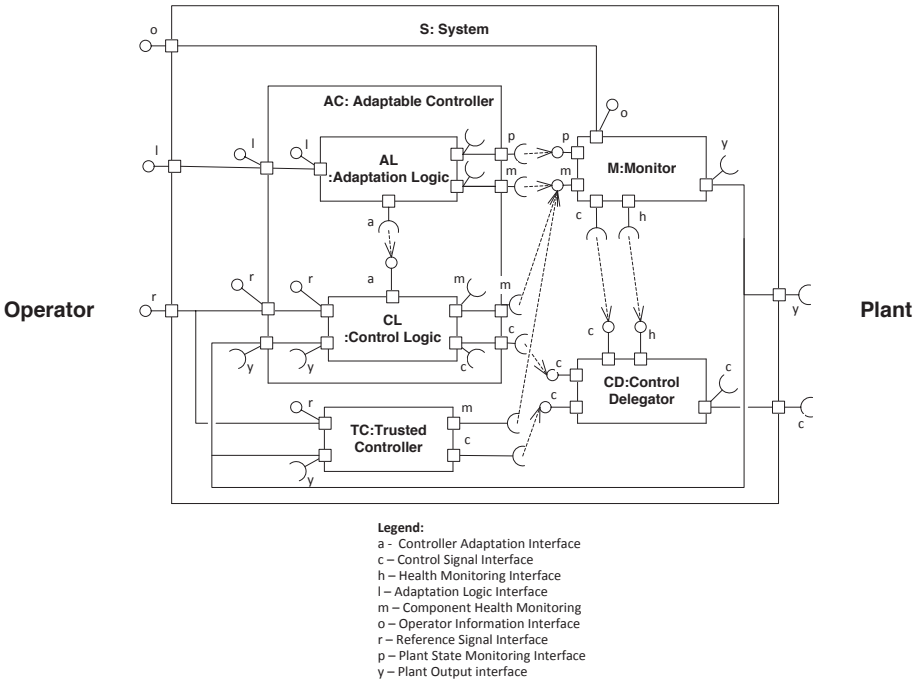


Figure 39 - Structure

Delegation of control is governed by the *Control Delegator* part that based on received/requested system and plant state information grant control privileges to one of its delegates. Given that there are not detected any failures to any of its delegates, the *Control Delegator* delegates control according to a defined *Delegation Scheme*.

Controller adaptation initiates in response to the result of the detection and analysis of change provided by the *Monitor*. Once the need to adapt is identified, the *Adaptation Logic* adapts the *Control Logic* accordingly. In order to assure a safe control process, the *Adaptable Controller* is deactivated for the period of time when this controller is modified. The *Trusted Controller* will then continuously provide service and thus assure safety.

The following participants and their respective responsibilities are represented in the Trusted Backup pattern:

- *S: System* – represents the system defined by the Trusted Backup pattern. The system interacts with a human operator and controls a plant.
- *AC: Adaptable Controller* – is here decomposed into an *Adaptation Logic* part and a *Control Logic* part.
- *CL: Control Logic* – responsible for providing control signals based upon the current plant state and reference data points.
- *AL: Adaptation Logic* – is responsible for accommodating changes to the plant or the environment in which the plant operates by adapting the *Control Logic*. It is here assumed that the absence of side effects due to adaptation may not be adequately evaluated, thus a high level of trust may not be awarded the control logic or the adaptation logic. The *Adaptation Logic* is here provided as an

external adaptation mechanism (external to the respective controller identified as *Control Logic*).

- *TC: Trusted Controller* – is responsible for providing control signals based upon the current plant state and reference data points. It is assumed that it may be demonstrated that the *Trusted Controller* always operates in a correct and sufficiently safe manner.
- *CD: Control Delegator* – is responsible for delegation of control such that control privileges are granted to the most suitable controller of a set of alternative controllers in such a manner that plant safety is always assured and secondly that the most optimal controller for a given situation is always sought. All controllers operate in parallel where a set of delegation rules define which controller is granted control privileges in each state, the remaining controllers being suppressed.
- *M: Monitor* – is responsible for monitoring the plant, monitoring the performance of the system, and identifying the need to adapt the adaptable controller to the plant and/or environmental changes.

The *Control Delegator* provides important functionality with respect to preserving system safety. It must be assured that it grants control privileges to the most suitable controller at all times. The pattern is suitable when there are no means of reverting to a previous stored plant state given a failure scenario, thus it would be important to support a forward recovery strategy. Given that an adaptable controller gives an erroneous control signal, the potential danger this signal inflicts must be mitigated from the current execution point. Means of achieving a forward recovery feature is provided here by the redundant controllers, health monitoring and conservative control delegation rules. Potentially hazardous control behaviour may arise if the *Control Delegator* grants control privileges to a non-trusted control component in a plant state where a worst-case erroneous control sequence may not be mitigated. With respect to providing guarantee of safe delegation of control, the following must be known: (1) the potential hazards associated with the plant; (2) worst case failure scenario and the failure modes of the adaptive controller, which may lead to identified hazards, and (3) the capability of the trusted controller with respect to stabilising a plant.

Software adaptations are intended to be limited to the adaptable controller only. The objective of the adaptable controller is to optimise control with respect to some performance goal, how this is achieved is of minor concern here, although a part named *Adaptation Logic* is assumed to contain the rules for how to adapt the *Control Logic* according to some identified need to adapt. The *Monitor* part is assumed to identify the need to adapt based on analysis of the plant and its environment.

Variability of the system is isolated to the *Control Logic* part of the *Adaptable Controller*. This *Control Logic* part might exist in various versions over time as the *Adaptation Logic* adapts the control function in response to plant or environmental events. A specification of the potential for system variability may thus be provided once the *Adaptation Logic* is specified.

There will be no variability of the system service in the part of the operating state space for which only the backup controller may operate as the backup controller is assumed to be a deterministic controller. Variability of the system service will be experienced in the part of the operating state space for which the adaptable controller may operate

and in those scenarios it is granted control privileges, as different versions of the Control Logic may provide different outputs when acting on the same input.

Instantiation Rule: An artefact *S* (see Figure 39 and Figure 38) instantiates the *Trusted Backup* pattern if:

- *S* is a specification of the hardware design and/or the software design of a control system.
- *S* specifies a system containing at least two diverse controller parts that operate in parallel.
- *S* specifies at least one adaptable controller and at least one trusted controller. By adaptable controller, it is meant a controller that can be automatically modified during operation. By trusted controller, it is meant a controller design that is able to be demonstrated adequately safe for its purpose.
- *S* specifies a system part that is responsible for delegating control privileges to (or switching between) the most suitable controller among a set of available controllers at any given time according to defined rules.

Related Patterns: The *Trusted Backup* design relates to the other patterns in the following manner:

- May be used together with any pattern that provides requirements for the behaviour of the system, or a detailing of the design of the system described.

Known Uses: A number of applications with adaptive control are referenced in [Åström and Wittenmark 1994]. One application is related to a chemical reactor control system where an adaptive controller was implemented on a reactor at Berol Kemi AB in 1982. As poor control may not only result in lowered production but also lead to potentially fatal events as explosions, it is vital to uphold the safety property of the plant. The system operator may switch between a conventional controller with manual mode tuning or an adaptive controller with automatic tuning to control the flow and temperature of the chemicals in the plant. Experiences showed that that temperature fluctuation was significantly reduced with the adaptive system. In addition, the operator could focus on other tasks as the time spent supervising reactor temperature was not necessary.

A representative example is described in [Schumann et al. 2006]. The authors refer to a NASA project called IFCS (Intelligent Flight Control System). The IFCS project utilised an on-line adaptive neural network in order to optimize aircraft performance during normal and adverse conditions. The neuro-controller was designed to enable a pilot to maintain control and safely land an aircraft that had suffered major systems failure or combat damage. Control surface failures may conflict with the design assumptions of an aircraft flight control system, with the effect that it is unable to handle the situation. The IFCS neuro-controller compensates for discrepancies between a reference model of the flight dynamics to any “new” flight dynamics model in order to maintain the best possible flight performance. The adaptive neural network software “learns” the new flight characteristics, on-board and in real time, thereby helping the pilot to maintain or regain control and prevent a potentially catastrophic aircraft accident.

The *Trusted Backup* pattern enables utilisation of adaptive control in a manner similar to the principles advocated in the Simplex architecture [Sha 2001]. Sha [Sha 2001] refer to the Boeing 777 flight control system as an example of a system that uses the ideas of the Simplex architecture in practise. The system uses two controllers, a

sophisticated “normal” controller that is customised for the Boeing 777 and a secondary controller that is simple and well proven through 25 years of use and based on the Boeing 747 control laws. The “normal” controller provides a higher level of performance and it must operate within the bounds given by the secondary controller, which assure safe flight control. The Boeing 777 case is comparable to the Trusted Backup pattern main line of though. Although an adaptable controller is not utilised in the Boeing 777, the “normal” controller represents a software controller that is an assessment and safety demonstration challenge. The secondary controller and the mechanism for switching among controllers is a means for providing safety assurance.

[Åström and Wittenmark 1994] K. J. Åström and B. Wittenmark, Adaptive Control, 2nd edition, Addison-Wesley, 1994.

[Schumann et al. 2006] J. Schumann, P. Gupta, and S. Jacklin. Toward Verification and Validation of Adaptive Aircraft Controllers. In IEEE Aerospace Conference. IEEE Press, USA, 2006.

[Sha 2001] L. Sha. Using Simplicity to Control Complexity. IEEE Software, 18:20-28, 2001.

B.6 Establish Concept

Name: Establish Concept

Pattern Signature: *Establish Concept* is defined with the signature illustrated in Figure 40.

In Figure 40, the following abbreviations are used for denoting the parameters of the pattern:

- *Env* is short for Environment.
- *Pur* is short for Purpose.
- *IdFunc* is short for Identification of Functions.
- *IdHuInt* is short for Identification of Human Interactions.
- *IdSyInt* is short for Identification of System Interactions.
- *IdHz* is short for Identification of Hazards
- *Conc* is short for Concept.

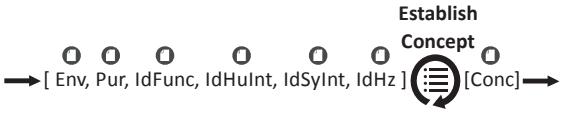


Figure 40 Establish Concept – Pattern Signature

Intent: Support the capture of an initial system concept description *Conc* in a systematic manner. The intent of the pattern is to define how an initial specification of a system concept shall be derived and documented by: i) outlining the main characteristics of the process of deriving a conceptual description; and ii) defining requirements for specifying the system concept. The pattern captures the problem of establishing the purpose *Pur* of the system under construction and its context *Env* of use to such a degree that enables subsequent conceptual development of a technical solution.

Applicability: The *Establish Concept* pattern is suitable to apply in the following situations:

- Prior to project initiation as a means to reflect upon the development challenges that shall be addressed.
- At the initial stages of conceptual development in order to systematically capture the main purpose and constraints applicable to the system under construction.

Problem: The main aspects to be considered when establishing the system concept are:

- *Capture the goals:* To clearly define the goals of the system under construction in order to define its purpose.
- *Define scope and context:* To clearly define any delimitation of the scope of operation and operating context.
- *Identify functions:* To clearly define the system functions and how these functions support the stated goals.

- *Identify interactions:* To clearly define the interactions that the system under construction has with the process/plant to be controlled, any human operator and any other system.
- *Identify know hazards:* To obtain on the basis of an early specification of a system concept, a high level definition of the potential hazards applicable to the system under construction.

Process Solution: Figure 41 illustrates the *Establish Concept* process annotated in a UML activity diagram.

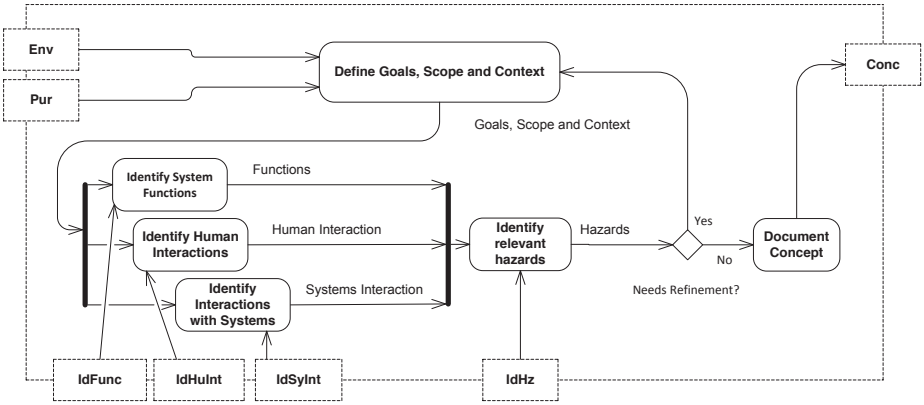


Figure 41 Establish Concept – Activity Diagram

The parameters associated with the activity diagram illustrated in Figure 41 may be interpreted as follows:

- *Env (Environment):* represents relevant data on the environment of the system under construction that may be used to determine influences that directly or indirectly affects the system.
- *Prp (Purpose):* represents relevant data providing information on goals, high-level requirements, functions or constraints that may be used as input to derive the purpose of the system under construction.
- *IdFunc (Identification of Functions):* represents the result from any method supporting the activity of deriving the functions of the system.
- *IdHuInt (Identification of Human Interactions):* represents the result from any method supporting the activity of identifying human interactions.
- *IdSyInt (Identification of System Interactions):* represents the result from any method supporting the activity of identifying interacting systems and their interactions.
- *IdHz (Identification of Hazards):* represents the result from any method supporting the identification of relevant hazards.

The activities identified in Figure 41 intend to serve the following purpose:

- *Define Goals, Scope, and Context:* based on available documentation, the intent of the activity is to clearly define the purpose of the system. The purpose of the

system is defined by establishing the goals to be achieved and by defining any operational and contextual constraints in the fulfilment of these goals.

- *Identify System Functions:* the intent of the activity is to define the high-level functions that shall be satisfied by the system under construction such that stated goals are fulfilled. Traceability between defined goals and system functions should be provided.
- *Identify Human Interactions:* the intent of the activity is to define how operators are intended to interact with the system under construction.
- *Identify Interactions with Systems:* the intent of the activity to define how the system under construction will interact with other systems. The required and provided services of the system under construction shall be detailed to such a level that dependencies to other systems are identified.
- *Identify Relevant Hazards:* the intent of the activity is to identify any relevant hazard with respect to operation of the system under construction in its intended context. The introduction of a hazard identification activity early in the design process facilitates effective identification of the potential dangers that the system under construction may inflict, allowing effective risk reduction and early application of mitigating measures.
- *Document Concept:* the intent of the activity is to document the findings of the *Establish Concept* process in a suitable format. The findings are the result of each activity defined in Figure 41.

Instantiation Rule: A documentation artefact *Conc* (see Figure 41 and Figure 40) instantiates the *Establish Concept* pattern if:

- *Conc* is a documentation of the results from applying the process described in Section “Process Solution”.
- *Conc* specifies the assumptions, and the justifications for design choices, made during the process of establishing the concept.
- *Conc* specifies the goals associated with the system under development.
- *Conc* specifies the scope, context of operation, and the functions to be fulfilled by the system under development.
- *Conc* specifies all intended interactions with other systems and human operators.
- *Conc* specifies all foreseeable hazards that are applicable to the system under development.
- *Conc* specifies the traces to all source documentation that are used in order to establish its content.

Related Patterns: The *Establish Concept* pattern may be used as a starting point to the development of the system to be constructed. The results obtained from applying the pattern may then be used in order to guide the selection of appropriate subsequent patterns to support the expressed concept. The *Establish Concept* is related to other patterns in the following manner:

- May be used with any design pattern, or set of design patterns, that satisfies the concept description obtained from instantiating *Establish Concept*.
- *May precede Hazard Identification.* The *Hazard Identification* pattern addresses the process of identifying hazards associated with the operation of given system that may be described in the results from instantiating *Establish Concept*.

Known Uses: The pattern captures the intent of the first stage of the system life cycle as presented in several safety standards, to notable standards are [IEC61508] and [EN50126].

[IEC61508] International Electrotechnical Commission, Functional safety of electrical/electronic/ programmble electronic safety-related systems, IEC 61508, Edition 2.0, 2010.

[EN50126] European Committee for Electrotechnical Standardization, Railway Applications – The specification and demonstration of Reliability, Availability, Maintainability and Safety (RAMS), EN 50126, 1999.

B.7 HAZARD IDENTIFICATION

Name: Hazard Identification

Pattern Signature: *Hazard Identification* is defined with the signature illustrated in Figure 42.

In Figure 42, the following abbreviations are used for denoting the parameters of the pattern:

- *ToA* is short for Target of Assessment.
- *Src* is short for Source.
- *IdHz* is short for Identified Hazards.
- *HzLg* is short for Hazard Log.

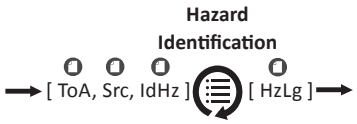


Figure 42 Hazard Identification – Pattern Signature

Intent: Support the capture of relevant hazards *IdHz* with respect to the operation of a target system *ToA* in a specific context and document these in a hazard log *HzLg*. The intent of the pattern is to define the process of identifying potential hazards associated with the operation of a target system in a given context. The pattern expresses: i) the main characteristics of the process of identifying hazards; and ii) the requirements for specifying process results in a hazard log.

Applicability: The *Hazard Identification* pattern is suitable to apply in the following situations:

- When the system under construction is intended to be used in a context where there exists safety concerns (e.g., potential threats to life or environment).
- When it is known, or there are indications that the system under construction may impact on safety.

Problem: The main aspects to be considered when identifying potential hazards are:

- *Scope:* To correctly delimit the scope of the analysis such that the cost of performing hazard identification is minimised, but where the scope is sufficiently broad to capture all relevant safety concerns.
- *Sources:* To provide confidence that all relevant historic data valuable for identifying potential hazards of the system under construction are accounted for. Relevant data sources may be data related to e.g., known incidents, known accidents, and assessments of comparable systems.
- *Methods:* To provide confidence that the methods applied in order to identify hazards are suitable and are able to provide the intended results. The intended result is to identify all relevant hazards.
- *Completeness:* To provide confidence that all relevant hazards within the scope are identified.

Process Solution: Figure 43 illustrates the *Hazard Identification* process annotated in a UML activity diagram.

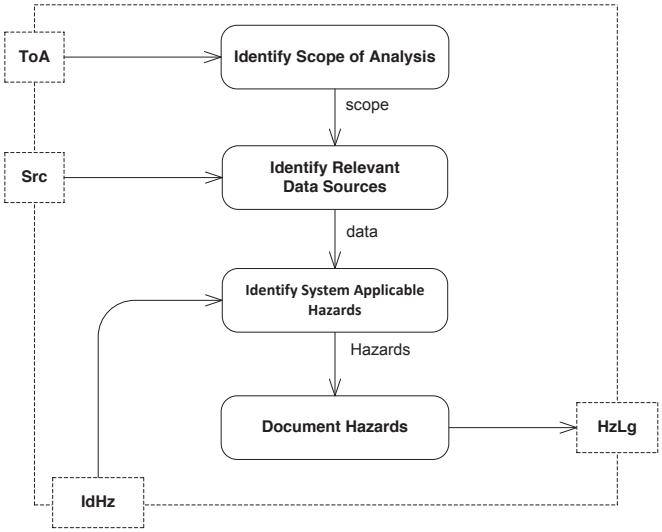


Figure 43 Hazard Identification – Activity Diagram

The input parameters associated with activity diagram may be interpreted as follows:

- *ToA (Target of Assessment)*: represents the entity that the hazard identification concerns.
- *Src (Source):* represents data providing information on potential causes of hazards.
- *IdHz (Identified Hazards):* represents the results from any method supporting the activity of identifying hazards.

The activities identified in Figure 43 intend to serve the following purpose:

- *Identify Scope of Analysis:* the intent of the activity is to define the boundary of the investigation, its objectives, and a strategy for how to satisfy the objectives. The strategy should involve a plan for the study, set the responsibilities of those involved, describe how data shall be collected and detail a schedule.
- *Identify Relevant Data Sources:* the intent of the activity is to systematically identify and document relevant sources of information that provide information on potential hazards relevant with respect to the intended use of the system under construction in its context. Relevant hazards data captured from sources shall be documented.
- *Identify System Applicable Hazards:* the intent of the activity is to provide an examination of a target (ToA) with respect to its context by the use of relevant methods. A set of complementary methods may be used as long as the total coverage is in accordance with the scope of the analysis.
- *Document Hazards:* the intent of the activity is to combine all relevant information with respect to the identification and analysis of hazards in a document form containing all identified hazards and their potential causes.

Instantiation Rule: An artefact *HzLg* (see Figure 43 and Figure 42) is the result of the instantiation of the *Hazard Identification* pattern if:

- *HzLg* is a documentation of the results from applying the process described in Section “Process Solution”.
- *HzLg* is a documentation of all relevant hazards associated with the use of a target system (identified by the instantiation of *ToA)* in a specific context.
- Every hazard of *HzLg* is uniquely identified.
- Every hazard of *HzLg* describes a potential danger of the use of a target system in its context.
- Every hazard of *HzLg* is traceable to its origin documentation, meaning the document where the hazard is identified (identified by the instantiation of *Src* or *IdHz)*.

Related Patterns: The *Hazard Identification* pattern is related to other patterns in the following manner:

- May succeed *Establish Concept* in terms of representing a natural follow up once a concept is established.
- May precede *Hazard Analysis* by providing an initial hazard log identifying relevant hazards associated with a target. The *Hazard Analysis* may then further address identified hazards in order to determine their potential causes.
- May be used in order to address an abstract design defined in a design pattern or the instantiation of a design pattern with respect to potential hazards in a given context
- May be used together with any method pattern that supports identification of hazards.

Known Uses: The pattern describes a process of identifying hazards in accordance with the practice as described in several safety standards and guidelines, to notable standards are [IEC61508] and [EN50129].

[IEC61508] International Electrotechnical Commission, Functional safety of electrical/electronic/ programmble electronic safety-related systems, IEC 61508, Edition 2.0, 2010.

[EN50129] European Committee for Electrotechnical Standardization, Railway Applications – Communications, signalling and processing systems – Safety related electronic systems for signalling, EN 50129, 2003.

B.8 HAZARD ANALYSIS

Name: Hazard Analysis

Pattern Signature: *Hazard Analysis* is defined with the signature illustrated in Figure 44.

In Figure 44, the following abbreviations are used for denoting the parameters of the pattern:

- *ToA* is short for Target of Assessment.
- *Src* is short for Source.
- *Haz* is short for Hazards.
- *IdCsHz* is short for Identified Causes of Hazards.
- *HzLg* is short for Hazard Log.

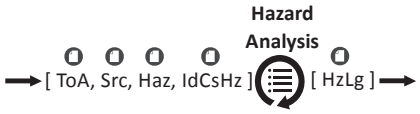


Figure 44 Hazard Analysis – Pattern Signature

Intent: Support the assessment of a target system *ToA* by defining the process of deriving the potential causes of hazards *Haz*. The expected result of applying the pattern is a hazard log *HzLg* that documents the hazards *Haz* and the potential causes of hazards *IdCsHz* associated with a system *ToA*. The hazard log *HzLg* facilitates the evaluation of the potential of the system to negatively affect safety. The pattern describes: 1) the main characteristics of the process of hazard analysis, and 2) the main requirements for specifying the result of the process in a hazard log.

Applicability: The *Hazard Analysis* pattern is suitable to apply in the following situations:

- When the system under construction is intended to be used in a context where there exists safety concerns (e.g. potential threats to life or environment).
- When hazards applicable to a system under construction are given and the potential causes of identified hazards are required to be identified.

Problem: The main aspects to be considered when performing hazards analysis are:

- *Scope*: To correctly delimit the scope of the analysis such that the cost of performing hazard analysis is minimised, but where the scope is sufficiently broad to capture all relevant safety concerns.
- *Hazards*: To ensure that the sources providing data on hazards relevant for the system under construction provide valid information.
- *Sources*: To ensure that the sources required for identifying potential causes of hazards are accounted for. Sources may represent experience data on incidents, accidents, and safety assessments.
- *Methods*: To ensure that the methods applied in order to assess the potential causes of hazards are suitable and are able to provide the intended results. The intended result is to identify all relevant causes of hazards.

- *Causes*: To ensure that the potential causes that can lead to hazards are identified.
- *Completeness*: To ensure that all relevant hazards are assessed with respect to potential causes.

Process Solution: Figure 45 illustrates the *Hazard Analysis* process annotated in a UML activity diagram.

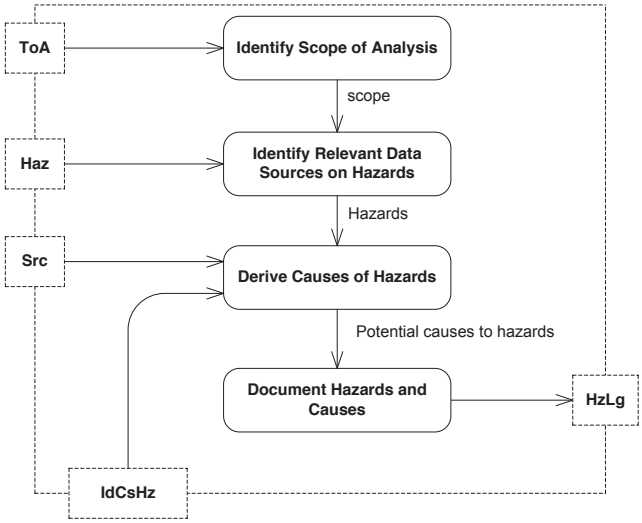


Figure 45 Hazard Analysis – Activity Diagram

The input parameters associated with the activity diagram may be interpreted as follows:

- *ToA (Target of Assessment*): represents the entity that the hazard analysis concerns.
- *Haz (Hazards*): represents information on identified hazards.
- *Src (Source*): represents information on potential causes of hazards.
- *IdCsHz (Identified Causes of Hazards*): represents the results from any method supporting the activity of identifying potential causes of hazards.

The main activities intend to serve the following purpose:

- *Identify Scope of Analysis*: the intent of the activity is to define the boundary of the investigation, its objectives, and a strategy for how to satisfy the objectives defined. The strategy should involve a plan for the study, set the responsibilities of those involved, describe how data shall be collected and detail a schedule.
- *Identify Relevant Data Sources on Hazards:* the intent of the activity is to gather information on identified hazards applicable to the system under construction, here the target of assessment.
- *Derive Causes of Hazards*: the intent of the activity is to assess a target in order to identify its potential contribution to events that may lead to hazards. The target should be assessed by examining relevant historic data and by the use of relevant assessment methods. One method or a set of complementary methods may be used as long as the total coverage is in accordance with the defined scope of analysis.

- *Document Hazards and Causes*: the intent of the activity is to combine all relevant information with respect to the identification and analysis of hazards in a document form containing all identified hazards and their potential causes.

Instantiation Rule: An artefact *HzLg* (see Figure 45 and Figure 44) is the result of the instantiation of the *Hazard Analysis* pattern if:

- *HzLg* is a documentation of the results from applying the process described in Section “Process Solution”.
- *HzLg* is a documentation of all relevant hazards associated with the use of a target system (identified by the instantiation of *ToA*) in a specific context.
- Every hazard of *HzLg* is uniquely identified.
- Every hazard of *HzLg* describes a potential danger of the application of a target system in a specific context.
- Every hazard of *HzLg* is traceable to its origin documentation, meaning the document where the hazard is identified (identified by the instantiation of *Haz* or *Src*).
- Every hazard is associated with a description of potential causes.
- Every potential cause associated with a hazard is traceable to its origin documentation, meaning the document (identified by the instantiation of *IdCsHz*) where the causal relationship may be identified.

Related Patterns: The *Hazard Analysis* pattern is related to other process requirement patterns in the following manner:

- May succeed *Hazard Identification* in terms of supplementing the hazard log provided as an output of *Hazard Identification* with information on potential causes of hazards.
- May precede *Risk Analysis* by providing as a deliverable, a set of hazards and potential causes associated with the operation of a target system (ToA) such that risk may be determined.
- May be used in order to address an abstract design defined in a design pattern or the instantiation of a design pattern with respect to its potential contribution to hazards.
- May be used together with other method patterns that support assessment of the causes of hazards.

Known Uses: The pattern describes a process of identifying the potential causes of hazards in accordance with the practice as described in several safety standards and guidelines, to notable standards are [IEC61508] and [EN50129].

[IEC61508] International Electrotechnical Commission, Functional safety of electrical/electronic/ programmble electronic safety-related systems, IEC 61508, Edition 2.0, 2010.

[EN50129] European Committee for Electrotechnical Standardization, Railway Applications – Communications, signalling and processing systems – Safety related electronic systems for signalling, EN 50129, 2003.

B.9 Risk Analysis

Name: Risk Analysis

Pattern Signature: *Risk Analysis* is defined with the signature illustrated in Figure 46.

In Figure 46, the following abbreviations are used for denoting the parameters of the pattern:

- *ToA* is short for Target of Assessment.
- *Haz* is short for Hazards.
- *ClsSev* is short for Classification of Severity.
- *ClsLi* is short for Classification of Likelihood.
- *ClsCr* is short for Classification of Criticality.
- *Risks* is not abbreviated.

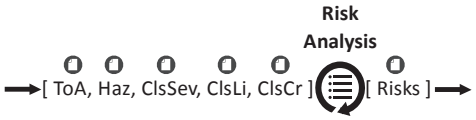


Figure 46 Risk Analysis – Pattern Signature

Intent: Support the assessment of a target system *ToA* with respect to risk. A process for risk assessment supports this where the result of an analysis of the likelihood of hazards as well as the result of an analysis of the severity of an accident associated with the occurrence of a hazard is combined into a notion of risk. The expected result *Risks* of applying the pattern represents a notion of the potential danger of applying the target system in a given context.

Applicability: The *Risk Analysis* pattern may be suitable to apply in the following situation:

- When the system under construction is already analysed with respect to its potential contribution to hazards in its intended context.

Problem: The main aspects to be considered when performing risk analysis are:

- *Defined target*: to clearly state what is the target of the assessment and correctly delimit the scope of the risk analysis.
- *Hazards*: assessment of risk is performed on the basis of hazards and their potential causes associated with a target. It is expected that data on hazards and hazard causes will be provided.
- *Classification*: in order to establish a notion of risk in a consistent manner such that risk associated with different systems or system entities can be easily compared. A categorisation scheme for discretising data on severity, likelihoods and risk is commonly applied.
- *Estimation*: in order to provide a notion of risk, severity and likelihood estimates are required to be provided either quantitatively or qualitatively.
- *Mitigations*: in order to sustain a certain risk level or reduce a risk to an acceptable level, mitigating means (or risk reduction means) are used. A part of risk analysis is to identify potential mitigations.

Process Solution: Figure 47 illustrates the *Risk Analysis* process specified using a UML activity diagram.

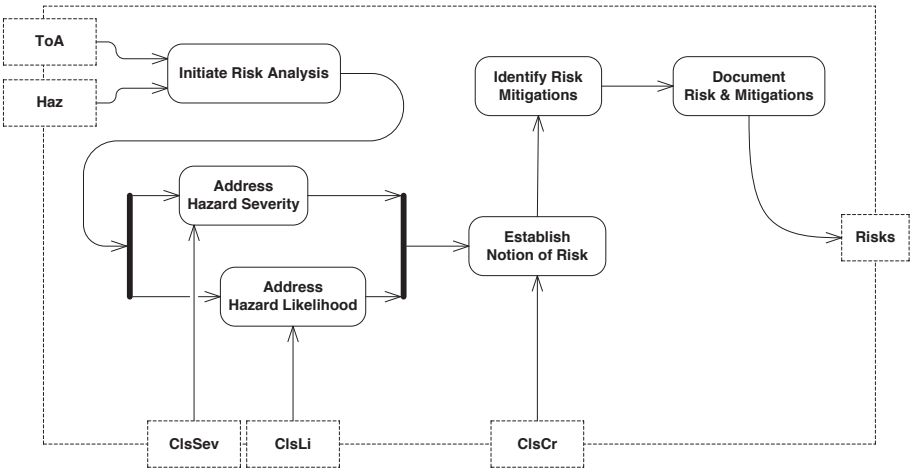


Figure 47 Risk Analysis – Activity Diagram

The input parameters associated with activity diagram may be interpreted as follows:

- *ToA (Target of Assessment*): represents the target of the assessment that is analysed with respect to risk.
- *Haz (Hazards*): represents the hazards that are associated with the target.
- *ClsSev*: represents the results from any method supporting the classification of the severity of hazards.
- *ClsLi*: represents the result from any method supporting the classification of likelihood of hazards.
- *ClsCr*: represents the results from any method supporting the classification of risk where risk is represented as a combined measure of the associated severity and the associated likelihood of hazards.

The main activities intend to serve the following purpose:

- *Initiate Risk Analysis*: the intent of the activity is to define the target of assessment, its intended context of operation, and identify hazards associated with the use of the target system in its context. It is expected that relevant information for risk analysis be provided as documentation of the target system. Relevant information include system hazards as well as the unwanted events associated with the operation of the target system that might lead to hazards.
- *Address Hazard Severity*: the intent of the activity is to address the severity of the consequence of hazards.
- *Address Hazard Likelihood*: the intent of the activity is to address the likelihood of occurrence of each hazard.
- *Establish Notion of Risk*: the intent of the activity is to combine data on likelihood of a hazard occurring and data on severity of the consequence of a hazard into a notion of risk.

- *Identify Risk Mitigations*: the intent of the activity is to identify risk reduction in the form of mitigations to reduce the likelihood of a hazard occurring and/or the severity of the consequence should a hazard occur.
- *Document Risk & Mitigations*: the intent of the activity is to combine all relevant information with respect to the assessment of risk in a document form containing all relevant risks and associated mitigations.

Instantiation Rule: An artefact *Risks* (see Figure 47 and Figure 46) is the result of instantiating the *Risk Analysis* pattern if:

- *Risks* is a documentation describing every relevant risk associated with the use of a target system (identified by the instantiation of *ToA*) in a specific context.
- Every risk of *Risks* is uniquely identified.
- Every risk of *Risks* is traceable to its origin documentation. By origin documentation, we mean: the document where the hazard relevant to risk is identified (identified by the instantiation of *Haz*), documentation on estimates on the severity of the consequence of hazards (identified by the instantiation of *ClsSev*), documentation on estimate of the likelihood of hazard (identified by the instantiation of *ClsLi*).
- Every risk of *Risks* is associated with a description of mitigation. The description should describe any dependencies or possible mitigations for reducing risk to an acceptable level.
- Every mitigation associated with a risk is stated such that its purpose and how it reduces risk can be clearly identified.

Related Patterns: The *Risk Analysis* pattern is related to other patterns in the following manner:

- May succeed *Hazard Analysis* in terms of supplementing the assessment of relevant hazards and their potential causes with information on the associated risk, e.g., by adding likelihood estimates and severity estimates associated with hazards.
- May precede *Establish System Safety Requirements* by providing information on the risks associated with the operation of the system, and information on required mitigations and potential mitigations in order to reduce risk. This ensures that the specification of safety requirements is based on a risk assessment of a target system.
- May be used in order to address the instantiation of a design pattern with respect to risk.
- May be used with other method patterns that support assessment or categorisation of the severity of the consequence of hazards.
- May be used with other method patterns that support the assessment or categorisation of the likelihood of the causes of hazards.
- May be used with other method patterns that support the classification of risk.

Known Uses: The pattern describes a process of assessing risk in accordance with the practice as described in several safety standards and guidelines, to notable standards are [IEC61508] and [EN50129].

[IEC61508] International Electrotechnical Commission, Functional safety of electrical/electronic/ programmble electronic safety-related systems, IEC 61508, Edition 2.0, 2010.

[EN50129] European Committee for Electrotechnical Standardization, Railway Applications – Communications, signalling and processing systems – Safety related electronic systems for signalling, EN 50129, 2003.

B.10 Establish System Safety Requirements

Name: Establish System Safety Requirements

Pattern Signature: *Establish System Safety Requirements* is defined with the signature illustrated in Figure 48.

In Figure 48, the following abbreviations are used for denoting the parameters of the pattern:

- *ToA* is short for Target of Assessment.
- *Reg* is short for Regulations.
- *Risks* is not abbreviated; represents the documentation of the risks associated with the application of *ToA* in its intended context.
- *Req* is short for Requirements.

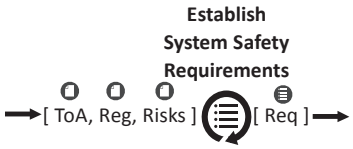


Figure 48 Establish System Safety Requirements – Pattern Signature

Intent: Support the specification of system safety requirements *Req* on the basis of a risk-based approach. The safety requirements describe the required measures to be satisfied by the system *ToA* to assure the necessary safety integrity. The general approach for defining safety requirements is to define them on the basis of the result of a risk assessment *Risks*, especially the mitigations identified as means to reduce risk to an acceptable level. The pattern describes the general process of capturing the requirements that must be satisfied in order to assure safety.

Applicability: The *Establish System Safety Requirements* pattern is intended for the following situations:

- When the system under construction may negatively affect the overall system safety.
- When there are identified measures that can mitigate identified risks and can be used as input to the specification of safety requirements.

Problem: The main aspects relevant to address when establishing the safety requirements are:

- *Characteristics*: To define the system characteristics to be satisfied such that the occurrence of unwanted events are minimised or avoided.
- *Functions*: To define the safety functions that assures safe operations.
- *Constraints*: To define the functional constrains that sufficiently delimit potentially hazardous operations.
- *Environment*: To define the operational environment that ensures safe operations.

- *Compliance*: To define the requirements that are required to be satisfied in order to comply with laws, regulation and standards, as a minimum the mandatory requirements related to assurance of safety. These requirements include requirements on applying some specific development process, perform certain activities, or make use of specific techniques.

Process Solution: Figure 49 illustrates the *Establish System Safety Requirements* process specified using a UML activity diagram.

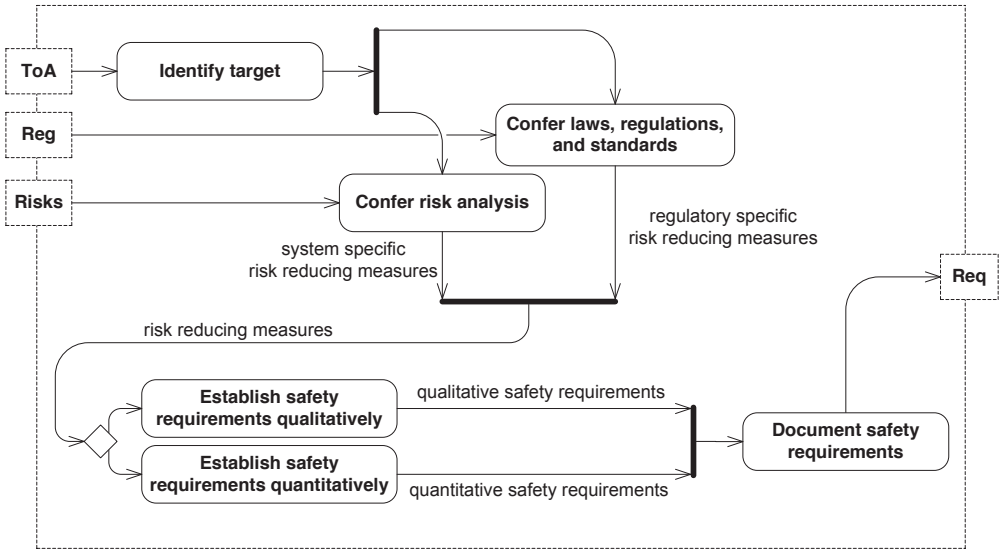


Figure 49 Establish System Safety Requirements – Process Flow

The input parameters associated with the activity diagram may be interpreted as follows:

- *ToA (Target of Assessment*): represents the target system for which safety requirements should be established.
- *Reg (Regulations*): represents any source of information describing mandatory or recommended practices (e.g. as provided in laws, regulations or standards) valuable for identifying risk reducing measures.
- *Risks*: represents risks associated with the target system.

The main activities serve the following purpose:

- *Identify target*: the intent of the activity is to identify *ToA*. The description of the target should as a minimum include a definition of the system and its boundaries, its operational profile, functional requirements, and safety integrity requirements.
- *Confer laws, regulations, and standards*: the intent of the activity is to capture all relevant data (requirements for risk reducing measures) from relevant sources (normative references) in order to outline the set of risk reducing measures that shall be met by compliance. Each source is inspected in order to identify, as a

minimum, the mandatory risk reducing measures that shall be met in order to be compliant.

- *Confer risk analysis*: the intent of the activity is to capture all the relevant data on risk analysis of the system that is under construction in order to outline the system specific risk reducing measures that shall be met.
- *Establish safety requirements qualitatively*: the intent of the activity is to define safety requirements on the basis of those identified risk reducing measures required applied, and which can be demonstrated fulfilled with qualitative reasoning.
- *Establish safety requirements quantitatively*: the intent of the activity is to define safety requirements on the basis of those identified risk reducing measures required applied, and which can be demonstrated fulfilled with quantitative reasoning.
- *Document safety requirements*: the intent of the activity is to detail all relevant information with respect to the requirements in a system safety requirements specification. For each requirement defined in the requirement specification, information detailing what influenced its definition should be provided, e.g., the associated risks, assumptions, calculations, and justifications.

Instantiation Rule: An artefact *Req* (see Figure 49 and Figure 48) is the result of a process that instantiates the *Establish System Safety Requirements* pattern if:

- *Req* is a set of requirements.
- *Req* is a result of applying a process illustrated in Figure 49 and described in Section “Process Solution”. The process is initiated by an activity on describing the target *ToA*. Once a description of the target system and its operational context is provided, the next activities shall identify the risk reducing measures to be applied to the target by conferring relevant laws, regulations and standards as well as the result of target specific risk analysis for guidance. Once all the relevant risk-reducing measures are identified, these shall be used as a basis to define the requirements to be met by the target system or by the process to be followed while developing the target. The requirements are defined quantitatively or qualitatively depending on the nature of the risk reducing measure that is addressed. The requirements are documented in a requirement specification *Req*.
- Every requirement of *Req* is traceable to relevant risks (identified by the instantiation of *Risks*), and/or regulatory requirements (identified by the instantiation of *Reg*).
- Every requirement of *Req* is justified such that any assumptions, calculations, and assessments that support the specification of the requirement as a safety requirement are provided.

Related Patterns: The *Establish System Safety Requirements* pattern is related to other patterns in the following manner:

- May succeed the *Risk Analysis* pattern that supports identifying risks. The *Establish System Safety Requirements* may be applied as support for defining the requirements to be fulfilled in order to reduce risk to an acceptable risk level.
- May be used in order to detail requirements for the design that is a result of an instantiation of a design pattern.

B.11 Overall Safety

Name: Overall Safety

Pattern Signature: *Overall Safety* is defined with the signature illustrated in Figure 50.

In Figure 50, the following abbreviations are used for denoting the parameters of the pattern:

- *ToD* is short for Target of Demonstration.
- *QualMng* is short for Quality Management.
- *SafMng* is short for Safety Management.
- *TechSaf* is short for Technical Safety.
- *Case* is short for Safety Case

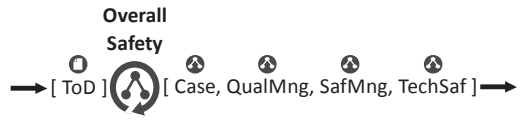


Figure 50 Overall Safety – Pattern Signature

Intent: Provide a structure for the specification of an overall strategy (or plan) for the safety demonstration *Case* of a target system *ToD*. The overall strategy is a means to document the set of practices that jointly provide confidence in the system being sufficiently safe for its intended use. The pattern describes a top-down deductive strategy to build the safety demonstration with a focus on establishing the overall argument for adequate managerial and technical safety. The focus of the pattern is on how to combine strategies on different safety concerns into an overall strategy such that it may be claimed and proven with confidence that the system is sufficiently safe.

Applicability: The *Overall Safety* pattern is intended for the following situations:

- At the initial stages of development in order to outline the main set of demonstration strategies, and how these strategies may be combined to demonstrate system safety.
- Prior to project initiation as a means for stakeholders (vendor, customer, safety authority) to reflect upon the main demonstration challenges and possible solutions.

Problem: The main aspects to be considered when providing a solution for the demonstration of safety are:

- *Quality management*: it is necessary to demonstrate that the quality of the system is controlled by an effective quality management system. Quality management is a means to minimise the occurrence of human errors at each stage of the development life cycle, and thus to reduce the risk of systematic faults in the system.
- *Safety management*: it is necessary to demonstrate that safety management is controlled by effective means. Effective safety management facilitates to reduce the occurrence of safety-related human errors throughout the life cycle and the risk of safety-related systematic faults.

- *Technical safety*: it is necessary to demonstrate the safety of the technical design.
- *Strategy identification*: in order to plan and effectively apply a suitable demonstration strategy for different systems, it is important to identify as early as possible the main lines of demonstration. This will reduce the cost related to performing safety assessment by avoiding those activities that does not support an effective safety demonstration. In addition, if the choice of demonstration strategies does not provide the necessary confidence, then there is a risk of not getting safety approval. The effect of not getting a safety approval may induce a great cost as additional work may be required in order to receive approval, or there will be a loss of returns if a project is discontinued.
- *Effectiveness*: it is important that the demonstration strategy is effective in providing the necessary safety demonstration. To develop safety related and safety critical systems is costly, mainly due to the activities required to assure and demonstrate that the system is sufficiently safe. It is important for a vendor to choose strategies that minimise cost but provide required results.
- *Confidence*: it is important to choose a demonstration strategy that has the potential to convince some safety authority reviewing the evidences that sufficient safety is achieved. In a development project, the effectiveness of different demonstration strategies must be weighed against the estimated cost and the ability of building confidence.
- *Acceptance*: it is important that the choice of demonstration strategies are acceptable for all involved parties such that they represent an effective means for demonstrating safety and may be applied in a manner that provides confidence that the system is safe.

Argument Structure Solution: Figure 51 and Table 12 together present the demonstration solution, and therefore should be read together. Figure 51 illustrates the decomposition of the safety argument in a tree structure using GSN notation. Table 12 details the tree structure by defining the type and expected content of each node.

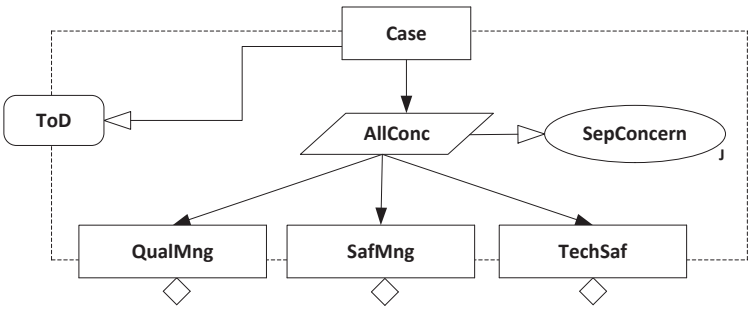


Figure 51 Overall Safety – Argument Structure

Associated with the contextual elements of the argument structure, the following is assumed:

- ToD: identifies the target system under consideration.

Table 12 Overall Safety – Argument Structure

Node	Node Type	Node Content Description
Case	Goal	*ToD* is safe
ToD	Context	Definition of parameter *ToD* (Target of Demonstration)
AllConc	Strategy	*ToD* is demonstrated safe by demonstrating that QM (Quality Management), SM (Safety Management), and TS (Technical Safety) is sufficiently addressed
SepConcern	Justification	Justification for the appropriateness of the separation of the demonstration concerning QM, SM, and TS
QualMng	Goal	*ToD* is safe with respect to proper QM (Quality Management)
SafMng	Goal	*ToD* is safe with respect to the SM (Safety Management) concern
TechSaf	Goal	*ToD* is safe with respect to the TS (Technical Safety) concern

Instantiation Rule: An artefact *Case* (see Figure 51 and Figure 50) instantiates the *Overall Safety* pattern if:

- *Case* represents the overall claim in a top-down decomposable safety argumentation on the safety of a target system (the target is identified by the instantiation of *ToD*).
- *Case* represents the root node in a tree like presentation of the safety argument as indicated in Figure 51**.**
- Every element of the decomposable safety argument where *Case* is the root node, is traceable to a safety case node as indicated in Figure 51**.** A safety case node is instantiated by adapting the descriptions in column “Node Content Description” in Table 12 to the context that is addressed in order to define a structure as given in Figure 51.
- *Case* and the associated safety argumentation expresses the decomposition of a main claim, that a target system is sufficiently safe, for its intended purpose by a strategy of claiming sufficient quality management (the claim expressed in the node *QualMng*), safety management (the claim expressed in the node *SafMng*), and technical safety (the claim expressed in the node *TechSaf*). Note: No guidance is given in this pattern on how to decompose the safety argument parts represented by the claims *QualMng*, *SafMng* and *TechSaf* down to its supporting evidences, suitable patterns supporting such a decomposition should be conferred.

Related Patterns: The *Overall Safety* pattern is related to other patterns in the following manner:

- May be supported by other safety case patterns for detailing those parts that are not fully developed, e.g., the pattern *Technical Safety* may be used to as support

on the issue of demonstrating that a system is technical safe and thereby detail the node *TechSaf* presented in Figure 51.

- May be used as support for the safety demonstration of a design derived from a design pattern.

B.12 TECHNICAL SAFETY

Name: Technical Safety

Pattern Signature: *Technical Safety* is defined with the signature illustrated in Figure 52.

In Figure 52, the following abbreviations are used for denoting the parameters of the pattern:

- *ToD* is short for Target of Demonstration.
- *Case* is short for Safety Case.
- *CoP* is short for Code of Practice.
- *CrRef* is short for Cross Reference.
- *ERE* is short for Explicit Risk Estimation.

Figure 52 Technical Safety – Pattern Signature

Intent: Provide a structure for the specification of a safety demonstration *Case* showing that a system *ToD* is safe by arguing that it is developed according to an accepted code of practice, is similar to an already accepted system, or risk being explicitly addressed and sufficiently reduced. The intent is to provide an argument structure for demonstration on sufficient technical safety. The pattern facilitates early identification of the set of different overall demonstration strategies that may be applied in order to provide confidence that a system is sufficiently safe.

Example: a system S is decomposed into the subsystems Sub 1 and Sub 2. Sub 1 is a system where showing compliance to a code of practice is a commonly accepted approach for safety demonstration. Sub 2 is a system includes novel technical solutions. Relying on a code of practice safety demonstration strategy of novel technical solutions may challenge our confidence as complying with a code is not necessarily sufficient. Therefore, for demonstration of Sub 2, a risk estimation strategy is chosen. The safety demonstration for system S consists of a combination of the safety demonstrations applied on the subsystems with the addition of the argumentation for this combination being sufficient.

Applicability: The *Technical Safety* pattern is suitable to apply in the following situations:

- At the initial stages of development in order to outline the main set of demonstration strategies to be applied for addressing technical safety concerns.
- Prior to project initiation as a means for stakeholders (e.g, vendor, customer) to reflect upon main challenges associated with the demonstration of technical safety and possible solutions.

Problem: The main aspects to be considered when providing a solution for the demonstration of technical safety are:

- *Implicit vs Explicit Safety Demonstration Strategy*: the necessary confidence in a safety demonstration may be provided by an implicit demonstration strategy, e.g., referring to the application of a code of practice that is known to mitigate certain risks. Another strategy is to compare the target of demonstration to a similar system that already has received approval. If an implicit demonstration strategy is not sufficient, an explicit demonstration strategy must be performed that requires information on the risks associated with the target.
- *Effectiveness*: it is important that the demonstration strategy is effective in providing the necessary safety demonstration. To develop safety related and safety critical systems is costly, mainly due to the activities required to assure and demonstrate that the system is sufficiently safe. It is important for a vendor to choose strategies that minimise cost but provide required results.
- *Confidence*: it is important to choose a demonstration strategy that has the potential to convince some safety authority reviewing the evidences that sufficient safety is achieved. In a development project, the effectiveness of different demonstration strategies must be weighed against the estimated cost and the ability of building confidence.
- *Acceptance*: it is important that the choice of demonstration strategies are acceptable for all involved parties such that they represent an effective means for demonstrating safety and used in a manner that provide confidence that the system is safe.

Argument Structure Solution: Figure 53 and Table 13 together present the demonstration solution, and therefore should be read together. Figure 53 illustrates the decomposition of the safety argument in a tree structure using the GSN notation. Table 13 details the tree structure by defining the type and expected content of each node.

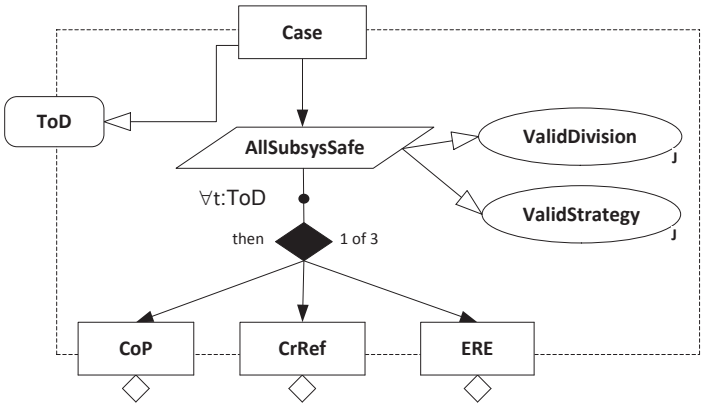


Figure 53 Technical Safety – Argument Structure

Table 13 Technical Safety – Argument Structure

Node	Node Type	Node Content Description
Case	Goal	*ToD* is safe
ToD	Context	Definition of parameter: ToD (Target of Demonstration)
AllSubsysSafe	Strategy	*ToD* is demonstrated safe by demonstrating that all subsystems *t* of *ToD* is safe
ValidDivision	Justification	Rationale for the division of *ToD* into subsystems
ValidStrategy	Justification	Rationale for the use of correct safety demonstration strategy on each subsystem of *ToD* as a means to demonstrate that *ToD* is safe
CoP	Goal	Subsystem *t* of *ToD* is safe by means of Code of Practice
CrRef	Goal	Subsystem *t* of *ToD* is safe by means of Cross Reference
ERE	Goal	Subsystem *t* of *ToD* is safe by means of Explicit Risk Estimation

Associated with the contextual elements of the argument structure, the following is assumed:

- *ToD*: identifies the target system under consideration.

Instantiation Rule: An artefact *Case* (see Figure 53 and Figure 52) instantiates the *Technical Safety* pattern if:

- *Case* is a safety demonstration of a target system (identified by the instantiation of ToD).
- Every element of *Case* is traceable to a safety case node as indicated in Figure 53. A safety case node is instantiated by applying the descriptions in column “Node Content Description” in Table 13 to the context that is addressed in order to define a structure as given in Figure 53.
- *Case* and the associated safety argumentation expresses the decomposition of a main claim, that a target system is technically safe for its purpose by a strategy of claiming that all subsystems are sufficiently safe.
- Subsystems are claimed sufficiently safe by a strategy of reference to a code of practice, or cross-reference and comparison to a similar and proven system, or explicit risk estimation.

Related Patterns: The *Technical Safety* pattern is related to other patterns in the following manner:

- May be used for detailing the claims on technical safety in the *Overall Safety* pattern.
- The *Codes Of Practice* pattern may be used to address the CoP node of this pattern by referencing common development practices.

- The *Cross Reference* pattern may be used to address the CrRef node in this pattern by comparing a target to a similar reference system that is an assured system and thus provide an implicit safety demonstration.
- The *Explicit Risk Evaluation* pattern may be used to address the ERE node in this pattern by explicitly addressing risk associated with a target.

Known Uses: The pattern describes a structure for arguing safety with a focus on technical aspects that may be seen as a variant of Part 4 of the safety case required by the standard [EN 50129] on issues related to demonstration of technical safety

[EN50129] European Committee for Electrotechnical Standardization, Railway Applications – Communications, signalling and processing systems – Safety related electronic systems for signalling, EN 50129, 2003.

B.13 CODE OF PRACTICE

Name: Code of Practice

Pattern Signature: *Code of Practice* is defined with the signature illustrated in Figure 54.

In Figure 54, the following abbreviations are used for denoting the parameters of the pattern:

- *ToD* is short for Target of Demonstration.
- *Haz* is short for Hazards.
- *Code* is short for Code of practice.
- *Case* is short for Safety Case.
- *AppOfCode* is short for Application of Code.

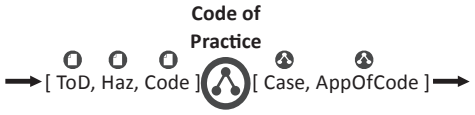


Figure 54 Code of Practice – Pattern Signature

Intent: Support the definition of a safety demonstration strategy *Case* claiming that a target system *ToD* is safe by the application of a code of practices *Code* known to provide control of a set of hazards *Haz*. By complying with the code, the underlying assumption is that some kinds of hazards are controlled, implying that a target *ToD* is safe with respect to some hazards assured through hazard control evidenced in the proper application of an accepted code of practice. The motivation for applying the pattern is to reduce the effort required to demonstrate that a particular practice is safe to apply by choosing practises that are pre-qualified and accepted in a domain for being effective in achieving required result.

Example #1: a software system S is developed to a high integrity level by formal methods. In domain D, the use of a formal method approach A for specifying requirements, design, and implementation is a highly recommended technique. As there is a code of practise with respect to the accepted use of approach A for high integrity systems, the approach may be claimed suitable and safe to apply with reference to the code.

Example #2: a software system S is assessed by different methods; one of these is a method M (e.g., fault tree analysis). In domain D, the use of M for software assessment is commonly used and accepted as an effective means to identify the causes that may lead to hazards. The method itself is then a pre-qualified method and needs no further motivation. Demonstration evidences associated with the method may then be reduced to providing confidence that the method was applied correctly, reducing the need to demonstrate that the method is suitable.

Applicability: The *Code of Practice* pattern is suitable to apply in the following situations:

- When there exist widely acknowledged codes of practice in the domain that have a generic acceptance as an effective means.
- When there exist acknowledged codes of practice in other relevant domain that have a generic acceptance as an effective means. The code needs then to be justified and be acceptable to any assessment body or approval body.

Problem: The main aspects to be considered when providing a solution for the demonstration of according to a code of practice are:

- *Hazard Control:* it needs to be established which code of practise is used to control which hazards such that it is clear what objective is fulfilled by the choice of a code.
- *Relevance:* is the code relevant for the system to be demonstrated safe. This needs to be verified.
- *Coverage:* with respect to cover a set of hazards, some by code of practise and some by other means, it is important to establish the coverage of hazards claimed by the set of code and define the residual set of hazards that are not addressed by the code.
- *Deviation from code*: is there some deviation from the referenced code to the applied code. Any deviation from code should be stated as well as the rationale for deviating.

Argument Structure Solution: Figure 55 and Table 14 together present the demonstration solution, and should therefore be read together. Figure 55 illustrates the decomposition of the safety argument in a tree structure using the GSN notation. Table 14 details the tree structure by defining the type and the expected content of each node.

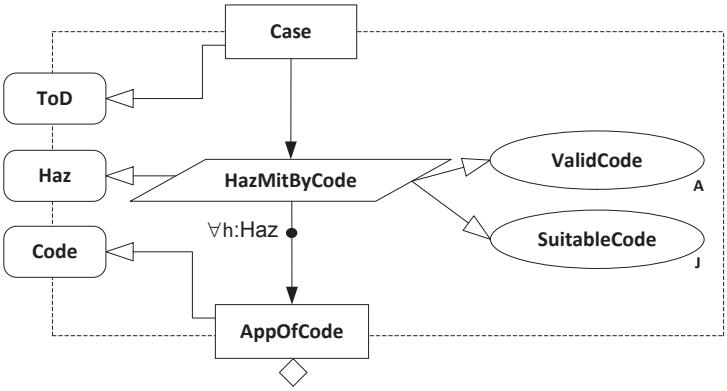


Figure 55 Code of Practice – Argument Structure

Table 14 Code of Practice – Argument Structure

Node	Node Type	Node Content Description
Case	Goal	*ToD* is safe
ToD	Context	Definition of *ToD*
HazMitByCode	Strategy	*ToD* is free from *Haz* by correct application a suitable set of codes
Haz	Context	Definition of *Haz,* denoting the hazards associated with system
ValidCode	Assumption	Assumption that the set of chosen practices is valid by reference to source
SuitableCode	Justification	Rationale over the set of codes provides suitable defences against hazards
AppOfCode	Goal	Application of code complied with *Code*
Code	Context	Definition of *Code*

A convenient structure of the argument is to add as many *AppOfCode* goal nodes as there are identified hazards (or groups of hazards if a code provides defence against a number of hazards), and directly attach these to the strategy *HazardMitByCode*. For each *AppOfCode* goal, attach a context description describing the code and develop the goal further with a suitable evidence pattern such that evidence of compliance to code is provided.

Associated with the contextual elements of the argument structure, the following is assumed:

- ToD (Target of Demonstration): identifies the target system under consideration.
- Haz: represents the hazards associated with the target system and its environment.
- h: may represent a hazard or a group of hazards in the set *Haz*.
- Code: represents the set of methods, procedures or other measures that is a code of practice.

Instantiation Rule: An artefact *Case* (see Figure 55 and Figure 54) instantiates the *Code of Practice* pattern if:

- *Case* is a safety demonstration of a target system (identified by the instantiation of *ToD)*.
- Every element of *Case* is traceable to a safety case node as indicated in Figure 55. A safety case node is instantiated by applying the descriptions in column “Node Content Description” in Table 14 to the context that is addressed in order to define a structure as given in Figure 55.
- *Case* and the associated argumentation expresses the decomposition of a main claim, that a target system is safe with respect to a given hazard by a strategy of claiming compliance to a code of practice.

Related Patterns: The *Code of Practise* pattern is related to other patterns in the following manner:

- May be used for detailing a demonstration goal of any safety case pattern where arguing compliance to a specific code is used as a strategy for arguing that the goal is satisfied.

Known Uses: The pattern defines a structure for arguing safety objectives being met on the basis of the application of well-proven practices. The strategy of arguing safety on the basis of the application of a code of practice is a strategy expressed in the European Regulation on common safety methods within railway and its associated application guideline [ERA/GUI/01-2008/SAF].

[ERA/GUI/01-2008/SAF] Guide for the application of the Commission Regulation on the adoption of a common safety method on risk evaluation and assessment as referred to in Article 6(3)(a) of the Railway Safety Directive, European Railway Agency, 2009.

B.14 CROSS REFERENCE

Name: Cross Reference

Pattern Signature: *Cross Reference* is defined with the signature illustrated in Figure 56.

In Figure 56, the following abbreviations are used for denoting the parameters of the pattern:

- *ToD* is short for Target of Demonstration.
- *ToR* is short for Target of Reference.
- *Crit* is short for Criteria.
- *Case* is short for Safety Case.
- *CritSat* is short for Criteria Satisfied.

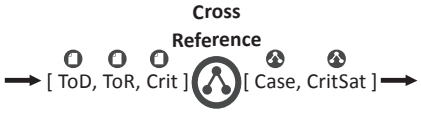


Figure 56 Cross Reference – Pattern Signature

Intent: Support the definition of a safety demonstration strategy *Case* claiming a target system *ToD* is sufficiently safe by comparison to a similar reference system *ToR* that already has received safety approval. The motivation is to reduce the effort required to demonstrate safe a system by comparison to a reference system. Given that hazards associated with the new system are similar and handled similarly to those of a reference system, the assumption is that the new system is equally safe as the reference system and thus is safe by cross reference.

Applicability: The *Cross Reference* pattern is suitable to apply in the following situations:

- When there exists a reference system that has a safety approval and is proven in use to have an acceptable safety level.
- When the system under consideration and the reference system is similar in the following manner:
  - Similar functions and interfaces.
  - Similar operating conditions.
  - Similar environmental conditions.

Problem: The main aspects to be considered when providing a demonstration solution according to a cross reference strategy are:

- *Comparison:* the whole case is founded on the ability to show that hazards, functions, operating conditions and other important aspects associated with a target system are similar to the reference system. It is imminent that the comparison shows that the two systems are similar enough to conclude the system is safe by comparison to an approved reference system.
- *Documentation:* in order to be able to do a comparison with a reference system, it is necessary to have available a proper set of documentation of the reference

system at a level of detail such that the characteristics of the system with respect to the criteria used for comparison are shown.

- *Assessment:* the assessment of similarity between the system under consideration and the reference system must motivate the conditions that establish the foundation for concluding that these systems are similar, or otherwise state the differences so that these conditions may be addressed with other means of demonstration.

Argument Structure Solution: Figure 57 and Table 15 together present the demonstration solution, and should be read together. Figure 57 illustrates the decomposition of the safety argument in a tree structure using the GSN notation. Table 15 details the tree structure by defining the type and expected content of each node.

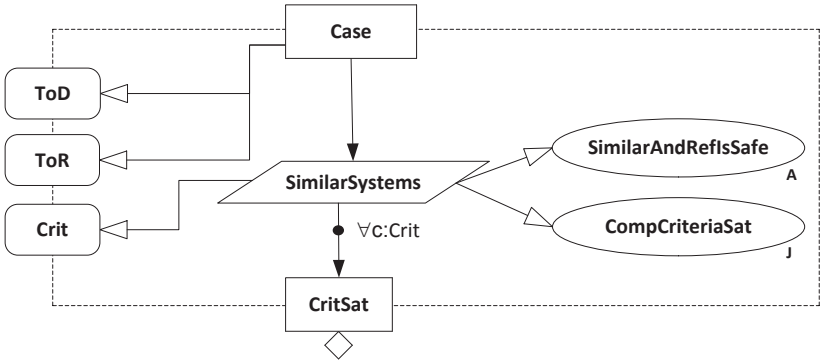


Figure 57 Cross Reference – Argument Structure

Table 15 Cross Reference – Argument Structure

Node	Node Type	Node Content Description
Case	Goal	*ToD* is safe by reference to a similar system *ToR* that has received safety approval
ToD	Context	Definition of *ToD*
ToR	Context	Definition of *ToR*
SimilarSystems	Strategy	*ToD* is safe as system *ToR* is safe and *ToD* is similar to *ToR* with respect to criteria *Crit* (e.g., Crit = {environment, operating conditions, hazards, functions, mitigations})
SimilarAndRefIs Safe	Assumption	*ToD* is similar by comparison to *ToR*. *ToR* has received safety approval. As *ToD* is similar to *ToR* then *ToD* is sufficiently safe.
CompCriteriaSat	Justification	Rationale over the criteria *Crit* for comparison is correct (e.g., the set {environment, operating conditions,

		hazards, functions, mitigations}) and offers adequate coverage of those aspects that is needed for comparison
Crit	Context	Definition of *Crit* (criteria for comparison)
CritSat	Goal	*ToD* is similar to *ToR* with respect to criteria *c* in *Crit*

Associated with the contextual elements of the argument structure, the following is assumed:

- ToD: identifies the target system under consideration.
- ToR: represents a reference system, the system that *ToD* is claimed to be similar to.
- Crit: represents the set of criteria or conditions that is used for comparison. If the two systems are sufficiently similar on all conditions in this set, then system S is sufficiently similar to the reference system and an implicit safety demonstration may be developed.
- c**:** represents an individual criteria part of the set of criteria defined by *Crit*.

Instantiation Rule: An artefact *Case* (see Figure 57 and Figure 56) instantiates the *Cross Reference* pattern if:

- *Case* is a safety demonstration of a target system (identified by the instantiation of *ToD)*.
- *Case* uses an acceptably safe reference system (identified by the instantiation of *ToR),* that is similar to the target system, as a means for comparison.
- Every element of *Case* is traceable to a safety case node as indicated in Figure 55. A safety case node is instantiated by applying the descriptions in column “Node Content Description” in Table 14 to the context that is addressed in order to define a structure as given in Figure 55.
- *Case* and the associated safety argumentation expresses the decomposition of a main claim, that a target system is safe, by an implicit safe demonstration strategy of comparing the target system with a similar reference system that is already found to be acceptably safe for use in a similar context.

Related Patterns:

The *Cross Reference* pattern is related to other safety case patterns in the following manner:

- May be used for detailing a demonstration goal of any safety case pattern concerned with arguing the suitability of a system for use in a particular context by a comparison to a similar and already approved reference system.

Known Uses: The pattern describes a structure for arguing safety objectives being satisfied by a system on the basis of a comparison with a similar and already accepted system. The strategy is expressed in the application guideline to the European Regulation on common safety methods within railway [ERA/GUI/01-2008/SAF].

[ERA/GUI/01-2008/SAF] Guide for the application of the Commission Regulation on the adoption of a common safety method on risk evaluation and assessment as referred to in Article 6(3)(a) of the Railway Safety Directive, European Railway Agency, 2009.

B.15 Explicit Risk Evaluation

Name: Explicit Risk Evaluation

Pattern Signature: *Explicit Risk Evaluation* is defined with the signature illustrated in Figure 58.

In Figure 58, the following abbreviations are used for denoting the parameters of the pattern:

- *ToD* is short for Target of Demonstration.
- *Risks* is not abbreviated.
- *QualACr* is short for Qualitative Acceptance Criteria.
- *QuantACr* is short for Quantitative Acceptance Criteria.
- *Case* is short for Safety Case.
- *QualAcc* is short for Qualitative Acceptance.
- *QuantAcc* is short for Quantitative Acceptance.

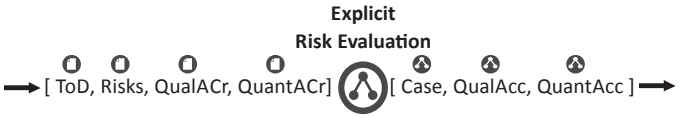


Figure 58 Explicit Risk Evaluation – Pattern Signature

Intent: Support the definition of a safety demonstration strategy *Case* claiming a target system *ToD* is safe by assuring that the presence of risk in the system is at an acceptable level by the use of adequate risk evaluation strategies. The demonstration structure is scaled systematically such that every identified risk is addressed individually. As different kinds of risks may require different handling, depending on their qualitative or quantitative nature, different risk estimation philosophies are supported.

Example #1: a software system S within the railway domains is associated with the risks R1 and R2. As S is intended to be operated within the railway domain, certain acceptance criteria may be required met as defined within the domain specific standard, e.g., compliance to SIL 3 requirements within the standard EN50128. All risk, here R1 and R2, mare be required to be addressed with respect to the acceptance criteria, here associated with the development of SIL 3 system defined in the standard EN50128.

Applicability: The *Explicit Risk Evaluation* pattern is suitable to apply in the following situations:

- When the system applies novel technical solution or is developed by novel techniques.
- When there are no existing acknowledged codes of practice that may be applied in order control the risk.
- When there are no existing similar systems that have received safety approval, and have proven quality in practise and may serve as a reference system in a *Cross Reference* demonstration approach.

Problem: The main aspects to be considered when providing a solution for the demonstration according to an explicit risk evaluation strategy are:

- *Risk Estimation:* it is necessary to provide a qualitative or quantitative estimate of the risk such that it may be possible to compare a stated risk with a risk acceptance criterion.
- *Safety Measures:* it is necessary to identify risk-reducing measures such that it is clear what renders a risk as acceptable.
- *Acceptance Criteria:* it is necessary to establish the acceptance criteria such that the boundary between non-acceptable risk and acceptable risk is clear.

Argument Structure Solution: Figure 59 and Table 16 together represent the demonstration solution, and should therefore be read together. Figure 59 illustrates the decomposition of the safety argument in a tree structure using the GSN notation. Table 16 details the tree structure by defining the type and the expected content of each node.

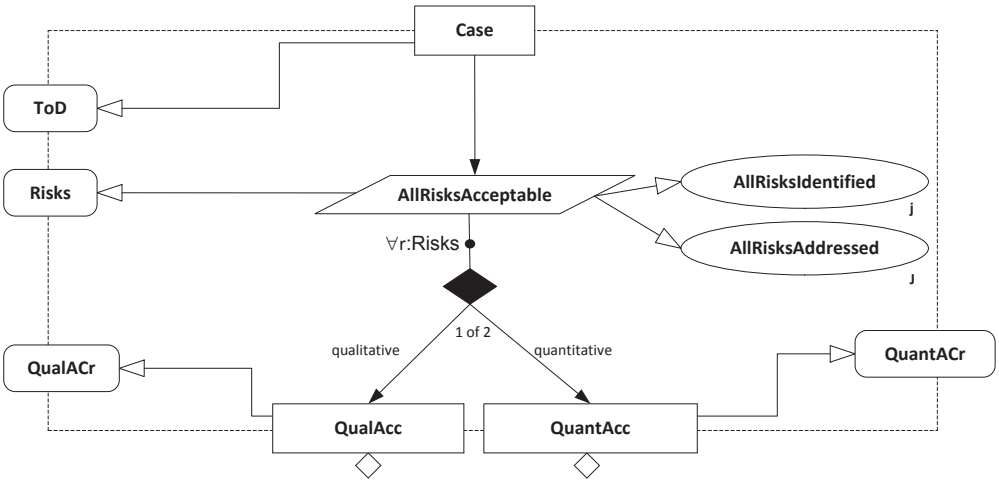


Figure 59 Explicit Risk Evaluation – Argument Structure

Table 16 Explicit Risk Evaluation – Argument Structure

Node	Node Type	Node Content Description
Case	Goal	*ToD* is safe
ToD	Context	Definition of *ToD*
AllRisksAcceptable	Strategy	*ToD* is acceptably safe as all operational scenarios only contain acceptable *Risks*
Risks	Context	Definition or *Risks*
AllRisksIdentified	Justification	The set *Risks* contains all important risks
AllRisksAddressed	Justification	All elements of the set *Risks* are addressed

		systematically, either qualitatively or quantitatively depending on the nature of each risk *r* in *Risks*
QualAcc	Goal	The risk *r,* element of *Risks*, is acceptable by comparison to qualitative acceptance criteria *QualACr*
QuantAcc	Goal	The risk *r*, element of *Risks*, is acceptable by comparison to quantitative acceptance criteria *QuantACr*
QualACr	Context	Definition of *QualACr*
QuantACr	Context	Definition of *QuantACr*

Associated with the contextual elements of the argument structure, the following is assumed:

- ToD: identifies the target system under consideration.
- Risks: represents a set of risk associated with the target system S.
- QualACr: represents a set of criteria, expressed in a qualitative manner, required to be satisfied in order to accept a specific risk as sufficiently reduced.
- QuantACr: represents a set of criteria, expressed in a quantitative manner, required to be satisfied in order to accept a specific risk as sufficiently reduced.

Instantiation Rule: An artefact *Case* (see Figure 59 and Figure 58) instantiates the *Explicit Risk Evaluation* pattern if:

- *Case* is a safety demonstration of a target system (identified by the instantiation of *ToD)*.
- Every element of *Case* is traceable to a safety case node as indicated in Figure 59. A safety case node is instantiated by applying the descriptions in column “Node Content Description” in Table 16 to the context that is addressed in order to define a structure as given in Figure 59.
- *Case* and the associated argumentation expresses the decomposition of a main claim, that a target system is safe, by a strategy of explicitly addressing all risks associated with the system (identified by the instantiation of *Risks*).

Related Patterns: The *Explicit Risk Evaluation* pattern is related to other patterns in the following manner:

- May be used for detailing the goal ERE in the *Technical Safety* pattern concerned with demonstrating that the risks associated with a system is explicitly addressed.
- May be used to develop a safety demonstration for an instantiation of a design pattern where it is assumed that no implicit argument is suitable such that the safety property of the system must be explicitly addressed.

Known Uses: The pattern describes a structure for arguing safety objectives being satisfied on the basis of explicitly addressing the risks associated with a target system. The strategy is expressed in the application guideline to the European Regulation on common safety methods within railway [ERA/GUI/01-2008/SAF].

[ERA/GUI/01-2008/SAF] Guide for the application of the Commission Regulation on the adoption of a common safety method on risk evaluation and assessment as referred to in Article 6(3)(a) of the Railway Safety Directive, European Railway Agency, 2009.

B.16 SAFETY REQUIREMENTS SATISFIED

Name: Safety Requirements Satisfied

Pattern Signature: *Safety Requirements Satisfied* is defined with the signature illustrated in Figure 60.

In Figure 60, the following abbreviations are used for denoting the parameters of the pattern:

- *ToD* is short for Target of Demonstration.
- *Req* is short for Requirements.
- *ReqSat* is short for Requirements Satisfied.
- *Case is short for Safety Case.*

Figure 60 Safety Requirements Satisfied – Pattern Signature

Intent: Provide a structure for safety demonstration *Case* that a system *ToD* is safe, by arguing that all safety requirements *Req* are satisfied.

Example: a system S shall satisfy the functional requirements R1 and R2, and safety requirements SR1 and SR2. In order to demonstrate that S is sufficiently safe for its intended purpose it is enough to demonstrate that SR1 and SR2 are satisfied. In order for system S to provide the intended function then R1 and R2 must be satisfied, but as these requirements do no impact safety they are not addressed in the safety demonstration.

Applicability: The *Safety Requirements Satisfied* pattern is intended for the following situation:

- When it is required to provide an explicit demonstration of the ability of a system to uphold safety invariants that are expressed as safety requirements.

Problem: The main aspects to be considered when providing a solution for the demonstration of safety are:

- *Completeness of specification*: it is necessary to provide confidence that the set of safety requirements is complete, i.e. contain all relevant requirements.
- *Correct demonstration approach*: it is necessary to provide confidence that for each requirement, the chosen approach for demonstrating that the requirement is satisfied is suitable.
- *Confirming evidences*: it is necessary to provide confidence that for each chosen demonstration approach, confirming evidences is available or will be provided.

Argument Structure Solution: Figure 61 and Table 17 together present the demonstration solution, and therefore should be read together. Figure 61 illustrates the decomposition of the safety argument in a tree structure using GSN notation. Table 17 details the tree structure by defining the type and expected content of each node.

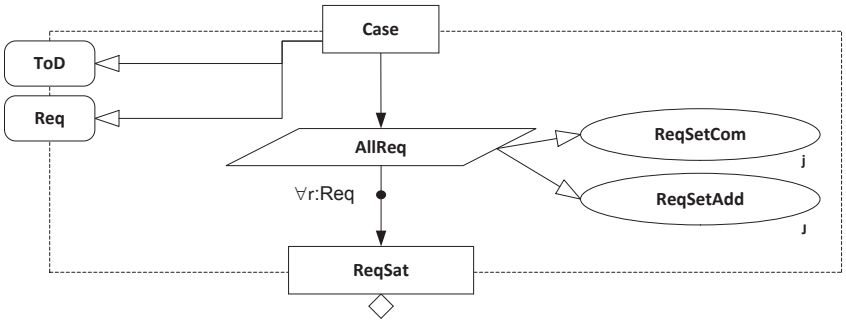


Figure 61 Safety Requirements Satisfied – Argument Structure

Associated with the contextual elements of the argument structure, the following is assumed:

- *ToD*: identifies the target system under consideration.
- *Req*: represents a set of safety requirements associated with the target system *ToD*.

Table 17 Safety Requirements Satisfied - Argument Structure

Node	Node Type	Node Content Description
Case	Goal	*ToD* satisfies *Req*
ToD	Context	Definition of *ToD*
Req	Context	Definition or *Req*
AllReq	Strategy	*ToD* is acceptably safe as all safety requirements are addressed and found satisfied
ReqSetCom	Justification	Justification for the requirements set *Req* being complete or contain all safety requirements that are relevant for *ToD*
ReqSetAdd	Justification	Justification for all elements of the set *Req* are accounted for. Elements may be addressed individually or in groups, any grouping of requirements needs to be justified.
ReqSat	Goal	The requirement *r,* element of *Req*, is satisfied

Instantiation Rule: A demonstration artefact *Case* (see Figure 61 and Figure 60) instantiates the *Safety Requirements Satisfied* pattern if:

- *Case* is a safety demonstration of a target system *ToD*.
- Every element of *Case* is traceable to a safety case node as indicated in Figure 61. A safety case node is instantiated by adapting the descriptions in column “Node Content Description” in Table 17 to the context that is addressed in order to define a structure as given in Figure 61.

- *Case* and the associated safety argumentation expresses the decomposition of a main claim, that a target system *ToD* is safe, by a strategy of demonstrating that all safety requirements *Req* associated with the system *ToD* are satisfied.

Related Patterns: The *Safety Requirements Satisfied* pattern is related to other patterns in the following manner:

- May be used to detail the ERE node of *Technical Safety* concerned with demonstrating that risk is explicitly addressed by demonstrating that the safety requirements, defined on the basis of risk assessment, are satisfied.
- May be used together with the *Establish System Safety Requirements. Establish System Safety Requirements* can be applied in order to derive the safety requirements. *Safety Requirements Satisfied* is then used to develop a demonstration that argues that identified requirements are satisfied.

B.17 ASSESSMENT EVIDENCE

Name: Assessment Evidence

Pattern Signature: *Assessment Evidence* is defined with the signature illustrated in Figure 62.

In Figure 62, the following abbreviations are used for denoting the parameters of the pattern:

- *ToD* is short for Target of Demonstration.
- *Cond* is short for Condition.
- *Crit* is short for Criteria.
- *Mtd* is short for Method.
- *CndSatEv* is short for Condition Satisfied Evidence
- *Case* is short for Safety *Case.*

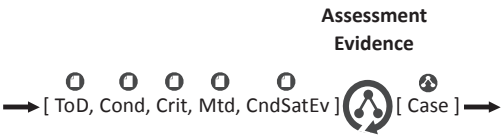


Figure 62 Assessment Evidence – Pattern Signature

Intent: Provide a structure for the specification of a safety demonstration *Case* where a piece of evidence *CndSatEv*, obtained on the basis of an assessment, provides support for claiming that some condition *Cond* associated with a target *ToD* is satisfied.

Applicability: The *Assessment Evidence* pattern is suitable to apply in the following situations:

- When there exists documentable evidence that is provided on the basis of a systematic assessment of a target.
- When the assessment method used is commonly accepted as suitable for the assessment performed and where the result of its application provides supporting evidence for safety claims.

Problem: The main aspects to be considered when providing a solution for the demonstration of safety are:

- *Target:* it is necessary that the assessment report clearly states the target of the assessment.
- *Objective:* It is necessary that the objective of the assessment be clearly stated.
- *Method:* it is necessary to clearly identify the method(s) applied in order to perform the assessment such that it may be determined if the applied method(s) was suitable to apply in order to address the target with respect to the assessment objective.
- Scope: it is necessary that the scope of the assessment is clearly stated such that the depth of the assessment may be obtained from the report.
- *Assumptions:* it is necessary that the assumptions on the application of the method or on the specification of scope is clearly stated such that it may be

determined if the assessment is suitable to reference with respect to a stated demonstration claim.

- *Results:* it is necessary that the result of the assessment be clearly stated such that the assessment results also may be assessed.
- *Conclusions:* it is necessary that assessment summarises its results into a conclusion such that what has been achieved is clearly stated as well as the conditions for which the result is valid.

Argument Structure Solution: Figure 63 and Table 18 together present the demonstration solution, and should therefore be read together. Figure 63 illustrates the decomposition of the safety argument in a tree structure using the GSN notation. Table 18 details the tree structure by defining the type and the expected content of each node.

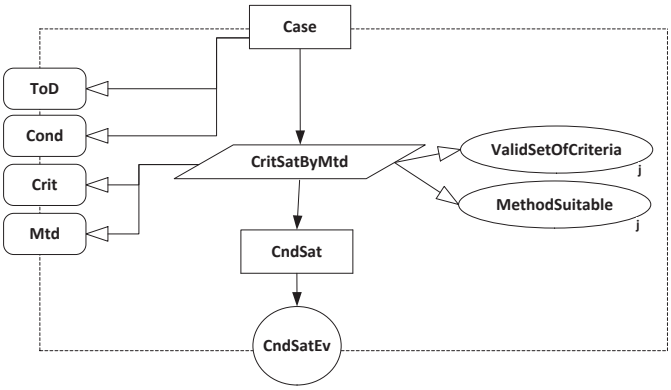


Figure 63 Assessment Evidence – Argument Structure

Table 18 Assessment Evidence – Argument Structure

Node	Node Type	Node Content Description
Case	Goal	*ToD* satisfies *Cond*
ToD	Context	Definition of *ToD*
Cond	Context	Definition of *Cond*
CritSatByMtd	Strategy	Assessment method *Mtd* satisfies criteria *Crit* thus the assessment result is valid
Crit	Context	Definition of *Crit*
Mtd	Context	Definition of *Mtd*
ValidSetOfCriteria	Justification	Justification for the set *Crit* represents the characteristics of a valid assessment approach
MethodSuitable	Justification	Justification for *Mtd* satisfies *Crit* and thus is a suitable assessment approach

CndSat	Goal	Condition *Cond* is assessed by *Mtd* and found fulfilled
CndSatEv	Solution	Description of *CndSatEv*

Associated with the contextual elements and the solution elements of the argument structure, the following is assumed:

- ToD: identifies the target system under consideration.
- Cond: identifies the condition that is assessed.
- Crit: represents the set of criterias that must be fulfilled in order for the assessment approach to represent a valid assessment.
- Mtd: identifies the method applied in order to obtain evidences.
- CndSatEv: represents the documentable evidence supporting the claim provided in the argument.

Instantiation Rule: An artefact *Case* (see Figure 63 and Figure 62) instantiates the *Assessment Evidence* pattern if:

- *Case* is a safety demonstration of a target system (identified by the instantiation of *ToD)*.
- Every element of *Case* is traceable to a safety case node as indicated in Figure 63. A safety case node is instantiated by adapting the descriptions in column “Node Content Description” in Table 18 to the context that is addressed in order to define a structure as given in Figure 63.
- *Case* and the associated argumentation expresses the decomposition of a main claim, that a target system is safe with respect to a stated condition (identified by the instantiation of *Cond*), by a strategy of arguing that a specific assessment method (identified by the instantiation of *Mtd*) satisfies criteria (identified by the instantiation of *Crit*) for assessing the condition in a suitable manner. The assessment evidence (identified by the instantiation of *CndSatEv*) confirms that *Cond* is satisfied.

Related Patterns: The *Assessment Evidence* pattern is related to other patterns in the following manner:

- May be used for detailing a demonstration goal of any safety case pattern that rely on evidences obtained on the basis of assessment in order to support the claim defined by the goal.

Known Uses: The pattern describes an argument structure where a claim is supported by evidence derived on the basis the application of a suitable assessment method. One example of the need to argue that a suitable assessment technique has been applied may be seen from Appendix A of [EN 50128]. Table A.9 within Appendix A provides recommendations on the application of specific techniques for assessing software depending on their software safety integrity level (SWSIL) classification.

[EN50128] European Committee for Electrotechnical Standardization, Railway Applications – Communications, signalling and processing systems – Software for railway control and protection systems, EN 50128, 2001.

B.18 Process Quality Evidence

Name: Process Quality Evidence

Pattern Signature *Process Quality Evidence* is defined with the signature illustrated in Figure 64.

In Figure 64, the following abbreviations are used for denoting the parameters of the pattern:

- *ToD* is short for Target of Demonstration.
- *Cond* is short for Condition.
- *Proc* is short for Process.
- *QualChar* is short for Quality Characteristics.
- *CharSatEv* is short for Characteristics Satisfied Evidence.
- *Case* is short for Safety *Case.*

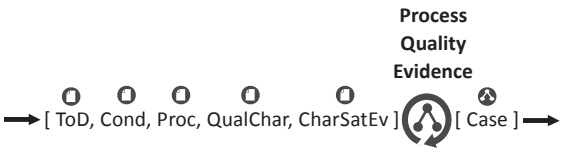


Figure 64 Process Quality Evidence – Pattern Signature

Intent: Provide a structure for the specification of a safety demonstration *Case* showing that some condition *Cond* is met by a system *ToD* on the basis of evidence that results from application of a non-standardised process that assures quality. The motivation for applying the pattern is when a non-standardised process is applied as a means to address a stated concern. As the applied process is not standardised or otherwise well established within its application domain, a compliance argument is not sufficient and an additional rationale of the suitability of the process must be provided.

Applicability: The *Process Quality Evidence* pattern is suitable to apply in the following situations:

- When an applied process that is well established in a company or other domain than in the domain for which the safety demonstration are developed.
- When the process is not a recommended practice within the target domain.
- When the process is novel but satisfies required qualities and/or has proven effective and thus is of adequate quality for its intended purpose.

Problem: The main aspects to be considered when providing a solution for the demonstration of safety are:

- *Characteristics*: it is necessary to identify the quality characteristics required to be supported by the process that establishes it as a good process.
- *Documentation*: it is necessary to identify the documentations that establish that the process comply to the required characteristics.

Argument Structure Solution: Figure 65 and Table 19 together present the demonstration solution, and should therefore be read together. Figure 65 illustrates the decomposition of the safety argument in a tree structure using the GSN notation. Table 19 details the tree structure by defining the type and expected content of each node.

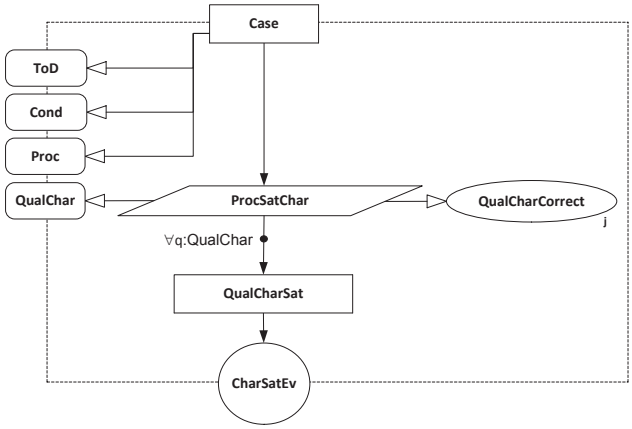


Figure 65 Process Quality Evidence – Argument Structure

Table 19 Process Quality Evidence – Argument Structure

Node	Node Type	Node Content Description
Case	Goal	Condition *Cond* associated with *ToD* is addressed by process *Proc*
ToD	Context	Definition of *ToD*
Cond	Context	Definition of *Cond*
Proc	Context	Definition of *Proc*
ProcSatChar	Strategy	Process *Proc* satisfies quality characteristics *QualChar* and thus is of adequate quality
QualChar	Context	Definition of *QualChar*
QualCharCorrect	Justification	Justification for the characteristics expressed in *QualChar* covers the characteristics of a process that is suitable in order to address condition *Cond*
QualCharSat	Goal	*CharSatEv* document the fulfilment of quality characteristic *QualChar*
CharSatEv	Solution	Definition of *CharSatEv*

Associated with the contextual elements of the argument structure, the following is assumed:

- ToD: identifies the target system under consideration.
- Cond: identifies the condition to be addressed by means of an established process.
- Proc: identifies the process applied in order to address the specified condition.
- QualChar: represents the set of quality characteristics associated with the process that jointly provide confidence in that the process is suitable to apply in order to address the stated condition.
- CharSatEv: represents the documentable description that establishes that the process satisfies the required quality characteristics.

Instantiation Rule: An artefact *Case* (see Figure 65 and Figure 64) instantiates the *Process Quality Evidence* pattern if:

- *Case* is a safety demonstration of a target system (identified by the instantiation of *ToD)*.
- Every element of *Case* is traceable to a safety case node as indicated in Figure 65. A safety case node is instantiated by applying the descriptions in column “Node Content Description” in Table 19 to the context that is addressed in order to define a structure as given in Figure 65.
- *Case* and the associated argumentation expresses the decomposition of a main claim, that a target system is safe with respect to a stated condition (identified by the instantiation of *Cond*), by a strategy of arguing that a specific process (identified by the instantiation of *Proc*) satisfies a set of quality characteristics (identified by the instantiation of *QualChar*). The assessment evidence (identified by the instantiation of *CharSatEv*) confirms that quality characteristics are satisfied.

Related Patterns: The *Process Quality Evidence* pattern is related to other patterns in the following manner:

- May be used together with any safety case pattern for detailing a demonstration goal where arguing and providing evidence for that a certain development process has been followed is adequate for addressing the concern addressed by the goal.

B.19 PROCESS COMPLIANCE EVIDENCE

Name: Process Compliance Evidence

Pattern Signature *Process Compliance Evidence* is defined with the signature illustrated in Figure 66.

In Figure 66, the following abbreviations are used for denoting the parameters of the pattern:

- *ToD* is short for Target of Demonstration.
- *Cond* is short for Condition.
- *Proc* is short for Process.
- *AppGde* is short for Application Guidance.
- *CompEv* is short for Compliance Evidence.
- *Case is short for Safety Case.*

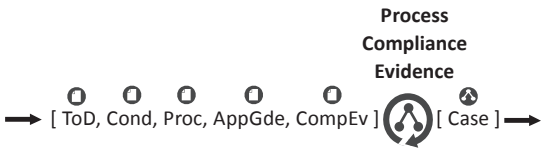


Figure 66 Process Compliance Evidence – Pattern Signature

Intent: Provide a structure for the specification of a safety demonstration *Case* showing that some condition *Cond* is met by a system *ToD* on the basis of evidence of compliance to a well-known and recommended practice. The motivation for applying the pattern is to reduce the effort required in order to motivate and demonstrate that a certain process is adequate by reference to a recommended or mandatory processes that is widely known within a domain to be effective. In such a situation, the demonstration effort is reduced to the demonstration of compliance, as the process itself is pre-approved.

Applicability: The *Process Compliance Evidence* pattern is suitable to apply in the following situations:

- When there exist recommended process practices within a domain that clearly state the situations for which a given practice is recommended and there is a match between the recommended situation of use and the objective for applying the process.
- When it may be established that there is a match between the recommended application of a process and the application of the process.

Problem: The main aspects to be considered when providing a solution for the demonstration of safety based on process compliance are:

- *Documentation*: it is necessary to identify the documentations that establish that the process is suitable for the intended purpose.
- *Compliance*: it is necessary to argue that the process is applied as intended.

Argument Structure Solution: Figure 67 and Table 20 together present the demonstration solution, and should therefore be read together. Figure 67 illustrates the

decomposition of the safety argument in a tree structure using GSN notation. Table 20 details the tree structure by defining the type and expected content of each node.

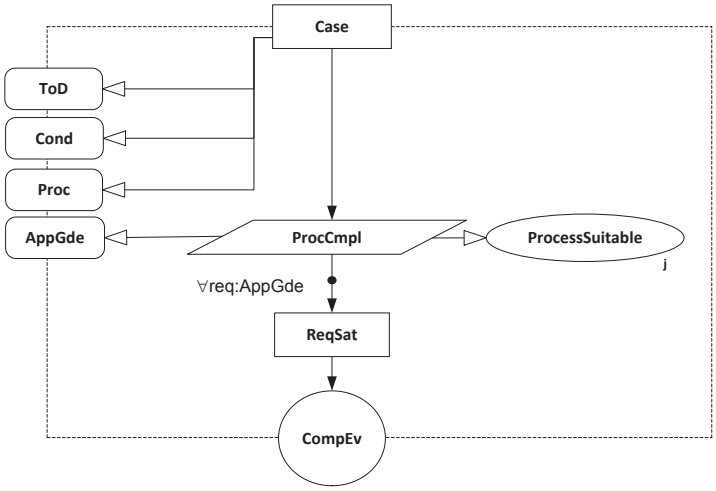


Figure 67 Process Compliance Evidence – Argument Structure

Table 20 Process Compliance Evidence – Argument Structure

Node	Node Type	Node Content Description
Case	Goal	*ToD* satisfies *Cond* by compliance to process *Proc*
ToD	Context	Definition of *ToD*
Cond	Context	Definition of *Cond*
Proc	Context	Definition of *Proc*
ProcCompl	Strategy	Process *Proc* is associated with application guideline. Compliance to *Proc* is shown by demonstrating compliance to application guideline *AppGde*
ProcessSuitable	Justification	Justification for the required characteristics defined in *Cond* are covered by the characteristics of the process *Proc*
AppGde	Context	Definition of *AppGde*
ReqSat	Goal	Documentation *CompEv* state the fulfilment of all requirements of *AppGde*
CompEv	Solution	Definition of *CompEv*

Associated with the contextual elements of the argument structure, the following is assumed:

- ToD: identifies the target system under consideration;
- Cond**:** identifies the condition to be addressed by means of an established process;
- Proc**:** identifies the process applied in order to address the specified condition;
- AppGde**:** represents the requirements and guidance to the application of the process;
- CompEv**:** represents the reference to a documentable description that establishes that process has been followed.

Instantiation Rule: A demonstration artefact *Case* (see Figure 67 and Figure 66) instantiates the *Process Compliance Evidence* pattern if:

- *Case* is a safety demonstration of a target system (identified by the instantiation of *ToD)*.
- Every element of *Case* is traceable to a unique abstract safety case element as indicated with GSN notation in Figure 67. An abstract safety case element is instantiated by applying the descriptions in column “Node Content Description” in Table 20 in order to define a structure as given in Figure 67.
- *Case* expresses the decomposition of a main claim, that a target system is safe with respect to a stated condition (identified by the instantiation of *Cond*), by demonstrating compliance to a commonly accepted process (identified by the instantiation of *Proc*). The evidence (identified by the instantiation of *ChompEv*) contains information on the application of the process in line with requirements (identified by the instantiation of *AppGde*).

Related Patterns: The *Process Compliance Evidence* pattern is related to other patterns in the following manner:

- May be used for detailing a demonstration goal of any safety case pattern where arguing and providing evidence for compliance to a certain development process is adequate for addressing the concern addressed by the goal.

Known Uses: The pattern describes an argument structure where the evidence of compliance to a process that is widely known as providing effective results is used to argue that a claim is met. A typical use of this strategy may be to claim that, e.g., the software in a given system is developed according to the process described in a particular software safety standard as [EN50128] or [IEC 60880] and thereby providing support to a claim on adequate software development process.

[EN50128] European Committee for Electrotechnical Standardization, Railway Applications – Communications, signalling and processing systems – Software for railway control and protection systems, EN 50128, 2001.

[IEC 60880] Nuclear power plants – Instrumentation and control systems important to safety – Software aspects for computer-based systems performing category A functions, IEC 60880, International Electrotechnical Commission, 2006.

B.20 Deterministic Evidence

Name: Deterministic Evidence

Pattern Signature: *Deterministic Evidence* is defined with the signature illustrated in Figure 68.

In Figure 68, the following abbreviations are used for denoting the parameters of the pattern:

- *ToD* is short for Target of Demonstration.
- *Cond* is short for Condition.
- *CondElmEv* is short for Condition Element Evidence.
- *CondEv* is short for Condition Evidence
- *Case* is short for Safety Case.
- *VldAsm* is short for Valid Assumption.

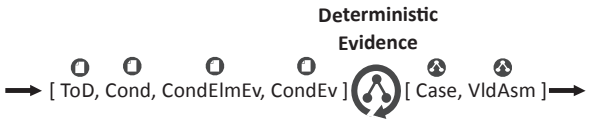


Figure 68 Deterministic Evidence – Pattern Signature

Intent: Provide a structure for the specification of a safety demonstration *Case* showing that a condition *Cond* is met by a target system *ToD* on the basis of a deterministic evaluation of the target. The motivation for applying the pattern is when it is possible to provide irrefutable evidence for some condition always being either true or false.

Example #1: a simple system S undergoes exhaustive testing where all possible system states are tested. Thus, the evidence from the exhaustive system test may be used to prove that the system cannot enter any potential hazardous states.

Example #2: a characteristic C of a system S is proven to be always true by a logical argument that establish that there are no events such that S may not be characterised as C.

The two examples both provide deterministic evidence, the former example by an empirical approach of testing all possibilities, the latter example by a logic reasoning approach.

Applicability: The *Deterministic Evidence* pattern is suitable to apply in the following situations:

- When there is some characteristic of the system that requires irrefutable evidence in order to establish required confidence. If this is the case, then measures must be taken in order to provide the ability to obtain the deterministic evidence.
- When there exists evidence that supports the establishment of a deterministic argument as it provides a high degree of confidence in support of a claim.

Problem: The main aspects to be considered when providing a solution for the demonstration of safety are:

- *Claim*: it is necessary to define the claim along with any exceptions such that it is clear what is always true and/or what is always false.
- *Reasoning*: it is necessary to define the reasoning steps that are used to combine assumptions to assertions.
- *Assumption*: it is necessary to establish the assumption on which the deterministic evidence is based, i.e. the axioms.
- *Data*: an assumption may be of a nature such that it expresses an established truth and thus requires no further evidence, or it may be of such a nature that the assumption needs to be confirmed in order to provide a valid foundation in an argument. Either way, a rationale or supporting data should be provided in order to establish an assumption as valid.

Argument Structure Solution: Figure 69 and Table 21 together present the demonstration solution, and therefore should be read together Figure 69 illustrates the decomposition of the safety argument in a tree structure using GSN notation. Table 21 details the tree structure by defining the type and expected content of each node.

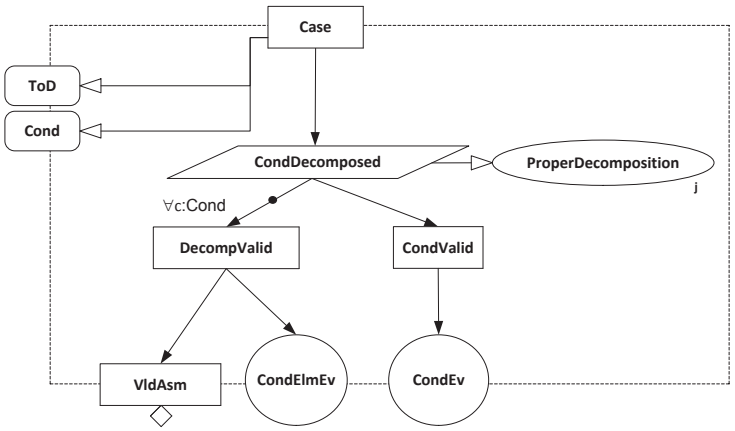


Figure 69 Deterministic Evidence – Argument Structure

Table 21 Deterministic Evidence – Argument Structure

Node	**Node Type**	**Node Content Description**
Case	Goal	*ToD* satisfies *Cond*
ToD	Context	Definition of *ToD*
Cond	Context	Definition of *Cond*
CondDecomposed	Strategy	*Cond* is systematically decomposed. Every decomposed element and the composition of elements supports *Cond*

ProperDecomposition	Justification	Justification of proper decomposition of the condition *Cond* into decomposed elements
DecompValid	Goal	Decomposed element *c* is determined on the basis of a valid assumption (VldAsm) and/or evidence (CondElmEv)
VldAsm	Goal	Assumption element *a* is valid
CondElmEv	Solution	Evidence establishing condition element *c*
CondValid	Goal	*Cond* is established
CondEv	Solution	Evidence establishing condition *Cond* on the basis of all proposition elements *c*

Associated with the contextual elements of the argument structure, the following is assumed:

- ToD: identifies the target system under consideration.
- Cond: represents the condition that is addressed.
- CondElmEv: represents the reference to a documentable description that establishes in a deterministic manner the truth associated with an element of the condition that is addressed.
- CondEv: represents the reference to a documentable description that establishes in a deterministic manner the truth associated with the conditions that is addressed based on the set of condition elements.

Instantiation Rule: An artefact *Case* (see Figure 69 and Figure 68) instantiates the *Deterministic Evidence* pattern if:

- *Case* is a safety demonstration of a target system (identified by the instantiation of *ToD)*.
- Every element of *Case* is traceable to a safety case node as indicated in Figure 69. A safety case node is instantiated by adapting the descriptions in column “Node Content Description” in Table 21 to the context that is addressed in order to define a structure as given in Figure 69.
- *Case* and the associated safety argumentation expresses the decomposition of a main claim, that a target system is safe with respect to or satisfies a stated condition (identified by the instantiation of *Cond*), by a strategy of establishing an irrefutable argument.

Related Patterns: The *Deterministic Evidence* pattern is related to other patterns in the following manner:

- May be used for detailing a demonstration goal of any safety case pattern where arguing and providing evidence for the concern addressed by the goal being satisfied must be based on irrefutable evidence.

Known Uses: The pattern describes an argument structure where a claim is supported by evidence that demonstrates the claim being fully predictable. One example of the need for deterministic evidence support is related to the recommendation expressed in Table A.12 within Appendix A of [EN 50128] requiring for SIL 4 classified software no dynamic objects, no dynamic variables, and no conditional jumps.

[EN50128] European Committee for Electrotechnical Standardization, Railway Applications – Communications, signalling and processing systems – Software for railway control and protection systems, EN 50128, 2001.

B.21 PROBABILISTIC EVIDENCE

Name: Probabilistic Evidence

Pattern Signature: *Probabilistic Evidence* is defined with the signature illustrated in Figure 70.

In Figure 70, the following abbreviations are used for denoting the parameters of the pattern:

- *ToD* is short for Target of Demonstration.
- *Cond* is short for Condition.
- *Mtd* is short for method.
- *Crit* is short for Criteria.
- *CritSatEv* is short for Criteria Satisfied Evidence.
- *ProbEv* is short for Probabilistic Evidence.
- *Case* is short for Safety Case.

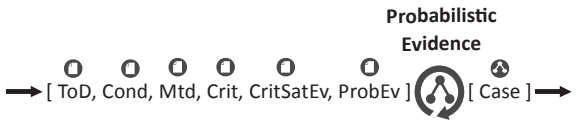


Figure 70 Probabilistic Evidence – Pattern Signature

Intent: Provide a structure for the specification of a safety demonstration *Case* showing that a condition *Cond* or a claim associated with a target system *ToD* is satisfactory met on the basis of evidence obtained by a quantitative assessment of the target. The motivation for applying the pattern is when there exists sufficient quantitative information such that the occurrence of some event may be predicted with sufficient confidence to support a safety claim.

Applicability: The *Probabilistic Evidence* pattern is suitable to apply in the following situations:

- When there exists quantitative information that may be used directly or systematically combined such that it supports a quantitatively expressed claim.
- Typically used in order to provide a notion of the degree for which a system fulfils a specific property (e.g., availability, reliability, safety) by a method of calculation and combination of random hardware failure probabilities obtained on individual hardware items.

Problem: The main aspects to be considered when providing a solution for the demonstration of safety are:

- *Result*: it is necessary that the probabilistic result is clearly stated in order to avoid ambiguity.
- *Validity of Result*: it is necessary to identify the characteristics of the process of how the result was obtained in order to provide confidence in the result and render possible to reproduce it.
- *Assumptions*: it is necessary to document the assumptions regarding any circumstances that render the probabilistic evidence valid or otherwise in which situation the result is not valid.

- *Method*: it is necessary to document by which method the result was obtained.

Argument Structure Solution: Figure 71 and Table 22 together present the demonstration solution, and therefore should be read together. Figure 71 illustrates the decomposition of the safety argument in a tree structure using the GSN notation. Table 22 details the tree structure by defining the type and expected context of each node.

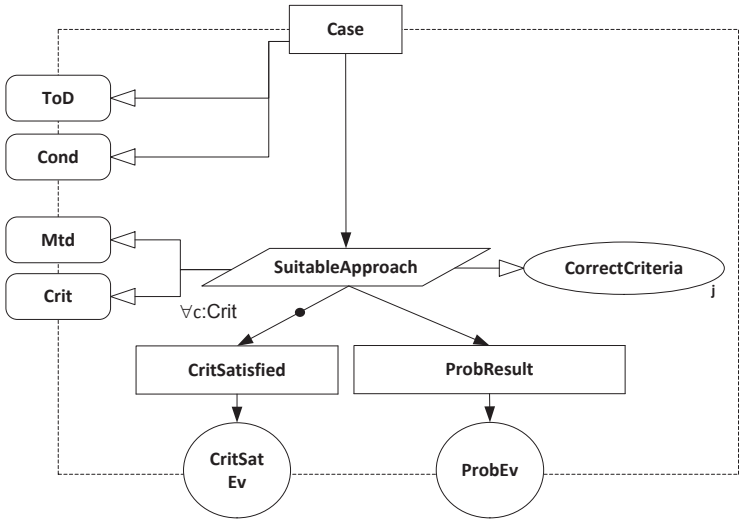


Figure 71 Probabilistic Evidence – Argument Structure

Table 22 Probabilistic Evidence – Argument Structure

Node	Node Type	Node Content Description
Case	Goal	*Cond* occurs with probability *p* in ToD.
ToD	Context	Definition of *ToD*
Cond	Context	Definition of *Cond*
SuitableApproach	Strategy	The occurrence of *Cond* is obtained based on *Mtd*. *Mtd* satisfies *Crit* thus result is trustworthy.
Mtd	Context	Definition of *Mtd*
Crit	Context	Definition of *Crit*
CorrectCriteria	Justification	Rationale over the *Crit* being suitable for characterising the ability of *Mtd* to provide trustworthy quantitative results
CritSatisfied	Goal	Application of *Mtd* satisfies criteria *c*
ProbResult	Goal	*Mtd* establishes that condition *Cond* occurs with probability *p*
CritSatEv	Solution	Evidence documenting that *Mtd* satisfies criteria *c*
ProbEv	Solution	Evidence documenting that *Cond* occurs with probability *p*

Associated with the contextual elements of the argument structure, the following is assumed:

- ToD: identifies the target system under consideration.
- Cond: represents conditions to be fulfilled that are associated with probability of occurrence.
- Mtd: represents the method applied in order to obtain a probability for some given event.
- Crit: represents the criteria that shall be fulfilled by the method that is applied in order to give confidence in the probabilistic result being correct.
- CritSatEv: represents the reference to a documentable description that establishes that the criteria are satisfied.
- ProbEv: represents the reference to a documentable description that establishes that *Cond* occurs with probability *p*.

Instantiation Rule: An artefact *Case* (see Figure 71 and Figure 70) instantiates the *Probabilistic Evidence* pattern if:

- *Case* is a safety demonstration of a target system (identified by the instantiation of *ToD)*.
- Every element of *Case* is traceable to a safety case node as indicated in Figure 71. A safety case node is instantiated by adapting the descriptions in column

"Node Content Description" in Table 22 to the context addressed in order to define a structure as given in Figure 71.

- *Case* and the associated argumentation expresses the decomposition of a main claim, that a condition (identified by the instantiation of *Cond)* occurs with a specific probability that is acceptable and thus is sufficiently safe. The probability is established on the basis of an acceptable method and backed up by evidences.

Related Patterns: The *Probabilistic Evidence* pattern is related to other patterns in the following manner:

- May be used for detailing a demonstration goal of any safety case pattern where it is sufficient to argue and provide evidence for the concern addressed by the goal occurs with a certain probability and thus is acceptable.

Known Uses: The pattern describes an argument structure where the evidence supporting a claim is derived on the basis of probabilistic methods. Appendix A of [EN 50128] identifies some of the recommended techniques to be used as support developing the software for railway signalling and protections systems. Several of these techniques such as reliability block diagram, fault tree analysis, and Markov models may be used to derive probabilistic results.

[EN50128] European Committee for Electrotechnical Standardization, Railway Applications – Communications, signalling and processing systems – Software for railway control and protection systems, EN 50128, 2001.

B.22 Basic Assumption Evidence

Name: Basic Assumption Evidence

Pattern Signature: *Basic Assumption Evidence* is defined with the signature illustrated in Figure 72.

In Figure 72, the following abbreviations are used for denoting the parameters of the pattern:

- *ToD* is short for Target of Demonstration.
- *Cond* is short for Condition.
- *AsmEv* is short for Assumption Evidence.
- *Case* is short for Safety Case.

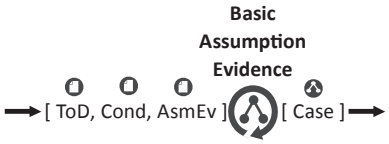


Figure 72 Basic Assumption Evidence – Pattern Signature

Intent: Provide a structure for the specification of a simple safety demonstration *Case* showing that some condition *Cond* is met by a system *ToD* evidenced in an assumption that is claimed valid on the basis of a rationale or by the support of evidence indicating that the assumption may be used as a basic fact. The assumption is not supported by "hard" evidence but expresses a phenomenon that is self-evident such that it may be treated as an axiom. If there is no information that supports an assumption, then the demonstration must terminate in a rationale and/or evidence that provide indications strong enough to establish the assumption as a basic fact.

Applicability: The *Basic Assumption Evidence* may be applied in the following situations:

- When there exists no evidence that may be referenced such that a claim may be backed up by facts.
- When there exist strong indications or a rationale such that the assumption may be treated as valid.

Problem: When developing a safety demonstration, some information is based on assumptions. An assumption needs to be motivated such that:

- *Rationale:* described to an extent that rationalise that the assumption is a basic fact that requires no further evidence and that may be accepted as a valid assumption.
- *Commonly accepted as truth*: described to an extent that provides convincing indications that the assumption is valid although no conclusive evidence may be provided.

Argument Structure Solution: Figure 73 and Table 23 together present the demonstration solution, and should therefore be read together. Figure 73 illustrates the decomposition of the safety argument in a tree structure using the GSN notation. Table 23 details the tree structure by defining the type and the expected content of each node.

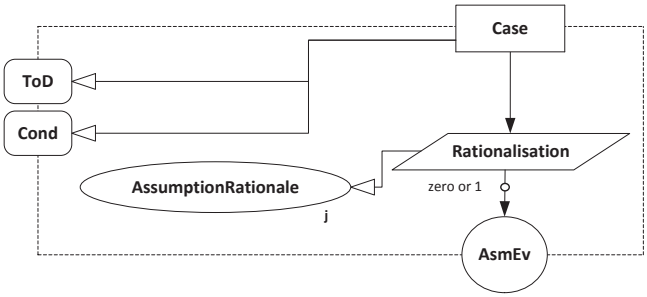


Figure 73 Basic Assumption Safety – Argument Structure

Table 23 Basic Assumption Evidence – Argument Structure

Node	Node Type	Node Content Description
Case	Goal	A valid assumption associated with *ToD* is *Cond*
ToD	Context	Definition of *ToD*
Cond	Context	Definition of assumption *Cond*
Rationalisation	Strategy	*Cond* is established by rationale or non-concluding evidence
AssumptionRationale	Justification	Rationale establishing that assumption *Cond* is a valid basic fact
AsmEv	Solution	Evidence documenting that assumption *Cond* is valid

Associated with the contextual elements of the argument structure, the following is assumed:

- ToD: identifies the target system under consideration.
- Cond: identifies the assumption associated with *ToD.*
- AsmEv: represents the reference to a documentable description that establishes the assumption as a basic fact.

Instantiation Rule: An artefact *Case* (see Figure 73 and Figure 72) instantiates the *Basic Assumption Evidence* pattern if:

- *Case* is a safety demonstration of a target system (identified by the instantiation of *ToD)*.
- Every element of *Case* is traceable to a safety case node as indicated in Figure 73. A safety case node is instantiated by adapting the descriptions in column

"Node Content Description" in Table 23 to the context that is addressed in order to define a structure as given in Figure 73.

- *Case* and the associated argumentation expresses the decomposition of a main claim, that an assumption is valid (identified by the instantiation of *Asm)* justified by a rationale and/or documentable evidence.

Related Patterns: The *Basic Assumption Evidence* pattern is related to other patterns in the following manner:

- May be used for detailing a demonstration goal of any safety case pattern that define a claim which are based on assumptions and where the assumptions are not required to be supported by evidences beyond giving a rationale or by referencing documentation that establish the assumption as a commonly accepted fact.

Known Uses: The pattern describes an argument structure where a kind of rationale or a justified assumption supports a claim such that no further evidence is required. In any kind of argumentation there are some axioms that are used as base facts. The assumptions used in assuring and justifying that safety objectives are met should be justified as required in [IEC61508], [EN50129], and [CommonPosition].

[IEC61508] International Electrotechnical Commission, Functional safety of electrical/electronic/ programmble electronic safety-related systems, IEC 61508, Edition 2.0, 2010.

[EN50129] European Committee for Electrotechnical Standardization, Railway Applications – Communications, signalling and processing systems – Safety related electronic systems for signalling, EN 50129, 2003.

[CommonPosition] Bel V, Bfs, CSN, ISTech, ONR, SSM, and STUK. Licensing of safety critical software for nuclear reactors: Common position of seven European nuclear regulators and authorised technical support organisations, 2013.

Chapter 11

Paper 3: Developing Safe Control Systems using Patterns for Assurance

Instead of the paper we have included the full technical report with the title "A Pattern-based Method for Safe Control Conceptualisation – Exemplified Within Railway Signalling", HWR-1037 rev 2, OECD Halden Reactor Project, Institute for energy technology, Halden, Norway, 2014.

OECD HALDEN REACTOR PROJECT

A Pattern-based Method for Safe Control Conceptualisation – Exemplified Within Railway Signalling

by

André A. Hauge[1,2] and Ketil Stølen[2,3]

[1]*Institute for energy technology, OECD Halden Reactor Project,*

[2]*Department of Informatics, University of Oslo, Norway*

[3]*Department of Networked Systems and Services, SINTEF ICT*

andre.hauge@hrp.no, ketil.stolen@sintef.no

2014-06-04

FOREWORD

The experimental operation of the Halden Boiling Water Reactor and associated research programmes are sponsored through an international agreement by:

- the Institutt for energiteknikk (IFE), Norway,

the Belgian Nuclear Research Centre SCK•CEN, acting also on behalf of other public or private organisations in Belgium,

- the Risø DTU National Laboratory for Sustainable Energy, Technical University of Denmark,
- the Finnish Ministry of Employment and the Economy (TYÖ),
- the Electricité de France (EDF),
- the Gesellschaft für Anlagen- und Reaktorsicherheit (GRS) mbH, representing a German group of companies working in agreement with the German Federal Ministry of Economics and Technology,
- the Japan Nuclear Energy Safety Organization (JNES),
- the Korean Atomic Energy Research Institute (KAERI), acting also on behalf of other public or private organisations in Korea,
- the Spanish Centro de Investigaciones Energéticas, Medioambientales y Tecnológicas (CIEMAT), representing a group of national and industry organisations in Spain,
- the Swedish Radiation Safety Authority (SSM), representing public and privat nuclear organisations in Sweden,
- the Swiss Federal Nuclear Safety Inspectorate ENSI, representing also the Swiss nuclear utilities (Swissnuclear) and the Paul Scherrer Institute,
- the National Nuclear Laboratory (NNL), representing a group of nuclear licensing and industry organisations in the United Kingdom, and
- the United States Nuclear Regulatory Commission (USNRC),

and as associated parties:

- Japan Atomic Energy Agency (JAEA),
- the Central Research Institute of Electric Power Industry (CRIEPI), representing a group of nuclear research and industry organisations in Japan
- the Mitsubishi Nuclear Fuel Co., Ltd. (MNF)
- the Czech Nuclear Research Institute (NRI),
- the French Institut de Radioprotection et de Sûreté Nucléaire (IRSN),
- the Ulba Metallurgical Plant JSC in Kazakhstan,
- the Hungarian Academy of Sciences, KFKI Atomic Energy Research Institute,
- the JSC "TVEL" and NRC "Kurchatov Institute", Russia,
- All-Russian Research Institute for Nuclear Power Plants Operation (VNIIAES), Russia,
- the Slovakian VUJE - Nuclear Power Plant Research Institute, and
- EU JRC Institute for Transuranium Elements, Karlsruhe,

and associated parties from USA:

- the Westinghouse Electric Power Company, LLC (WEC),
- the Electric Power Research Institute (EPRI),
- the Global Nuclear Fuel (GNF) – Americas, LLC and GE-Hitachi Nuclear Energy, LLC, and
- the US Department of Energy (DOE)

The right to utilise information originating from the research work of the Halden Project is limited to persons and undertakings specifically given this right by one of these Project member organisations.

Recipients are invited to use information contained in this report to the discretion normally applied to research and development programmes. Recipients are urged to contact the Project for further and more recent information on programme items of special interest.

Institutt for energiteknikk
OECD HALDEN REACTOR PROJECT

Title

A Pattern-based Method for Safe Control Conceptualisation – Exemplified Within Railway Signalling

Author:

André A. Hauge and Ketil Stølen

Document ID:		First issued:
HWR-1037 rev 2		March 2013

Keywords:

conceptual design; pattern; pattern language; safety; safety case: requirements elicitation

Abstract:

HWR-1037 exemplifies the application of a pattern-based method, called Safe Control Systems (SaCS), on a case taken from the railway domain. The method is supported by a pattern language and provides guidance on development of conceptual safety designs. By a conceptual safety design, we mean an early stage specification of system requirements, system design, and safety case for a safety critical system.

We argue that SaCS facilitates efficient and effective elicitation of system requirements, system design concepts according to requirements, and a safety case for demonstrating that the derived concept satisfies safety objectives by the systematic application of patterns.

The pattern language supports the method by offering six different kinds of basic patterns, operators for combining patterns, and a graphical notation for visualising pattern compositions. Intended users of SaCS are system developers, safety engineers, hardware engineers, and software engineers.

Issue Date:		Name	Signature	Date
2014-06-04	Prepared by:	André A. Hauge	Sign.	2014-04-21
Confidential grade:	Reviewed by:	Vikash Katta	Sign.	2014-05-19
HRP Only	Approved by:	Terje Johnsen	Sign.	2014-06-04

MAIL
P.O. BOX 173
NO-1751 HALDEN
Norway

TELEPHONE
+47 69 21 22 00

TELEFAX
Administration + 47 69 21 22 01
Nuclear Safety + 47 69 21 22 01
Purchasing Office + 47 69 21 24 40

TELEFAX
IND/OC div. + 47 69 21 24 90
VISIT/RID/COSS div. + 47 69 21 24 60
Reactor Plant + 47 69 21 24 70

TABLE OF CONTENTS

1. INTRODUCTION

This report demonstrates and presents experiences from the use of a pattern-based method called Safe Control Systems (SaCS) for the development of a conceptual safety design for a railway interlocking system. A railway interlocking system is a typical safety critical system; an example interlocking system serves as a case for evaluating the SaCS method. An error in the specification and/or in the implementation of an interlocking system may lead to unacceptable consequences, e.g., deaths due to collision of trains caused by signalling failure.

The SaCS method provides guidance on the development of safety critical systems by the application of patterns. SaCS has also been tested out in the nuclear domain for the conceptualisation of a reactor control system design [10].

The six kinds of basic patterns offered by the SaCS pattern language are categorised according to two development perspectives: process assurance and product assurance. Both perspectives detail patterns according to three aspects: requirement; solution; and safety case. We distinguish between basic and composite patterns. Each basic pattern contains an instantiation rule that may be used to assess whether it is correctly instantiated.

The basic SaCS patterns capture commonly accepted safety engineering practices, e.g., development processes and activities, risk assessment methods, and other concepts for providing safety assurance as reflected in international safety standards and guidelines, e.g., [13], [14], [15], and [16]. The basic SaCS patterns are defined in a format inspired by classical literature on patterns, e.g., the format for expressing patterns as defined in [1], [3], and [7]. SaCS patterns differs from patterns defined in classical literature with respect to its explicit definition of inputs, outputs, and the instantiation rules that define the transition from input to output for each pattern. The SaCS way of explicitly defining the parameters (i.e., inputs and outputs) facilitates easy combination of several patterns. The instantiation rules facilitate validation of the result of pattern instantiation based on the pattern definition and the given inputs. A composite pattern defines a specific combination of patterns and is expressed graphically. The notation for expressing composite patterns is inspired by languages for system modelling, e.g., the modelling of patterns by collaborations and modularisation of a specification by decomposition in UML [20]. The graphical notation is also inspired by the literature related to visualisation, e.g., visualisation of risk analysis [19] and general literature on visualisation of complex data [4], [18], and [22].

This report describes the SaCS method, and its supporting language in an example driven manner based on a case from the railway domain.

The remainder of this report is structured as follows: Section 2 provides a background on the SaCS method and the SaCS pattern language. Section 3 describes the railway case. Section 4 defines the success criteria associated with the application of SaCS on a railway case. Sections 5 to 9 exemplify the application of the SaCS method. Section 10 summarises the main results from the application of the SaCS method. Section 11 discusses the fulfilment of success criteria of the case. Section 12 presents related work. Finally, Section 13 concludes.

Definitions of the basic SaCS patterns used in this report are given in Appendix A. Detailed results from pattern instantiation are given in Appendix B.

2. THE CASE: RAILWAY INTERLOCKING SYSTEM

Figure 1 illustrates the main appliances of a train station. The station has two tracks and a level crossing (shown with two vertical lines).

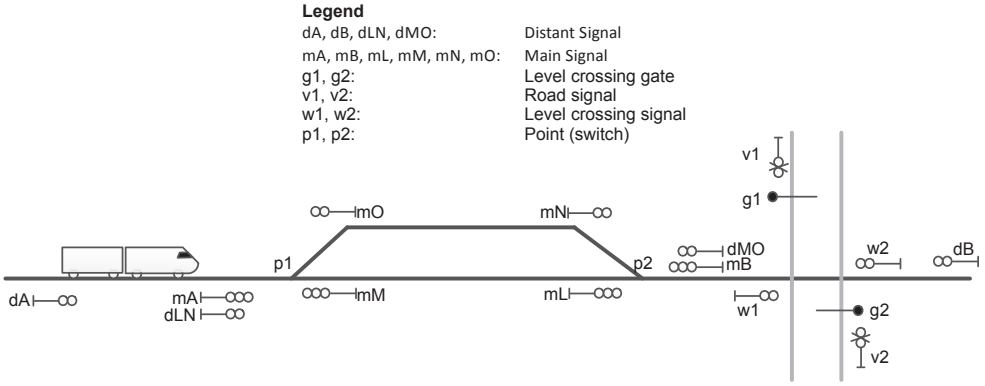


Figure 1 A station with two tracks and a level crossing

The station in Figure 1 is connected in both ends to neighbouring stations with a single track. An interlocking system controls the appliances associated with the station in order to safely control the movements of trains along defined train routes. The annotations in Figure 1 denote the following:

- A short line that ends with two or three adjacent circles represents either a distant light signal (i.e., dA, dB, dLN, and dMO) or a main light signal (i.e., mA, mB, mL, mM, mN, and mO). Distant signals give indication of the expected value of an upcoming main signal in the same direction. The purpose of a distant signal is to allow a train driver to know in advance if it is required to stop or slow down before a main signal is reached. Main signals indicate if a train is allowed to proceed or is required to stop at a specific position on the track.
- There are two points for switching traffic onto different tracks; these are identified as p1 and p2.
- Road signals, v1 and v2, are used to indicate to road users if it is allowed to cross the railway track or if they must stop.
- Level crossing signals, w1 and w2, are used to indicate to train drivers if the level crossing is secured from road traffic.
- Level crossing gates, g1 and g2, are used to hinder road users from crossing the track when a train is passing the level crossing.

In the following sections, we will exemplify the application of the SaCS method for deriving the specification of the interlocking system. The system is referred to as 2TLCI (2 Tracks and Level Crossing Interlocking) system. 2TLCI is intended as a concept for the replacement of an existing interlocking system. The identifiers for points, signals, etc. depicted in Figure 1 represent the naming convention used in this report.

Figure 2 outlines the system context of the 2TLCI system where it is expected to replace the system named Interlocking System.

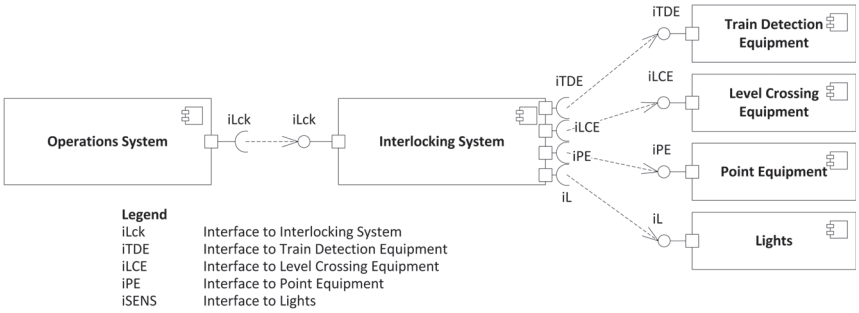


Figure 2 Overview of the overall system

The different entities in Figure 2 represent the following:

- *Operations System*: is a system used by an operator (e.g., train dispatcher) to interact with the interlocking system and may represent a local operator position (operator position at site) or a centralised control position (operator position off-site for remote control).
- *Train Detection Equipment*: any equipment that is used to detect the position of a train on a specific track section, e.g. track circuit detection, axle-counting equipment or other positioning equipment.
- *Level Crossing Equipment*: is equipment that is used for controlling a level crossing and it includes detectors for the passage of trains at specific points on the track, level crossing gates, road signals, and train signals.
- *Point Equipment*: is equipment that is used to move trains from one track to another.
- *Lights*: represents distant signals and main signals that are used to control the movements of trains.

3. BACKGROUND – SACS

3.1 The SaCS Method

The method interleaves three main activities, each of which is divided into sub-activities:

- *Pattern Selection* – The purpose of this activity is to support the conception of a design with respect to a given development case by:
  - Selecting SaCS patterns for requirement elicitation.
  - Selecting SaCS patterns for establishing design basis.
  - Selecting SaCS patterns for establishing safety case.
- *Pattern Composition* – The purpose of this activity is to specify the use of the selected patterns by:
  - Specifying compositions of patterns.
  - Specifying instantiations of patterns.
- *Pattern Instantiation* – The purpose of this activity is to instantiate the composite pattern specification by:
  - Selecting pattern instantiation order.
  - Conducting stepwise instantiation.

A selection map that is introduced in Section 5.1 supports pattern selection. The pattern language outlined in Section 3.2 supports pattern composition. Pattern instantiation is supported by instantiation rules defined for every basic pattern (basic patterns are defined in Appendix A).

3.2 The SaCS Pattern Language

The SaCS pattern language consists of a set of patterns (basic SaCS patterns) of different kinds, and a graphical syntax for specifying how patterns can be combined (creating composite SaCS patterns) and applied in order to derive a conceptual safety design. A composition of patterns (composite for short) may be expressed by, e.g., mapping an output parameter of a pattern to an input parameter to a second pattern, thereby defining a relationship between the patterns. Relations can be used to combine patterns; the relations will be explained by the examples provided in Section 5 to Section 9. A composite may reference and use any type of SaCS pattern (i.e., basic pattern or composite pattern) in its definition. The syntax and semantics of the SaCS pattern language is described in [11]. Every basic SaCS pattern is defined such that it can be used stand-alone. Every pattern is also defined such that it is easy to use several patterns together as every input and output parameter of a pattern is explicitly detailed.

Figure 3 illustrates the main graphical elements used for referring to different types of patterns within the specification of a composite pattern.

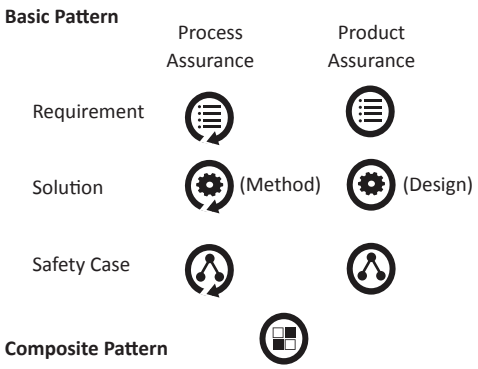


Figure 3 Icons used in pattern references

The six types of basic patterns are:

- *Process Assurance Requirement Pattern*: this pattern type is inspired by the recommended practises for development of safety critical systems as given in safety standards, e.g. [13], [14] and [15], where confidence in achieving a safe product is to a large degree provided by following a recommended process.
- *Product Assurance Requirement Pattern*: this pattern type is inspired by the problem frames approach [17]. The problem frames approach is a means to specify the system and its domains as well as the interaction phenomena between the system and domains in order to derive requirements for the system under development.
- *Process Assurance Solution (Method) Pattern*: this pattern type is defined as a means to document specific methods (e.g., fault tree analysis) that are widely used within safety engineering.
- *Product Assurance Solution (Design) Pattern*: this pattern type is inspired by the format for describing effective design solutions as patterns as given by Alexander et al. [1] and Gamma et al. [7].
- *Process Assurance Safety Case Pattern*: this pattern type is defined as a means to document the structure of claims and evidences such that it may be logically deduced and concluded that a target system is sufficiently safe. This safety case pattern type is used for patterns that primarily depend on process-oriented claims, arguments and evidences.

- *Product Assurance Safety Case Pattern:* this pattern type is similar to the Process Safety Case type, but is used for patterns that primarily depend on product-oriented claims, arguments and evidences.

4. SUCCESS CRITERIA

The SaCS method intends to support users like system developers, safety engineers, hardware engineers, and software engineers in the early stages of system development. The primary objective is to facilitate development of conceptual safety designs of safety critical systems. A safety critical system represents here a system that upon failure may result in losses that are determined unacceptable, e.g., loss of life, significant damage to environment, or unacceptable economic losses. A system may be regarded as sufficiently safe for its intended purpose if it is free form unacceptable failures. In order for a stakeholder to take informed decisions upon further development of a design concept, a conceptual safety design should contain information on the ability of a design to enforce safety.

A conceptual safety design is defined in a suitable manner for a given context if it satisfies the following definition:

Definition *A conceptual safety design is a triplet consisting of an early stage specification of:*

- *system* requirements *(R),*
- *system design (D),*
- *safety case (S).*

An implementation *I* satisfies a conceptual safety design *(R, D, S)* if there exists a safety case *SC* in accordance with *S* that demonstrates that *I* is safe with respect to the specifications of *R* and *D*.

The SaCS method provides a pattern-based method supporting the development of conceptual safety designs by systematically building a context specific composite pattern that is instantiated into a conceptual safety design. The composite pattern that is provided by applying SaCS represents an abstract concept solution whereas its instantiation is a conceptual safety design according to the definition.

We define instantiation in the context of SaCS as follows:

Definition A *conceptual safety design instantiates a SaCS composite pattern if each element of the triple can be instantiated from the SaCS composite pattern according to the instantiation rules of the individual patterns within the composite and according to the rules for composition.*

In the context of SaCS, abstract concepts are provided by patterns, e.g., requirement patterns, design patterns and safety case patterns. Then instantiation of a pattern, e.g., a system design specification, a requirement specification, or a safety case specification, is the result of the application of a pattern in a specific context. Each basic SaCS pattern describes its instantiation rule. The instantiation rule provides guidance on valid instantiations of the pattern. A conceptual safety design according to the definition is a triple of different and related specifications. In order to be effective for safety design conceptualisation, the SaCS method must therefore facilitate the procurement of such an artefact. We claim that SaCS is effective for this task by facilitating the specification of a composite pattern that is easily instantiated into a conceptual safety design and define our hypothesis as follows:

H: *The SaCS method facilitates effective and efficient development of conceptual safety designs that are:*

- *In accordance with safety objectives.*
- *At a* sufficient *level of detail.*

- *Easy to use.*

In order to test our hypothesis, we deduce the following predictions that should hold for the conceptual safety design that is produced when applying SaCS on the interlocking system described in Section 2:

P: *Application of the SaCS method on the interlocking case described in Section 2 results in a conceptual safety design that characterises the interlocking case and is easily instantiated from a composite SaCS pattern. Furthermore, the conceptual safety design:*

a) Is in accordance with safety objectives – the conceptual safety design is defined in agreement with safety objectives.

b) Is at a sufficient level of detail – the conceptual safety design is defined by parts that are appropriate, necessary, sufficiently detailed for an early stage specification, and may be easily understood.

c) Is easy to use – the conceptual safety design may be easily extended, detailed, or refined.

5. ELICIT FUNCTIONAL REQUIREMENTS

5.1 Pattern Selection

Figure 4[1] provides an overview of the patterns available in the interlocking case, which are organised into a pattern selection map. A pattern selection map visualises a specific kind of relationship between the patterns. The kind of relationship that is visualised is the order in which a user is expected to consider the relevancy of a pattern for application in a given context. The patterns that are used in this case are defined in Appendix A, the definition of those patterns that are not used may be found in HWR-1029 [10].

The selection process starts at the pattern referenced in the upper left corner of Figure 4 and follows the direction of the arrows. Pattern selection ends when all patterns have been considered. The diamond symbol represents a choice. Associated with a choice there is a group of relevant patterns where more than one pattern may be selected. The letter above a choice is a reference to a correspondingly named group of patterns indicated in the lower part of Figure 4. The symbol "*" is used to identify the patterns that are used in the interlocking case.

We assume that the information provided in Section 2 sufficiently details the development objective. Furthermore, we assume that the hazards associated with the operation of the interlocking system have been identified and that these are: derailment, collision train-train, collision train-object, and level crossing accident. Based on these assumptions, the application of the first pattern in the selection map named *Establish Concept* is found to be unnecessary as it supports clarifying the initial objectives of a system under construction.

[1] Abbreviations in Figure 4: HAZID – HAZard IDentification; FMEA – Failure Modes and Effects Analysis; HAZOP – HAZard and OPerability Studies; FTA – Fault Tree Analysis; SIL – Safety Integrity Level.

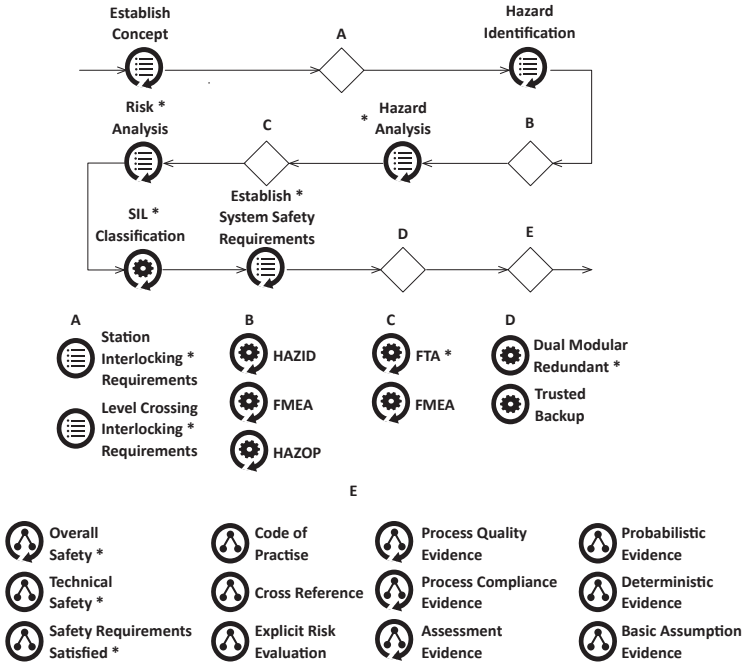


Figure 4 Pattern Selection Map

The patterns *Station Interlocking Requirements* (defined in Appendix A.1) and *Level Crossing Interlocking Requirements* (defined in Appendix A.2) associated with choice (A) in Figure 4 capture the problem of eliciting functional requirements for a system that shall control the appliances available in a station with two tracks and a level crossing, respectively, and therefore have been selected for the case.

5.2 Pattern Instantiation

Functional requirements in the case are elicited by instantiating the *Station Interlocking Requirements* pattern (defined in Appendix A.1) and the *Level Crossing Interlocking Requirements* pattern (defined in Appendix A.2).

The *Station Interlocking Requirements* and the *Level Crossing Interlocking Requirements* patterns are both product assurance requirement patterns, intended as support for deriving the functional requirements for the product that shall be developed. In the case, the product to be developed is an interlocking system for controlling the appliances of a station as illustrated in Figure 1 in Section 2.

Figure 5 is an excerpt from the pattern *Station Interlocking Requirements* and represents a problem frame diagram [17] adapted with SaCS specific annotations for explicitly denoting input and output parameters. The problem frame diagram and the associated textual description within *Station Interlocking Requirements* provide guidance to the user in analysing the challenges addressed by the pattern.

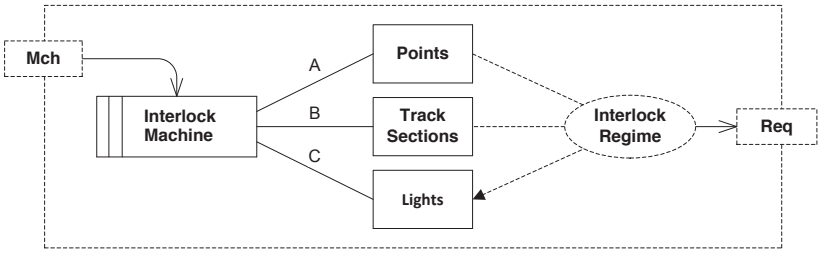


Figure 5 Excerpt of the pattern Station Interlocking Requirements

When the *Station Interlocking Requirements* pattern is instantiated, its parameter *Mch* (short for Machine) indicated in Figure 5 is initially associated with the description of the 2TLCI system as outlined in Section 2. The pattern *Station Interlocking Requirements* is thus interpreted in the context of developing the 2TLCI system. The outcome of applying the pattern according to its instantiation rule is a set of requirements, represented by the output parameter *Req* (short for requirements), specifying how train movements shall be controlled along the tracks by the use of the different appliances as outlined in Figure 1.

Instantiation results are summarised in Section 10 of this report. The result of instantiating the *Station Interlocking Requirements* pattern is described in Table 1 in Section 10.1.

Once the *Station Interlocking Requirements* pattern is instantiated, the *Level Crossing Interlocking Requirements* pattern is instantiated. The pattern *Level Crossing Interlocking Requirements* is interpreted in the context of development of the 2TLCI system (as with the *Station Interlocking Requirements* pattern) and applied according to its instantiation rule to provide the relevant requirements for assuring safe control of a level crossing.

The result of instantiating the *Level Crossing Requirements* pattern according to its instantiation rule is described in Table 2 in Section 10.1.

5.3 Pattern Composition

A composite pattern named *Functional Requirements* defined according to the syntax of the SaCS pattern language is presented in Figure 6. The syntax and semantics for the SaCS pattern language is defined in [11]. The *Functional Requirements* composite pattern defines a relationship between the following basic patterns:

- *Station Interlocking Requirements* – defined in Appendix A.1.
- *Level Crossing Interlocking Requirements* – defined in Appendix A.2.

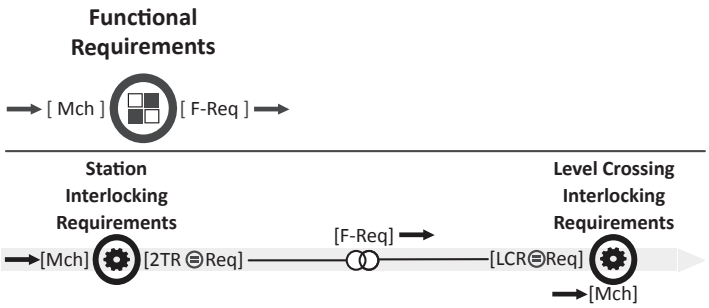


Figure 6 Functional Requirements – Composite Pattern

A composite pattern consists of a declaration and its content. In Figure 6, the declaration is visualised above the horizontal line and identifies the name of the composite pattern that is defined (here the name *Functional Requirements* is given to the composite). In the declaration, the inputs to the composite (here a parameter *Mch*) as well as the outputs (here a parameter *F-Req*) are also defined. The content of the composite is visualised below the horizontal line.

Whether a parameter of a pattern is an input or an output is indicated by the use of a thick arrow pointing either towards (input) or from (output) a parameter. The symbols [_] indicates a list of parameters. The inputs and outputs of the composite (visualised in the declaration) are matched with the identifiers to the inputs and outputs of the patterns visualised in the content part of the composite. Thus, any instantiation of the input parameter *Mch* of *Functional Requirements* implies that the input *Mch* to *Station Interlocking Requirements* and the input *Mch* to *Level Crossing Interlocking Requirements* are similarly instantiated.

Every parameter indicated in Figure 6 may be found in the respective definition of the associated pattern. Parameters are classified into different kinds in the SaCS pattern language and Figure 6 can be annotated to indicate the classification of parameters. However, in order to simplify the visual presentation we have not included symbols indicating the classification of parameters. The annotation 2TR=Req (the sign "=" enclosed by a circle in Figure 6 symbolises an alias operator) creates an alias 2TR for the parameter named Req. The annotation LCR=Req creates an alias LCR for the parameter Req.

A *combines* relation (a line with an icon consisting of two overlapping white circles with black stroke) models the relationship between the patterns *Station Interlocking Requirements* and *Level Crossing Interlocking Requirements*. The result of the combines relation is a set that consists of the requirements resulting from instantiating 2TR and the requirements resulting from instantiating LCR, the combined set is named F-Req.

The grey wide arrow illustrated in the background in Figure 6 indicates the recommended pattern instantiation order and thus only gives guidance to the process of applying the patterns.

6. ELICIT SAFETY REQUIREMENTS

6.1 Pattern Selection

Once the functional requirements have been elicited on the basis of selected patterns from choice (A) of Figure 4, the further traversal of the pattern selection map leads to the pattern *Hazard Identification*. This pattern defines the process of identifying potential hazards associated with the use of a target system

under assessment in a given context. In choice B, patterns expressing different methods supporting hazard identification are offered. None of these are selected as we assumed in Section 5.1 that the hazards associated with the system are already identified.

The *Hazard Analysis* pattern (defined in Appendix A.4) is selected as it provides guidance on the process of deriving the potential causes of hazards. In choice (C) of Figure 4, different process solution patterns (representing methods) that support hazard analysis may be selected. The *FTA* pattern (defined in Appendix A.7) is selected as support for *Hazard Analysis* under the assumption that a top-down fault tree analysis is an acceptable and effective method for identifying potential causes of hazards.

Traversal of the pattern selection map in Figure 4 from the current point leads to the pattern *Risk Analysis* (defined in Appendix A.5). The pattern provides guidance on how to address identified hazards and to establish a notion of risk. The *SIL Classification* pattern (defined in Appendix A.8) is selected as it defines the method for classifying railway systems and their components with respect to criticality.

The pattern *Establish System Safety Requirements* (defined in Appendix A.6) describes the process of establishing safety requirements based on inputs from risk assessment. It is regarded as relevant for the case and selected as support.

6.2 Pattern Instantiation

Safety requirements are defined based on the results of risk assessment. The process assurance requirement patterns selected in Section 6.1 support the process of eliciting safety requirements and are applied sequentially according to the following order:

1. *Hazard Analysis* – used as support in the process of identifying potential causes of hazards based on inputs on applicable hazards.
2. *Risk Analysis* – used as support in the process of addressing hazards with respect to their severity and likelihood and combine this information into a notion of risk.
3. *Establish System Safety Requirements* – used as support in the process of defining requirements based on identified risks.

The instantiation of the patterns selected in Section 6.1 are outlined in the following subsections:

- Section 6.2.1 describes the instantiation of the patterns *Hazard Analysis* and *FTA*.
- Section 6.2.2 describes the instantiation of the patterns *Risk Analysis* and *SIL Classification*.
- Section 6.2.3 describes the instantiation of the pattern *Establish System Safety Requirements*.

6.2.1 Hazard Analysis and Fault Tree Analysis

Figure 7 is an excerpt of the pattern *Hazard Analysis* and describes the main process for performing hazard analysis. The process described in Figure 7 is illustrated as a UML activity diagram [20] (adapted with SaCS specific annotations for explicitly denoting input and output parameters).

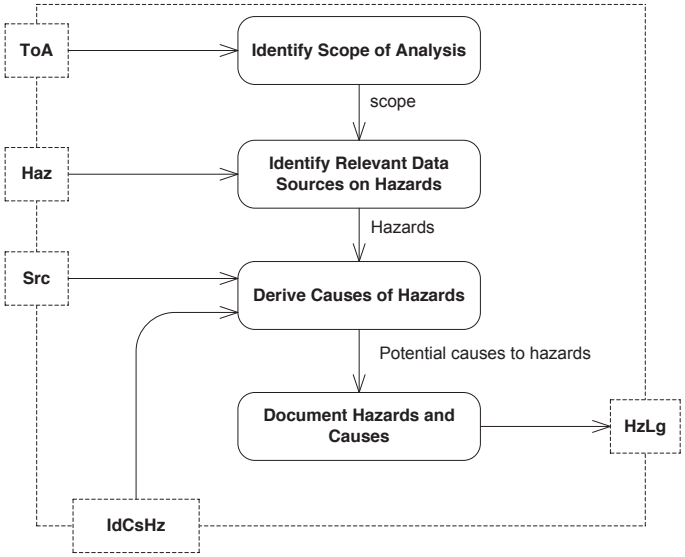


Figure 7 Excerpt of the pattern Hazard Analysis

The *Hazard Analysis* pattern is instantiated such that:

- Parameter ToA (short for Target of Assessment) is associated with the system description provided in Section 2.
- Parameter Haz (short for Hazards) is associated with the following hazards (see Section 5.1 for the assumption on relevant hazards):
  - Derailment
  - Collision train-train
  - Collision train-object
  - Level crossing accidents
- Parameter IdCsHz (short for Identified Causes of Hazards) was associated with the results of applying the *FTA* pattern according to its instantiation rule. The input to the instantiation of *FTA* is the system description provided in Section 2 and the list of hazards presented above. The result of pattern instantiation is a fault tree expressing the likely causes of hazards.
- Parameter HzLg (short for Hazard Log) was associated with the result of applying the *Hazard Analysis* pattern according to its instantiation rule.

The result of applying the *Hazard Analysis* pattern is detailed in Appendix B.2. The result of applying the *FTA* pattern used as support for *Hazard Analysis* is detailed in Appendix B.1.

6.2.2 Risk Analysis and SIL Classification

Figure 8 is an excerpt from the pattern *Risk Analysis*. The *Risk Analysis* pattern (defined in Appendix A.5) is applied once the hazards associated with our target are analysed. The *SIL Classification* pattern (defined in Appendix A.8) is applied as support for *Risk Analysis*.

The outline of the 2TLCI system as presented in Section 2 and the hazard log established in Section 6.2.1 (detailed in Appendix B.2) are used as inputs to the application of *Risk Analysis,* associated with the parameters *Toa* and *Haz,* respectively.

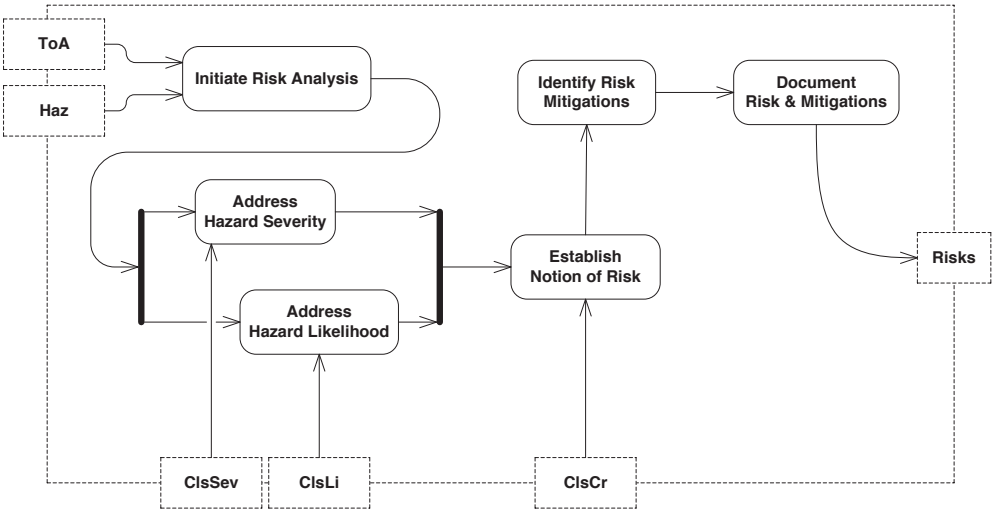


Figure 8 Excerpt from the pattern Risk Analysis

The intention of performing risk analysis is to identify the potential severity of an accident associated with a hazard, and to estimate the likelihood of events leading to the hazard.

The hazard analysis as described in Section 6.2.1 identifies potential causes of hazards. These potential causes of hazards are further used to identify the potential for the occurrence of hazards. As the hazard analysis performed in Section 6.2.1 is a qualitative analysis, we do not provide quantitative risk estimates. Instead, we deduce a System Integrity level (SIL) from tolerable hazard rates and use the results from hazard analysis as an input for identifying the relevant functions that potentially may give rise to hazards.

The results of applying the *Risk Analysis* pattern are detailed in Appendix B.3. The result indicates, based on the assessment of the main functions, that the 2TLCI system is a SIL4 system. The SIL4 classification implies that a compliance argument must be provided in order to provide assurance of the system being developed according to appropriate practices. With appropriate practices it is here meant complying with the requirements stated in [13], [14], and [15] that applies to development of SIL4 systems.

6.2.3 Establishing the Safety Requirements

Once the risk analysis associated with our target is performed (represented by the instantiation of the *Risk Analysis* pattern), the following activity is to establish the safety requirements by instantiating the *Establish System Safety Requirements* pattern (defined in Appendix A.6).

Figure 9 is an excerpt from the definition of the *Establish System Safety Requirements* pattern and presents a UML activity diagram (adapted with SaCS specific annotations for explicitly denoting input and output parameters) illustrating the main process for establishing safety requirements.

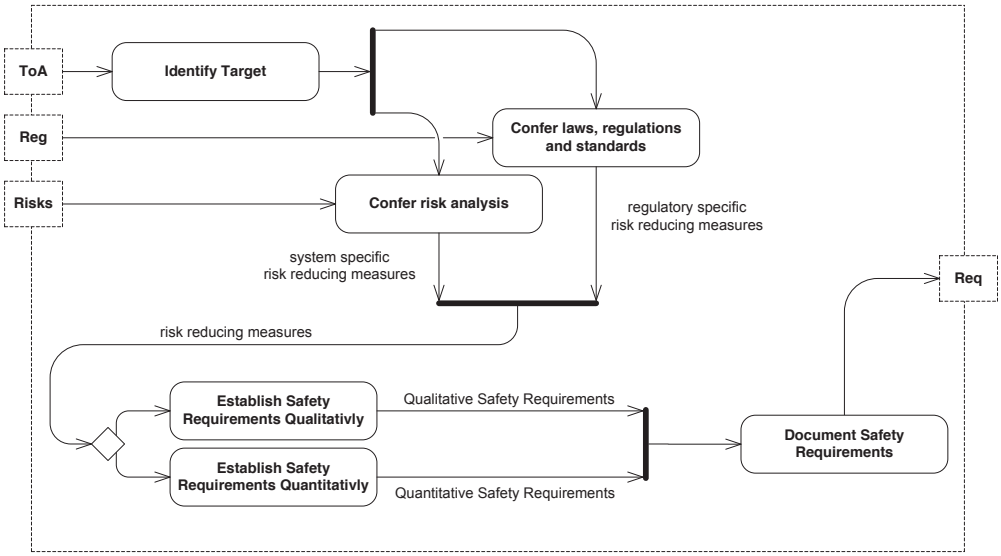


Figure 9 Excerpt from pattern Establish System Safety Requirements

The system description provided in Section 2 represents the instantiation of the input parameter ToA of *Establish System Safety Requirements* indicated in Figure 9. The hazard log referred to in Section 6.2.1 (detailed in Appendix B.2) and the extension of this hazard log represented by the result referred to in Section 6.2.2 (detailed in Appendix B.3) represent the instantiation of the input parameter *Risks* of the pattern *Establish System Safety Requirements* indicated in Figure 9.

A part of the results described in Section 6.2.2 is a classification of the system under development as a SIL4 system, requiring compliance with the standards [13], [14], and [15]. We delimit the scope in this case study and do not address compliance issues. Although compliance is important in order to acquire the necessary approval from a relevant safety authority, we rather focus on the risks identified in the previous sections. The mentioned standards would be associated with the *Reg* parameter indicated in Figure 9 if compliance issues were part of our scope.

The requirements produced as a result of instantiating the *Establish System Safety Requirements* are presented in Table 3 in Section 10.1. A detailed description of the safety requirements including traceability information is given in Appendix B.4.

6.3 Pattern Composition

Figure 10 illustrates a composite pattern named *Safety Requirements.* The composite pattern specifies a relationship between the following patterns:

- *Hazard Analysis*
- *Risk Analysis*
- *Establish System Safety Requirements*
- *FTA*
- *SIL Classification*

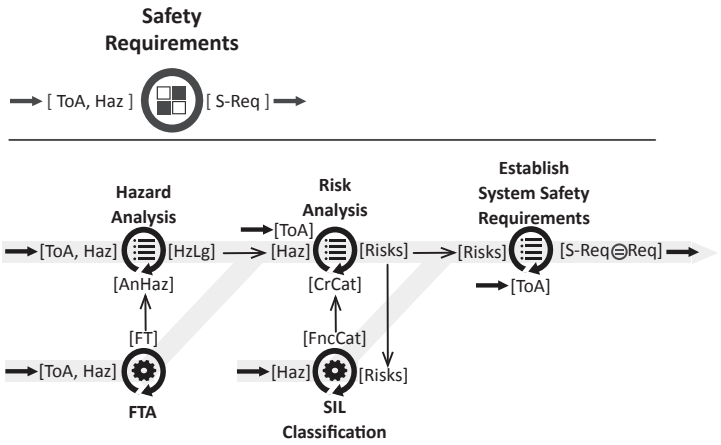


Figure 10 Safety Requirements – Composite Pattern

Figure 10 specifies the *Safety Requirements* composite. The composite takes as input ToA (short for Target of Assessment) and Haz (short for Hazards) and provides as output S-Req (short for Safety Requirements). The input parameters ToA and Haz to the composite *Safety Requirements* are used by its contained patterns. The contained patterns also define input parameters named ToA and Haz with public accessibility (indicated by the thick black arrow pointing towards a given parameter), thereby creating an implied relationship between the similarly named parameters. The output parameter S-Req of the pattern *Establish System Safety Requirements* is denoted as a public output (by the thick black arrow pointing away from the parameter S-Req) and is thus listed as an output of the composite.

The output identified as HzLg of *Hazard Analysis* is assigned to the input identified as Haz of *Risk Analysis*. The arrow connecting HzLg with Haz denotes an *assigns* relation. The assigns relation, used several times in Figure 10, denotes that the instantiation of an output parameter associated with a source pattern shall be assigned to an input parameter associated with a target pattern.

Figure 11 defines the *Requirements* composite. The composite is defined in order to make illustrations presented later simpler. The *Requirements* composite consists of the composite for establishing functional requirements (defined in Figure 6) and the composite for establishing safety requirements (defined in Figure 10).

The contained patterns in the composite in Figure 11 are related with a *combines* relation indicating that the instantiation of the F-Req parameter and the S-Req parameter shall be combined resulting in Req (an identifier for the union of F-Req and S-Req).

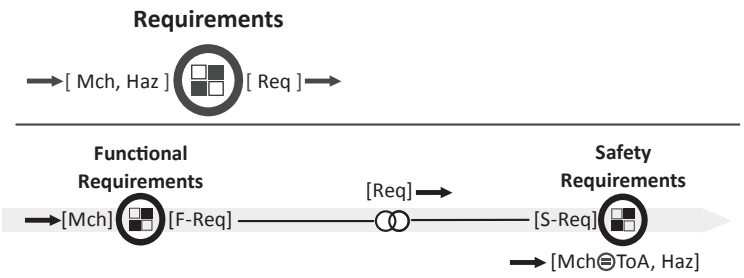


Figure 11 Requirements – Composite Pattern

7. ESTABLISH DESIGN

7.1 Pattern Selection

At choice (D) of Figure 4 there are two available design patterns that may be selected, these are named *Trusted Backup* and *Dual Modular Redundant*.

The *Trusted Backup* pattern describes a system concept where an adaptable controller may operate freely in a delimited operational state space. Safety is assured by a redundant non-adaptable controller that operates in a broader state space and in parallel with the adaptable controller. A control delegator grants control privileges to the most suitable controller at any given time on the basis of switching rules and information from safety monitoring.

The *Dual Modular Redundant* pattern describes a system concept where two similar controllers operate in parallel. The parallel operating redundant controllers and a voting unit provides mitigations against random error.

The *Dual Modular Redundant* was selected as guidance for establishing the design basis for 2TLCI system on the basis of an evaluation of the strengths and weaknesses of the different design patterns with respect to facilitating designs that satisfies the requirements.

7.2 Pattern Instantiation

Figure 12 is an excerpt from the pattern *Dual Modular Redundant* (defined in Appendix A.3). It describes a system S consisting of two parallel operating controllers, C1 and C2, a component named Cmd that provides an interface towards an overall system, and a voter unit named V.

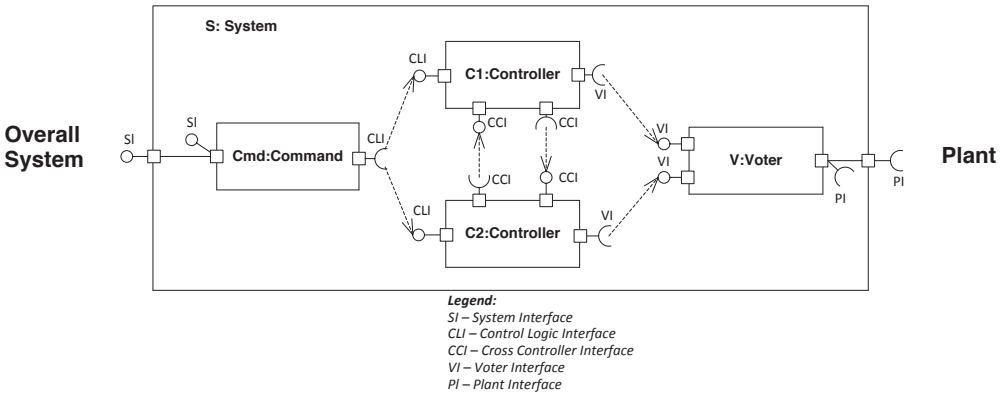


Figure 12 Excerpt from the pattern Dual Modular Redundant

Figure 13 specifies the 2TLCI system and represents the results from instantiating the *Dual Modular Redundant* pattern in the context described in Section 2.

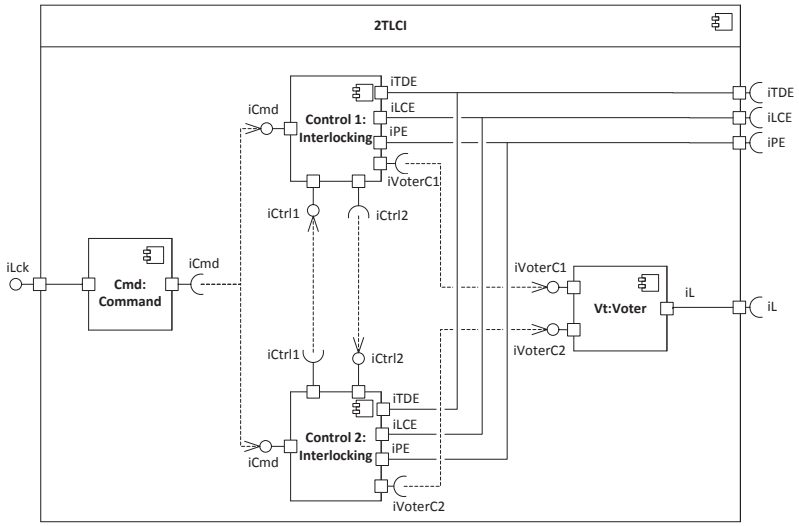


Figure 13 2TLCI – Component Diagram

The following is a mapping between parts/components of the 2TLCI design, as expressed in Figure 13, and parts required by an instantiation of the *Dual Modular Redundant*:

- 2TLCI in Figure 13 is an instantiation of S in Figure 12.
- Control 1 in Figure 13 is an instantiation of C1 in Figure 12.
- Control 2 in Figure 13 is an instantiation of C2 in Figure 12.
- Cmd in Figure 13 is an instantiation of Cmd in Figure 12.
- Vt in Figure 13 is an instantiation of V in Figure 12.
- The interfaces iTDE, iLCE, iPE and iL in Figure 13 are point-to-point interfaces between the 2TLCI system and the equipment that is controlled and represents the instantiation of PI in Figure 12.

Only a partial description of the 2TLCI system design is described here, the remaining description is provided in Section 10.2.

7.3 Pattern Composition

Figure 14 defines a composite named *Intermediate Pattern Solution*. In Figure 14, it is defined that the result of instantiating the pattern *Dual Modular Redundant* is the output S. It is also defined in the figure that the result from instantiating the pattern *Requirements* is the output Req and the S shall satisfy Req. From the specification of the pattern *Requirements* and the pattern *Dual Modular Redundant* we can find that the instantiation of Req is expected to be a specification of requirements and that the instantiation of S is expected to be a specification of a system design.

The relationship between the patterns named *Requirements* and *Dual Modular Redundant* is illustrated by a *satisfies* relation, the black filled-in circle/bullet is associated with the output that describes the requirements that should be satisfied and the check mark is associated with the system design that should satisfy the requirements.

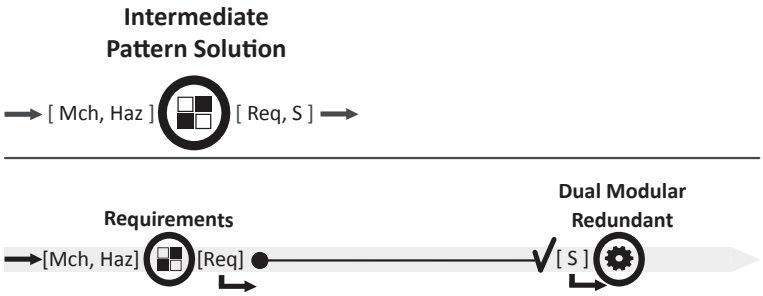


Figure 14 Intermediate Pattern Solution – Composite Pattern

8. ESTABLISH SAFETY CASE

8.1 Pattern Selection

A safety case is a logical structured argument connecting a main claim through a decomposition of this main claim down to evidences. The *Overall Safety* pattern associated with choice (E) in Figure 4 is selected as guidance for developing a safety demonstration strategy on claiming the system in question being sufficiently safe by arguing satisfactory quality management, safety management and technical safety. The definition of the pattern is provided in Appendix A.9.

The *Technical Safety* pattern associated with choice (E) in Figure 4 is selected as guidance for developing a demonstration on satisfactory technical safety. The strategy defined by the pattern is to explicitly address all risks associated with the system. The definition of the pattern is provided in Appendix A.10.

The *Safety Requirements Satisfies* pattern associated with choice (E) in Figure 4 is selected as guidance for developing a demonstration on satisfaction of safety requirements. Safety requirements are derived based on a risk-based process, and therefore this pattern is selected for detailing that risk is explicitly addressed (a demonstration strategy that accommodates the *Technical Safety* pattern) through the safety requirements. The definition of the pattern is provided in Appendix A.11.

8.2 Pattern Instantiation

Figure 15 is an excerpt of the pattern *Overall Safety* and illustrates an argument structured using GSN notation [8]. The dotted drawn rectangular box represents a SaCS adaptation of the GSN notation, each element visualised on one of the edges of the box represents either an input (i.e., the context element ToD) or an output (i.e., the goal named Case and the undeveloped goals named QualMng, SafMng, and TechSaf).

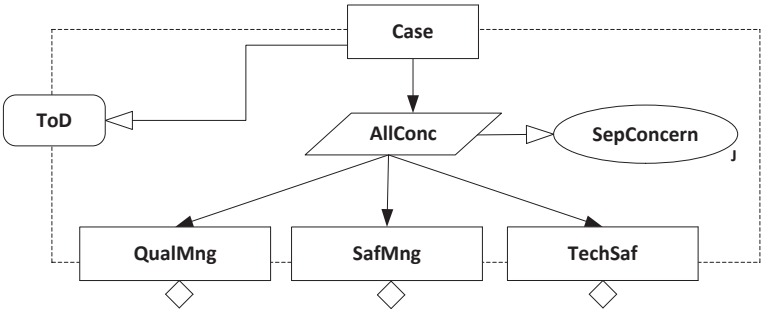


Figure 15 Excerpt of Overall Safety

The *Overall Safety* pattern describes a generic argument structure as visualised in Figure 15 for arguing that a target system (represented by the parameter ToD) is sufficiently safe for its intended purpose by arguing adequate quality management, safety management, and technical safety. The target of demonstration is the system specified in Section 7.2. Once the pattern is instantiated according to its instantiation rule, the instantiation of the Case parameter represents the root node of a logical argument visualised as a tree-like structure. The *Overall Safety* pattern only provides a partial argument structure and the undeveloped goals (denoted using a box with a diamond beneath) must be developed down to evidences in order to complete the argument.

The undeveloped goals identified as QualMng and SafMng in Figure 15 express the need to demonstrate satisfactory quality management and safety management, respectively. We do not detail the safety argument with respect to quality management and safety management under the assumption that a vendor has established routines that are normally common for many projects. Therefore, arguments and evidences demonstrating that these routines are suitable and followed will to a large degree consist of generic material. The case specific arguments related to demonstration of technical safety are more interesting and is thus detailed in this report.

Figure 16 is an excerpt from the *Technical Safety* pattern and illustrates a generic structure for arguing satisfactory technical safety. The argument structure provides choices in terms of different alternative strategies for arguing that a given target is technically safe. One of the strategies described is to argue that the system is developed according to an accepted code of practice (represented by the undeveloped goal CoP) and thus is safe. A second strategy is to argue that the system is basically identical to a reference system (e.g., similar technical platform, similar context of use, similar operational use) that is accepted as safe and consequentially the new system should be accepted as being equally safe (represented by the undeveloped goal CrRef). A third strategy is to explicitly address all risks associated with the system (represented by the undeveloped goal ERE) and show that these are reduced to an acceptable level. The *Safety Requirements Satisfied* pattern is instantiated in order to explicitly address risk (by detailing the undeveloped goal ERE of the pattern *Technical Safety*) by arguing that the safety requirements (that are established on the basis of a risk-based process) are satisfied.

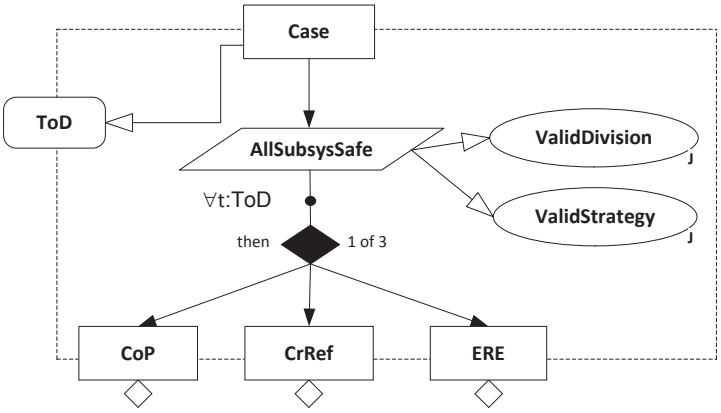


Figure 16 Excerpt of Technical Safety

Figure 17 is an excerpt from the *Safety Requirements Satisfied* pattern and illustrates a generic argument structure where a claim is made that a system is safe if every safety requirement associated with the system is satisfied. In order to apply *Safety Requirements Satisfied*, the inputs ToD that represents the target that of the safety demonstration as well as Req that represents the safety requirements associated with the target needs to be defined.

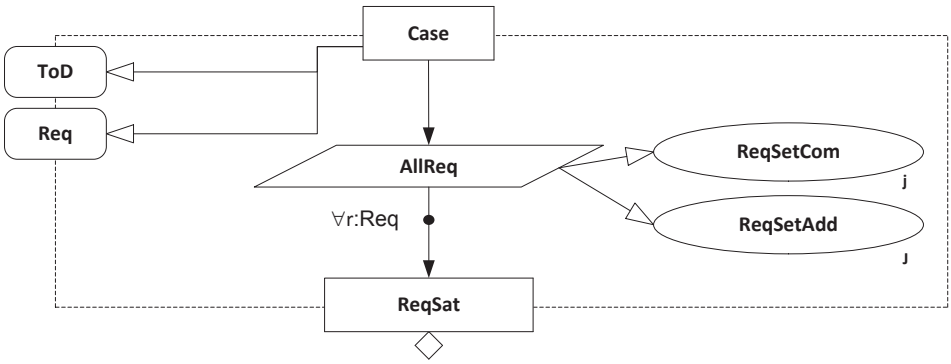


Figure 17 Excerpt of Safety Requirements Satisfied

Different strategies can be employed in order to provide a convincing argument showing that a specific safety requirement is satisfied. Depending on how a requirement is expressed (e.g., qualitatively or quantitatively), a user of the pattern language may choose between different safety case patterns for expressing the inference between a claim and its supporting evidence. The different safety case patterns with the keyword “Evidence” in Figure 4 provide guidance on bridging the gap between different kinds of claims and the evidences used as supporting, e.g., *Assessment Evidence*, *Probabilistic Evidence,* or *Deterministic Evidence*. Definition of these patterns is available in HWR-1029 [10]. Here, we simplify the argument structure and describe the inference between claims and their supporting evidences by relating evidences (represented as GSN solution elements that include references to the associated evidence information) directly with claims (represented as GSN goals) in the safety case structure provided upon pattern instantiation.

The results of instantiating the patterns *Overall Safety*, *Technical Safety* and *Safety Requirements Satisfied* provide a safety case structured according to the GSN notation. The instantiation results are provided in Section 10.3.

8.3 Pattern Composition

Figure 18 illustrates a composite pattern named *Safety Case*. The *Safety Case* composite consists of the following patterns:

- *Overall Safety*.
- *Technical Safety*.
- *Safety Requirements Satisfied*.

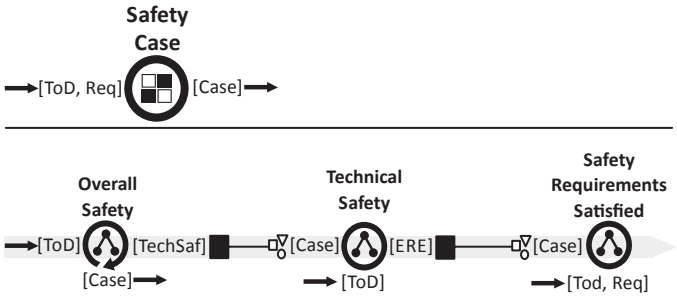


Figure 18 Safety Case – Composite Pattern

Figure 18 specifies that the *Safe Case* pattern takes as input ToD (short for Target of Demonstration) and Req (short for Requirements) and provides as output Case (short for Safety Case).

The parameter Case that is listed as output of *Safety Case* is associated with the output Case of the pattern *Overall Safety*, this is indicated by the similarly named parameters and the thick arrows that symbolise that the parameters have public accessibility. The *Technical Safety* pattern also provides an output named Case. However, the parameter Case of *Technical Safety* is defined with local accessibility (local accessibility is default) and only acts as a detailing of a part described by the *Overall Safety* pattern as indicated by a *details* relation. A details relation is illustrated as a black box connected with a solid drawn line to three small icons (a circle, a square and a triangle). The black box indicates the output parameter that is detailed and the three smaller icons indicate the output parameter that provides a detailing. The output Case of *Safety Requirements Satisfied* pattern details the output ERE of *Technical Safety*.

9. PATTERN SOLUTION FOR THE CASE

9.1 Pattern Composition

Figure 19 illustrates the composite pattern named *Pattern Solution* that captures every pattern applied in the case. *Pattern Solution* contains the following patterns:

- *Requirements* – defined in Figure 11.
- *Dual* Modular *Redundant* – defined in Appendix A.3.
- *Safety* Case – defined in Figure 18.

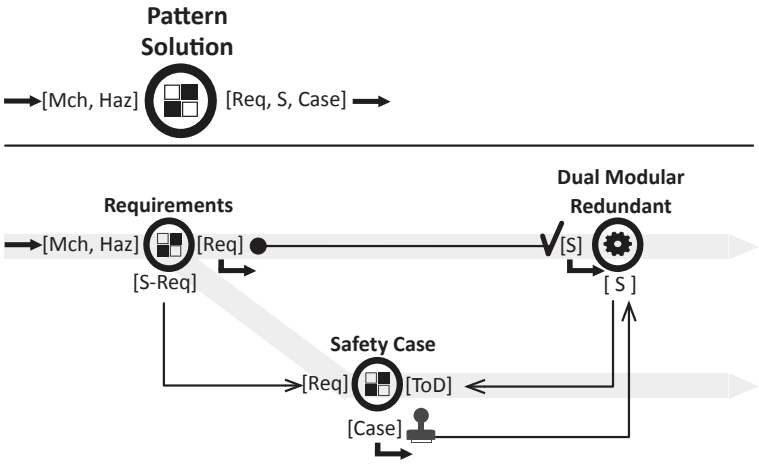


Figure 19 Pattern Solution – Composite Pattern

Figure 19 specifies that an instantiation of S associated with *Dual Modular Redundant* shall satisfy the instantiation of Req associated with the composite *Requirements* (defined in Figure 11). Further, it is specified that the parameter Case of the composite *Safety Case* demonstrates (illustrated by a demonstrates relation) that the output S of *Dual Modular Redundant* is sufficiently safe. It is also specified that the parameter S-Req (represents safety requirements, see Figure 11) of the composite *Requirements* is an input to the *Safety Case* composite (defined in Figure 18). The instantiation order of the patterns is defined by the grey arrow in the background that indicates that the *Requirement* composite shall be instantiated first, and then the *Dual Modular Redundant* and the *Safety Case* patterns may be instantiated in parallel.

9.2 Pattern Instantiation

The composite pattern described in Figure 19 may be applied on several cases (e.g., for developing an interlocking system for a station similar to the one described in this report). In order to describe a specific application of a composite pattern, annotations for denoting the instantiation of parameters can be added to the composite.

Figure 20 illustrates the different icons and the respective kinds of artefacts they represent. The icons are used for indicating the instantiation of parameters in composite SaCS patterns.

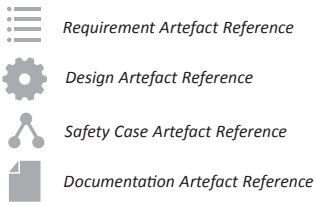


Figure 20 Artefact References in SaCS

Figure 21 is identical to Figure 19 with the addition of annotations for visualising instantiations. An instantiation is illustrated within a composite by an icon symbolising the type of artefact referred to followed by a string identifying the referred artefact. A dotted line connects an instantiation with the parameter it instantiates. The instantiations illustrated in Figure 21 refers to the artefacts that are

described in this report. When every composite pattern diagram that is applied in a case is annotated with their instantiations, the traceability between the input and output of every pattern and between patterns are provided.

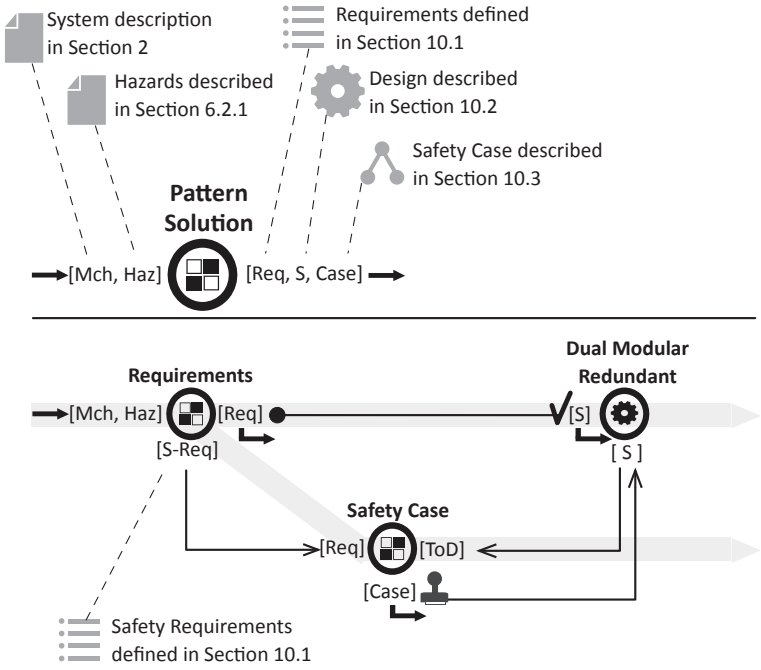


Figure 21 Pattern Solution (annotated with instantiations) – Composite Pattern

10. MAIN RESULTS

The main results of applying the SaCS method is a set of requirements elicited for the system under development (documented in Section 10.1); a design for the system (documented in Section 10.2); and a safety case arguing that the system design is sufficiently safe for application in its intended context (documented in Section 10.3). Additional results, e.g., the results from hazard identification and hazard analysis that are used as a basis for defining the safety requirements, are documented in Appendixes.

10.1 Requirements Specification

The requirements that shall be satisfied by the interlocking system for controlling the appliances of the station, as outlined in Section 2, are detailed in Table 1, Table 2 and Table 3. The following abbreviations and keywords are used in the specification of the requirements:

Abbreviations and Keywords
v1, v2: Road signals (see Figure 1)
w1, w2: Level crossing signals (see Figure 1)
dA, dB, dLN, dMO: Distant signals (see Figure 1)
mA, mM, mO, mN, mL: Main signals (see Figure 1)
p1, p2: Point (see Figure 1)
g1, g2: Level crossing gate (see Figure 1)

A, B, L, M, X, Y: Track Sections (adapted from the example system described in Appendix A.1)
AX, AY, BX, BY, L, M, N, O: Train routes (adapted from the example system described in Appendix A.1)

Table 1 Station Interlocking Requirements

ID	Requirement
2T.1	The system shall manage train movements along the following train routes: AX, AY, BX, BY, M, L, N, O.
2T.2	The system may set a train route *AX* as *locked* when: a) the train routes *AY, M, O, N, BX* and *BY* are in the state *not locked*; and b) point *p1* is *aligned*; and c) track sections *A, X and B* are in the state *vacant*.
2T.3	The system shall when a train route *AX* is set as *locked* signal: a) *ExpectProceed* for the light signal dA; and b) *ExpectStop* for the light signal dB and dMO; and c) *Proceed* for the light signal mA; and d) *Stop* for the light signal mB, mO, mM and mN.
2T.4	The system shall when track section *A* becomes occupied and train route *AX* is *locked* signal: a) *ExpectStop* for the light signal *dA*; and b) *Stop* for the light signal *mA*.
2T.5	The system may only release a locked train route *AX* if the following events occur: a) track section *A* becomes occupied, then track section *X* becomes occupied; b) track section *M* becomes vacant, then track section *A* becomes vacant.
2T.6	The system may only set a train route *AY* as locked when: a) the train routes *AX, M, O, L, BX* and *BY* are in the state *not locked*; and b) point *p1* is diverging; and c) track sections *A, Y and B* are in the state *vacant*.
2T.7	The system shall when a train route *AY* is set as locked signal: a) *ExpectProceedSlow* for the light signal dA; and b) *ExpectStop* for the light signal dB and dMO; and c) *ProceedSlow* for the light signal mA; and d) *Stop* for the light signal mB, mO and mM.
2T.8	The system shall when track section *A* becomes occupied and train route *AY* is *locked* signal: a) *ExpectStop* for the light signal *dA*; and b) *Stop* for the light signal *mA*.
2T.9	The system may only release a locked train route *AY* if the following events occur: a) track section *A* becomes occupied, then track section *Y* becomes occupied; b) track section *M* becomes vacant, then track section *A* becomes vacant.
2T.10	The system may only set a train route *BX* as *locked* when: a) the train routes *AX, AY, O, L, N* and *BY* are in the state *not locked*, and b) point *p2* is *aligned*; and c) track sections *A, X* and *B* are in the state vacant.
2T.11	The system shall when a train route *BX* is set as locked signal: a) *ExpectStop* for the light signal dA, dLN and dMO; and b) *ExpectProceed* for the light signal *dB*; and c) *Proceed* for the light signal *mB*; and d) *Stop* for the light signal *mA, mO, mN* and *mL*.
2T.12	The system shall when track section *B* becomes occupied and train route *BX* is locked signal: a) *ExpectStop* for the light signal *dB*; and b) *Stop* for the light signal *mB*.
2T.13	The system may only release a locked train route *BX* if the following events occur: a) track section *B* becomes occupied, then track section *X* becomes occupied; b) track section *L* becomes vacant, then track section *B* becomes vacant.
2T.14	The system may only set a train route *BY* as *locked* when: a) the train routes *AX, AY, M, L, N* and *BX* are in the state *not locked*, and b) point *p2* is *diverging*; and b) track sections *A, Y* and *B* are in the state *vacant*.
2T.15	The system shall when a train route *BY* is set as locked signal: a) *ExpectStop* for the light signal dA, dLN and dMO; and b) *ExpectProceedSlow* for the light signal *dB*; and c) *ProceedSlow* for the light signal mB; and d) *Stop* for the light signal *mA, mM, mN* and *mL*.

2T.16	The system shall when track section *B* becomes *occupied* and train route *BY* is *locked* signal: a) *ExpectStop* for the light signal *dB*; and b) *Stop* for the light signal *mB*.
2T.17	The system may only release a locked train route *BY* if the following events occur: a) track section *B* becomes occupied, then track section *Y* becomes occupied; b) track section *L* becomes vacant, then track section *B* becomes vacant.
2T.18	The system may only set a train route *M* as *locked* when: a) the train routes *AX, AY, O* and *BY* are in the state *not locked*; and b) point *p1* is *aligned*; and c) track sections *M and A* are in the state vacant.
2T.19	The system shall when a train route *M* is set as *locked* signal: a) *ExpectStop* for the light signal *dA* and *dB*; and b) *ExpectProceed* for the light signal *dMO*; and c) *Proceed* for the light signal *mM*; and d) *Stop* for the light signal *mA, mB* and *mO*.
2T.20	The system shall when track section *A* becomes *occupied* and train route *M* is *locked* signal: a) *ExpectStop* for the light signal *dMO*; and b) *Stop* for the light signal *mM*.
2T.21	The system may only release a locked train route *M* if the following events occur: a) track section *A* becomes occupied, then track section *M* becomes occupied; b) track section *A* becomes vacant.
2T.22	The system may only set a train route *L* as *locked* when: a) the train routes *AY, N, BX* and *BY* are in the state *not locked*; and b) point *p2* is *aligned*; and b) track sections *B and L* are in the state vacant.
2T.23	The system shall when a train route *L* is set as *locked* signal: a) *ExpectStop* for the light signal *dA* and *dB*; and b) *ExpectProceed* for the light signal *dLN*; and c) *Stop* for the light signal *mA, mB* and *mN*; and d) *Proceed* for the light signal *mL*.
2T.24	The system shall when track section *B* becomes *occupied* and train route *L* is *locked* signal: a) *ExpectStop* for the light signal *dLN*; and b) *Stop* for the light signal *mL*.
2T.25	The system may only release a locked train route *L* if the following events occur: a) track section *B* becomes occupied, then track section *L* becomes occupied; b) track section *B* becomes vacant.
2T.26	The system may only set a train route *N* as *locked* when: a) the train routes *AX, L, BX* and *BY* are in the state *not locked*; and b) point *p2* is *diverging*; and c) track sections *B* and *L* are in the state vacant.
2T.27	The system shall when a train route *N* is set as *locked* signal: a) *ExpectStop* for the light signal *dA* and *dB*; and b) *ExpectProceedSlow* for the light signal *dLN*; and c) *Stop* for the light signal *mA*, *mB* and *mL*; and d) *ProceedSlow* for the light signal *mN*.
2T.28	The system shall when track section *B* becomes *occupied* and train route *N* is *locked* signal: a) *ExpectStop* for the light signal *dLN*; and b) *Stop* for the light signal *mN*.
2T.29	The system may only release a locked train route *N* if the following events occur: a) track section *B* becomes occupied, then track section *L* becomes occupied; b) track section *B* becomes vacant.
2T.30	The system may only set a train route *O* as *locked* when: a) the train routes *AX, AY, M* and *BX* are in the state *not locked*; and b) point *p1* is *diverging*; and c) track section *A* and *X* are in the state *vacant*.
2T.31	The system shall when a train route *O* is set as *locked* signal: a) *ExpectStop* for the light signal dA and dB; and b) *ExpectProceedSlow* for the light signal dMO; and c) *Stop* for the light signal mA, mB and mM; and d) *ProceedSlow* for the light signal mO.
2T.32	The system shall when track section *A* becomes *occupied* and train route *O* is *locked* signal: a) *ExpectStop* for the light signal *dMO*; and b) *Stop* for the light signal *mO*.

2T.33	The system may only release a locked train route *O* if the following events occur: a) track section *A* becomes occupied, then track section *M* becomes occupied; b) track section *A* becomes vacant.

Table 2 Level Crossing Interlocking Requirements

ID	Requirement
LC.1	The system shall when the level crossing is in the *normal* state: a) signal *ExpectStopBeforeCrossing* for the light signal wa and wb; and b) signal *StopBeforeCrossing* for the light signal w1 and w2; and c) signal *Go* for the light signal v1 and v2; and d) set the gates g1 and g2 in the state *Elevated*.
LC.2	The system shall when the level crossing is in the *Closed* state: a) signal *ExpectPassageOfCrossingAllowed* for the light signal wa and wb; and b) signal *PassageOfCrossingAllowed* for the light signal w1 and w2; and c) signal *Stop* for the light signals v1 and v2; and d) set the gates g1 and g2 in the state *Closed*.
LC.3	The system shall assure that the level crossing transition to the *closed* state when: a) the coupling point cA is set to true; or b) the coupling poin cB is set to *true*.
LC.4	The system shall assure that the level crossing transition to normal state (and thus is released from *closed* state) when: a) the train that induced a closed state (by passing the coupling point cA or cB) has passed the level crossing (detected by passing the decouple point dA or dB).

Table 3 Safety Requirements

ID	Requirement
SR.1	A proceed aspect for a main signal belonging to a train route may only be given once the conditions for locking the train route are satisfied
SR.2	A train route may not be locked unless all points belonging to the train route are positioned correctly
SR.3	A train route may not be locked if a track section in the train route belongs to another train route
SR.4	A train route may not be locked unless points belonging to the train route are controlled in correct position
SR.5	A points position may only be altered if conditions for altering its position are satisfied
SR.6	A points position may not be altered if it belongs to a locked train route
SR.7	A main signal for a train route that has a point positioned to a diverging track shall not signal less restrictive than proceed slow
SR.8	A distant signal shall show at least as restrictive aspect as its belonging main signal
SR.9	A train route may not be locked unless all track sections in the train route are vacant
SR.10	A train route may not be locked unless its safety zone is vacant
SR.11	A train route may not be locked if it has a part that is shared with another locked train route
SR.12	An incoming train route that is locked may not be released before the train has arrived the station
SR13	A train route may not be locked if there is another train route locked over its safety zone
SR.14	An outgoing train route that is locked may not be released before the train has left the station
SR.15	An outgoing train route may not be locked unless the line is set in the direction from the train routes starting point
SR.16	An outgoing train route may not be locked unless the line is vacant
SR.17	The line may only be set in the direction of an arrival station if the main signals (exit) of the arrival station shows ‘Stop’
SR.18	In order to signal proceed aspect for a main signal that has a dependency to a level crossing interlocking, the road traffic must be blocked
SR.19	In order to signal proceed aspect for a distant signal that has a dependency to a level crossing interlocking, the road traffic must be blocked
SR.20	No work is allowed on a track section unless it is blocked

SR.21	A train route may not be locked if a track section in the train route is blocked

10.2 Design Specification

In the following sections, the 2TLCI system design will be outlined. Figure 2 in Section 2 outlines the context for which the 2TLCI system is intended to replace the component identified as Interlocking System. Section 10.2.1 describes the main concepts by a UML class diagram. Section 10.2.2 describes the main parts of the 2TLCI system illustrated by UML component diagrams. Section 10.2.3 describes the behaviour of the system by UML state machine diagrams.

10.2.1 Main Concepts

Figure 22 illustrates the main concepts in our system with a UML class diagram presenting the main classes and the associations between classes that are modelled. A central concept is the notion of a train route, our system is defined in order to control train movements according to eight train routes. A train route is composed of several track sections. A given track section may be a part of several train routes. A track section may have a point or crossing gates. The train movements along a train route and the crossing of a track section by road users are controlled by light signals of different kinds (see configuration of light signals along the tracks in Figure 1). The enumerations defined in Figure 22 detail the different states for the different appliances.

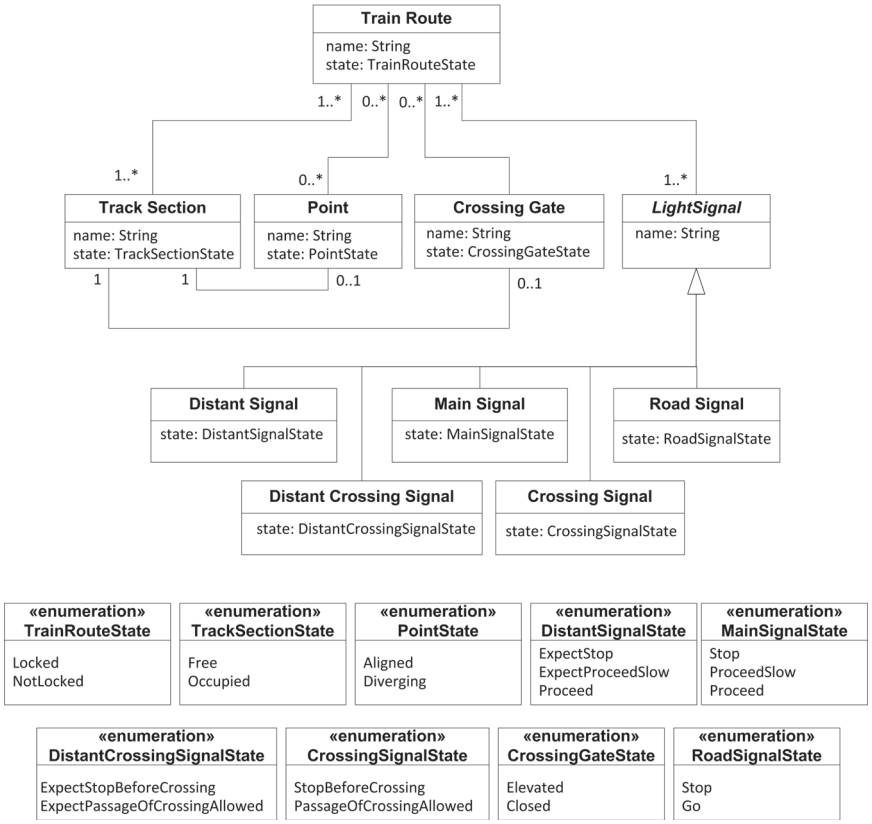


Figure 22 Main Classes

10.2.2 Overview of the Interlocking System

The 2TLCI system is illustrated in Figure 23.

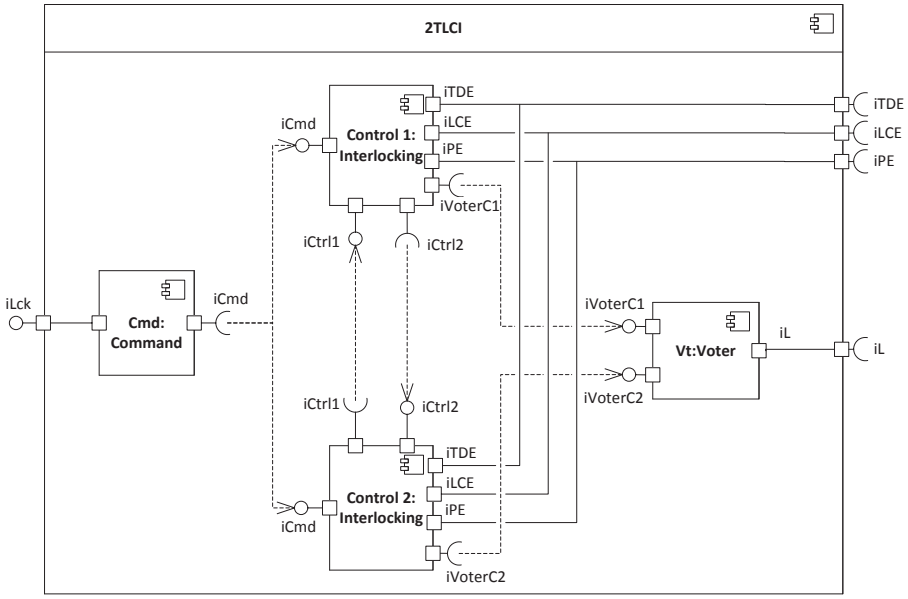


Figure 23 Illustration of the 2TLCI system

The 2TLCI system illustrated in Figure 23 is intended to operate in the following manner:

- Cmd is responsible for the communication between an overall system and 2TLCI and shall be able to:
  - Receive commands from an overall system, e.g., Centralised Train Control (CTC) or an operator on a local control unit, through the interface iLck.
  - Read the states of Control 1 and Control 2 through the interface iCmd and communicate this to the overall system on interface iLck.
- Control 1 and Control 2 are two components with similar functionality that operates in parallel and are responsible for the control of appliances (e.g, lights, points, and level crossing equipment) in accordance with commands from a local operator or from an operator associated with CTC based on interlocking rules and inputs on train movements (train detection equipment).
- Vt is responsible for assuring that in the event of disagreement between the commands given by the dual controllers, appliances are controlled in a safe manner (e.g., signal stop on all light signals).

10.2.3 Main Functionality

The UML state machine diagrams in this section describe the main functionality of the 2TLCI system, the interlocking component behaviour. The behaviour of the command component and the voter component are not detailed.

Figure 24 illustrates the interlocking state machine. The state machine named *Interlocking* represents the functionality provided by Control 1 and Control 2 in Figure 23. The composite states of the state machine are described such that:

- Lock Train Route is detailed in Figure 25.

- Monitor is not detailed. It is expected that Monitor is detailed once the conceptual design has matured to describe the mechanisms for performing self checks and performing cross monitoring of the other interlocking component (e.g., Control 1 monitors Control 2 and vice versa).

Every signal is identified with a prefix SIG. A monitoring signal is identified with a prefix SIG_Mon whereas a request signal is identified with a prefix SIG_Req. A signal for locking a train route has the prefix SIG_Req_LckTR. A signal representing a request for locking a specific train route, e.g., the train route AX, is identified as SIG_Req_LckTR_AX. The kind of monitoring signal or request signal is indicated by their name in the illustrations that follows, we do not give further details on the signals.

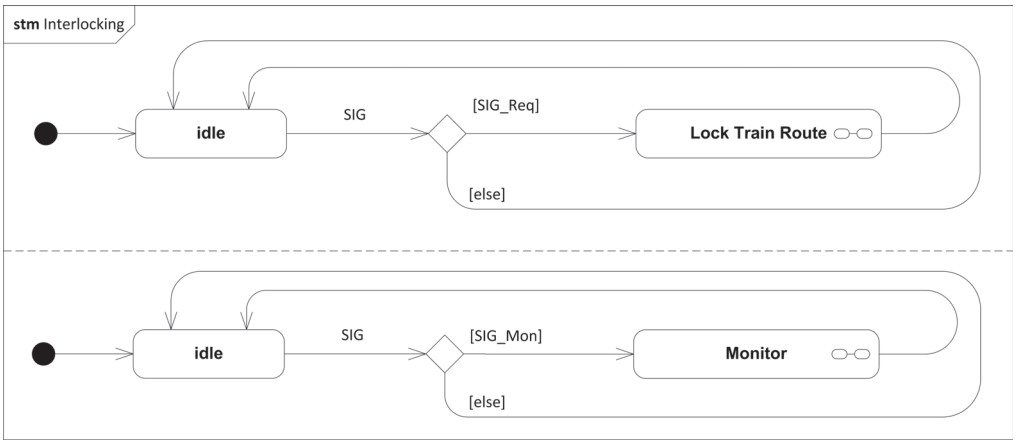


Figure 24 Interlocking - State Machine

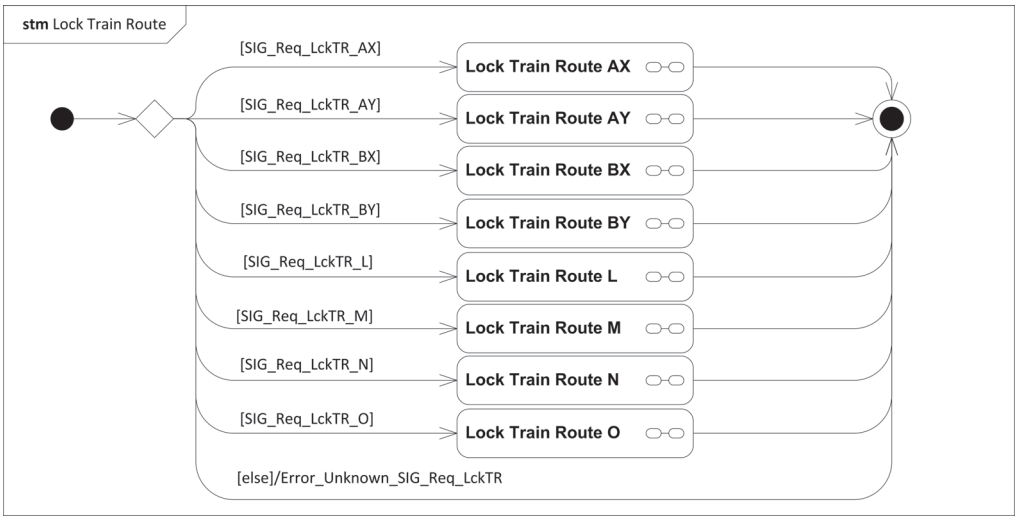


Figure 25 Lock Train Route - State Machine

Figure 25 describes the state machine named *Lock Train Route,* which was indicated as a composite state in Figure 24. Figure 26 defines *Lock Train Route AX. Lock Train Route AX* is indicated in Figure 25 as a composite state The remaining composite states in Figure 25 are very similar to *Lock Train Route AX*, therefore, we only provide details with respect to the functionality for locking the train route AX to avoid unnecessary repetition.

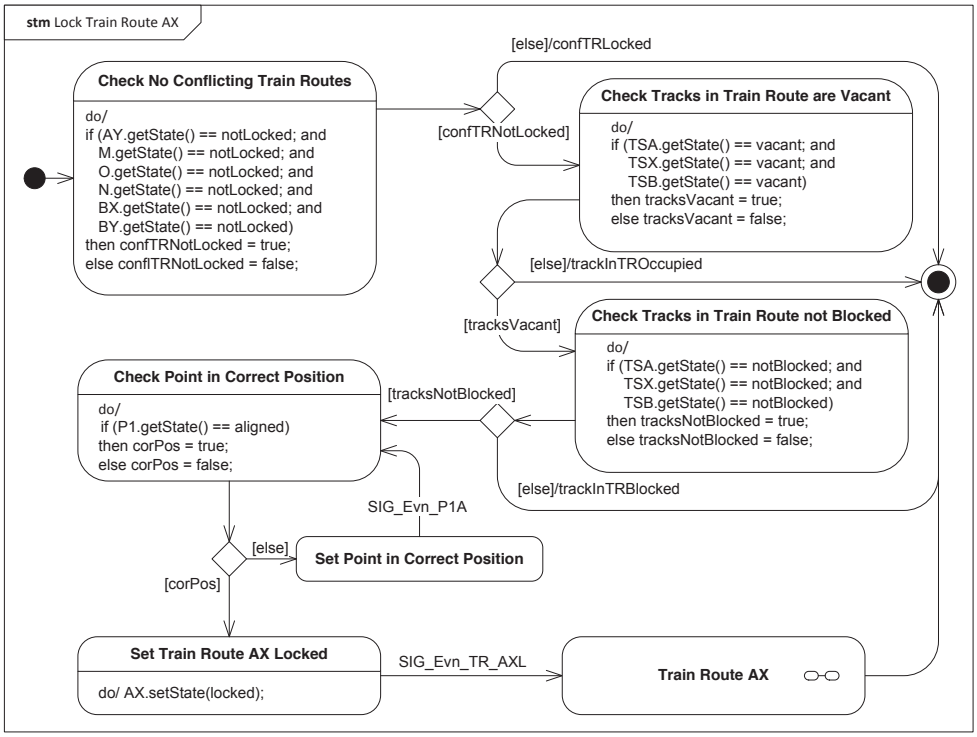


Figure 26 Lock Train Route AX - State Machine

The state machine defined in Figure 26 uses a composite state named *Train Route AX* that is defined in Figure 27.

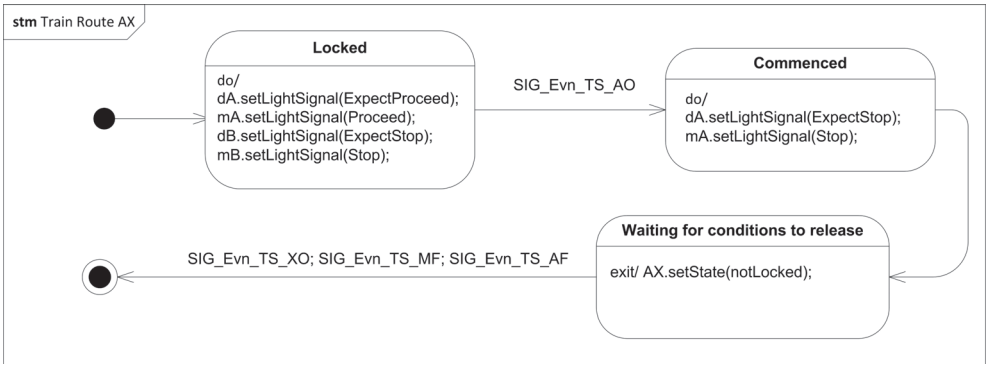


Figure 27 Train Route AX - State Machine

In the specification of the interlocking system as presented above, we have assumed that no train route is simultaneously handled such that simultaneous incoming or outgoing traffic is not supported. The design specification presented in this report only captures the initial design and the design is underspecified in the sense that only the main features are addressed. Issues that typically should be

further detailed are, e.g., tolerance to handle failure events and robustness with respect to unexpected changes to the states of appliances.

10.3 Safety Case Specification

Figure 28 and Figure 29 outline the main argument, annotated in GSN notation [8], of the safety case that is used to argue that the 2TLCI design is suitable for use in the context of controlling the appliances of the station.

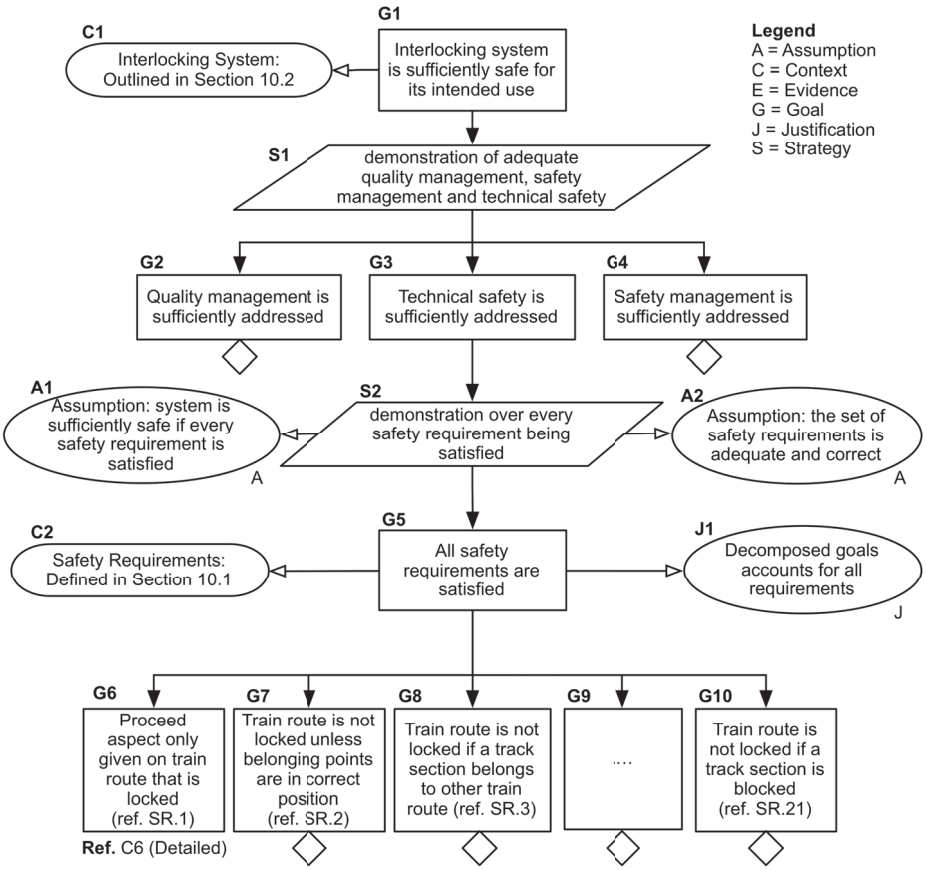


Figure 28 Safety Case (Part 1)

The goals in Figure 28 that are not developed are denoted as undeveloped goals. An undeveloped goal is annotated with a diamond. The undeveloped goals are assumed to be provided in a later stage of the development of the 2TLCI system. Some of the undeveloped goals may be supported by generic safety demonstrations common to many development projects within an organisation, e.g., demonstration on adequate quality management and safety management is assumed to follow company procedures. Thus, we do not explicitly address the undeveloped goals in this report.

The focus of the safety case in this report is on arguing the satisfactory technical safety of the design concept developed for the interlocking system identified as the 2TLCI system. Satisfactory technical safety is here primarily demonstrated by addressing the safety requirements. We delimit the detailing in this report to the demonstration for requirement SR.1, see goal G6 in Figure 28.

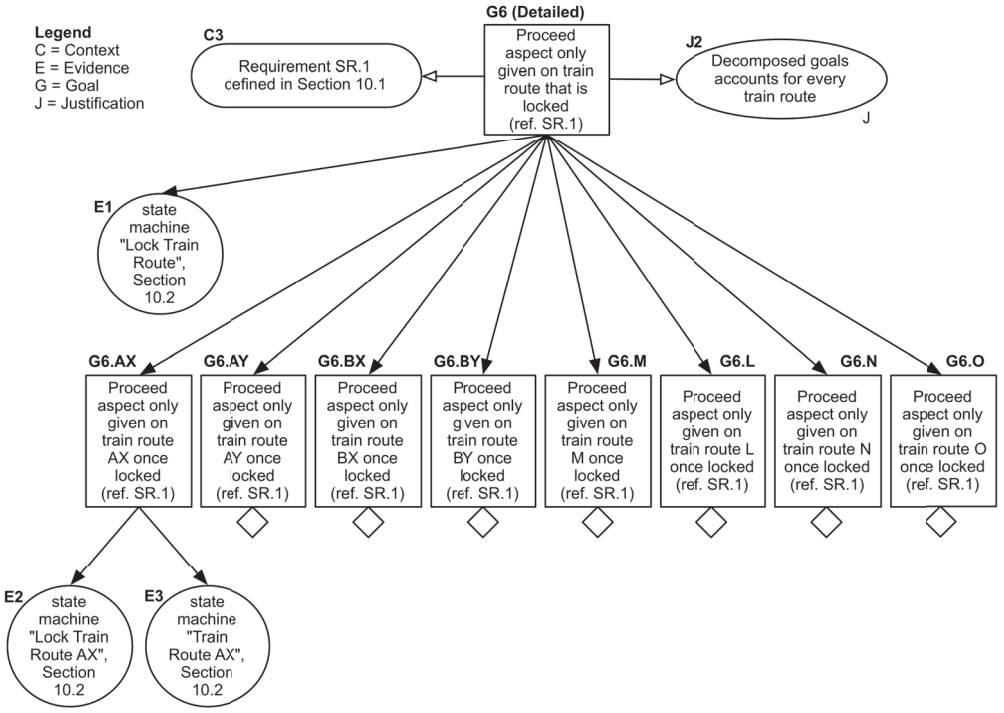


Figure 29 Safety Case (Part 2)

Figure 29 represents the detailing of the goal G6 and addresses the safety requirement SR.1 that is defined in Section 10.1. The claim put forward in the goal G6 is supported by a reference to the *Lock Train Route* state machine as evidence. There are defined sub-goals such that the fulfilment of the claim expressed in C6 is addressed separately for each train route. The design presented in Section 10.2 only details the behaviour for locking train route AX. Thus, we only develop the goal G6.AX for which this report provides documentation that is referenced as supporting evidence.

11. DISCUSSION

The successfulness of applying the SaCS method in the case is here discussed with respect to the success criteria outlined in Section 4. The following discussion represents a self-evaluation as the method has been applied in the case by the authors. However, each step is documented in a manner we believe is sufficient to allow others to perform an independent evaluation of the feasibility of the SaCS approach. The authors have also tested the SaCS method in a case on the development of a reactor control system concept [10]. This discussion is influenced by the discussion in [10].

We claim that the conceptual safety design specified in Section 10 fulfils the success criteria defined in Section 4. Success criteria states that:

***P:** Application of the SaCS method on the interlocking case described in Section 2 results in a conceptual safety design that characterises the interlocking case and is easily instantiated from a composite SaCS pattern. Furthermore, the conceptual safety design:*

a) Is in accordance with safety objectives – the conceptual safety design is defined in agreement with safety objectives.

b) Is at a sufficient level of detail – the conceptual safety design is defined by parts that are appropriate, necessary, sufficiently detailed for an early stage specification, and may be easily understood.

c) Is easy to use – the conceptual safety design may be easily extended, detailed, or refined.

Regarding the application of the SaCS method resulting in a conceptual safety design that characterises the interlocking case and being easily instantiated from a composite pattern, we argue this is satisfied as follows.

Firstly, according to the definition in Section 4, a conceptual safety design is a triplet consisting of an early stage specification of system requirements, system design and safety case. The result of applying the SaCS method in the case, which is the conceptual safety design, is presented in Section 10. Each part of the triplet that constitutes the conceptual safety design in Section 10 is presented in a separate sub-section, Section 10.1, Section 10.2, and Section 10.3, respectively. Thus, the conceptual safety design consists of the specification parts required by the definition. The sequence of steps in Section 5 to Section 9 represents the application of the process of the SaCS method defined in Section 3.1.

Secondly, the conceptual safety design characterises the case defined in Section 2 by specifying the requirements, a technical solution, and a safety case for an interlocking system as indicated in Figure 2 that can control the train movements at a station with the track configuration as depicted in Figure 1. The names for the appliances as indicated in Figure 1 are used consistently in the different specification parts presented in Section 10, e.g., the names dA and dB identifying different distant signals. We find it fair to argue that the conceptual safety design presents a solution to the control problem presented in Section 2 and in this manner characterises the case.

Thirdly, according to the definition in Section 4, a conceptual safety design instantiates a composite SaCS pattern if each element of the triplet can be instantiated from a composite according to the instantiation rule of the individual patterns within the composite and according to the rules for composition. A pattern within a composite is either a basic pattern or a composite pattern. The rules for composition are defined by the syntax and semantics of the SaCS pattern language as presented in [11]. Each basic pattern defines its instantiation rule. The basic patterns applied in the case are defined in Appendix A. The composite pattern for the case is specified in Figure 21. Figure 21 defines a composition of the patterns *Dual Modular Redundant* (defined in Appendix A.3), *Requirements* (defined in Figure 11), and *Safety Case* (defined in Figure 18). Figure 21 also defines how each of these patterns was instantiated in terms of identifying the artefacts that acted as inputs to pattern instantiation as well as the outputs produced upon pattern instantiation. A pattern needs to be interpreted by a user in its application context in order to be useful, which is a process that may be difficult to document. Furthermore, the outcome of this process is also influenced by the knowledge and skills of the user. However, it is the pattern definitions, the descriptions of the application of patterns in the case, and the descriptions of the results of pattern instantiation that represents the documentation that allows others to perform an independent evaluation of the feasibility of the SaCS method. While Section 5 to Section 9 documents how the SaCS method has been applied in the case and Section 10 documents the end result, Appendix B documents the intermediate development artefacts that are not part of the conceptual safety design but influenced its definition. In the following, we argue that each of the specifications described in Section 10.1, Section 10.2, and Section 10.3 are easily instantiated from the composite in Figure 21.

The specification of the system requirements in Section 10.1 is according to Figure 21 the result of applying the *Requirements* composite. Figure 21 specifies the use of the composite *Requirements*, and the pattern itself is defined in Figure 11. Figure 11 specifies that the requirements representing the outcome of applying *Functional Requirements* shall be combined with the requirements representing the

outcome of applying *Safety requirements* and together represent the outcome of applying *Requirements*. *Functional Requirements* and *Safety Requirements* are composites, which are defined in Figure 6 and Figure 10, respectively. Both patterns also define a combination of basic patterns defined in Appendix A.

The result of applying *Functional Requirements* consists of the combined result of two patterns, named *Station Interlocking Requirements* and *Level Crossing Interlocking Requirements*. These two patterns are defined in Appendix A.1 and Appendix A.2. The instantiation of *Station Interlocking Requirements* and *Level Crossing Interlocking Requirements* are described in Section 5.2.

The result of applying *Safety Requirements* is according to the definition presented in Figure 10 a result of applying a combination of five basic patterns. The detailed description of the instantiation of the patterns contained in *Safety Requirements* is given in Section 6.2. In short, the safety requirements are the outcome of instantiating the pattern *Establish System Safety Requirements* contained within *Safety Requirements*. The other patterns within *Safety Requirements* are used as support for deriving the necessary information required for eliciting the safety requirements, such as identifying the hazards associated with the system in question and the potential causes of these hazards. While the safety requirements derived from pattern instantiation are documented in Section 10.1, the intermediate development artefacts from instantiating the patterns within *Safety Requirements* are documented in Appendix B.

The specification of the system design in Section 10.2 is according to Figure 21 the result of instantiating the pattern *Dual Modular Redundant.* The *Dual Modular Redundant* pattern is defined in Appendix A.3. The instantiation of the *Dual Modular Redundant* pattern is described in Section 7, including a detailed explanation of the traceability between the abstract design presented in the pattern and the specification of the interlocking system.

The specification of the safety case in Section 10.3 is according to Figure 21 the result of instantiating the pattern *Safety Case*. The safety case specifies the main principles for arguing that the proposed interlocking system is sufficiently safe for its intended purpose. In Figure 18, the *Safety Case* composite defines that its output is produced by instantiating the pattern named *Overall Safety*. It is also specified in Figure 18 that the pattern *Technical Safety* is used as support in detailing one of the specification parts produced when instantiating *Overall Safety*. Furthermore, *Safety Requirements Satisfied* is used as support for detailing one of the specification parts produced when instantiating *Technical Safety*. Section 8.2 describes how each pattern is applied as support for specifying the safety case. The definitions of the patterns *Overall Safety*, *Technical Safety*, and *Safety Requirements Satisfied* are provided in Appendix A.

Regarding a), we argue that the conceptual safety design is in accordance with safety objectives as follows.

The conceptual safety design represented by the specification in Section 10.1, Section 10.2, and Section 10.3 is defined in accordance with safety objectives in the following manner:

- Section 10.1 defines the safety objectives in the case. The safety objectives are represented by the safety requirements described in Table 3. The safety requirements are established by the application of the composite pattern named *Safety Requirements* as described earlier. *Safety Requirements* describe the systematic application of several patterns as support for establishing the risk associated with the system under development as well how these risks are used to define safety requirements. Table 33 provides the traceability between the safety requirements and some of the important results from risk analysis that served as input to requirements specification. The system under development is outlined in Section 2 along with a description of its intended context of operation. Section 6 demonstrates how the objectives of the case defined

Section 2 is addressed by applying a series of patterns that results in the definition of safety requirements. Intermediate development artefacts such as the list of hazards associated with the operation of the system, or the assessment of the potential causes of hazards, are documented in Appendix B.

- Section 10.2 contains a specification of a system design that accommodates the safety requirements presented in Table 3. Although the system design is defined in accordance with the safety requirements, it is mainly the safety case (defined in Section 10.3) that expresses to which extent the safety requirements are satisfied. We have chosen to delimit the safety case specification by only detailing the argument structure for requirement SR.1. However, the undeveloped goals within the specification give a clear indication of how the safety case should be developed.
- Section 10.3 outlines a safety demonstration strategy where the interlocking system under development is claimed to be sufficiently safe for its intended purpose by a demonstration of adequate quality management, safety management and technical safety. However, it is only the claim associated with demonstration of technical safety that is developed, supported by a strategy of arguing that the safety requirements are met. The safety case in Section 10.3 demonstrates the main lines for arguing that the system design presented in Section 10.2 is sufficiently safe for its intended purpose. The intended strategy is to show as far as possible that the system design accommodates the safety requirements by referencing the relevant part of the system design specification as evidence.

The three specifications serve different purposes. The system requirements specification captures the objectives of the system under development. The system design specification represents an early stage technical description of a system that fulfils objectives. The safety case expresses in what way the safety objectives can be argued fulfilled.

Regarding b), we argue that the conceptual safety design is at a sufficient level of detail as follows.

According to the definition in Section 4, a conceptual safety design is an early stage specification. By early stage specification, it is here meant a description that shows the main features of a solution for a given problem. Although the specification is not expected to be complete, a conceptual safety design is of little use if it does not clearly convey a potential solution for the problem in question in a manner that can be easily understood. We find it reasonable to argue that the conceptual safety design is at a sufficient level of detail if it is expressed in a manner that clearly shows how the objectives of the case is intended solved. Furthermore, the conceptual safety design should be expressed in a format that can be easily understood by its intended users.

The objective in the case as it is expressed in Section 2 is to derive an interlocking system that can safely control the train movements at train station with two tracks and level crossing. The starting point in the application of the SaCS method is the development objective stated in Section 2, and the outcome is the conceptual safety design presented in Section 10. Through the process of applying the SaCS method, the overall development objective is refined into a requirements specification and further into a system design. Each step in the application of the SaCS method is detailed in Section 5 to Section 9. Furthermore, Appendix A details every basic SaCS patterns applied in this report. In this sense, the necessary information required in order to trace the end result step-by-step through the application of the SaCS method back to the initial development objective is available.

The conceptual design is a triplet where each part is described in the following formats:

- The requirements specification is defined textually in natural language. Requirements specifications are commonly expressed in natural language.

- The design specification is defined by a combination of UML diagrams and textual descriptions. UML is a commonly used modelling language for describing systems in terms of their structure and behaviour.
- The safety case specification is defined with GSN. The GSN notation facilitates structuring a safety argument with simple graphical constructs that requires little effort to be understood.

Although textual requirements, GSN, and UML models in general should be understandable for the intended users of SaCS, this may not be the case for the specifications provided here. We have not tested the specifications on potential users of SaCS in order to investigate if the specifications are easy to understand, but rather provide the specifications themselves.

Regarding c), we argue that the conceptual safety design is easy to use as follows.

The conceptual safety design described in Section 10 represents a specification described at a high level, it may be difficult at this stage to evaluate if the design is easy to use in the sense of being easy to implement. However, we find the specification informative enough for experts to judge whether the described concept is feasible and detailed enough to use as a starting point for further refinement. The system design in Section 10.2 is a refinement of the requirements in Section 10.1, and the requirements are derived on the basis of systematic analysis of the development objectives and the context described in Section 2. The conceptual design may be refined by repeating the SaCS process and iterate over pattern selection, pattern instantiation, and pattern composition until the design is in an implementable state.

The requirements specification in Section 10.1 consists of requirements that are uniquely identified. The requirement can be further detailed or rephrased; the requirement will then exist in another version. Requirements may be added to the specification easily by adding a unique requirement identifier along with the associated requirement text.

With respect to the design part of the conceptual design, the component diagram in Figure 23 illustrates the main structure of the interlocking system under development. Figure 23 is a UML composite diagram that clearly identifies the internal components of the interlocking system, the interfaces between internal components, as well the interfaces with the external systems. The interlocking system modelled in Figure 23 interacts with the other parts of the overall system via interfaces as illustrated in Figure 2. The main functionality of the interlocking system is described in Section 10.2.3, exemplified by the description of the functionality for controlling train movements along a train route identified as AX. Figure 24, Figure 25, and Figure 26 presents the behaviour of the most important component, e.g., the controllers that are responsible for providing the interlocking functionality. The functionality for controlling the train movements along the train routes other than AX can be described by specifications similar to the specification related to the train route AX. In order to avoid unnecessary repetition, we have not included the complete specification. However, we believe that the main features of the design should be easy to understand from those specifications provided, e.g., the specification related to AX.

The safety case specification is described by GSN. GSN offers constructs for modularising a safety argument such that existing argument structures can be easily reused. The safety case in Section 10.3 is limited to only detail how the requirement SR.1 has been met. The undeveloped goals in the safety case structure represent an under specification. Although only the strategy for arguing the satisfaction of SR.1 is detailed in the safety case, the different undeveloped goals in the argument structure indicates how an overall strategy for safety demonstration is approached. The undeveloped goals also identify those parts of the safety demonstration that needs to be further developed. The scope of the safety case can be extended easily by, e.g., adding nodes to the existing tree structure, adding sub trees to the structure, or make the existing safety case a sub tree in a larger argument structure. The safety case is a living

document, which should be established at the same time as the design is established, and should be maintained throughout the lifecycle of the system. As far as evidences for the suitability of a system for its intended use may be provided in the early stages, the safety case at least provides traceability between the required functions of the system and features of the design as far as these are specified. Any refinement of the design into a more detailed design or into an implementation requires refinement of the safety case in order to demonstrate that safety requirements are still satisfied.

12. RELATED WORK

To the best of our knowledge, there exists no pattern-based method that combines diverse kinds of patterns into compositions like SaCS. However, the concept of systematically applying a set of patterns is not new and is inspired by the work of Alexander et al. [1] on patterns supporting the architecture of buildings. Important sources of inspiration for our work that are applicable to development of software-based systems are pattern approaches for: requirements elicitation [12][17], software design [3][7][9], and safety demonstration [2][8]. Two challenges associated with the referenced pattern approaches are: the integration of patterns is detailed informally; each pattern approach only reflects on one important perspective when developing critical systems (e.g., software design [7]). SaCS offers the ability to combine different kinds of patterns and to detail the combination.

The notation for detailing the application of patterns and the focus on establishing a complementing set of development artefacts offered by SaCS is motivated by the need within the safety domain for providing safety assurance. International safety standards, e.g., [13][14][15], express requirements related to providing safety assurance. While safety standards to a large degree describe the required practices during development in order to produce systems that may be accepted within a particular domain, SaCS provides guidance on applying accepted safety engineering practices. The European railway regulation [5] and the associated guideline [6] on common safety methods for risk evaluation and assessment has influenced the work on defining SaCS patterns. While the guideline [6] addresses the railway domain, SaCS can be applied in different domains by selecting the patterns that are applicable to a given domain.

13. CONCLUSIONS

In this report, we have exemplified the application of the SaCS method on the development of a conceptual safety design of a railway interlocking system.

We have tried to demonstrate how the conceptual safety design is instantiated from several basic SaCS patterns within a specific composite SaCS pattern. Each constituent basic pattern of the composite pattern defined for the case has clearly defined inputs and outputs. Each basic pattern provides guidance on how it is intended to be instantiated through defined instantiation rules. The composite pattern details: the identifier and type of every pattern applied in order to derive the conceptual design; the pattern instantiation order; and the flow of data through the network of patterns giving traceability between development artefacts.

The conceptual safety design is built systematically in manageable steps as exemplified in Section 5 to 9 by a clearly defined process for pattern selection, instantiation and merging of results. The conceptual safety design is a triplet consisting of the system requirements in Section 10.1, the system design in Section 10.2, and the safety case in Section 10.3. The conceptual safety design is expressed in a manner we think is easy to understand for its intended users.

14. REFERENCES

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

[2] R. Alexander, T. Kelly, and J. McDermid, “Safety Cases for Advanced Control Software: Safety Case Patterns”, Technical Report FA8655-07-1-3025, Department of Computer Science, University of York, 2008.

[3] F. Buschmann, K. Henney, and D. Schmidt, *Pattern-oriented Software Architecture: On Patterns and Pattern Languages, Vol. 5.*, Wiley, 2007.

[4] W. D. Ellis, *A Source Book of Gestalt Psychology*, The Gestalt Journal Press, 1997.

[5] European Union, *Commission Regulation (EC) No 352/2009 on the Adoption of Common Safety Methods on Risk Evaluation and Assessment*. Official Journal of the European Union, 2009.

[6] ERA, Guide for the Application of the Commission Regulation on the Adoption of a Common Safety Method on Risk Evaluation and Assessment, European Railway Agency, 2009.

[7] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addision-Wesley, 1995.

[8] GSN Working Group, *GSN Community Standard*, Version 1.0, GSN Working Group, York, England, 2011.

[9] R.S. Hanmer, *Patterns for Fault Tolerant Systems*, Wiley, 2007.

[10] A. A. Hauge and K. Stølen, A Pattern-based Method for Safe Control Systems – Exemplified Within Nuclear Power Production, HWR-1029 rev 2, OECD Halden Reactor Project, 2014.

[11] A. A. Hauge and K. Stølen, Syntax & Semantics of the SaCS Pattern Language, HWR-1052, OECD Halden Reactor Project, 2013.

[12] E. Hull, K. Jackson, and J. Dick, *Requirements Engineering*, Springer, 2004.

[13] IEC62278, Railway Applications – Specification and Demonstration of Reliability, Availability, Maintainability and Safety (RAMS), International Electrotechnical Commission, 2002.

[14] IEC62279, Railway Applications – Communication, Signalling and Processing Systems – Software for Railway Control and Protection Systems, Edition 1.0, International Electrotechnical Commission, 2002.

[15] IEC62425, Railway Applications – Communication, Signalling and Processing Systems – Safety Related Electronic Systems for Signalling, Edition 1.0, International Electrotechnical Commission, 2007.

[16] IEC61025, Fault Tree Analysis (FTA), Edition 2.0, International Electrotechnical Commission, 2006.

[17] M. Jackson, *Problem Frames: Analysing and Structuring Software Development Problems*, Addison-Wesley, 2001.

[18] W. Lidwell, K. Holden, and J. Butler, *Universal Principles of Design*, Rockport Publishers, 2010.

[19] M.S. Lund, B. Solhaug, and K. Stølen, *Model-Driven Risk Analysis: The CORAS Approach*, 1st Edition, Springer, 2010.

[20] OMG, Unified Modeling Language Specification, Version 2.4.1, Object Management Group, 2011.

[21] T. Sivertsen, “Metodikk for formell verifisering”, Jernbaneverket Banedivisjonen Teknikk, Jernbaneverket, 2009.

[22] E.R. Tufte, *The Visual Display of Quantitative Information*, 2nd Edition, Graphics Press, 2001.

APPENDIX A BASIC SACS PATTERNS

Appendix A provides the definitions of the patterns used in this report. The reports HWR-1029 and HWR-1037 describes the application of the SaCS method on two different cases. Some of the patterns are used in both cases, thus some of the pattern definitions in this report are also given in HWR-1029. The definitions of the patterns *Station Interlocking Requirements*, *Level Crossing Interlocking Requirements*, *Dual Modular Redundant, and SIL Classification* given in appendix A.1, A.2, A.3 and A.8, respectively, are only presented in this report.

A.1 STATION INTERLOCKING REQUIREMENTS

Name: Station Interlocking Requirements

Pattern Signature: *Station Interlocking Requirements* is defined with the signature illustrated in Figure 30.

In Figure 30, the following abbreviations are used for denoting the parameters of the pattern:

- *Mch* is short for Machine.
- *Req* is short for Requirements.

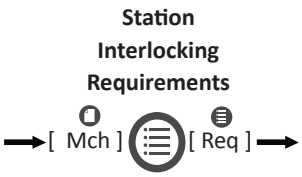


Figure 30 Station Interlocking Requirements – Pattern Signature

Intent: Support eliciting functional requirements *Req* for an interlock system *Mch* that shall control the appliances of a train station with two or more tracks at the station area. A typical example of a station with two tracks is given in Figure 31.

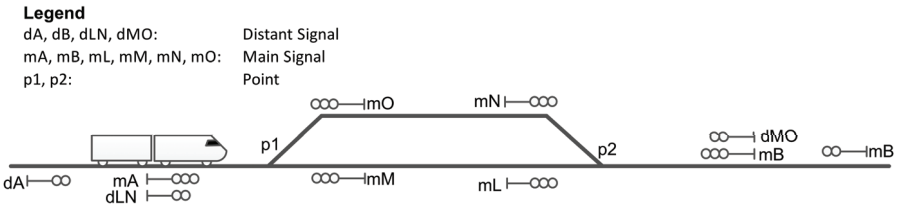


Figure 31 Example – Signal layout at a two track station

Figure 32 illustrates a typical sectioning of the railway tracks of a two-track station.

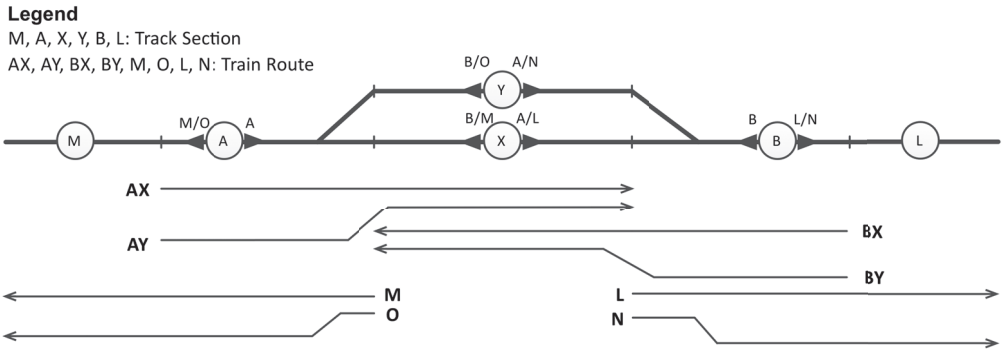


Figure 32 Example – Track sections and train routes at a station with two tracks

Applicability: The *Station Interlocking Requirements* pattern is intended for the following situation:

- To derive the functional requirements while building or upgrading an interlocking system that shall control the appliances of a station with two or more tracks.

Problem: The main aspects relevant for establishing functional requirements for the interlock system are:

- *States of appliances*: What are the possible states of the different appliances like distant signals, main signals and points.
- *System states*: What are the possible system states.
- *Transitions*: What are the expected transitions between system states.
- *Safety*: Is there some system states or transitions that shall never be allowed and that may be hazardous (e.g., simultaneous incoming or outgoing traffic or drive through).

Problem Frame Analysis Solution: Figure 33 illustrates the *Station Interlocking Requirements* problem frames diagram.

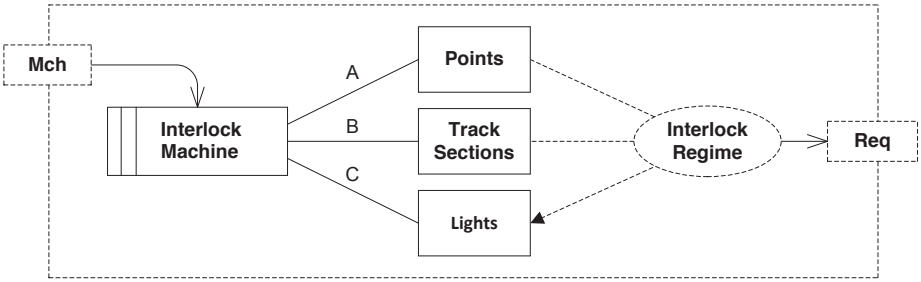


Figure 33 Station Interlocking Requirements – Problem Frame Diagram

The input parameter associated with the problem frame diagram shall be interpreted as follows:

- *Mch*: represents the machine to be constructed or an existing system that is modernised, and that is responsible for controlling train movements in a safe manner.

The expected output from instantiating the pattern is identified as:

- *Req*: represents the set of requirements derived on the basis of instantiating the pattern according to the defined instantiation rule.

The problem domains that are represented in the problem frame diagram shall be interpreted as:

- *Points*: represents any point part of the track layout (e.g., the two points exemplified in Figure 31).
- *Track Sections*: represents the different sections of the tracks (e.g., as exemplified in Figure 32).
- *Lights*: represents the distant signals and the main signals (e.g., as exemplified in Figure 31).

The *Interlock Machine* shall assure safe allocation of train routes for trains. There are eight possible train routes with the configuration exemplified in Figure 32. A train route may be in two states, we name these states *Locked* (allocated to a train) and *Not Locked*. The *Interlock Machine* interacts with the appliances represented as problem domains in Figure 33 through the following interfaces in order to secure train routes:

- *A*: represents the interaction between the *Interlock Machine* and the *Points*. The points may be in two positions; we name these positions *Aligned* and *Diverging*.
- *B*: represents the interaction between the *Interlock Machine* and the *Track Sections*. A track section may be in two states, we name these states *Vacant* and *Occupied*.
- *C*: represents the interaction between the *Interlock Machine* and the *Lights*. We assume here that the lights may be in the following states depending on the type of signal:
  - A distant signal may be in the states: *Expect Proceed, Expect Proceed Slow* or *Expect Stop*.
  - A main signal may be in the states: *Proceed, Proceed Slow* and *Stop.* An outbound main signal on a diverging track (e.g., mO or mN in Figure 31) can only be in the states *Proceed Slow* or *Stop* because the train when moving shall only proceed with a limited speed.

Requirements for the *Interlock Machine* may be elicited by instantiating the abstract requirements given in Table 4.

Table 4 Station Interlocking Requirements – Abstract Requirements

ID	Requirement	Note on instantiation
R.1	*Mch* shall manage train movements along the following train routes: *train routes*	Identify every *train route* that is required to be controlled (e.g., AX, AY,M,… as in Figure 32) and that is part of all *train routes.*
R.2	*Mch* may set *train route* as locked when: *conditions*	For each *train route (e.g.,* AX in Figure 32*)*: define each *condition* (e.g., A in state *Not Locked*, X in state *Not Locked*) part of *conditions* that must be satisfied in order to lock a train route (assign a train route to a train).
R.3	*Mch* shall when *train route* becomes locked signal: *signalling*	For each *train route:* define the signalling to be performed once the train route is locked.
R.4	*Mch* shall detect *train route* as commenced when: *conditions*	For each *train route* (e.g., AX in Figure 32*)*: define each *condition* (*e.g., A* goes to state *Occupied)* part of *conditions* that must be satisfied in order to detect the *train route* as commenced.
R.5	*Mch* shall when *train route* is detected as commenced signal: *signals*	For each train route (e.g. AX in Figure 32*)*: define the signalling to be performed once the train route is detected as commenced (e.g. mA goes to state Stop)
R.6	*Mch* may only unlock *train route* when: *conditions*	For each *train route* (e.g., AX in Figure 32*)*: define each *condition* (e.g., *A* goes to state *Vacant*) part of *conditions* that must be satisfied in order to unlock a locked train route.

Instantiation Rule: An artefact *Req* (see Figure 33 and Figure 30) instantiates the *Station Interlocking Requirements* pattern if:

- *Req* is a set of requirements.
- *Req* is the result of an analysis, e.g., by the use the problem frames analysis approach as outlined in Figure 33 and the guidance provided in Section “Problem Frame Analysis Solution”, of how a system *Mch* shall operate in order to assure safe train movements at a train station. In order to perform the analysis, the system *Mch* must be identified and described. Relevant problem domains that interact with *Mch* or in other ways are important for the interlocking scenario must be identified and their possible states must be defined. The objective of the analysis is to define the interlocking rules, in the form of requirements *Req* for *Mch* that assures the required functionality and safety.
- Every requirement of *Req* is an instance of an abstract requirement. Abstract requirements are defined in column “Requirement” of Table 4.
- An abstract requirement is instantiated by applying the guidance provided in the column “Note on instantiation” of Table 4.
- Every requirement of *Req* is traceable to a unique abstract requirement.
- Every requirement of *Req* describes a property, behaviour or a constraint of the system (identified by the instantiation of *Mch*).

A.2 LEVEL CROSSING INTERLOCKING REQUIREMENTS

Name: Level Crossing Interlocking Requirements

Pattern Signature: *Level Crossing Interlocking Requirements* is defined with the signature illustrated in Figure 34.

In Figure 34, the following abbreviations are used for denoting the parameters of the pattern:

- *Mch* is short for Machine.
- *Req* is short for Requirements.

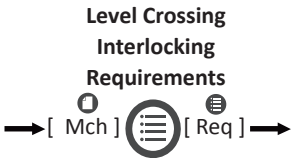


Figure 34 Level Crossing Interlocking Requirements – Pattern Signature

Intent: Support the elicitation of functional requirements *Req* for an interlocking system *Mch* that shall control the appliances of a level crossing. An example of a railway track with a level crossing is given in Figure 35. A level crossing is illustrated as a road lane crossing a single railway track.

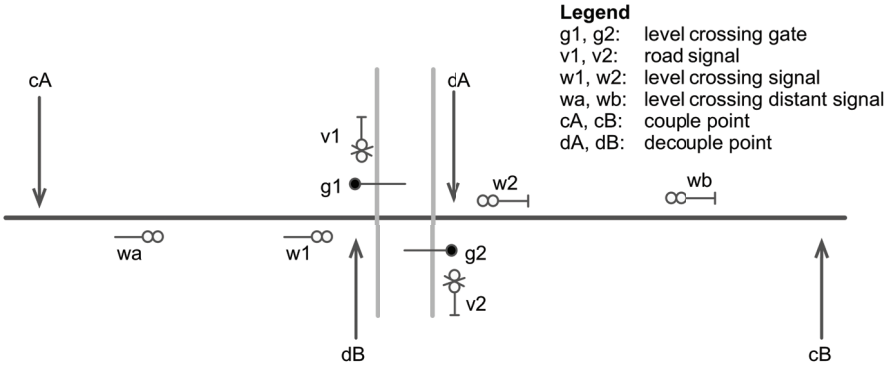


Figure 35 Example signal layout illustration

Applicability: The *Level Crossing Interlocking Requirements* pattern is suitable to apply in the following situation:

- To derive the functional requirements while building or upgrading an interlocking system that shall control the appliances of a level crossing.

Problem: The main aspects relevant for establishing functional requirements for a level crossing interlocking system are:

- *Possible states of appliances*: What are the possible states of the different appliances like level crossing signals, road signals and gates.
- *Possible System States*: What are the possible system states.
- *Transitions*: What are the expected transitions between system states.
- *Safety*: Is there some system states or transitions that shall never be allowed and that may be hazardous.

- *Dependencies*: Is there any dependencies towards other interlocking equipment, e.g., the interlocking on a nearby station.

Problem Frame Analysis Solution: Figure 36 illustrates the *Level Crossing Interlocking Requirements* problem frames diagram.

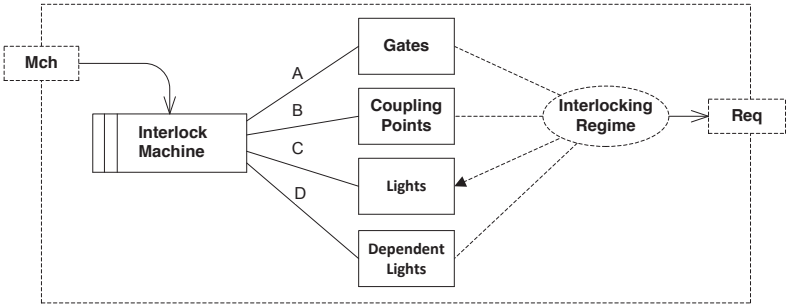


Figure 36 Level Crossing Interlocking Requirements – Problem Frames Diagram

The input parameter associated with the problem frame diagram shall be interpreted as:

- *Mch*: represents the machine to be constructed or an existing system that is modernised, and that is responsible for controlling train movements in a safe manner.

The expected output from instantiating the pattern is identified as:

- *Req*: represents the set of requirements derived on the basis of instantiating the pattern according to the defined instantiation rule.

The problem domains that are represented in the problem frame diagram shall be interpreted as:

- *Gates*: represents the level crossing gates referred to in Figure 35.
- *Coupling Points*: represents the different couple and decouple points indicated in Figure 35.
- *Lights*: represents the different road signals referred to in Figure 35.

The *Interlock Machine* shall safely control the appliance for securing a level crossing and assure proper signalling to road users and trains. The *Interlock Machine* interacts with the appliances represented as problem domains in Figure 36 through the following interfaces in order to secure the level crossing:

- *A*: represents the interaction between the *Interlocking Machine* and the *Gates*. The gates may be in two states, we name these states *Closed* and *Elevated*.
- *B*: represents the interaction between the *Interlock Machine* and the *Coupling Points*. A coupling point may be in two states, we name these states *True* and *False*.
- *C*: represents the interaction between the *Interlock Machine* and the *Lights*. The lights may be in the following states depending on type of signal:
  - A road signal may be in two states; we name these states *Go and Stop*.
  - A level crossing signal may be in two states; we name these states *Passage of Crossing Allowed and Stop Before Crossing*.
  - A level crossing distant signal may be in two states; we name these states *Expect Passage of Crossing Allowed and Expect Stop Before Crossing*.
- *D*: represents the interaction between the *Interlock Machine and Dependent Lights*. The dependent lights can be light signals that are associated with the interlocking at a station. If the level crossing is placed in between light signals at a station (e.g., scenarios illustrated in Figure 37), then the level crossing shall have a dependency towards relevant station signals. The two scenarios in Figure 37 depict different types of dependencies. In scenario A, the road crosses

the track between an incoming main signal and two outgoing main signals. In scenario B, the road is placed between a distant signal and a main signal.

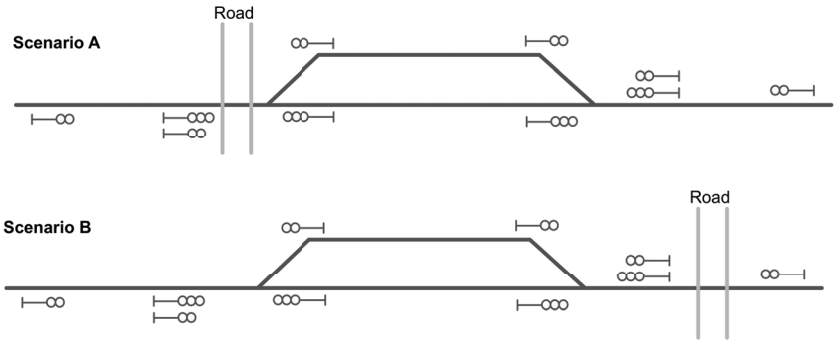


Figure 37 Scenarios for defining dependencies between interlocking and level crossing

Requirements for the *Interlock Machine* can be elicited by the use of the abstract requirements and notes on instantiating the abstract requirements given in Table 5.

Table 5 Level Crossing Interlocking Requirements – Abstract Requirements

ID	Requirement	Note on instantiation
R.1	*Mch* shall facilitate safe level crossings by controlling the level crossing appliances in the following system states: *states*	Define each state associated with the level crossing system, e.g., open state, closed state.
R.2	*Mch* shall by default control the level crossing appliances in state *state*	Define any default state, e.g., open state.
R.3	*Mch* shall set the level crossing system in state *state* when: *conditions*	For each system state (e.g., closed): detail the conditions that must be satisfied in order to reach state (e.g., gates *g1* and *g2* indicated in Figure 35 are in the state *Elevated*)
R.4	*Mch* shall when in system state *state* assure*: appliance* is in state *state*	For each system state: detail the required state of each appliance (e.g., when in state *open* then gates *g1* and *g2* indicated in Figure 35 are in the state *Elevated*)
R.5	*Mch* shall when in *state* set the level crossing system to default state when: *conditions*	For each system state: detail the conditions for transferring to a default state (if applicable)

Instantiation Rule: An artefact *Req* (see Figure 36 and Figure 34) is the result of instantiating the *Level Crossing Interlocking Requirements* pattern if:

- *Req* is a set of requirements.
- *Req* is the result of an analysis, e.g., a result of the use the problem frames analysis approach as outlined in Figure 36 and the guidance provided in Section “Problem Frames Analysis Solution”, on how a system *Mch* shall operate in order to assure safe train movements at a level crossing. In order to perform the analysis, the system *Mch* must be identified and described. Relevant problem domains that interact with *Mch* or in other ways are important for the interlocking scenario must be identified and their possible states must be defined. The objective of the

analysis is to define the interlocking rules, in the form of requirements *Req* for *Mch*, that assures the required functionality and safety.

- Every requirement of *Req* is an instance of an abstract requirement. Abstract requirements are defined in column “Requirement” of Table 5. An abstract requirement is instantiated by applying the guidance provided in the column “Note on instantiation” of Table 5.
- Every requirement of *Req* is traceable to a unique abstract requirement.
- Every requirement of *Req* describes a property, behaviour or a constraint of the system (identified by the instantiation of *Mch*).

A.3 DUAL MODULAR REDUNDANT

Pattern Signature: The *Dual Modular Redundant* is defined with the signature illustrated in Figure 38.

In Figure 38, the following abbreviations are used for denoting the parameters of the pattern:

- *Cmd* is short for Command.
- *C1* is short for Controller 1.
- *C2* is short for Controller 2.
- *V* is short for Voter.
- *S* is short for System.

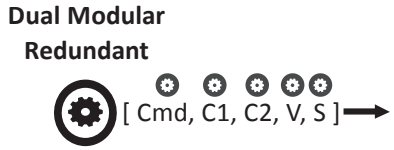


Figure 38 Dual Modular Redundant – Pattern Signature

Intent: Describe the design of a system *S* that offers a simple and cost effective protection against random hardware failure by the use of two redundant controllers, *C1* and *C2* that operate in parallel. Command signals to the two controllers and status indications from these are handled by the component *Cmd*. *Cmd* also provides the interface between the components performing control operations and the overall system. A voter *V* is used to implement functionality for reacting upon discrepancies between the parallel operating controllers.

Applicability: The *Dual Modular Redundant* pattern is intended for the following situations:

- When the reliability of the individual controllers in the dual controller setup and supporting system parts is high enough, and there is satisfactory protection against random failure, to achieve adequate system reliability and safety.
- When the safety management and quality management are performed to such an extent that ensures sufficient protection against systematic errors and furthermore that the dual modular redundant configuration to be sufficiently robust.

Problem: The main aspects to be considered when providing protection against failure are:

- *Random hardware failure*: hardware is susceptible to wear and tear and failure may manifest at random. Protection should be incorporated against random hardware failure.
- *Separation of functions*: it must be ensured that the potential failures of non-critical functions do not negatively affect critical functions. Therefore, there should be a clear separation between functional parts of different safety integrity.
- *Reliability of components*: it must be ensured that reliability of individual components is satisfactory in order to meet reliability targets.
- *Common cause failure*: it must be ensured that the system is adequately protected against common cause failures.
- *Fail-safe behaviour*: it should be ensured that any potential failure of the system should not negatively affect safety.

Design Solution: Figure 39 illustrates the main structure of components and their interfaces of the *Dual Modular Redundant design*, specified using UML notation.

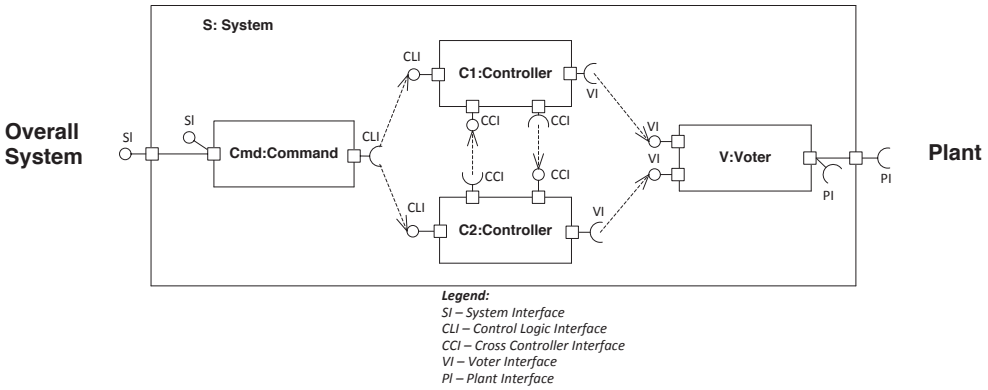


Figure 39 Dual Modular Redundant Design

The component *Cmd* is responsible for communicating with an overall system and the parts performing critical operations, *C1* and *C2*. *C1* and *C2* are two identical parts/components that operate in parallel and perform critical operations (e.g., interlocking operations). A voter interacts with the plant or Equipment Under Control (EUC) and carry out the control operations from *C1* and *C2*. The voter will implement mechanisms to provide 1-out-of-2 or 2-out-of-2 voting functionality.

An important feature with respect to utilising the proposed design for safety critical systems is the physical and functional separation of non-critical functionality (resided in *Cmd*) from the critical functionality (resided in *C1* and *C2*). The redundant controllers provide protection against random hardware failure that may affect control operations and the voter assures that no potential erroneous signalling is performed due to random hardware failure. Protection against systematic failure can be handled by following adequate activities during the development process (e.g., safety management activities and quality management activities); this is out of scope of the pattern.

The roles and responsibilities represented in the pattern are:

- *S: System* – represents the system defined by the *Dual Modular Redundant*. The system (e.g., a railway interlocking system) interacts with an overall system or human operator (e.g., a train leader through a Centralised Train Control (CTC) system) and the plant that is controlled (e.g., distant light signals, main light signals, and track switches).
- *C1: Controller* – responsible for providing control signals (e.g., interlocking control signals) based on plant states and commands from the overall system or some operator.
- *C2: Controller* – responsible for providing control signals (e.g., interlocking control signals) based on plant states and commands from the overall system or some operator.
- *Cmd: Command* – responsible for the interaction with an overall system such that commands (e.g., from an operator) may be communicated to the control system. *Cmd* can also be responsible for secondary functions such as data logging.
- *V: Voter* – responsible for activating plant control in accordance with the commands given by the dual controllers and take proper safe actions in the case of disagreement (that is an indication of a component failure) between the controllers.

Instantiation Rule: An artefact *S* (see Figure 39 and Figure 38) instantiates the *Dual Modular Redundant* pattern if:

- *S* is a specification of a control system with hardware and/or software parts/components.
- *S* specifies a system containing at least two redundant controller parts that operate in parallel and that interface with a voting mechanism.

- *S* specifies a system that has a part that provides an interface with an overall system and the dual controllers such that orders may be given to, and indications obtained from, the controllers.
- *S* specifies a system that has a part that is responsible for providing a voting mechanism such that protection against random failure is detected and may be mitigated.

Related Patterns: The pattern relates to the other patterns in the following manner:

- Any product requirement pattern that might be used to derive the functional requirements for the system.
- Any process requirement pattern that might be used to assess and derive safety functional requirements for the system.
- Any safety case pattern that might be used as a means to argue that the system is adequately safe for its purpose.

Known Uses: In Norway, a commonly used interlocking system known as NSB-94 uses a dual modular redundant design similar to the one described here. Critical functions, such as the interlocking functionality, are implemented in terms of two identical PLCs (Programmable Logic Controllers) that operate in parallel. A voting mechanism ensures that no random hardware failure may lead to potentially harmful signalling, e.g., no train receives a go (green light signal) if there is any disagreement between the controllers (2-out-of-2 voting is required to give a go). In case of disagreement, e.g., one controller commands a stop signal (red light signal) and the other controller commands a go signal for a given light then stop is signalled (1-out-of-2 voting to give a stop).

A.4 HAZARD ANALYSIS

Name: Hazard Analysis

Pattern Signature: *Hazard Analysis* is defined with the signature illustrated in Figure 40.

In Figure 40, the following abbreviations are used for denoting the parameters of the pattern:

- *ToA* is short for Target of Assessment.
- *Src* is short for Source.
- *Haz* is short for Hazards.
- *IdCsHz* is short for Identified Causes of Hazards.
- *HzLg* is short for Hazard Log.

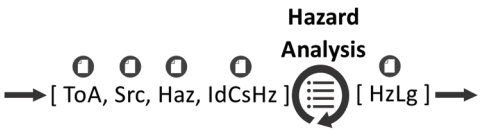


Figure 40 Hazard Analysis – Pattern Signature

Intent: Support the assessment of a target system *ToA* by defining the process of deriving the potential causes of hazards *Haz*. The expected result of applying the pattern is a hazard log *HzLg* that documents the hazards *Haz* and the potential causes of hazards *IdCsHz* associated with a system *ToA*. The hazard log *HzLg* facilitates the evaluation of the potential of the system to negatively affect safety. The pattern describes: 1) the main characteristics of the process of hazard analysis, and 2) the main requirements for specifying the result of the process in a hazard log.

Applicability: The *Hazard Analysis* pattern is suitable to apply in the following situations:

- When the system under construction is intended to be used in a context where there exists safety concerns (e.g. potential threats to life or environment).
- When hazards applicable to a system under construction are given and the potential causes of identified hazards are required to be identified.

Problem: The main aspects to be considered when performing hazards analysis are:

- *Scope*: To correctly delimit the scope of the analysis such that the cost of performing hazard analysis is minimised, but where the scope is sufficiently broad to capture all relevant safety concerns.
- *Hazards*: To ensure that the sources providing data on hazards relevant for the system under construction provide valid information.
- *Sources*: To ensure that the sources required for identifying potential causes of hazards are accounted for. Sources may represent experience data on incidents, accidents, and safety assessments.
- *Methods*: To ensure that the methods applied in order to assess the potential causes of hazards are suitable and are able to provide the intended results. The intended result is to identify all relevant causes of hazards.
- *Causes*: To ensure that the potential causes that can lead to hazards are identified.
- *Completeness*: To ensure that all relevant hazards are assessed with respect to potential causes.

Process Solution: Figure 41 illustrates the *Hazard Analysis* process annotated in a UML activity diagram.

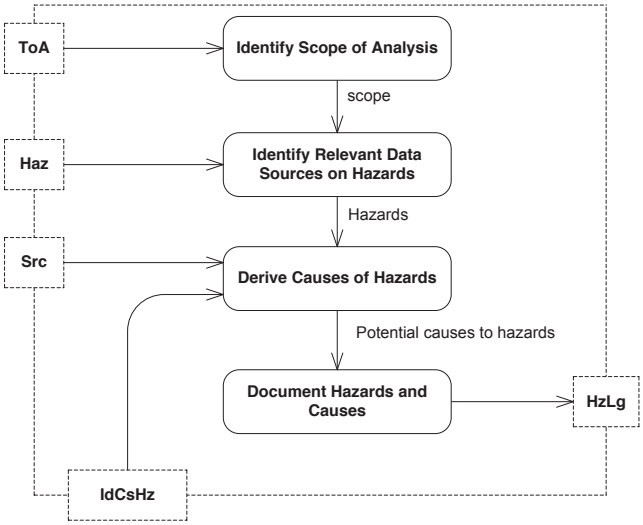


Figure 41 Hazard Analysis – Activity Diagram

The input parameters associated with the activity diagram may be interpreted as follows:

- *ToA (Target of Assessment*): represents the entity that the hazard analysis concerns.
- *Haz (Hazards*): represents information on identified hazards.
- *Src (Source*): represents information on potential causes of hazards.
- *IdCsHz (Identified Causes of Hazards*): represents the results from any method supporting the activity of identifying potential causes of hazards.

The main activities intend to serve the following purpose:

- *Identify Scope of Analysis*: the intent of the activity is to define the boundary of the investigation, its objectives, and a strategy for how to satisfy the objectives defined. The strategy should involve a plan for the study, set the responsibilities of those involved, describe how data shall be collected and detail a schedule.
- *Identify Relevant Data Sources on Hazards:* the intent of the activity is to gather information on identified hazards applicable to the system under construction, here the target of assessment.
- *Derive Causes of Hazards*: the intent of the activity is to assess a target in order to identify its potential contribution to events that may lead to hazards. The target should be assessed by examining relevant historic data and by the use of relevant assessment methods. One method or a set of complementary methods may be used as long as the total coverage is in accordance with the defined scope of analysis.
- *Document Hazards and Causes*: the intent of the activity is to combine all relevant information with respect to the identification and analysis of hazards in a document form containing all identified hazards and their potential causes.

Instantiation Rule: An artefact *HzLg* (see Figure 41 and Figure 40) is the result of the instantiation of the *Hazard Analysis* pattern if:

- *HzLg* is a documentation of the results from applying the process described in Section “Process Solution”.
- *HzLg* is a documentation of all relevant hazards associated with the use of a target system (identified by the instantiation of *ToA*) in a specific context.
- Every hazard of *HzLg* is uniquely identified.

- Every hazard of *HzLg* describes a potential danger of the application of a target system in a specific context.
- Every hazard of *HzLg* is traceable to its origin documentation, meaning the document where the hazard is identified (identified by the instantiation of *Haz* or *Src*).
- Every hazard is associated with a description of potential causes.
- Every potential cause associated with a hazard is traceable to its origin documentation, meaning the document (identified by the instantiation of *IdCsHz*) where the causal relationship may be identified.

Related Patterns: The *Hazard Analysis* pattern is related to other process requirement patterns in the following manner:

- May succeed *Hazard Identification* in terms of supplementing the hazard log provided as an output of *Hazard Identification* with information on potential causes of hazards.
- May precede *Risk Analysis* by providing as a deliverable, a set of hazards and potential causes associated with the operation of a target system (ToA) such that risk may be determined.
- May be used in order to address an abstract design defined in a design pattern or the instantiation of a design pattern with respect to its potential contribution to hazards.
- May be used together with other method patterns that support assessment of the causes of hazards.

Known Uses: The pattern describes a process of identifying the potential causes of hazards in accordance with the practice as described in several safety standards and guidelines, to notable standards are [IEC61508] and [EN50129].

[IEC61508] International Electrotechnical Commission, Functional safety of electrical/electronic/ programmble electronic safety-related systems, IEC 61508, Edition 2.0, 2010.

[EN50129] European Committee for Electrotechnical Standardization, Railway Applications – Communications, signalling and processing systems – Safety related electronic systems for signalling, EN 50129, 2003.

A.5 Risk Analysis

Name: Risk Analysis

Pattern Signature: *Risk Analysis* is defined with the signature illustrated in Figure 42.

In Figure 42, the following abbreviations are used for denoting the parameters of the pattern:

- *ToA* is short for Target of Assessment.
- *Haz* is short for Hazards.
- *ClsSev* is short for Classification of Severity.
- *ClsLi* is short for Classification of Likelihood.
- *ClsCr* is short for Classification of Criticality.
- *Risks* is not abbreviated.

Figure 42 Risk Analysis – Pattern Signature

Intent: Support the assessment of a target system *ToA* with respect to risk. A process for risk assessment supports this where the result of an analysis of the likelihood of hazards as well as the result of an analysis of the severity of an accident associated with the occurrence of a hazard is combined into a notion of risk. The expected result *Risks* of applying the pattern represents a notion of the potential danger of applying the target system in a given context.

Applicability: The *Risk Analysis* pattern may be suitable to apply in the following situation:

- When the system under construction is already analysed with respect to its potential contribution to hazards in its intended context.

Problem: The main aspects to be considered when performing risk analysis are:

- *Defined target*: to clearly state what is the target of the assessment and correctly delimit the scope of the risk analysis.
- *Hazards*: assessment of risk is performed on the basis of hazards and their potential causes associated with a target. It is expected that data on hazards and hazard causes will be provided.
- *Classification*: in order to establish a notion of risk in a consistent manner such that risk associated with different systems or system entities can be easily compared. A categorisation scheme for discretising data on severity, likelihoods and risk is commonly applied.
- *Estimation*: in order to provide a notion of risk, severity and likelihood estimates are required to be provided either quantitatively or qualitatively.
- *Mitigations*: in order to sustain a certain risk level or reduce a risk to an acceptable level, mitigating means (or risk reduction means) are used. A part of risk analysis is to identify potential mitigations.

Process Solution: Figure 43 illustrates the *Risk Analysis* process specified using a UML activity diagram.

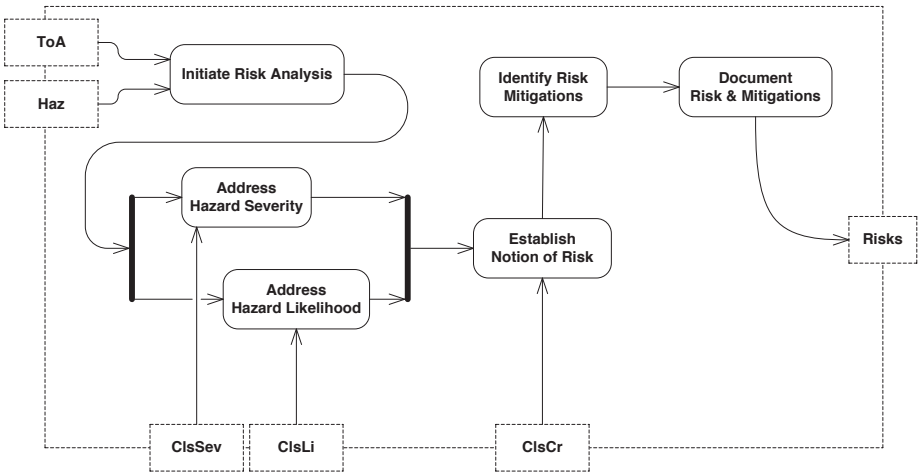


Figure 43 Risk Analysis – Activity Diagram

The input parameters associated with activity diagram may be interpreted as follows:

- *ToA (Target of Assessment*): represents the target of the assessment that is analysed with respect to risk.
- *Haz (Hazards*): represents the hazards that are associated with the target.
- *ClsSev*: represents the results from any method supporting the classification of the severity of hazards.
- *ClsLi*: represents the result from any method supporting the classification of likelihood of hazards.
- *ClsCr*: represents the results from any method supporting the classification of risk where risk is represented as a combined measure of the associated severity and the associated likelihood of hazards.

The main activities intend to serve the following purpose:

- *Initiate Risk Analysis*: the intent of the activity is to define the target of assessment, its intended context of operation, and identify hazards associated with the use of the target system in its context. It is expected that relevant information for risk analysis be provided as documentation of the target system. Relevant information include system hazards as well as the unwanted events associated with the operation of the target system that might lead to hazards.
- *Address Hazard Severity*: the intent of the activity is to address the severity of the consequence of hazards.
- *Address Hazard Likelihood*: the intent of the activity is to address the likelihood of occurrence of each hazard.
- *Establish Notion of Risk*: the intent of the activity is to combine data on likelihood of a hazard occurring and data on severity of the consequence of a hazard into a notion of risk.
- *Identify Risk Mitigations*: the intent of the activity is to identify risk reduction in the form of mitigations to reduce the likelihood of a hazard occurring and/or the severity of the consequence should a hazard occur.
- *Document Risk & Mitigations*: the intent of the activity is to combine all relevant information with respect to the assessment of risk in a document form containing all relevant risks and associated mitigations.

Instantiation Rule: An artefact *Risks* (see Figure 43 and Figure 42) is the result of instantiating the *Risk Analysis* pattern if:

- *Risks* is a documentation describing every relevant risk associated with the use of a target system (identified by the instantiation of *ToA*) in a specific context.
- Every risk of *Risks* is uniquely identified.
- Every risk of *Risks* is traceable to its origin documentation. By origin documentation, we mean: the document where the hazard relevant to risk is identified (identified by the instantiation of *Haz*), documentation on estimates on the severity of the consequence of hazards (identified by the instantiation of *ClsSev*), documentation on estimate of the likelihood of hazard (identified by the instantiation of *ClsLi*).
- Every risk of *Risks* is associated with a description of mitigation. The description should describe any dependencies or possible mitigations for reducing risk to an acceptable level.
- Every mitigation associated with a risk is stated such that its purpose and how it reduces risk can be clearly identified.

Related Patterns: The *Risk Analysis* pattern is related to other patterns in the following manner:

- May succeed *Hazard Analysis* in terms of supplementing the assessment of relevant hazards and their potential causes with information on the associated risk, e.g., by adding likelihood estimates and severity estimates associated with hazards.
- May precede *Establish System Safety Requirements* by providing information on the risks associated with the operation of the system, and information on required mitigations and potential mitigations in order to reduce risk. This ensures that the specification of safety requirements is based on a risk assessment of a target system.
- May be used in order to address the instantiation of a design pattern with respect to risk.
- May be used with other method patterns that support assessment or categorisation of the severity of the consequence of hazards.
- May be used with other method patterns that support the assessment or categorisation of the likelihood of the causes of hazards.
- May be used with other method patterns that support the classification of risk.

Known Uses: The pattern describes a process of assessing risk in accordance with the practice as described in several safety standards and guidelines, to notable standards are [IEC61508] and [EN50129].

[IEC61508] International Electrotechnical Commission, Functional safety of electrical/electronic/ programmble electronic safety-related systems, IEC 61508, Edition 2.0, 2010.

[EN50129] European Committee for Electrotechnical Standardization, Railway Applications – Communications, signalling and processing systems – Safety related electronic systems for signalling, EN 50129, 2003.

A.6 Establish System Safety Requirements

Name: Establish System Safety Requirements

Pattern Signature: *Establish System Safety Requirements* is defined with the signature illustrated in Figure 44.

In Figure 44, the following abbreviations are used for denoting the parameters of the pattern:

- *ToA* is short for Target of Assessment.
- *Reg* is short for Regulations.
- *Risks* is not abbreviated; represents the documentation of the risks associated with the application of *ToA* in its intended context.
- *Req* is short for Requirements.

Figure 44 Establish System Safety Requirements Pattern Signature

Intent: Support the specification of system safety requirements *Req* on the basis of a risk-based approach. The safety requirements describe the required measures to be satisfied by the system *ToA* to assure the necessary safety integrity. The general approach for defining safety requirements is to define them on the basis of the result of a risk assessment *Risks*, especially the mitigations identified as means to reduce risk to an acceptable level. The pattern describes the general process of capturing the requirements that must be satisfied in order to assure safety.

Applicability: The *Establish System Safety Requirements* pattern is intended for the following situations:

- When the system under construction may negatively affect the overall system safety.
- When there are identified measures that can mitigate identified risks and can be used as input to the specification of safety requirements.

Problem: The main aspects relevant to address when establishing the safety requirements are:

- *Characteristics*: To define the system characteristics to be satisfied such that the occurrence of unwanted events are minimised or avoided.
- *Functions*: To define the safety functions that assures safe operations.
- *Constraints*: To define the functional constrains that sufficiently delimit potentially hazardous operations.
- *Environment*: To define the operational environment that ensures safe operations.
- *Compliance*: To define the requirements that are required to be satisfied in order to comply with laws, regulation and standards, as a minimum the mandatory requirements related to assurance of safety. These requirements include requirements on applying some specific development process, perform certain activities, or make use of specific techniques.

Process Solution: Figure 45 illustrates the *Establish System Safety Requirements* process specified using a UML activity diagram.

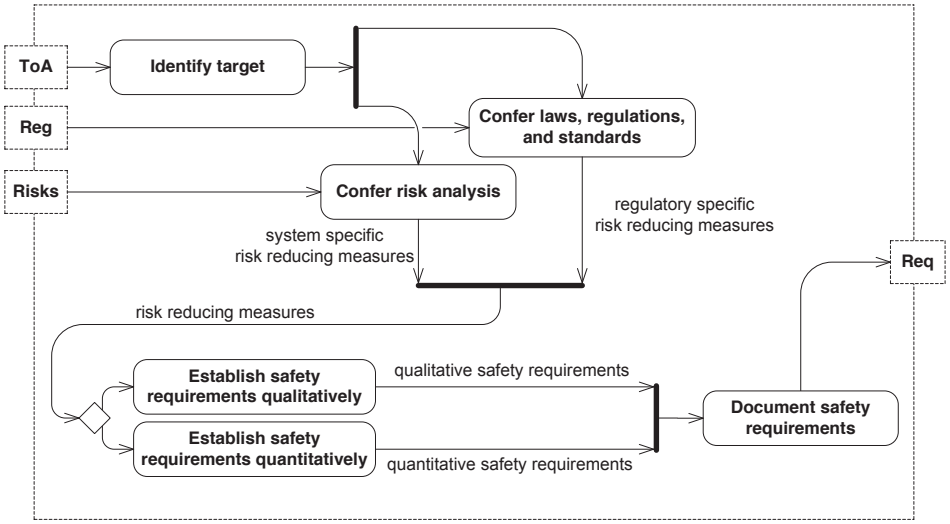


Figure 45 Establish System Safety Requirements – Process Flow

The input parameters associated with the activity diagram may be interpreted as follows:

- *ToA (Target of Assessment*): represents the target system for which safety requirements should be established.
- *Reg (Regulations*): represents any source of information describing mandatory or recommended practices (e.g. as provided in laws, regulations or standards) valuable for identifying risk reducing measures.
- *Risks*: represents risks associated with the target system.

The main activities serve the following purpose:

- *Identify target*: the intent of the activity is to identify *ToA*. The description of the target should as a minimum include a definition of the system and its boundaries, its operational profile, functional requirements, and safety integrity requirements.
- *Confer laws, regulations, and standards*: the intent of the activity is to capture all relevant data (requirements for risk reducing measures) from relevant sources (normative references) in order to outline the set of risk reducing measures that shall be met by compliance. Each source is inspected in order to identify, as a minimum, the mandatory risk reducing measures that shall be met in order to be compliant.
- *Confer risk analysis*: the intent of the activity is to capture all the relevant data on risk analysis of the system that is under construction in order to outline the system specific risk reducing measures that shall be met.
- *Establish safety requirements qualitatively*: the intent of the activity is to define safety requirements on the basis of those identified risk reducing measures required applied, and which can be demonstrated fulfilled with qualitative reasoning.
- *Establish safety requirements quantitatively*: the intent of the activity is to define safety requirements on the basis of those identified risk reducing measures required applied, and which can be demonstrated fulfilled with quantitative reasoning.
- *Document safety requirements*: the intent of the activity is to detail all relevant information with respect to the requirements in a system safety requirements specification. For each requirement defined in the requirement specification, information detailing what influenced its definition should be provided, e.g., the associated risks, assumptions, calculations, and justifications.

Instantiation Rule: An artefact *Req* (see Figure 45 and Figure 44) is the result of a process that instantiates the *Establish System Safety Requirements* pattern if:

- *Req* is a set of requirements.
- *Req* is a result of applying a process illustrated in Figure 45 and described in Section “Process Solution”. The process is initiated by an activity on describing the target *ToA*. Once a description of the target system and its operational context is provided, the next activities shall identify the risk reducing measures to be applied to the target by conferring relevant laws, regulations and standards as well as the result of target specific risk analysis for guidance. Once all the relevant risk-reducing measures are identified, these shall be used as a basis to define the requirements to be met by the target system or by the process to be followed while developing the target. The requirements are defined quantitatively or qualitatively depending on the nature of the risk reducing measure that is addressed. The requirements are documented in a requirement specification *Req*.
- Every requirement of *Req* is traceable to relevant risks (identified by the instantiation of *Risks*), and/or regulatory requirements (identified by the instantiation of *Reg*).
- Every requirement of *Req* is justified such that any assumptions, calculations, and assessments that support the specification of the requirement as a safety requirement are provided.

Related Patterns: The *Establish System Safety Requirements* pattern is related to other patterns in the following manner:

- May succeed the *Risk Analysis* pattern that supports identifying risks. The *Establish System Safety Requirements* may be applied as support for defining the requirements to be fulfilled in order to reduce risk to an acceptable risk level.
- May be used in order to detail requirements for the design that is a result of an instantiation of a design pattern.

A.7 FTA

Name: FTA (Fault Tree Analysis)

Pattern Signature: *FTA* is defined with the signature illustrated in Figure 46.

In Figure 46, the following abbreviations are used for denoting the parameters of the pattern:

- *ToA* is short for Target of Assessment.
- *UE* is short for Unwanted Event.
- *FT* is short for Fault Tree

Figure 46 Pattern Signature – FTA

Intent: Support a systematic and deductive assessment of the potential causes of identified unwanted events *UE* of a target *ToA*. The pattern describes the fault tree analysis method and provides guidance on its application. A fault tree *FT* is expressed as a tree where the root of the tree denotes the unwanted event *UE* (e.g. representing a system hazard) that is analysed and leaf nodes denotes basic events (e.g. component failure modes) that may lead to the unwanted event. The primary goal is to identify minimal cut sets by the use of the tree structure. A cut set expresses a set of basic events such that if these occur at the same time, the unwanted event will occur. A minimal cut set expresses the minimal set of basic events that may lead to an unwanted event.

Applicability: The *FTA* pattern is intended for the following situations:

- As a means to deduce potential causes of unwanted events.
- As a means to identify combinations of faults that may lead to unwanted events.
- As a means to analyse system concepts, simple systems or complex systems.

Problem: The main challenges associated with methods for assessment of the potential causes of an unwanted event are:

- *Efficiency*: To support efficient application with minimal use of resources.
- *Logical*: To support a logical specification of the dependencies between an unwanted event that is investigated and the potential causes that is identified through the assessment.
- *Communication*: To support communication of results on a form that allows different actors to understand the findings with minimal effort.

Method Solution: Figure 47 exemplifies a fault tree and outlines the relationships between a fault tree and the inputs and outputs of the FTA pattern.

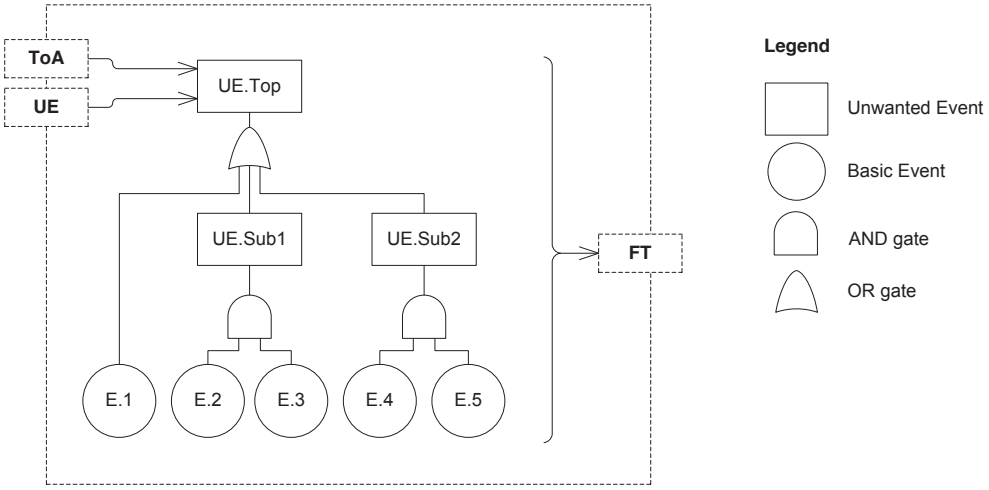


Figure 47 FTA example diagram

The following steps may apply the FTA method systematically:

- Step 1 – Define the top event: The intent is to define the top event that represent the root of a fault tree and the event that is analysed (UE.Top in Figure 47).
- Step 2 – Construct fault tree: The intent is to construct the fault tree by deductively constructing the tree in a top-down manner by the use of graphical elements symbolising events and gates. An unwanted event should state the fault/failure under consideration. Gates (e.g., the or gate in Figure 47) are used to define the combination of antecedents (e.g. basic event E.1, UE.Sub1, or UE.Sub2) that imply the consequent (e.g. UE.Top). Decomposition of an unwanted event (e.g., UE.Top) should not be described by connecting a series of gates down to basic events, instead intermediate unwanted events should be defined (e.g. UE.Sub1, and UE.Sub2). Basic events define how a system component or element can fail.
- Step 3 – Find minimal cut sets: The intent is to deduce from the tree structure the combination of basic events that may lead to the top event and to arrange these in order. The set with lowest order (contains least number of basic events) is most significant. In Figure 47, the event UE.Top occurs if event E.1 occurs (minimal cut set of order 1). The top event also occurs if events E.2 and E.3 occur or if events E.4 and E.5 occur (cut sets of order 2). There are cut sets of order 3, 4, and 5 but these are less significant.
- Step 4 – Qualitative Analysis: The intent is to analyse the fault tree qualitatively in order identify weak points of the system with respect to the top event. This may be performed on the basis of cut sets. If there exists cut-sets of order 1 then the system is vulnerable to single point of failure. If there exists no cut sets of order 1, but the events in the cut sets of order 2 have identical characteristics, then the system may be susceptible to common cause failures.
- Step 5 – Quantitative Analysis: The intent is to calculate the probability of a top event by combining probabilities of basic events (given that probabilities of basic events are provided). The probability of UE.Top in Figure 47 may be calculated by a combination of the following general rules:
  - The probability of an event A ‘’and’’ an event B is expressed as: P(A*B) = P(A) * P(B)
  - The probability of an event A ‘’or’’ an event B is expressed as: P(A+B) = P(A) + P(B) – (P(A) * P(B))

Instantiation Rule: A documentation artefact *FT* (see Figure 47 and Figure 46) instantiates the *FTA* pattern if:

- *FT* is a set of fault trees.
- Every fault tree of *FT* is instantiated by applying the guidance provided in Section “Method Solution” for all unwanted events (identified by the instantiation of *UE*). For a specific target *ToA*, this means that every unwanted event *UE* that represents a suitable top event in the analysis is systemically assessed according to the process outlined through Steps 1 to Step 5.
- Every fault tree of *FT* is traceable to a unique unwanted event (identified by the instantiation of *UE*).
- Every fault tree of *FT* describes the relation between an unwanted event, represented as a top node in the tree structure, and potential initiating events (e.g. HW/SW failures) associated with a target system (identified by the instantiation of *ToA*).

Related Patterns: The *FTA* pattern is related to other patterns in the following manner:

- May support *Hazard Analysis* pattern by providing a method for identification of potential causes of hazards.
- May support *Risk Analysis* pattern by providing a method for performing qualitative and quantitative analysis of hazards.

Known Uses: Fault tree analysis is a commonly applied method within safety critical domains and is described in several standards and guidelines, e.g. nuclear domain [NUREG–0492], aerospace [ARP4761], and the generic standard [IEC61025].

[NUREG–0492] NRC, Fault Tree Handbook, U.S. Nuclear Regulatory Commission (1981)

[ARP4761] SAE, Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment, Society of Automotive Engineers (1996)

[IEC61025] IEC, Fault Tree Analysis (FTA), Edition 2.0, International Electrotechnical Commission (2006)

A.8 SIL CLASSIFICATION

Name: SIL Classification

Pattern Signature: *SIL Classification* pattern is defined with the signature illustrated in Figure 48.

In Figure 48, the following abbreviations are used for denoting the parameters of the pattern:

- *ToA* is short for Target of Assessment.
- *Risks* is not abbreviated.
- *THR* is short for Tolerable Hazard Rate.
- *FncCat* is short for Function Categorisation.

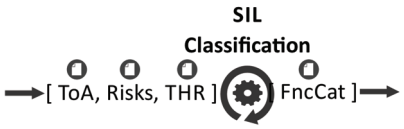


Figure 48 SIL Classification – Pattern Signature

Intent: Support the classification of railway system *ToA* with respect to its importance to safety. Railway systems may be classified according to a Safety Integrity Level (SIL) categorisation that defines five levels (SIL 4 to SIL 0, where SIL 4 categorises the most critical systems and SIL 0 categorises non-safety related systems). A SIL for a specific system is determined on the basis of the safety functions it shall perform and represents a measure of the ability to perform the safety functions (the higher the SIL, the lower the likelihood of failing to perform the required safety functions).

Applicability: The *SIL Classification* pattern is suitable to apply in the following situations:

- When developing railway systems, as SIL classification is a mandatory activity.
- When the classification of systems according to their importance to safety shall be performed according to [EN50128, EN50129].

Problem: The main challenges associated with categorisation of the functions of critical systems are:

- *Criticality*: provide categorisation of systems based on an evaluation of the criticality of the functions it offers.
- *Conformity*: provide categorisation of systems such that systems of same category may be associated with the same set of rules.

Method Solution: Figure 49 illustrates the *SIL Classification* method pattern diagram. The following information should be provided in order to apply the method:

- *ToA (Target of Assessment*): Specification of the target system and its function.
- *THR (Tolerable Hazard Rate*): Description of the tolerable rates of occurrence of hazards.
- *Risks*: Risks associated with *ToA* comprising a set of hazards associated with the operation of *ToA* such that for each hazard the potential severity and likelihood of occurrence is described.

The safety integrity level of a system depends on the required level of protection against systematic and random failures. A high degree of safety integrity is achieved by:

- Systematic failure integrity: represents the ability of a system not to contain hazardous systematic faults. The property is non-quantifiable and faults are caused by human errors in the various stages of the system/sub-system/equipment life cycle, e.g., specification error, design error, manufacturing error, installation error, operation error, and maintenance error. Systematic failure integrity is achieved by means of quality management and safety management.

- Random failure integrity: represents the ability of a system not to fail due to hazardous random failure events. Random failure is particularly associated with random hardware failure (e.g. material fatigue, short-circuit) and may be quantified based on reliability data of hardware components.

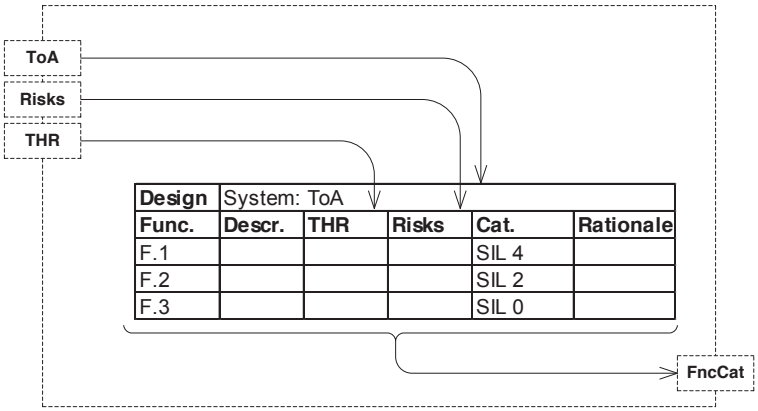


Figure 49 SIL Classification - Method Pattern Diagram

In order to establish a correct SIL of a ToA, the THR (Tolerable Hazard Rate) for each hazard associated with the operation of the ToA is defined first. Then, a SIL table is used to determine the SIL for functions from THR. A SIL is defined for each function. Thus, a SIL for an item (module, component, sub-system, or a system) is determined on the basis of the SIL associated with the functions performed by the item. If an item comprises several functions with different SIL, the highest SIL is selected as the SIL for the item. The steps that may be applied in order to determine SIL are:

- Step 1 – If THR are not available, define the safety assumptions and the relevant hazards associated with the intended use of ToA. If THR is given, proceed to Step 2.
- Step 2 – Describe the functions of the ToA, and for every function detail the relationship (available from risk analysis) to identified system hazards.
- Step 3 – For every identified safety related function, allocate SIL to the function based on the relationship between the function and potential hazards by the use of the SIL table given in Table 6 that defines the relationship between safety integrity levels and tolerable hazard rates per function.
- Step 4 – Describe the fulfilment of functions with respect to the items (module/component/sub-system/system) providing its HW or HW/SW implementation as given in the architecture specification of ToA.
- Step 5 – Define SIL for HW or HW/SW items based on the safety functions they fulfil.

Table 6 SIL Table (made up from [EN50128] and [EN50129] SIL tables)

THR (Tolerable Hazard Rate) per hour and per function	Description of software safety integrity	SIL (Safety Integrity Level)
$10^{-9} \leq THR < 10^{-8}$	Very High	4
$10^{-8} \leq THR < 10^{-7}$	High	3
$10^{-7} \leq THR < 10^{-6}$	Medium	2
$10^{-6} \leq THR < 10^{-5}$	Low	1
	Non safety-related	0

Instantiation Rule: An artefact *FncCat* (see Figure 49 and Figure 48) instantiates the *SIL Classification* pattern if:

- *FncCat* is a documentation of the results from a classification of the functions of a target system (identified by the instantiation of *ToA*).
- *FncCat* represents the results from applying the steps 1 to 5 described in Section “Method Solution” or represents the full results from applying steps 1 to 7 in Section “Method Solution”.
- Every function described in FncCat is categorised as SIL4 to SIL0 according to categorisation given in [EN50128] and [EN50129].

Related Patterns: The *SIL Classification* pattern is related to other patterns in the following manner:

- May support *Risk Analysis* pattern by providing a categorisation scheme for the categorisation of a system that is assessed with respect to risk.

Known Uses: The pattern represents the method for categorising systems within a railway context which is similar to the categorisation defined in the standard [EN50128] and [EN50129].

[EN50128] Railway Applications – Communication, Signalling and Processing Systems – Software for Railway Control and Protection Systems (2011)

[EN50129] Railway Applications – Communication, Signalling and Processing Systems – Safety Related Electronic Systems for Signalling (2003)

A.9 Overall Safety

Name: Overall Safety

Pattern Signature: *Overall Safety* is defined with the signature illustrated in Figure 50.

In Figure 50, the following abbreviations are used for denoting the parameters of the pattern:

- *ToD* is short for Target of Demonstration.
- *QualMng* is short for Quality Management.
- *SafMng* is short for Safety Management.
- *TechSaf* is short for Technical Safety.
- *Case* is short for Safety Case.

Figure 50 Overall Safety – Pattern Signature

Intent: Provide a structure for the specification of an overall strategy (or plan) for the safety demonstration *Case* of a target system *ToD*. The overall strategy is a means to document the set of practices that jointly provide confidence in the system being sufficiently safe for its intended use. The pattern describes a top-down deductive strategy to build the safety demonstration with a focus on establishing the overall argument for adequate managerial and technical safety. The focus of the pattern is on how to combine strategies on different safety concerns into an overall strategy such that it may be claimed and proven with confidence that the system is sufficiently safe.

Applicability: The *Overall Safety* pattern is intended for the following situations:

- At the initial stages of development in order to outline the main set of demonstration strategies, and how these strategies may be combined to demonstrate system safety.
- Prior to project initiation as a means for stakeholders (vendor, customer, safety authority) to reflect upon the main demonstration challenges and possible solutions.

Problem: The main aspects to be considered when providing a solution for the demonstration of safety are:

- *Quality management*: it is necessary to demonstrate that the quality of the system is controlled by an effective quality management system. Quality management is a means to minimise the occurrence of human errors at each stage of the development life cycle, and thus to reduce the risk of systematic faults in the system.
- *Safety management*: it is necessary to demonstrate that safety management is controlled by effective means. Effective safety management facilitates to reduce the occurrence of safety-related human errors throughout the life cycle and the risk of safety-related systematic faults.
- *Technical safety*: it is necessary to demonstrate the safety of the technical design.
- *Strategy identification*: in order to plan and effectively apply a suitable demonstration strategy for different systems, it is important to identify as early as possible the main lines of demonstration. This will reduce the cost related to performing safety assessment by avoiding those activities that does not support an effective safety demonstration. In addition, if the choice of demonstration strategies does not provide the necessary confidence, then there is a risk of not getting safety approval. The effect of not getting a safety approval may induce a great cost as additional work may be required in order to receive approval, or there will be a loss of returns if a project is discontinued.

- *Effectiveness*: it is important that the demonstration strategy is effective in providing the necessary safety demonstration. To develop safety related and safety critical systems is costly, mainly due to the activities required to assure and demonstrate that the system is sufficiently safe. It is important for a vendor to choose strategies that minimise cost but provide required results.
- *Confidence*: it is important to choose a demonstration strategy that has the potential to convince some safety authority reviewing the evidences that sufficient safety is achieved. In a development project, the effectiveness of different demonstration strategies must be weighed against the estimated cost and the ability of building confidence.
- *Acceptance*: it is important that the choice of demonstration strategies are acceptable for all involved parties such that they represent an effective means for demonstrating safety and may be applied in a manner that provides confidence that the system is safe.

Argument Structure Solution: Figure 51 and Table 7 together present the demonstration solution, and therefore should be read together. Figure 51 illustrates the decomposition of the safety argument in a tree structure using GSN notation. Table 7 details the tree structure by defining the type and expected content of each node.

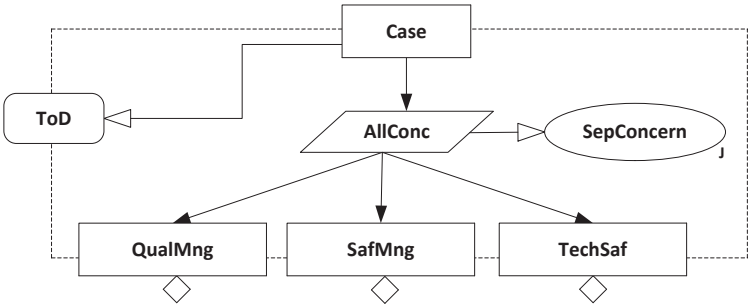


Figure 51 Overall Safety – Argument Structure

Associated with the contextual elements of the argument structure, the following is assumed:

- *ToD*: identifies the target system under consideration;

Table 7 Overall Safety – Argument Structure Details

Node	Node Type	Node Content Description
Case	Goal	*ToD* is safe
ToD	Context	Definition of parameter *ToD* (Target of Demonstration)
AllConc	Strategy	*ToD* is demonstrated safe by demonstrating that QM (Quality Management), SM (Safety Management), and TS (Technical Safety) is sufficiently addressed
SepConcern	Justification	Justification for the appropriateness of the separation of the demonstration concerning QM, SM, and TS
QualMng	Goal	*ToD* is safe with respect to the QM (Quality Management) concern
SafMng	Goal	*ToD* is safe with respect to the SM (Safety Management) concern
TechSaf	Goal	*ToD* is safe with respect to the TS (Technical Safety) concern

Instantiation Rule: An artefact *Case* (see Figure 51 and Figure 50) instantiates the *Overall Safety* pattern if:

- *Case* represents the overall claim in a top-down decomposable safety argumentation on the safety of a target system (the target is identified by the instantiation of *ToD*).
- *Case* represents the root node in a tree like presentation of the safety argument as indicated in Figure 51**.**
- Every element of the decomposable safety argument where *Case* is the root node, is traceable to a safety case node as indicated in Figure 51. A safety case node is instantiated by adapting the descriptions in column “Node Content Description” in Table 7 to the context that is addressed in order to define a structure as given in Figure 51.
- *Case* and the associated safety argumentation expresses the decomposition of a main claim, that a target system is sufficiently safe, for its intended purpose by a strategy of claiming sufficient quality management (the claim expressed in the node *QualMng*), safety management (the claim expressed in the node *SafMng*), and technical safety (the claim expressed in the node *TechSaf*). Note: No guidance is given in this pattern on how to decompose the safety argument parts represented by the claims *QualMng*, *SafMng* and *TechSaf* down to its supporting evidences, suitable patterns supporting such a decomposition should be conferred.

Related Patterns: The *Overall Safety* pattern is related to other patterns in the following manner:

- May be supported by other safety case patterns for detailing those parts that are not fully developed, e.g., the pattern *Technical Safety* may be used to as support on the issue of demonstrating that a system is technical safe and thereby detail the node *TechSaf* presented in Figure 51.
- May be used as support for the safety demonstration of a design derived from a design pattern.

A.10 TECHNICAL SAFETY

Name: Technical Safety

Pattern Signature: *Technical Safety* is defined with the signature illustrated in Figure 52.

In Figure 52, the following abbreviations are used for denoting the parameters of the pattern:

- *ToD* is short for Target of Demonstration.
- *Case* is short for Safety Case.
- *CoP* is short for Code of Practice.
- *CrRef* is short for Cross Reference.
- *ERE* is short for Explicit Risk Estimation.

Figure 52 Technical Safety – Pattern Signature

Intent: Provide a structure for the specification of a safety demonstration *Case* showing that a system *ToD* is safe by arguing that it is developed according to an accepted code of practice, is similar to an already accepted system, or risk being explicitly addressed and sufficiently reduced. The intent is to provide an argument structure for demonstration on sufficient technical safety. The pattern facilitates early identification of the set of different overall demonstration strategies that may be applied in order to provide confidence that a system is sufficiently safe.

Example: a system S is decomposed into the subsystems Sub 1 and Sub 2. Sub 1 is a system where showing compliance to a code of practice is a commonly accepted approach for safety demonstration. Sub 2 is a system includes novel technical solutions. Relying on a code of practice safety demonstration strategy of novel technical solutions may challenge our confidence as complying with a code is not necessarily sufficient. Therefore, for demonstration of Sub 2, a risk estimation strategy is chosen. The safety demonstration for system S consists of a combination of the safety demonstrations applied on the subsystems with the addition of the argumentation for this combination being sufficient.

Applicability: The *Technical Safety* pattern is suitable to apply in the following situations:

- At the initial stages of development in order to outline the main set of demonstration strategies to be applied for addressing technical safety concerns.
- Prior to project initiation as a means for stakeholders (e.g, vendor, customer) to reflect upon main challenges associated with the demonstration of technical safety and possible solutions.

Problem: The main aspects to be considered when providing a solution for the demonstration of technical safety are:

- *Implicit vs Explicit Safety Demonstration Strategy*: the necessary confidence in a safety demonstration may be provided by an implicit demonstration strategy, e.g., referring to the application of a code of practice that is known to mitigate certain risks. Another strategy is to compare the target of demonstration to a similar system that already has received approval. If an implicit demonstration strategy is not sufficient, an explicit demonstration strategy must be performed that requires information on the risks associated with the target.
- *Effectiveness*: it is important that the demonstration strategy is effective in providing the necessary safety demonstration. To develop safety related and safety critical systems is costly, mainly due to the activities required to assure and demonstrate that the system is sufficiently

safe. It is important for a vendor to choose strategies that minimise cost but provide required results.

- *Confidence*: it is important to choose a demonstration strategy that has the potential to convince some safety authority reviewing the evidences that sufficient safety is achieved. In a development project, the effectiveness of different demonstration strategies must be weighed against the estimated cost and the ability of building confidence.
- *Acceptance*: it is important that the choice of demonstration strategies are acceptable for all involved parties such that they represent an effective means for demonstrating safety and used in a manner that provide confidence that the system is safe.

Argument Structure Solution: Figure 53 and Table 8 together present the demonstration solution, and therefore should be read together. Figure 53 illustrates the decomposition of the safety argument in a tree structure using the GSN notation. Table 8 details the tree structure by defining the type and expected content of each node.

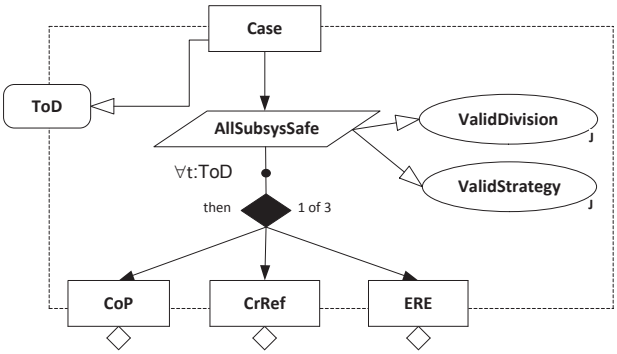


Figure 53 Technical Safety - Argument Structure

Table 8 Technical Safety - Argument Structure

Node	Node Type	Node Content Description
Case	Goal	*ToD* is safe
ToD	Context	Definition of parameter: ToD (Target of Demonstration)
AllSubsysSafe	Strategy	*ToD* is demonstrated safe by demonstrating that all subsystems *t* of *ToD* is safe
ValidDivision	Justification	Rationale for the division of *ToD* into subsystems is valid
ValidStrategy	Justification	Rationale for the use of correct safety demonstration strategy on each subsystem of *ToD* as a means to demonstrate that *ToD* is safe
CoP	Goal	Subsystem *t* of *ToD* is safe by means of Code of Practice
CrRef	Goal	Subsystem *t* of *ToD* is safe by means of Cross Reference
ERE	Goal	Subsystem *t* of *ToD* is safe by means of Explicit Risk Estimation

Associated with the contextual elements of the argument structure, the following is assumed:

- ToD: identifies the target system under consideration;

Instantiation Rule: An artefact *Case* (see Figure 53 and Figure 52) instantiates the *Technical Safety* pattern if:

- *Case* is a safety demonstration of a target system (identified by the instantiation of ToD).
- Every element of *Case* is traceable to a safety case node as indicated in Figure 53. A safety case node is instantiated by applying the descriptions in column “Node Content Description” in Table 8 to the context that is addressed in order to define a structure as given in Figure 53.
- *Case* and the associated safety argumentation expresses the decomposition of a main claim, that a target system is technically safe for its purpose by a strategy of claiming that all subsystems are sufficiently safe.
- Subsystems are claimed sufficiently safe by a strategy of reference to a code of practice, or cross-reference and comparison to a similar and proven system, or explicit risk estimation.

Related Patterns: The *Technical Safety* pattern is related to other patterns in the following manner:

- May be used for detailing the claims on technical safety in the *Overall Safety* pattern.
- The *Codes Of Practice* pattern may be used to address the CoP node of this pattern by referencing common development practices.
- The *Cross Reference* pattern may be used to address the CrRef node in this pattern by comparing a target to a similar reference system that is an assured system and thus provide an implicit safety demonstration.
- The *Explicit Risk Evaluation* pattern may be used to address the ERE node in this pattern by explicitly addressing risk associated with a target.

Known Uses: The pattern describes a structure for arguing safety with a focus on technical aspects that may be seen as a variant of Part 4 of the safety case required by the standard [EN 50129] on issues related to demonstration of technical safety

[EN50129] European Committee for Electrotechnical Standardization, Railway Applications – Communications, signalling and processing systems – Safety related electronic systems for signalling, EN 50129, 2003.

A.11 SAFETY REQUIREMENTS SATISFIED

Name: Safety Requirements Satisfied

Pattern Signature: *Safety Requirements Satisfied* is defined with the signature illustrated in Figure 54.

In Figure 54, the following abbreviations are used for denoting the parameters of the pattern:

- *ToD* is short for Target of Demonstration.
- *Req* is short for Requirements.
- *ReqSat* is short for Requirements Satisfied.
- *Case is short for Safety Case.*

Figure 54 Safety Requirements Satisfied - Pattern Signature

Intent: Provide a structure for safety demonstration *Case* that a system *ToD* is safe, by arguing that all safety requirements *Req* are satisfied.

Example: a system S shall satisfy the functional requirements R1 and R2, and safety requirements SR1 and SR2. In order to demonstrate that S is sufficiently safe for its intended purpose it is enough to demonstrate that SR1 and SR2 are satisfied. In order for system S to provide the intended function then R1 and R2 must be satisfied, but as these requirements do no impact safety they are not addressed in the safety demonstration.

Applicability: The *Safety Requirements Satisfied* pattern is intended for the following situation:

- When it is required to provide an explicit demonstration of the ability of a system to uphold safety invariants that are expressed as safety requirements.

Problem: The main aspects to be considered when providing a solution for the demonstration of safety are:

- *Completeness of specification*: it is necessary to provide confidence that the set of safety requirements is complete, i.e. contain all relevant requirements.
- *Correct demonstration approach*: it is necessary to provide confidence that for each requirement, the chosen approach for demonstrating that the requirement is satisfied is suitable.
- *Confirming evidences*: it is necessary to provide confidence that for each chosen demonstration approach, confirming evidences is available or will be provided.

Argument Structure Solution: Figure 55 and Table 9 together present the demonstration solution, and therefore should be read together. Figure 55 illustrates the decomposition of the safety argument in a tree structure using GSN notation. Table 9 details the tree structure by defining the type and expected content of each node.

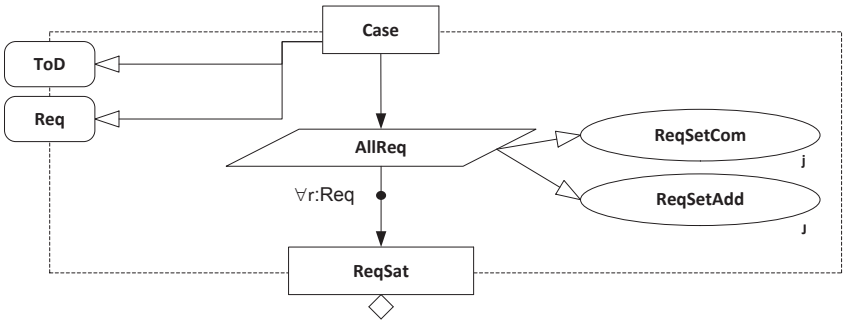


Figure 55 Safety Requirements Satisfied – Argument Structure

Associated with the contextual elements of the argument structure, the following is assumed:

- *ToD*: identifies the target system under consideration.
- *Req*: represents a set of safety requirements associated with the target system *ToD*.

Table 9 Safety Requirements Satisfied - Argument Structure Details

Node	Node Type	Node Content Description
Case	Goal	*ToD* satisfies *Req*
ToD	Context	Definition of *ToD*
Req	Context	Definition or *Req*
AllReq	Strategy	*ToD* is acceptably safe as all safety requirements are addressed and found satisfied
ReqSetCom	Justification	Justification for the requirements set *Req* being complete or contain all safety requirements that are relevant for *ToD*
ReqSetAdd	Justification	Justification for all elements of the set *Req* are accounted for. Elements may be addressed individually or in groups, any grouping of requirements needs to be justified.
ReqSat	Goal	The requirement *r,* element of *Req*, is satisfied

Instantiation Rule: A demonstration artefact *Case* (see Figure 55 and Figure 54) instantiates the *Safety Requirements Satisfied* pattern if:

- *Case* is a safety demonstration of a target system *ToD*.
- Every element of *Case* is traceable to a safety case node as indicated in Figure 55. A safety case node is instantiated by adapting the descriptions in column “Node Content Description” in Table 9 to the context that is addressed in order to define a structure as given in Figure 55.
- *Case* and the associated safety argumentation expresses the decomposition of a main claim, that a target system *ToD* is safe, by a strategy of demonstrating that all safety requirements *Req* associated with the system *ToD* are satisfied.

Related Patterns: The *Safety Requirements Satisfied* pattern is related to other patterns in the following manner:

- May be used to detail the ERE node of *Technical Safety* concerned with demonstrating that risk is explicitly addressed by demonstrating that the safety requirements, defined on the basis of risk assessment, are satisfied.

- May be used together with the *Establish System Safety Requirements*. *Establish System Safety Requirements* can be applied in order to derive the safety requirements. *Safety Requirements Satisfied* is then used to develop a demonstration that argues that identified requirements are satisfied.

APPENDIX B DETAILS ON PATTERN INSTANTIATION

B.1 RESULTS FROM INSTANTIATING FTA

Table 10 represents a fault tree expressed in a tabular form. The table should be read from top to bottom. The fault tree represents an assessment of the hazards that are relevant with respect to the intended operation of the 2TLCI system. The fault tree analysis presented in the following tables represents the results of applying the *FTA* pattern (defined in Appendix A.7). The analysis is based on the fault tree analysis presented in [21].

Table 10 FTA of Hazards

<table>
<tr><td colspan="4">Hazards</td></tr>
<tr><td colspan="4">OR</td></tr>
<tr><td rowspan="2">H1 – Derailment
(See Table 11)</td><td colspan="2">H2 – Collision train-train</td><td rowspan="2">H3 – Collision train-object
(See Table 28)</td><td rowspan="2">H4 – Level crossing accidents
(See Table 30)</td></tr>
<tr><td colspan="2">OR</td></tr>
<tr><td></td><td>H2.1 – Collision train-train at station
(See Table 16)</td><td>H2.2 – Collision train-train at line
(See Table 22)</td><td></td><td></td></tr>
</table>

Table 11 FTA of H1 – Derailment

<table>
<tr><td colspan="2">H1 - Derailment</td></tr>
<tr><td colspan="2">OR</td></tr>
<tr><td>H1.1 – Train derail in point</td><td>H1.2 – Train derail other place than at point</td></tr>
<tr><td>See Table 12</td><td>E1 – Wrong speed, lack of track maintenance, rock fall, projecting error, etc.</td></tr>
</table>

Table 12 FTA of H1.1 – Train derail in point

<table>
<tr><td colspan="5">H1.1 – Train derail in point</td></tr>
<tr><td colspan="5">AND</td></tr>
<tr><td rowspan="5">E2 –Train occupies track section at point</td><td colspan="4">OR</td></tr>
<tr><td rowspan="4">E3 – Point does not belong to train route</td><td colspan="3">AND</td></tr>
<tr><td rowspan="3">E4 – Point belongs to train route</td><td colspan="2">OR</td></tr>
<tr><td rowspan="2">E5 – Train does not have proceed aspect</td><td>AND</td></tr>
<tr><td>E6 – Train has proceed aspect</td><td>H1.1.A – Derailing in a point belonging to a train route with proceed aspect</td></tr>
<tr><td colspan="2"></td><td></td><td>See Table 13</td></tr>
</table>

Table 13 FTA of H1.1.A – Derailing in point belonging to train route with proceed aspect

<table>
<tr><td colspan="5">H1.1.A - Derailing in point belonging to train route with proceed aspect</td></tr>
<tr><td colspan="5">OR</td></tr>
<tr><td rowspan="5">E7 – Point is not controlled in correct position</td><td colspan="4">Derailing in point that is controlled in correct position</td></tr>
<tr><td colspan="4">AND</td></tr>
<tr><td rowspan="3">E8 – Point is controlled in correct position</td><td colspan="3">OR</td></tr>
<tr><td rowspan="2">E9 – Point position altered</td><td colspan="2">AND</td></tr>
<tr><td>E10 – Point position not altered</td><td>H.1.1.B – Train passes point at too high speed</td></tr>
<tr><td colspan="2"></td><td></td><td>See Table 14</td></tr>
</table>

Table 14 FTA of H1.1.1.B – Train passes point at too high speed

<table>
<tr><td colspan="5">H1.1.B – Train passes point at too high speed</td></tr>
<tr><td colspan="5">AND</td></tr>
<tr><td rowspan="5">E13 – Point is in a diverging position</td><td colspan="4">Train do not proceed slow (cautious)</td></tr>
<tr><td colspan="4">OR</td></tr>
<tr><td rowspan="3">E14 – Main signal at beginning of train route signals "Proceed"</td><td colspan="3">AND</td></tr>
<tr><td rowspan="2">E15 – Main signal does not signal "Proceed"</td><td>H1.1.C – Train driver disregard restrictive signal or respond to late</td><td rowspan="2">E16 – Derailment at point due to lack of braking</td></tr>
<tr><td>See Table 15</td></tr>
</table>

Table 15 FTA of H1.1.C – Train driver disregard restrictive signal or respond to late

<table>
<tr><td colspan="5">H1.1.C – Train driver disregard restrictive signal or respond to late</td></tr>
<tr><td colspan="5">OR</td></tr>
<tr><td rowspan="3">E17 – Distant signal does not show restrictive enough signal</td><td colspan="4">AND</td></tr>
<tr><td rowspan="2">E18 – Distant signal shows correct signal</td><td colspan="3">OR</td></tr>
<tr><td>Human error</td><td>Too short sighting distance to signal</td><td>Too bad sighting conditions</td></tr>
<tr><td></td><td>E19 – Human factors</td><td>E20 – Projecting error</td><td>E21 – Projecting error, vegetation, weather conditions, etc.</td></tr>
</table>

Table 16 FTA of H2.1 – Collision train-train at station

<table>
<tr><td colspan="4">H2.1 – Collision train-train at station</td></tr>
<tr><td colspan="4">OR</td></tr>
<tr><td>H2.1.1 – Collision incoming train against stationary train at station (See Table 17)</td><td>H2.1.2 – Collision incoming train against incoming train at station (See Table 19)</td><td>H2.1.3 - Collision incoming train against outgoing train at station (See Table 20)</td><td>H2.1.4 – Collision outgoing train against outgoing train at station (See Table 21)</td></tr>
</table>

Table 17 FTA - H2.1.1 – Collision incoming train against stationary train at station

<table>
<tr><td colspan="5">H2.1.1 – Collision incoming train against stationary train at station</td></tr>
<tr><td colspan="5">OR</td></tr>
<tr><td colspan="4">Collision incoming train with proceed aspect against stationary train at station</td><td>Collision incoming train without proceed aspect against stationary train at station</td></tr>
<tr><td colspan="4">OR</td><td rowspan="4">E26 – Incoming train passes main signal in "Stop"</td></tr>
<tr><td colspan="2">Collision incoming train with proceed aspect against stationary train at incoming train route</td><td colspan="2">Collision incoming train with proceed aspect against stationary train at the safety zone of the incoming train route</td><td>H2.1.1.A – Collision incoming train with proceed aspect against stationary train outside the incoming train route and its safety zone</td></tr>
<tr><td colspan="2">AND</td><td colspan="2">AND</td><td>See Table 18</td></tr>
<tr><td>E22 – Main signal (entry) signals proceed aspect</td><td>E23 – Another train present in train route</td><td>E24 – Main signal (entry) signals proceed aspect</td><td>E25 – Another train present in safety zone</td><td></td></tr>
</table>

Table 18 FTA of H2.2.1.A – Collision incoming train with proceed aspect against stationary train outside the incoming train route and its safety zone

<table>
<tr><td colspan="5">H2.2.1.A – Collision incoming train with proceed aspect against stationary train outside the incoming train route and its safety zone</td></tr>
<tr><td colspan="5">AND</td></tr>
<tr><td rowspan="3">E27 – Main signal (entry) signals proceed aspect</td><td colspan="4">OR</td></tr>
<tr><td colspan="2">AND</td><td colspan="2">AND</td></tr>
<tr><td>E28 – Point in incoming train route puts train onto another train route than the locked train routed</td><td>E29 – Other train occupies other train route that is not extension of the incoming train route and its safety zone</td><td>E30 – Incoming trains passes the incoming train route and its safety zone</td><td>E31 – Other train occupies train route in extension of the incoming train route and its safety zone</td></tr>
</table>

Table 19 FTA of H2.1.2 – Collision incoming train against incoming train at station

<table>
<tr><td colspan="6">H2.1.2 – Collision incoming train against incoming train at station</td></tr>
<tr><td colspan="6">OR</td></tr>
<tr><td colspan="5">Collision incoming train against incoming train at station, both trains has proceed aspect</td><td>Collision incoming train against incoming train at station, at least one train does not have proceed aspect</td></tr>
<tr><td colspan="5">OR</td><td rowspan="8">E39 – Incoming trains passes main signal in "Stop"</td></tr>
<tr><td>Collision incoming train against incoming train at same train route, both trains have proceed aspect</td><td colspan="4">Collision incoming train against incoming train with different train routes, both trains have proceed aspect</td></tr>
<tr><td rowspan="5">E32 – Identical to: Collision incoming train against stationary train at station</td><td colspan="4">AND</td></tr>
<tr><td>Both trains have proceed aspect</td><td colspan="3">OR</td></tr>
<tr><td rowspan="3">E33 – Both train routes are locked</td><td colspan="2">Collision incoming train against incoming train in shared part of train route</td><td>Collision incoming train against incoming train in the safety zone of one of the train routes</td><td>Collision incoming train against incoming train in shared safety zone of the train routes</td></tr>
<tr><td colspan="2">OR</td><td rowspan="2">E37 –Train route crosses safety zone of another locked train route</td><td rowspan="2">E38 – Train route share safety zone with another locked train route</td></tr>
<tr><td>E35 – Train route has shared part with another locked train route</td><td>E36 – Train route has shared part with another locked train route that has been released too early</td></tr>
</table>

Table 20 FTA of H2.1.3 – Collision incoming train against outgoing train at station

<table>
<tr><td colspan="6">H2.1.3 – Collision incoming train against outgoing train at station</td></tr>
<tr><td colspan="6">OR</td></tr>
<tr><td colspan="4">Collision incoming train against outgoing train at station, both trains have received proceed aspect</td><td colspan="2">Collision incoming train against outgoing train at station, at least one train has not received proceed aspect</td></tr>
<tr><td colspan="4">AND</td><td colspan="2">OR</td></tr>
<tr><td rowspan="2">Both trains have received proceed aspect</td><td colspan="2">OR</td><td rowspan="2">Collision incoming train against outgoing train in the safety zone of the incoming train route</td><td rowspan="3">E44 – Incoming train passes main signal in "Stop"</td><td rowspan="3">E45 – Outgoing train passes main signal in "Stop"</td></tr>
<tr><td colspan="2">Collision incoming train against outgoing train in shared part of train route</td></tr>
<tr><td rowspan="2">E40 – Both train routes are locked</td><td colspan="2">OR</td><td rowspan="2">E43 – Train route crosses safety zone of another locked train route</td></tr>
<tr><td>E41 – Train route has shared part with another locked train route</td><td>E42 – Train route has shared part with another locked train route that has been released to early</td></tr>
</table>

Table 21 FTA of H2.1.4 – Collision outgoing train against outgoing train at station

<table>
<tr><td colspan="5">H2.1.4 – Collision outgoing train against outgoing train at station</td></tr>
<tr><td colspan="5">OR</td></tr>
<tr><td>Collision outgoing train against outgoing train at station, at least one train has not received proceed aspect</td><td colspan="4">Collision outgoing train against outgoing train at station, both trains has received proceed aspect</td></tr>
<tr><td rowspan="4">E46 – At least one of the trains passes main signal in "Stop"</td><td colspan="4">AND</td></tr>
<tr><td>Both trains have received proceed aspect</td><td colspan="3">Collision outgoing train against outgoing train in shared part of train route</td></tr>
<tr><td rowspan="2">E47 – Both train routes are locked</td><td colspan="3">OR</td></tr>
<tr><td>Collision outgoing train against outgoing train in different outgoing train routes</td><td>Collision outgoing train against outgoing train in the same outgoing train route</td></tr>
<tr><td colspan="2"></td><td>E48 – Train route has shared part with another locked train route</td><td>E49 – Outgoing train route is released before first train has left the station</td></tr>
</table>

Table 22 FTA of H2.2 - Collision train-train at line

<table>
<tr><td colspan="5">H2.2 - Collision train-train at line</td></tr>
<tr><td colspan="5">OR</td></tr>
<tr><td>H2.2.1 – Train 1 at St. 1 has proceed aspect towards St. 2, Train 2 has left St. 2 but not arrived at St. 1. (See Table 23)</td><td>H2.2.2 – Train 1 at St. 1 has proceed aspect towards St. 2, Train 2 is at St. 2 with proceed aspect towards St. 1. (See Table 24)</td><td>H2.2.3 – Train 1 at St. 1 has proceed aspect towards St. 2, Train 2 is at St. 2 and receives proceed aspect towards St. 1. (See Table 25)</td><td>H2.2.4 – Train 1 at St. 1 receives proceed aspect towards St. 2, Train 2 at St. 2 receives proceed aspect towards St. 1. (See Table 26)</td><td>H2.2.5 – Train 1 at St. 1 has proceed aspect towards St. 2, Train 2 has passed main signal (exit) at St. 2 and is enroute towards St. 1 (See Table 27)</td></tr>
</table>

Table 23 FTA of H2.2.1

H2.2.1 – Train 1 at St. 1 has proceed aspect towards St. 2, Train 2 has left St. 2 but not arrived at St. 1			
AND			
Train 2 is en route towards station 2 or returns to station 1		Train 1 is at station 1 with proceed aspect	
		Conditions for receiving proceed aspect are satisfied	
AND		**AND**	
E50 – Train 2 has proceed aspect	E51– Train 2 occupies block line	E52 – Signal received from station 2 that line is vacant	E53 – The line is set in direction of station 2

Table 24 FTA of H2.2.2

H2.2.2 – Train 1 at St. 1 has proceed aspect towards St. 2, Train 2 is at St. 2 with proceed aspect towards St. 1			
AND			
Train 1 has proceed aspect		Train 2 has proceed aspect	
Conditions for receiving proceed aspect are satisfied		Conditions for receiving proceed aspect are satisfied	
AND		**AND**	
E54 – Signal received from station 2 that line is vacant	E55 – The line is set in direction of station 2	E56 – Signal received from station 1 that line is vacant	E57 – The line is set in direction of station 1

Table 25 FTA of H2.2.3

H2.2.3 - Train 1 at St. 1 has proceed aspect towards St. 2, Train 2 is at St. 2 and receives proceed aspect towards St. 1	
AND	
Train 1 has proceed aspect	Train 2 receives proceed aspect
E58 – Main signal (exit) signals proceed	E59 – Line is set in direction of towards station 1

Table 26 FTA of H2.2.4

H2.2.4 – Train 1 at St. 1 receives proceed aspect towards St. 2, Train 2 at St. 2 receives proceed aspect towards St. 1			
AND			
Train 1 at station 1 satisfies conditions for receiving proceed aspect		Train 2 at station 3 satisfies conditions for receiving proceed aspect	
AND		**AND**	
E60 – Line is in direction of station 2	E61 – The line is vacant	E62 – Line is in direction of station 1	E63 – The line is vacant

Table 27 FTA of H2.2.5

H2.2.5 – Train 1 at St. 1 has proceed aspect towards St. 2, Train 2 has passed main signal (exit) at St. 2 and is en route towards St. 1	
AND	
Train 2 had proceed aspect, the line was previously set in direction towards station 1	Train 1 has proceed aspect
AND	E66 – Line is set in direction of station 2
E64 – Train 2 has not arrived station 1	E65 – Line is released

Table 28 FTA of H3 - Collision train-object

<table>
<tr><td colspan="5">H3 – Collision train-object</td></tr>
<tr><td colspan="5">OR</td></tr>
<tr><td colspan="3">Collision at level crossing with level crossing interlocking</td><td>H3.1 – Collision related to work in/at track</td><td rowspan="4">E71 – Other conditions that might be prevented by signalling "Stop"</td></tr>
<tr><td colspan="3">AND</td><td rowspan="3">See Table 29</td></tr>
<tr><td rowspan="3">E67 – Road traffic is not blocked</td><td colspan="2">Train arrive at level crossing</td></tr>
<tr><td colspan="2">OR</td></tr>
<tr><td rowspan="2">E68 – Train do not pass main signal or distant signal that has dependency to level crossing interlocking</td><td>Train passes main signal that has a dependency to level crossing interlocking</td><td>Train passes distant signal that has a dependency to level crossing interlocking</td></tr>
<tr><td>E69 – Main signal signals proceed aspect</td><td>E70 – Distant signal signals proceed aspect</td></tr>
</table>

Table 29 FTA of H3.1 – Collision related to work in/at track

<table>
<tr><td colspan="3">H3.1 – Collision related to work in/at track</td></tr>
<tr><td colspan="3">OR</td></tr>
<tr><td colspan="2">Collision related to work in/at track between two stations</td><td>Collision related to work in/at track at station</td></tr>
<tr><td colspan="2">Proceed aspect for outgoing traffic is given from one of the stations</td><td>Proceed aspect is given onto a track circuit that is under work</td></tr>
<tr><td colspan="2">Outgoing train route is locked</td><td rowspan="3">E74 – Train route is locked that consists of a track section that is under work</td></tr>
<tr><td colspan="2">AND</td></tr>
<tr><td>E72 – Line is set in an outgoing direction from a station</td><td>E73 – Line is set in incoming direction to neighbouring station</td></tr>
</table>

Table 30 FTA of H4 – Level Crossing Accident

<table>
<tr><td colspan="5">H6 – Level Crossing Accident</td></tr>
<tr><td colspan="5">OR</td></tr>
<tr><td colspan="4">Collision at level crossing with level crossing interlocking</td><td rowspan="5">E79 – Other accidents</td></tr>
<tr><td colspan="4">AND</td></tr>
<tr><td colspan="3">Train arrive at level crossing</td><td>Road traffic is not blocked</td></tr>
<tr><td colspan="3">OR</td><td rowspan="2">E78 – Level crossing equipment (gates and signals) do not signal “Stop”</td></tr>
<tr><td>E75 – Train do not pass main signal or distant signal that has a dependency to level crossing interlocking</td><td>E76 – Train passes main signal that has a dependency to level crossing interlocking</td><td>E77 – Train passes distant signal that has a dependency to level crossing interlocking</td></tr>
</table>

B.2 Results from Instantiating Hazard Analysis

Table 31 represents the hazard log provided as a result of instantiating the *Hazard Analysis* pattern. The table provides traceability between hazards and their potential causes that are relevant with respect to the 2TLCI system (excluding potential causes of error associated with e.g. projecting error, human errors, etc. that are beyond the conceptualisation of the 2TLCI system).

In the column “Potential Cause” of Table 31, potential causes of hazards are given by referencing the identifiers for events found in the fault trees (documented in Appendix B.1). The tables where identified events may be found are given in brackets. An “and” dependency between events in the fault tree are written by separating the identifiers for events with the sign “*”, and an “or” dependency in the fault trees are resolved by adding rows to Table 31.

Table 31 Hazard Log

Hazard: H1 - Derailment			
Cs. ID	**Potential Cause**	**Problem description**	**Mitigating Actions**
C1	E2*E3 (Table 12)	Train occupies a point that does not belong to its train route.	Assure that a proceed aspect is not given unless all conditions for locking the train route are satisfied. Assure that a train route may not be locked if a track section with a point in the train route belongs to another train route. Assure that a train route may not be locked unless all track sections belonging to the train route are vacant. Assure that a train route may not be locked unless all points belonging to the train route are positioned correctly.
C2	E2*E4*E5 (Table 12)	Point belongs to the train route but the train has no proceed aspect.	No mitigation defined – the train has no proceed aspect. Automatic train control as mitigation is out of scope.
C3	E2*E4*E6*E7 (Table 12, Table 13)	Train has proceed aspect on a train route but the point is not in the correct position.	Assure that a train route may not be locked unless points belonging to the train route are controlled in correct position.
C4	E2*E4*E6*E8*E9 (Table 12, Table 13)	Train has proceed on a train route where a point is initially in correct position but where the position is altered.	Assure that the position of a point may only be altered if conditions for altering its position are satisfied. Assure that the position of a point may not be altered if it belongs to a locked train route.
C5	E2*E4*E6*E8*E10 *E13*E14 (Table 12, Table 13,Table 14)	Train has proceed aspect on a train route where a point is controlled correct in a diverging position but main signal does not signal proceed slow.	Assure that a main signal for a train route that has a point positioned to a diverging track does not signal less restrictive signal than proceed slow.
C6	E2*E4*E6*E8*E10 *E13*E15* *E16* E17 (Table 12, Table 13,Table 14,Table 15)	Train has proceed aspect on a train route where a point is controlled correct in a diverging position, main signal does not signal proceed but distant signal does not give correct restrictive signal.	Assure that a distant signal shows at least as restrictive a signal aspect as its belonging main signal.
Hazard: H2 – Collision train-train			
Cs. ID	**Potential Cause**	**Problem description**	**Mitigating Actions**
C7	E22*E23 (Table 17)	Collision when main (entry) signal signals proceed when another train is present in train route.	Assure that a proceed aspect for a main signal belonging to a train route may only be given once the conditions for locking the train route are satisfied. Assure that a train route may not be locked unless all track sections in the train route are vacant.
C8	E24*E25 (Table 17)	Collision as main (entry) signal signals proceed when another train present in safety zone.	Assure that train route may not be locked unless its safety zone is vacant.
C9	E26 (Table 17)	Incoming train passes signal in ‘stop’.	No mitigation defined – the train has no proceed aspect. Automatic train control as mitigation is out of scope.
C10	E27*E28*E29 (Table 18)	Point is positioned onto wrong train route where there is a train.	Assure that a train route may not be locked unless all points belonging to the train route is positioned correctly.

C11	E27*E30*E31 (Table 18)	Train has proceed aspect but passes its train route or safety zone where there is a train.	No mitigation defined – the train should stop at end of train route. Automatic train control as mitigation is out of scope.
C12	E33*E35 (Table 19)	Two train routes with shared pars are both locked.	Assure that a proceed aspect for a main signal belonging to a train route may only be given once the conditions for locking the train route are satisfied. Assure that a train route may not be locked if it has a part that is shared with another locked train route.
C13	E33*E36 (Table 19)	Two train routes with shared parts and proceed as one route is released too early.	Assure that an incoming train route that is locked may not be released before the train has arrived the station.
C14	E33*E37 (Table 19)	Two locked train routes where one train crosses safety zone of other train route.	Assure that a train route may not be locked if there is another train route locked over its safety zone.
C15	E37*E38 (Table 19)	Two locked train routes with same safety zone.	Assure that a train route may not be locked if there is another train route locked over its safety zone.
C16	E39 (Table 19)	Train passes signal in stop.	Requires no mitigation – the train shall stop when signal is stop. Automatic train control as mitigation is out of scope.
C17	E40*E41 (Table 20)	Two train routes are locked and has shared parts.	Assure that a train route may not be locked if it has a part that is shared with another locked train route.
C18	E40*E42 (Table 20)	Two train routes have shared parts and the second route gets locked as the first is released too early.	Assure that an outgoing train route that is locked may not be released before the train has left the station
C19	E40*E43 (Table 20)	Two locked train routes where one train crosses safety zone of other train route.	Assure that a train route may not be locked if other train routes are locked over its safety zone.
C20	E44 (Table 20)	Train passes signal in ‘stop’.	Requires no mitigation – the train has no proceed aspect. Automatic train control as mitigation is out of scope.
C21	E45 (Table 20)	Train passes signal in ‘stop’.	Requires no mitigation – the train has no proceed aspect. Automatic train control as mitigation is out of scope.
C22	E46 (Table 21)	Collision at station as at least one train passes ‘stop’.	Requires no mitigation – the train has no proceed aspect. Automatic train control as mitigation is out of scope.
C23	E47*E48 (Table 21)	Collision at station as two train routes are locked where these have a shared part.	Assure that a train route may not be locked if it has a part that is shared with another locked train route
C24	E47*E49 (Table 21)	Collision at station as two train routes are locked due to one released before train has left the station.	Assure that an outgoing train route that is locked may not be released before the train has left the station.
C25	E50*E51*E52*E53 (Table 23)	Collision at line due to proceed is given although a train occupies line.	Assure that a proceed aspect for a main signal, belonging to a train route, may only be given once the conditions for locking the train route are satisfied. Assure that a train route may not be locked unless all track sections in the train route are vacant.
C26	E54*E55*E56*E57 (Table 24)	Collision at line due to proceed given to both stations.	Assure that an outgoing train route may not be locked unless the line is vacant. Assure that an outgoing train route may not be locked unless the line is set in the direction from the starting point of the train route.
C27	E58*E59 (Table 25)	Collision at line due to one train gets line set and receives proceed aspect in opposite direction as a train that already has proceed aspect.	Assure that the line may only be set in the direction of an arrival station if the main signals (exit) of the arrival station shows ‘stop’.
C28	E60*E61*E62*E63 (Table 26)	The line is vacant and two trains at different stations satisfy conditions for receiving proceed.	Requires no mitigation – a line may only be set in one direction.

C29	E64*E65*E66 (Table 27)	Collision at line as the line is released although the first train has not arrived at its arrival station.	Assure that an incoming train route that is locked may not be released before the train has arrived the station. Assure that an outgoing train route may not be locked unless the line is set in the direction from the train routes starting point. Assure that an outgoing train route may not be locked unless the line is vacant.
Hazard: H3 – Collision train-object			
Cs. ID	**Potential Cause**	**Problem description**	**Mitigating Actions**
C30	E67*E68 (Table 28)	Road traffic is not blocked and train arrive at level crossing without passing main signal or distant signal with dependency to interlocking.	Requires no mitigation – this may be a projecting error or the distance between level crossing and any distant signal or main signal makes it impractical to define dependency.
C31	E67*E69 (Table 28)	Main signal with dependency to interlocking signals proceed and road traffic is not blocked.	Assure that a proceed aspect for a main signal that has a dependency to a level crossing interlocking is only given if road traffic is blocked.
C32	E67*E70 (Table 28)	Distant signal with dependency to interlocking signals proceed and road traffic is not blocked.	Assure that a proceed aspect for a distant signal that has a dependency to a level crossing interlocking is only given if road traffic is blocked.
C33	E71 (Table 28)	Event that might be prevented by signalling ‘stop’.	Requires no mitigation – there is no other condition identified that requires ‘stop’.
C34	E72*E73 (Table 29)	Collision related to work in/at line where the line is set outgoing from one station and incoming to neighbouring station.	Assure that a train route may not be locked if a track section in the train route is blocked. Assure that no work is allowed on a track section unless it is blocked.
C35	E74 (Table 29)	Collision related to work in/at track at station where a track route consisting of a track under work has been locked.	Assure that no work is allowed on a track section unless it is blocked.
Hazard: H4 – Level crossing accident			
Cs. ID	**Potential Cause**	**Problem description**	**Mitigating Actions**
C36	E75*E78 (Table 30)	Road traffic is not blocked and train do not pass main signal or distant signal with dependency to interlocking.	Requires no mitigation – this may be a projecting error or the distance between level crossing and any distant signal or main signal makes it impractical to define dependency.
C37	E76*E78 (Table 30)	Road traffic is not blocked and main signal with dependency to interlocking signals proceed.	Assure that a proceed aspect for a main signal that has a dependency to a level crossing interlocking is not given unless road traffic is blocked.
C38	E77*E78 (Table 30)	Road traffic is not blocked and distant signal with dependency to interlocking signals proceed.	Assure that a proceed aspect for a distant signal that has a dependency to a level crossing interlocking is not given unless road traffic is blocked.

B.3 Results from Instantiating Risk Analysis

According to the *SIL Classification pattern*, the first step is to define tolerable hazard rates (THR) for system functions. The assumed THR for the hazards that are found relevant with respect to the operation of the 2TLCI system is defined in Table 32. We have assumed a THR of 10^{-9} per hour per function as each hazard may in a worst-case scenario lead to multiple fatalities.

In Table 32, the first two columns identify hazards and THR (defined according to step 1 of *SIL Classification*). The column "Functions" list relevant system functions that are associated with the hazard. The "SIL" column lists the classification of the function into a SIL level based on the THR for the hazard and the potential contribution of the function (assumed "likely" here as we have not performed a quantitative estimation) to the occurrence of the hazard (defined according to step 3 of *SIL Classification* pattern).

Table 32 Hazard Log Extension - Hazards, THR, Functions and SIL

Hazard	THR (Tolerable Hazard Rate) per hour per function	Functions		SIL
		Short description	**Ref.** (Table 31)	
H1 – Derailment	10^{-9}	F1 – Lock a train route	C1, C3	SIL4
		F2 – Control point is free	C1	SIL4
		F3 – Control point in correct position	C1,C3	SIL4
		F4 – Set position of point	C4	SIL4
		F5 – Signal proceed aspect	C3, C5, C6	SIL4
H2 – Collision train-train	10^{-9}	F1 – Lock a train route	C7, C12, C14, C15, C17, C19, C23, C26, C29	SIL4
		F5 – Signal proceed aspect	C7	SIL4
		F6 – Control track section is vacant	C7, C8	SIL4
		F3 – Control point in correct position	C10	SIL4
		F7 – Release a locked train route	C13, C18, C24	SIL4
		F8 – Set direction of line	C27	SIL4
		F9 – Release the line (set to neutral position)	C29	SIL4
H3 – Collision train-object	10^{-9}	F1 – Lock a train route	C31, C34, C35	SIL4
		F5 – Signal proceed aspect	C31, C32	SIL4
		F10 – Block road traffic	C31, C32	SIL4
		F11 – Block track section	C34, C35	SIL4
H6 – Level crossing accidents	10^{-9}	F10 – Block road traffic	C37, C38	SIL4
		F5 – Signal proceed aspect	C37, C38	SIL4

Every hazard listed in Table 32 is given a THR that in the SIL table of the pattern *SIL Classification* maps to a SIL4 level for associated system functions. Instead of identifying and classifying every system function, Table 32 identifies the main functions and their SIL. We simplify the classification procedure and classify the system as a SIL4 system. A complete detailing of SIL for every function could be a benefit if it is possible to modularise the system such that there is a sufficient separation between functions with different SIL.

The SIL4 classification implies that a compliance argument must be given that provides assurance that the system is developed according to appropriate practices. The appropriate practices associated with the development of SIL4 systems are stated in standards. In the context of the railway case the practices are defined by the requirements given in [13], [14], and [15] that applies to SIL4 systems.

B.4 Results from Instantiating Establish System Safety Requirements

The safety requirements defined in Table 33 were established on the basis of the proposed mitigations of hazards collected in the hazard log. The hazard log is presented in Table 31 in Appendix B.2. Potential mitigations for hazards are given in the column named “Mitigating Actions” in Table 31.

The safety requirements in Table 33 have unique identifiers, which are given in column “ID”. Each safety requirement is traceable to the respective mitigated hazards by the references provided in the column “Mitigates”.

Table 33 Safety Requirements

ID	Requirement	Mitigates (Ref. Table 31)
SR.1	A proceed aspect for a main signal belonging to a train route may only be given once the conditions for locking the train route are satisfied	C1, C7, C12, C25
SR.2	A train route may not be locked unless all points belonging to the train route are positioned correctly	C1, C10
SR.3	A train route may not be locked if a track section in the train route belongs to another train route	C1
SR.4	A train route may not be locked unless points belonging to the train route are controlled in correct position	C3
SR.5	A points position may only be altered if conditions for altering its position are satisfied	C4
SR.6	A points position may not be altered if it belongs to a locked train route or if it belongs to a track section that is not vacant	C4
SR.7	A main signal for a train route that has a point positioned to a diverging track shall not signal less restrictive than proceed slow	C5
SR.8	A distant signal shall show at least as restrictive aspect as its belonging main signal	C6
SR.9	A train route may not be locked unless all track sections in the train route are vacant	C7, C25
SR.10	A train route may not be locked unless its safety zone is vacant	C8
SR.11	A train route may not be locked if it has a part that is shared with another locked train route	C12, C17, C23
SR.12	An incoming train route that is locked may not be released before the train has arrived the station	C13, C29
SR13	A train route may not be locked if there is another train route locked over its safety zone	C14, C15,C19
SR.14	An outgoing train route that is locked may not be released before the train has left the station	C18, C24
SR.15	An outgoing train route may not be locked unless the line is set in the direction from the train routes starting point	C26, C29
SR.16	An outgoing train route may not be locked unless the line is vacant	C26, C29
SR.17	The line may only be set in the direction of an arrival station if the main signals (exit) of the arrival station shows ‘Stop’	C27
SR.18	In order to signal proceed aspect for a main signal that has a dependency to a level crossing interlocking, the road traffic must be blocked	C31, C37
SR.19	In order to signal proceed aspect for a distant signal that has a dependency to a level crossing interlocking, the road traffic must be blocked	C32, C38
SR.20	No work is allowed on a track section unless it is blocked	C34, C35
SR.21	A train route may not be locked if a track section in the train route is blocked	C34

Chapter 12

Paper 4: An Analytic Evaluation of the SaCS Pattern Language – Including Explanations of Major Design Choices

An Analytic Evaluation of the SaCS Pattern Language – Including Explanations of Major Design Choices

André Alexandersen Hauge
Institute for energy technology, Halden, Norway
University of Oslo, Norway
andre.hauge@hrp.no

Ketil Stølen
SINTEF ICT, Oslo, Norway
University of Oslo, Norway
ketil.stolen@sintef.no

***Abstract*—In this paper, we present an analytic evaluation of the Safe Control Systems (SaCS) pattern language for the development of conceptual safety designs. By a conceptual safety design we mean an early stage specification of system requirements, system design, and safety case for a safety critical system. The SaCS pattern language may express basic patterns on different aspects of relevance for conceptual safety designs. SaCS may also be used to combine basic patterns into composite patterns. A composite pattern may be instantiated into a conceptual safety design. A framework for evaluating modelling languages is used to conduct the evaluation. The quality of a language is within the framework expressed by six appropriateness factors. A set of requirements is associated with each appropriateness factor. The extent to which these requirements are fulfilled are used to judge the quality. We discuss the fulfilment of the requirements formulated for the SaCS language on the basis of the theoretical, technical, and practical considerations that were taken into account and shaped the SaCS language.**

Keywords–pattern language, analytic evaluation, design conceptualisation, safety.

I. Introduction

A pattern describes a particular recurring problem that arises in a specific context and presents a well-proven generic scheme for its solution [1]. A pattern language is a language for specifying patterns making use of patterns from a vocabulary of existing patterns and defined rules for combining these [2]. A safety critical system [3] is a system *"whose failure could result in loss of life, significant property damage, or damage to the environment"*. With a conceptual safety design we mean an early stage specification of system requirements, system design, and safety case for a safety critical system. The Safe Control Systems (SaCS) pattern language has been designed to facilitate the specification of patterns to support the development of conceptual safety designs. The intended users of SaCS are system engineers, safety engineers, hardware and software engineers.

This paper conducts an analytic evaluation of the suitability of the SaCS pattern language for its intended task. A framework for analysing languages known as the semiotic quality framework (SEQUAL) [4] is used as a basis for the evaluation. The appropriateness of a language for its intended task is in the framework characterised by six appropriateness factors [4]: domain, modeller, participant, comprehensibility, tool, and organisational. A set of requirements is presented for each appropriateness factor in order to characterise more precisely what is expected from our language in order to be appropriate. The requirements represent the criteria for judging what is appropriate of a language for conceptual safety design, independent of SaCS being appropriate or not. We motivate our choices and discuss to what extent the requirements are fulfilled.

The remainder of this article is structured as follows: Section II provides a short introduction to the SaCS pattern language. Section III discusses evaluation approaches and motivates the selection of SEQUAL. Section IV motivates the selection of requirements and conducts an evaluation of the SaCS language with respect to these requirements for each appropriateness factor. Section V presents related work on pattern-based development. Section VI draws the conclusions.

II. Background on The SaCS Pattern Language

Fig. 1 defines a composite pattern according to the syntax of SaCS [5]. The composite described in Fig. 1 is named *Safety Requirements* and consists of the basic patterns *Hazard Analysis*, *Risk Analysis*, and *Establish System Safety Requirements*. The basic patterns are specified separately in a structured manner comparable to what can be found in the literature [1][2][6][7][8][9][10][11] on patterns.

In Fig. 1, the horizontal line separates the declaration part of the composite pattern from its content. The icon placed below the identifier *Safety Requirements* signals that this is a composite pattern. Every pattern in SaCS is parametrised. An input parameter represents the information expected to be provided when applying a pattern in a context. An output parameter represents the expected outcome of applying a pattern in a given context. The inputs to *Safety Requirements* are listed inside square brackets to the left of the icon, i.e., *ToA* and *Haz*. The arrow pointing towards the brackets symbolises input. The output of the pattern is also listed inside square brackets, but on the right-hand side of the icon, i.e., *Req*. The arrow pointing away from the brackets symbolises output. An icon placed adjacent to a parameter identifier denotes its type. The parameters *ToA*, *Haz*, *HzLg*, and *Risks* in Fig. 1 are of type *documentation*, while *Req* is of type *requirement*. The inputs and outputs of a composite are always publicly accessible.

 ISBN: 978-1-61208-343-8

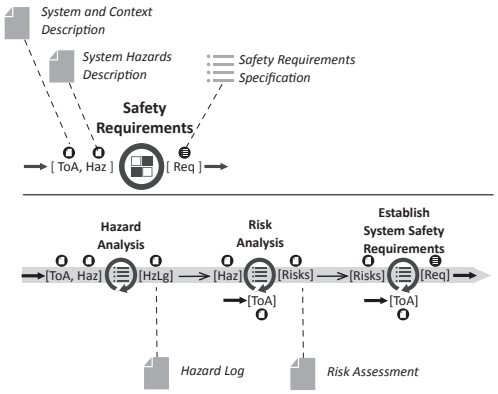


Figure 1. A composite pattern named Safety Requirements

A particular instantiation of a parameter is documented by a relation that connects a parameter with its associated development artefact. In Fig. 1, a grey icon placed adjacent to an identifier of a development artefact classifies what kind of artefact that is referenced. A dotted drawn line connecting a parameter with an artefact represents an *instantiates* relation. Instantiations of parameters expressed in Fig. 1 are:

- the document artefact *System and Context Description* instantiates *ToA*.
- the document artefact *System Hazards Description* instantiates *Haz*.
- the requirement artefact *Safety Requirements Specification* instantiates *Req*.
- the document artefact *Hazard Log* instantiates *HzLg*.
- the document artefact *Risk Assessment* instantiates *Risks*.

A one-to-many relationship exists between inputs in the declaration part of a composite and similarly named inputs with public accessibility (those pointed at by fat arrows) in the content part. The relationship is such that when *ToA* of *Safety Requirements* is instantiated (i.e., given its value by the defined relation to *System and Context Description*) then every correspondingly named input parameter contained in the composite is also similarly instantiated. A one-to-one relationship exists between an output parameter in the declaration part of a composite and a correspondingly named output parameter with public accessibility (those followed by a fat arrow) in the content part. The relationship is such that when *Req* of *Establish System Safety Requirements* is produced then *Req* of *Safety Requirements* is similarly produced.

The arrows (thin arrows) connecting basic patterns in the content part of *Safety Requirements* represent two instances of an operator known as the *assigns* relation. The *assigns* relations within *Safety Requirements* express that:

- The output *HzLg* of the pattern *Hazard Analysis* is assigned to the input *Haz* of the pattern *Risk Analysis*.
- The output *Risks* of the pattern *Risk Analysis* is assigned to the input *Risks* of the pattern *Establish System Safety Requirements*.

That the three basic patterns are process patterns follows from the icon below their respective identifiers. There are six different kinds of basic patterns in SaCS, each represented by a specific icon.

III. Evaluation framework

Mendling et al. [12] describe two dominant approaches in the literature for evaluating the quality of modelling approaches: (1) top-down quality frameworks; (2) bottom-up metrics that relate to quality aspects. The most prominent top-down quality framework according to [12] is SEQUAL [4][13][14]. The framework is based on semiotic theory (the theory of signs) and is developed for evaluating the quality of conceptual models and languages of all kinds. Moody et al. [15] report on an empiric study involving 194 participants on the use of SEQUAL and concludes that the study provides strong support for the validity of the framework. Becker et al. [16] present a guideline-based approach as an alternative to SEQUAL. It addresses the six factors: correctness, clarity, relevance, comparability, economic efficiency, and systematic design. Mendling et al. [12] also discuss a number of bottom-up metrics approaches. Several of these contributions are theoretic without empirical validation according to the authors.

We have chosen to apply the SEQUAL framework for our evaluation as it is a general framework applicable to different kinds of languages [4] whose usefulness has been confirmed in experiments [15]. Furthermore, an analytic evaluation is preferred over a metric-based approach due to project limitations. An analytic evaluation is also a suitable complement to the experience-based evaluations of SaCS presented in [17][18].

The appropriateness of a modelling language for a specific task is in SEQUAL related to the definition of the following sets: the set of goals G for the modelling task; its domain D in the form of the set of all statements that can be stated about the situation at hand; the relevant knowledge of the modeller Km and other participants Ks involved in the modelling task; what persons involved interpret the models to say I; the language L in the form of the set of all statements that can be expressed in the language; relevant tool interpretation T of the models; and what is expressed in the models M.

Fig. 2 is adopted from [13] and illustrates the relationships between the different sets in SEQUAL. The quality of a language L is expressed by six appropriateness factors. The quality of a model M is expressed by nine quality aspects.

In the following, we will not address the different quality aspects of a model M but rather address the quality of the SaCS pattern language.

The appropriateness factors indicated in Fig. 2 are related to different properties of the language under evaluation. The appropriateness factors are [4]:

- *Domain appropriateness*: the language should be able to represent all concepts in the domain.

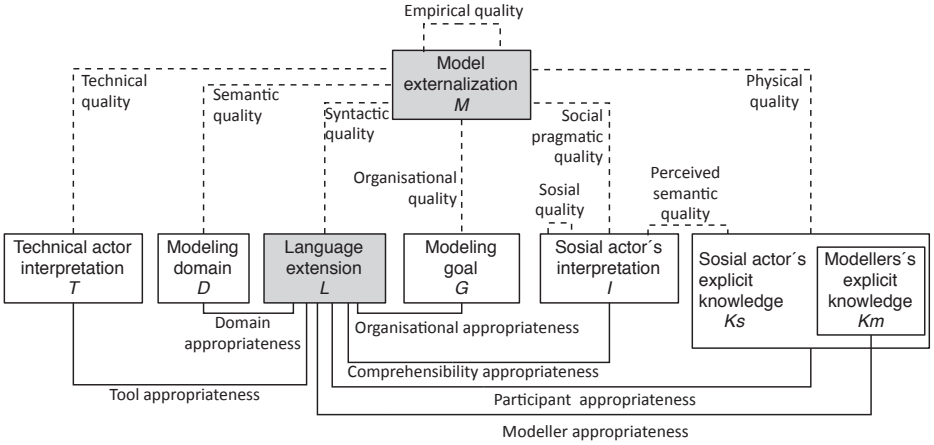


Figure 2. The quality framework (adopted from [19])

- *Modeller appropriateness*: there should be no statements in the explicit knowledge of the modeller that cannot be expressed in the language.
- *Participant appropriateness*: the conceptual basis should correspond as much as possible to the way individuals who partake in modelling perceive reality.
- *Comprehensibility appropriateness*: participants in the modelling should be able to understand all the possible statements of the language.
- *Tool appropriateness*: the language should have a syntax and semantics that a computerised tool can understand.
- *Organisational appropriateness*: the language should be usable within the organisation it targets such that it fits with the work processes and the modelling required to be performed.

A set of requirements is associated with each appropriateness factor. The extent to which the requirements are fulfilled are used to judge the quality of the SaCS pattern language for its intended task. The requirements are defined on the basis of requirements found in the literature on SEQUAL [4].

IV. THE EVALUATION

A necessary step in the application of SEQUAL [4][13] is to adapt the evaluation to account for the modelling needs. This amounts to expressing what the different appropriateness factors of the framework represent in the particular context of the evaluation in question. In particular, the modelling needs are detailed by the definition of a set of criteria for each of the appropriateness factors.

Table I introduces the criteria for evaluating the suitability of the SaCS pattern language for its intended task. In the first column of Table I, the two letters of each requirement identifier identify the appropriateness factor addressed by the requirement, e.g., DA for Domain Appropriateness.

TABLE I. OVERVIEW OF EVALUATION CRITERIA

ID	Requirement
DA.1	The language must include the concepts representing best practices within conceptual safety design.
DA.2	The language must support the application of best practices within conceptual safety design.
MA.1	The language must facilitate tacit knowledge externalisation within conceptual safety design.
MA.2	The language must support the modelling needs within conceptual safety design.
PA.1	The terms used for concepts in the language must be the same terms used within safety engineering.
PA.2	The symbols used to illustrate the meaning of concepts in the language must reflect these meanings.
PA.3	The language must be understandable for people familiar with safety engineering without specific training.
CA.1	The concepts and symbols of the language should differ to the extent they are different.
CA.2	It must be possible to group related statements in the language in a natural manner.
CA.3	It must be possible to reduce model complexity with the language.
CA.4	The symbols of the language should be as simple as possible with appropriate use of colour and emphasis.
TA.1	The language must have a precise syntax.
TA.2	The language must have a precise semantics.
OA.1	The language must be able to express the desired conceptual safety design when applied in a safety context.
OA.2	The language must ease the comprehensibility of best practices within conceptual safety design for relevant target groups like system engineers, safety engineers, hardware and software engineers.
OA.3	The language must be usable without the need of costly tools.

The different appropriateness factors are addressed successively in Section IV-A to Section IV-F according to the order in Table I. Each requirement from Table I is discussed. A requirement identifier is presented in a **bold** font when first introduced in the text followed by the associated requirement and an evaluation of the extent to which the requirement is fulfilled by SaCS.

A. Domain appropriateness

DA.1 The language must include the concepts representing best practices within conceptual safety design.

In the SaCS language, there are currently 26 basic patterns [17][18] on different concepts within conceptual safety design.

 ISBN: 978-1-61208-343-8

Each pattern may be referenced by its unique name. Three of the currently available basic patterns are referenced in Fig. 1 and are named *Hazard Analysis*, *Risk Analysis* and *Establish System Safety Requirements*.

Fig. 3 presents the icons used for basic SaCS patterns and indicates a categorisation. The three icons to the left are used for categorising patterns providing development guidance with a strong processual focus. The three icons to the right are used for categorising patterns providing development guidance with a strong product focus. Different kinds of patterns express different concepts and best practices within development of safety critical systems. The combined use of patterns from different categories facilitates development of conceptual safety designs.

Figure 3. Icons for the different kinds of basic pattern references

Habli and Kelly [20] describe the two dominant approaches in safety standards for providing assurance of safety objectives being met. These are: (1) the process-based approach; (2) the product-based approach. Within the process-based approach, safety assurance is achieved on the basis of evidence from the application of recommended or mandatory development practices in the development life cycle. Within the product-based approach, safety assurance is achieved on the basis of product specific evidences that meet safety requirements derived from hazard analysis. The practice within safety standards as described above motivate our categorisation into the process assurance and the product assurance pattern groups.

The safety property of a system is addressed on the basis of a demonstration of the fulfilment of safety objectives. Seven nuclear regulators [21] define a safety demonstration as *"a set of arguments and evidence elements that support a selected set of dependability claims - in particular the safety - of the operation of a system important to safety used in a given plant environment"*. Although it is the end system that is put into operation, evidences supporting safety claims are produced throughout the system life cycle and need to be systematically gathered from the very beginning of a development project [21]. The safety case approach represents a means for explicitly presenting the structure of claims, arguments, and evidences in a manner that facilitates evaluation of the rationale and basis for claiming that safety objectives are met. The safety case approach is supported by several authors [10][20][21][22]. What is described above motivates the need for patterns supporting safety case specification in addition to patterns on requirements elicitation and system design specification.

As indicated above, in the design of the SaCS pattern language we have as much as possible selected keywords and icons in the spirit of leading literature within the area. This indicates that we at least are able to represent a significant part of the concepts of relevance for conceptual safety design.

DA.2 The language must support the application of best practices within conceptual safety design.

Safety standards [23] may demand a number of activities to be performed in which certain activities must be applied in a specific sequence. Safety standards [23] may also describe the expected inputs and outputs of different activities and in this sense state what is the expected content of deliverables that allows a transition from one activity to the next. According to Krogstie [4], the main phenomena in languages that accommodate a behavioural modelling perspective are states and transitions between states. In this sense, the language should support the modelling of the application of best practices according to a behavioural modelling perspective.

Fig. 4 presents the icons for the different kinds of parameters and artefact references in SaCS. The *documentation parameter* and the *documentation artefact reference* types (represented visually by the icons presented in Fig. 4) are defined in order to allow a generic classification of parameters and artefacts that may not be classified as requirement, design, or safety case. An example may be the result of risk analysis that is an intermediate result in conceptual safety design and an input to an activity on the specification of safety requirements [23][24]. The process of deriving safety requirements on the basis of an assessment of hazards is expressed by a chain of patterns as presented in Fig. 1. The outcome of applying the last pattern in the chain is a requirements specification. The last pattern cannot be applied before the required inputs are produced.

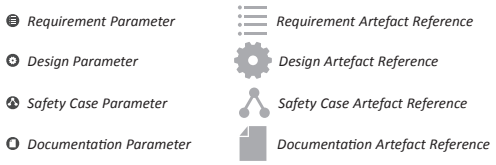


Figure 4. Icons for the different kinds of parameters and artefact references

Fig. 5 presents the symbolic representation of the different relations in SaCS. Relations define transitions between patterns or dependencies between elements within a composite pattern definition. The reports [17][18] define the concepts behind the different relations and exemplify the practical use of all the concepts in different scenarios. Fig. 1 is explained in Section II and exemplify a composite pattern containing five instances of the *instantiates* relation and two instances of the *assigns* relation.

Figure 5. Symbols for the different kinds of relations

The need for the different relations presented in Fig. 5 is motivated by the practices described in different standards and guidelines, e.g., IEC 61508 [23], where activities like hazard identification and hazard analysis are required to be performed sequentially and where the output of one activity is assigned as input to another activity. Thus, we need a concept of assignment. In SaCS, this is defined by an *assigns*

 ISBN: 978-1-61208-343-8

relation between patterns. When performing an activity like hazard analysis, the results from the application of a number of methods may be combined and used as input. Two widely known methods captured in two different basic SaCS patterns are Failure Modes and Effects Analysis (FMEA) and Fault Tree Analysis (FTA). A concept for combining results is needed in order to model that the results from applying several patterns as FMEA and FTA are combined into a union consisting of every individual result. In SaCS, this is defined by a *combines* relation between patterns. A *details* relation is used to express that the result of applying one pattern is further detailed by the application of a second pattern. Functional safety is an important concept in IEC 61508 [23]. Functional safety is a part of the overall safety that depends on a system or equipment operating correctly in response to its inputs. Furthermore, functional safety is achieved when every specified safety function is carried out and the level of performance required of each safety function is met. A *satisfies* relation between a pattern for requirements elicitation and a pattern for system design expresses that the derived system satisfies the derived requirements. Safety case patterns supports documenting the safety argument. A *demonstrates* relation between a safety case pattern and a design pattern expresses that the derived safety argument represents a safety demonstration for the derived system.

Fig. 6 and Fig. 7 illustrate how the intended instantiation order of patterns may be visualised. The direction of the arrow indicates the pattern instantiation order; patterns (or more precisely the patterns referred to graphically) placed closer to the starting point of the arrow are instantiated prior to patterns placed close to the tip of the arrow. Patterns may be instantiated in parallel and thus have no specific order; this is visualised by placing pattern references on separate arrows.

Figure 6. Serial instantiation

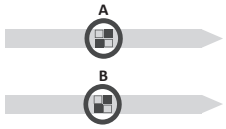


Figure 7. Parallel instantiation

As argued above, the SaCS language facilitates the application of best practices within safety design and mirrors leading international standards within the area; in particular IEC 61508. We therefore think it is fair to say that the language to a large extent fulfils DA.2.

B. Modeller appropriateness

MA.1 The language must facilitate tacit knowledge externalisation within conceptual safety design.

As already mentioned, the current version of the language contains 26 basic patterns. The basic patterns are documented in [17] and [18]. The patterns are defined on the basis of safety engineering best practices as defined in international standards and guidelines [21][23][24][25][26][27] and other sources on safety engineering. The limited number of basic patterns currently available delimit what can be modelled in a composite pattern. Defining more basic patterns will provide a better coverage of the tacit knowledge that can be externalised. A user may easily extend the language. A basic pattern, e.g., the pattern *Hazard Analysis* [17] referenced in Fig. 1, is defined in a simple structure of named sections containing text and illustrations according to a common format. The format is thoroughly detailed in [5].

Table II compares the overall format of basic SaCS patterns to pattern formats in the literature. We have chosen a format that resembles that of Alexander et al. [2] with the addition of the sections “Pattern signature”, “Intent”, “Applicability”, and “Instantiation rule”. The signature, intent, and applicability sections of basic patterns are documented in such a manner that the context section provided in [2] is not needed. The format in [2] is a suitable basis as it is simple, well-known, and generally applicable for specifying patterns of different kinds. The format provided by Gamma et al. [8] is also simple and well-known, but tailored specifically for capturing patterns for software design.

All in all, we admit that there may be relevant tacit knowledge that is not easily externalised as the SaCS language is today. However, the opportunity of increasing the number of basic patterns makes it possible to at least reduce the gap.

MA.2 The language must support the modelling needs within conceptual safety design.

IEC 61508 [23] is defined to be applicable across all industrial domains developing safety-related systems. As already mentioned, a key concept within IEC 61508 is functional safety. Functional safety is achieved according to [23] by adopting a broad range of principles, techniques and measures.

A key concept within SaCS is that principles, techniques, methods, activities, and technical solutions of different kinds are defined within the format of basic patterns. A limited number of concerns are addressed by each basic pattern. A

TABLE II. PATTERN FORMATS IN THE LITERATURE COMPARED TO BASIC SACS PATTERNS [5]

	[6]	[2]	[8]	[9]	[10]	[28]	[29]	[30]	[5]
Name	✓	✓	✓	✓	✓	✓	✓	✓	✓
Also known as			✓		✓				
Pattern signature									✓
Intent			✓		✓				✓
Motivation			✓		✓				
Applicability			✓		✓				✓
Purpose							✓		
Context	✓	✓				✓	✓		
Problem	✓	✓		✓		✓		✓	✓
Forces	✓					✓			
Solution	✓	✓		✓		✓	✓	✓	✓
Structure			✓		✓				
Participants			✓		✓				
Collaborations			✓		✓				
Consequences			✓		✓				
Implementation			✓		✓				
Sample code			✓						
Example					✓				
Compare							✓		
Instantiation rule									✓
Related patterns		✓	✓	✓	✓	✓			✓
Known uses	✓	✓		✓		✓			✓

 ISBN: 978-1-61208-343-8

specific combination of patterns is defined within a composite pattern. A composite pattern is intended to address the overall challenges that appear in a given development context. Individual patterns within a composite only address a subset of the challenges that need to be solved in the context. A composite may be defined prior to work initiation in order to define a plan for the application of patterns. Another use may be to refine a composite throughout the work process. This is exemplified in [17] and [18]. A composite may also be defined once patterns have been applied in order to document the work process. A composite representing a plan may be easily reused for documentation purposes by adding information on the instantiation of parameters.

C. Participants appropriateness

PA.1 The terms used for concepts in the language must be the same terms used within safety engineering.

Activities such as *hazard identification* and *hazard analysis* [26], methods such as *fault tree analysis* [31] and *failure mode effects analysis* [32], system design solutions including *redundant modules* and voting mechanisms [33], and practices like arguing safety on the basis of arguing that safety requirements are satisfied [21], are all well known safety engineering practices that may be found in different standards and guidelines [23][24][27]. The different concepts mentioned above are all reflected in basic SaCS patterns. Moreover, as already pointed out, keywords such as process assurance, product assurance, requirement, solution, safety case, etc. have all been selected based on leading terminology within safety engineering.

PA.2 The symbols used to illustrate the meaning of concepts in the language must reflect these meanings.

One commonly cited and influential article within psychology is that of Miller [34], on the limit of human capacity to process information. The limit, according to Miller, is seven plus or minus two elements. When the number of elements increases past seven, the mind may be confused in correctly interpreting the information. Thus, the number of symbols should be kept low in order to facilitate effective human information processing.

Lidwell et al. [35] describe iconic representation as *"the use of pictorial images to make actions, objects, and concepts in a display easier to find, recognize, learn, and remember"*. The authors describe four forms for representation of information with icons: similar, example, symbolic, and arbitrary. We have primarily applied the symbolic form to identify a concept at a higher level of abstraction than what may be achieved with the similar and example forms. We have also tried to avoid the arbitrary form where there is little or no relationship between a concept and its associated icon. Fig. 3, Fig. 4, Fig. 5, and Fig. 6 present the main icons in SaCS. In order to allow a flexible use of icons and keep the number of icons low, we have chosen to not define a dedicated icon for each concept but rather define icons that categorises several related concepts. A relatively small number of icons was designed in a uniform manner in order to capture intuitive representations of related concepts. As an example, the referenced basic patterns in Fig. 1 have the same icons linking them by category, but unique identifiers separating them by name.

PA.3 The language must be understandable for people familiar with safety engineering without specific training.

The SaCS language is simple in the sense that a small set of icons and symbols are used for modelling the application of patterns, basically: pattern references as in Fig. 3, parameters and artefact references as in Fig. 4, relations as in Fig. 5, and instantiation order as in Fig. 6. Guidance to the understanding of the language is provided in [5], where the syntax and the semantics of SaCS patterns are described in detail. The SaCS language comes with a structured semantics [5] that offers a schematic mapping from syntactical elements into text in English. Guidance to the application of SaCS is provided by the examples detailed in [17][18]. Although we have not tested SaCS on people unfamiliar with the language, we expect that users familiar with safety engineering may comprehend the concepts and the modelling on the basis of [5][17][18] within 2-3 working days.

D. Comprehensibility appropriateness

CA.1 The concepts and symbols of the language should differ to the extent they are different.

The purpose of the graphical notation is to represent a structure of patterns in a manner that is intuitive, comprehensible, and that allows efficient visual perception. The key activities performed by a reader in order to draw conclusion from a diagram are according to Larkin and Simon [36]: searching and recognising relevant information.

Lidwell et al. [35] present 125 patterns of good design based on theory and empirical research on visualisation. The patterns describe principles of designing visual information for effective human perception. The patterns are defined on the basis of extensive research on human cognitive processes. Some of the patterns are commonly known as Gestalt principles of perception. Ellis [37] provides an extensive overview of the Gestalt principles of perception building upon classic work from Wertheimer [38] and others. Gestalt principles capture the tendency of the human mind to naturally perceive whole objects on the basis of object groups and parts.

One of several Gestalt principles applied in the SaCS language is the principle of similarity. According to Lidwell et al. [35], the principle of similarity is such that similar elements are perceived to be more related than elements that are dissimilar.

The use of the similarity principle is illustrated by the composite pattern in Fig. 1. Although each referenced pattern has a unique name, their identical icons indicate relatedness. Different kinds of patterns are symbolised by the icons in Fig. 3. The icons are of the same size with some aspects of similarity and some aspects of dissimilarity such that a degree of relatedness may be perceived. An icon for pattern reference is different in shape and shading compared to an icon used for artefact reference (see Fig. 3 and Fig. 4). Thus, an artefact and a pattern should be perceived as representing quite different concepts.

CA.2 It must be possible to group related statements in the language in a natural manner.

There are five ways to organise information according to Lidwell et al. [35]: category, time, location, alphabet, and

 ISBN: 978-1-61208-343-8

continuum. The *category* refers to the organisation of elements by similarity and relatedness. An example of the application of the principle of categorisation [35] in SaCS is seen in the possibility to reduce the number of relations drawn between patterns when these are similar. Patterns in SaCS may have multiple inputs and multiple outputs as indicated in Fig. 1. Relations between patterns operate on the parameters. The brackets [ ] placed adjacent to a pattern reference denotes an ordered list of parameters. In order to avoid drawing multiple relations between two patterns, relations operate on the ordered parameter lists of the patterns by list-matching of parameters.

Fig. 8 exemplifies two different ways for expressing visually the same relationships between the composite patterns named *A* and *B*. The list-matching mechanism is used to reduce the number of relation symbols drawn between patterns to one, even though the phenomena modelled represents multiple similar relations. This reduces the visual complexity and preserves the semantics of the relationships modelled.

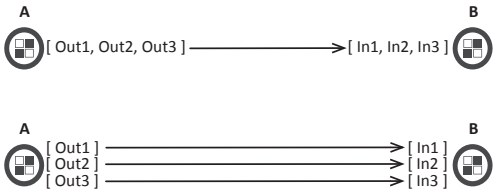


Figure 8. Alternative ways for visualising multiple similar relations

CA.3 It must be possible to reduce model complexity with the language.

Hierarchical organisation is the simplest structure for visualising and understanding complexity according to Lidwell et al. [35]. The SaCS language allows concepts to be organised hierarchically by specifying that one pattern is detailed by another or by defining composite patterns that reference other composite patterns in the content part.

Fig. 9 presents a composite pattern named *Requirements* that reference other composites as part of its definition. The contained pattern *Safety Requirements* is defined in Fig. 1. The contained pattern *Functional Requirements* is not defined and is referenced within Fig. 9 for illustration purposes. *Requirements* may be easily extended by defining composites supporting the elicitation of, e.g., performance requirements and security requirements, and later model the use of such patterns in Fig. 9. In Fig. 9, the output of applying the *Requirements* pattern is represented by the parameter *ReqSpec*. The *ReqSpec* parameter represents the result of applying the *combines* relation on the output *Req* of the composite *Safety Requirements* and the output *Req* of the composite *Functional Requirements*.

CA.4 The symbols of the language should be as simple as possible with appropriate use of colour and emphasis.

A general principle within visualisation according to Lidwell et al. [35] is to use colour with care as it may lead to misconceptions if used inappropriately. The authors points out that there is no universal symbolism for different colours.

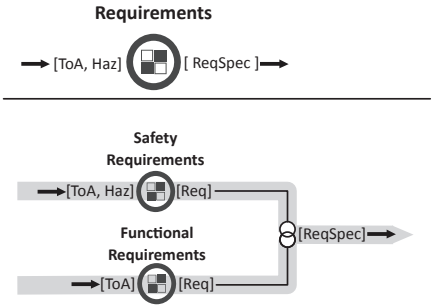


Figure 9. Composition of composites

As colour blindness is common the SaCS language applies different shades of grey in visualisations.

Fig. 10 illustrates how the SaCS language makes use of the three Gestalt principles of perception [35][39][38] known as: Figure-Ground; Proximity; and Uniform Connectedness. The Gestalt principles express mechanisms for efficient human perception from groups of visual objects.

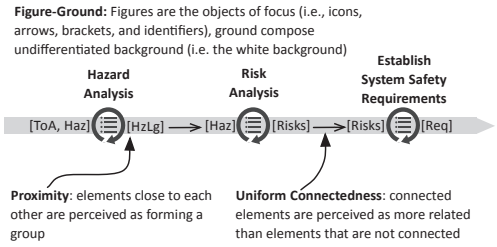


Figure 10. A fragment of Fig. 1 illustrating the use of Gestalt principles

E. Tool appropriateness

TA.1 The language must have a precise syntax.

The syntax of the SaCS language (see [5]) is defined in the EBNF [40] notation. EBNF is a meta-syntax that is widely used for describing context-free grammars.

TA.2 The language must have a precise semantics.

A structured semantics for SaCS patterns is defined in [5] in the form of a schematic mapping from pattern definitions, via its textual syntax in EBNF [40], to English. The non-formal representation of the semantics supports human interpretation rather than tools, although the translation procedure as described in [5] may be automated. The presentation of the semantics of patterns as a text in English was chosen in order to aid communication between users, possibly with different technical background, on how to interpret patterns.

F. Organisational appropriateness

OA.1 The language must be able to express the desired conceptual safety design when applied in a safety context.

 ISBN: 978-1-61208-343-8

The application of the SaCS pattern language produces composite patterns that are instantiated into conceptual safety designs. A composite pattern expresses a combination of basic patterns. The basic patterns express safety engineering best practices and concepts inspired by international safety standards and guidelines, e.g., [23][24][27]. International safety standards and guidelines describe concepts and practices for development of safety critical systems that may be perceived as commonly accepted. The SaCS pattern language is tested out in two cases. The first concerned the conceptualisation of a nuclear power plant control system, while the second addressed the conceptualisation of a railway interlocking system, fully detailed in [17] and [18], respectively. In both cases it was possible to derive a conceptual safety design using the SaCS language as support as well as model how patterns were applied as support.

OA.2 The language must ease the comprehensibility of best practices within conceptual safety design for relevant target groups like system engineers, safety engineers, hardware and software engineers.

We have already explained how basic patterns represent concepts and best practices inspired by safety standards and guidelines. Each basic pattern addresses a limited number of phenomena. Basic patterns are combined into a composite pattern where the composite addresses all relevant challenges that occur in a specific context. A composite pattern as the one presented in Fig. 1 ease the explanation of how several concepts within conceptual safety design are combined and applied.

Wong et al. [41] reviewed several large development projects and software safety standards from different domains with respect to cost effectiveness and concludes that although standards provide useful and effective guidance, safety and cost effectiveness objectives are successfully met by effective planning and by applying safety engineering best practices evidenced in company best practices throughout the development life cycle. Compared to a standard or a guideline, a composite pattern in the SaCS language may be used to capture such a company specific best practice. In order to accommodate different situations, different compositions of patterns may be defined.

OA.3 The language must be usable without the need of costly tools.

Every pattern used in the cases described in [17][18] was interpreted and applied in its context by a single researcher with background from safety engineering. A conceptual safety design was produced for each case. Every illustration in [5][17][18] and in this paper is created with a standard drawing tool.

V. Related Work

In the literature, pattern approaches supporting development of safety critical systems are poorly represented. In the following we shortly discuss some different pattern approaches and their relevancy to the development of conceptual safety designs.

Jackson [42] presents the problem frames approach for requirements analysis and elicitation. Although the problem frames approach is useful for detailing and analysing a problem and thereby detailing requirements, the problem classes presented in [42] are defined on a very high level of abstraction.

The use of boilerplates [43][44] for requirement specification is a form of requirement templates but nonetheless touches upon the concept of patterns. The boilerplate approach helps the user phrase requirements in a uniform manner and to detail these sufficiently. Although boilerplates may be useful for requirement specification, the focus in SaCS is more towards supporting requirement elicitation and the understanding of the challenges that appear in a specific context.

Withall [45] describes 37 requirements patterns for assisting the specification of different types of requirements. The patterns are defined at a low level; the level of a single requirement. The patterns of Withall may be useful, but as with the boilerplates approach, the patterns support more the specification of requirements rather than requirements elicitation.

Patterns on design and architecture of software-based systems are presented in several pattern collections. One of the well-known pattern collections is the one of Gamma et al. [8] on recurring patterns in design of software based systems. Without doubt, the different pattern collections and languages on system design and architecture represent deep insight into effective solutions. However, design choices should be founded on requirements, and otherwise follow well established principles of good design. The choice of applying one design pattern over another should be based on a systematic process of establishing the need in order to avoid design choices being left unmotivated.

The motivations for a specific design choice are founded on the knowledge gained during the development activities applied prior to system design. Gnatz et al. [46] outline the concept of process patterns as a means to address the recurring problems and known solutions to challenges arising during the development process. The patterns of Gnatz et al. are not tailored for development of safety critical systems and thus do not necessarily reflect relevant safety practices. Fowler presents [7] a catalogue of 63 analysis patterns. The patterns do not follow a strict format but represent a body of knowledge on analysis described textually and by supplementary sketches.

While process patterns and analysis patterns may be relevant for assuring that the development process applied is suitable and leads to well informed design choices, Kelly [10] defines patterns supporting safety demonstration in the form of reusable safety case patterns. The patterns expressed are representative for how we want to address the safety demonstration concern.

A challenge is to effectively combine and apply the knowledge on diverse topics captured in different pattern collections and languages. Henninger and Corrêa [47] survey different software pattern practices and states *"software patterns and collections tend to be written to solve specific problems with little to no regard about how the pattern could or should be used with other patterns"*.

Zimmer [48] identifies the need to define relationships between system design patterns in order to efficiently combine them. Noble [49] builds upon the ideas of Zimmer and defines

 ISBN: 978-1-61208-343-8

a number of relationships such as *uses, refines, used by, combine*, and *sequence of* as a means to define relationships between system design patterns. A challenge with the relations defined by Noble is that they only specify relations on a very high level. The relations do not have the expressiveness for detailing what part of a pattern is *used, refined*, or *combined*. Thus, the approach does not facilitate a precise modelling of relationships.

Bayley and Zhu [50] define a formal language for pattern composition. The authors argue that design patterns are almost always to be found composed with each other and that the correct applications of patterns thus relies on precise definition of the compositions. A set of six operators is defined for the purpose of defining pattern compositions. The language is exemplified on the formalisation of the relationships expressed between software design patterns described by Gamma et al. [8]. As we want the patterns expressed in the SaCS language to be understandable to a large community of potential users, we find this approach a bit too rigid.

Smith [51] presents a catalogue of elementary software design patterns in the tradition of Gamma et al. [8] and proposes the Pattern Instance Notation (PIN) for expressing compositions of patterns graphically. The notation uses simple rounded rectangles for abstractly representing a pattern and its associated roles. Connectors define the relationships between patterns. The connectors operate on the defined roles of patterns. The notation is comparable to the UML collaboration notation [52].

UML collaborations [52] are not directly instantiable. Instances of the roles defined in a collaboration that cooperates as defined creates the collaboration. The main purpose is to express how a system of communicating entities collectively accomplishes a task. The notation is particularly suitable for expressing system design patterns.

Several notations [53][54][55] for expressing patterns graphically use UML [52] as its basis. The notations are simple, but target the specification of software.

VI. Conclusion

We have presented an analytical evaluation of the SaCS pattern language with respect to six different appropriateness factors. We arrived at the following conclusions:

- *Domain*: In the design of the SaCS language we have as much as possible selected keywords and icons in the spirit of leading literature within the area. This indicates that we at least are able to represent a significant part of the concepts of relevance for conceptual safety design.
- *Modeller*: There may be relevant tacit knowledge that is not easily externalised as the SaCS language is today. However, the opportunity of increasing the number of basic patterns makes it possible to at least reduce the gap.
- *Participants*: The terms used for concepts have been carefully selected based on leading terminology within safety engineering. The SaCS language facilitates representing the application of best practices within safety design and mirror leading international standards; in particular IEC 61508.
- *Comprehensibility*: The comprehension of individual patterns and pattern compositions is supported by the use of terms commonly applied within the relevant industrial domains as well as by the application of principles of good design in visualisations, such as the Gestalt principles of perception [35][38].
- *Tool*: Tool support may be provided on the basis of the syntax and semantics of the SaCS language [5].
- *Organisational*: Organisations developing safety critical systems are assumed to follow a development process in accordance to what is required by standards. Wong et al. [41] reviewed several large development projects and software safety standards from different domains with respect to cost effectiveness and concludes that although standards provide useful and effective guidance, safety and cost effectiveness objectives are successfully met by effective planning and by applying safety engineering best practices evidenced in company best practices throughout the development life cycle. SaCS patterns may be defined, applied, and combined in a flexible manner to support company best practices and domain specific best practices.

Acknowledgment

This work has been conducted and funded within the OECD Halden Reactor Project, Institute for energy technology (IFE), Halden, Norway. The second author has partly been funded by the ARTEMIS project CONCERTO.

References

[1] F. Buschmann, K. Henney, and D. C. Schmidt, Pattern-Oriented Software Architecture: On Patterns and Pattern Languages. Wiley, 2007, vol. 5.

[2] C. Alexander, S. Ishikawa, and M. Silverstein, A Pattern Language: Towns, Buildings, Construction. Oxford University Press, 1977, vol. 2.

[3] J. C. Knight, "Safety Critical Systems: Challenges and Directions," in Proceedings of the 24th International Conference on Software Engineering (ICSE'02). ACM, 2002, pp. 547–550.

[4] J. Krogstie, Model-based Development and Evolution of Information Systems: A Quality Approach. Springer, 2012.

[5] A. A. Hauge and K. Stølen, "Syntax & Semantics of the SaCS Pattern Language," Institute for energy technology, OECD Halden Reactor Project, Halden, Norway, Tech. Rep. HWR-1052, 2013.

[6] A. Aguiar and G. David, "Patterns for Effectively Documenting Frameworks," in Transactions on Pattern Languages of Programming II, ser. LNCS, J. Noble, R. Johnson, P. Avgeriou, N. Harrison, and U. Zdun, Eds. Springer, 2011, vol. 6510, pp. 79–124.

[7] M. Fowler, Analysis Patterns: Reusable Object Models. Addison-Wesley, 1996.

[8] E. Gamma, R. Helm, R. E. Johnson, and J. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, 1995.

[9] R. S. Hanmer, Patterns for Fault Tolerant Software. Wiley, 2007.

[10] T. P. Kelly, "Arguing Safety – A Systematic Approach to Managing Safety Cases," Ph.D. dissertation, University of York, United Kingdom, 1998.

[11] B. Rubel, "Patterns for Generating a Layered Architecture," in Pattern Languages of Program Design, J. Coplien and D. Schmidt, Eds. Addison-Wesley, 1995, pp. 119–128.

 ISBN: 978-1-61208-343-8

[12] J. Mendling, G. Neumann, and W. van der Aalst, "On the Correlation between Process Model Metrics and Errors," in Proceedings of 26th International Conference on Conceptual Modeling, vol. 83, 2007, pp. 173–178.

[13] A. G. Nysetvold and J. Krogstie, "Assessing Business Process Modeling Languages Using a Generic Quality Framework," in Proceedings of the 17th Conference on Advanced Information Systems Engineering (CAiSE'05) Workshops. Idea Group, 2005, pp. 545–556.

[14] J. Krogstie and S. D. F. Arnesen, "Assessing Enterprise Modeling Languages Using a Generic Quality Framework," in Information Modeling Methods and Methodologies. Idea Group, 2005, pp. 63–79.

[15] D. L. Moody, G. Sindre, T. Brasethvik, and A. Sølvberg, "Evaluating the Quality of Process Models: Empirical Testing of a Quality Framework," in Proceedings of the 21st International Conference on Conceptual Modeling, ser. LNCS. Springer, 2013, vol. 2503, pp. 380–396.

[16] J. Becker, M. Rosemann, and C. von Uthmann, "Guidelines of Business Process Modeling," in Business Process Management, ser. LNCS, vol. 1806. Springer, 2000, pp. 30–49.

[17] A. A. Hauge and K. Stølen, "A Pattern-based Method for Safe Control Conceptualisation – Exemplified Within Nuclear Power Production," Institute for energy technology, OECD Halden Reactor Project, Halden, Norway, Tech. Rep. HWR-1029, 2013.

[18] ——, "A Pattern-based Method for Safe Control Conceptualisation – Exemplified Within Railway Signalling," Institute for energy technology, OECD Halden Reactor Project, Halden, Norway, Tech. Rep. HWR-1037, 2013.

[19] I. Hogganvik, "A Graphical Approach to Security Risk Analysis," Ph.D. dissertation, Faculty of Mathematics and Natural Sciences, University of Oslo, 2007.

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

[21] The members of the Task Force on Safety Critical Software, "Licensing of safety critical software for nuclear reactors: Common position of seven european nuclear regulators and authorised technical support organisations," http://www.belv.be/, 2013, [accessed: 2014-04-10].

[22] C. Haddon-Cave, "The Nimrod Review: An independent review into the broader issues surrounding the loss of the RAF Nimrod MR2 aircraft XV230 in Afghanistan in 2006," The Stationery Office (TSO), Tech. Rep. 1025 2008-09, 2009.

[23] IEC, "IEC 61508 Functional safety of electrical/electronic/programmble electronic safety-related systems, 2nd edition," International Electrotechnical Commission, 2010.

[24] CENELEC, "EN 50129 Railway applications – Communications, signalling and processing systems – Safety related electronic systems for signalling," European Committee for Electrotechnical Standardization, 2003.

[25] European Commission, "Commission Regulation (EC) No 352/2009 on the Adoption of Common Safety Method on Risk Evaluation and Assessment," Official Journal of the European Union, 2009.

[26] ERA, "Guide for the Application of the Commission Regulation on the Adoption of a Common Safety Method on Risk Evaluation and Assessment as Referred to in Article 6(3)(a) of the Railway Safety Directive," European Railway Agency, 2009.

[27] IEC, "IEC 61513 Nuclear power plants – Instrumentation and control systems important to safety – General requirements for systems, 2nd edition," International Electrotechnical Commission, 2001.

[28] A. Ratzka, "User Interface Patterns for Multimodal Interaction," in Transactions on Pattern Languages of Programming III, ser. LNCS, J. Noble, R. Johnson, U. Zdun, and E. Wallingford, Eds. Springer, 2013, vol. 7840, pp. 111–167.

[29] D. Riehle and H. Züllinghoven, "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. Addison-Wesley, 1995, pp. 9–42.

[30] K. Wolf and C. Liu, "New Client with Old Servers: A Pattern Language for Client/Server Frameworks," in Pattern Languages of Program Design, J. Coplien and D. Schmidt, Eds. Addison-Wesley, 1995, pp. 51–64.

[31] IEC, "IEC 61025 Fault Tree Analysis (FTA), 2nd edition," International Electrotechnical Commission, 2006.

[32] ——, "IEC 60812 Analysis techniques for system reliability – Procedure for failure mode and effects analysis (FMEA), 2nd edition," International Electrotechnical Commission, 2006.

[33] N. Storey, Safety-critical Computer Systems. Prentice Hall, 1996.

[34] G. A. Miller, "The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information," Psychological Review, vol. 63, no. 2, 1956, pp. 81–97.

[35] W. Lidwell, K. Holden, and J. Butler, Universal Principles of Design, 2nd ed. Rockport Publishers, 2010.

[36] J. H. Larkin and H. A. Simon, "Why a Diagram is (Sometimes) Worth Ten Thousand Words," Cognitive Science, vol. 11, no. 1, 1987, pp. 65–100.

[37] W. D. Ellis, A Source Book of Gestalt Psychology. The Gestalt Journal Press, 1997.

[38] M. Wertheimer, "Laws of Organization in Perceptual Forms," in A sourcebook of Gestalt Psychology, W. D. Ellis, Ed. Routledge and Kegan Paul, 1938, pp. 71–88.

[39] J. Wagemans, J. H. Elder, M. Kubovy, S. E. Palmer, M. A. Peterson, M. Singh, and R. von der Heydt, "A Century of Gestalt Psychology in Visual Perception: I. Perceptual Grouping and Figure-Ground Organization," Psychological Bulletin, vol. 138, no. 6, 2012, pp. 1172–1217.

[40] ISO/IEC, "14977:1996(E) Information technology - Syntactic metalanguage - Extended BNF," International Organization for Standardization / International Electrotechnical Commission, 1996.

[41] W. E. Wong, A. Demel, V. Debroy, and M. F. Siok, "Safe Software: Does It Cost More to Develop?" in Fifth International Conference on Secure Software Integration and Reliability Improvement (SSIRI'11), 2011, pp. 198–207.

[42] M. Jackson, Problem Frames: Analysing and Structuring Software Development Problems. Addison-Wesley, 2001.

[43] E. Hull, K. Jackson, and J. Dick, Requirements Engineering, 3rd ed. Springer, 2010.

[44] G. Sindre, "Boilerplates for application interoperability requirements," in Proceedings of 19th Norsk konferanse for organisasjoners bruk av IT (NOKOBIT'12). Tapir, 2012.

[45] S. Withall, Software Requirement Patterns (Best Practices), 1st ed. Microsoft Press, 2007.

[46] M. Gnatz, F. Marschall, G. Popp, A. Rausch, and W. Schwerin, "Towards a Living Software Development Process based on Process Patterns," in Proceedings of the 8th European Workshop on Software Process Technology (EWSPT'01), ser. LNCS, vol. 2077. Springer, 2001, pp. 182–202.

[47] S. Henninger and V. Corrêa, "Software Pattern Communities: Current Practices and Challenges," in Proceedings of the 14th Conference on Pattern Languages of Programs (PLOP'07). ACM, 2007, pp. 14:1–14:19, article No. 14.

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

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

[50] I. Bayley and H. Zhu, "A Formal Language for the Expression of Pattern Compositions," International Journal on Advances in Software, vol. 4, no. 3, 2012, pp. 354–366.

[51] J. M. Smith, Elemental Design Patterns. Addison-Wesley, 2012.

[52] OMG, "Unified Modeling Language Specification, Version 2.4.1," Object Management Group, 2012.

[53] H. Byelas and A. Telea, "Visualization of Areas of Interest in Software Architecture Diagrams," in Proceedings of the 2006 ACM Symposium on Software Visualization (SoftVis'06), 2006, pp. 105–114.

[54] J. Dong, S. Yang, and K. Zhang, "Visualizing Design Patterns in Their Applications and Compositions," IEEE Transactions on Software Engineering, vol. 33, no. 7, 2007, pp. 433–453.

[55] J. M. Vlissides, "Notation, Notation, Notation," C++ Report, 1998, pp. 48–51.

 ISBN: 978-1-61208-343-8