An Analytic Evaluation of the SaCS Pattern Language for Conceptualisation of Safety Critical Systems

André Alexandersen Hauge
Institute for Energy Technology, Halden, 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 the Safe Control Systems (SaCS) pattern language for the development of conceptual safety designs and conduct an analytical evaluation of the appropriateness of the language for its intended task. 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 can express basic patterns on different aspects of relevance for conceptual safety designs. SaCS can also be used to combine basic patterns into composite patterns. A composite pattern can 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 language on the basis of the theoretical, technical, and practical considerations that were taken into account and shaped the SaCS language.**

Keywords–pattern language; evaluation; design; conceptualisation; safety.

I. INTRODUCTION

This paper presents the Safe Control Systems (SaCS) pattern language and an evaluation of the suitability of the language as support for the development of safety critical systems. A shorter version of this paper is presented in [1].

A pattern describes a particular recurring problem that arises in a specific context and presents a well-proven generic scheme for its solution [2]. 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 [3]. The SaCS pattern language has been designed to facilitate the specification of patterns to support the development of conceptual safety designs. With a conceptual safety design, we mean an early stage specification of system requirements, system design, and safety case for a safety critical system. A safety critical system [4] is a system *"whose failure could result in loss of life, significant property damage, or damage to the environment"*. The intended users of the SaCS pattern language are system engineers, safety engineers, hardware and software engineers.

According to McGrath [5], there are eight common methods for evaluation. However, there is no single evaluation method 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). Based on the strengths and weaknesses of the different evaluation methods and the questions that are required to be answered, the researcher has to choose how different kinds of methods should be combined. It is desirable to maximise precision, realism, and generality simultaneously but, as argued by McGrath, this is not possible with one single research method.

The suitability of the SaCS pattern language for its intended task is investigated by complementing kinds of evaluations; two case studies and the analytic evaluation presented in this paper. The two case studies, fully documented in [6] and [7], can be seen as variants of what McGrath terms *field experiment*, a method that scores high on realism. According to Eisenhardt [8], the case study approach is especially appropriate in new topic areas and describes how to build theories from case study research. The analytic evaluation is most closely related to *non-empirical evidence* in the McGrath classification, a method that scores high on generality.

A framework for analysing languages known as the Semiotic Quality (SEQUAL) framework [9] is used as a basis for the analytic evaluation. The appropriateness of a language for its intended task is in the framework characterised by six appropriateness factors [9]: 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 paper is structured as follows: Section II provides an introduction to the SaCS pattern language. Section III demonstrates the applicability of the SaCS pattern language in an example. Section IV discusses analytic evaluation approaches and motivates the selection of the SEQUAL framework. Section V presents the analytic evaluation of the SaCS pattern language according to the SEQUAL framework. Section VI presents related work on pattern-based development. Section VII draws the conclusions.

II. Background on the SaCS Pattern Language

The SaCS pattern language is an integrated part of the SaCS method. In order explain the language, we outline the SaCS method. The SaCS method consists of the following three artefacts:

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

B. *The SaCS library*: 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.

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

In order to apply the SaCS method, an assumed user of the SaCS process employs the patterns within the library as guidance to problem solving. Furthermore, the user of the process employs the language for expressing a solution to a problem in the form of a pattern in order to extend the library. The language has a formal syntax as well as a structured semantics [10] that supports users in specifying patterns and 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 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 produces the conceptual safety design.

In the following sub-sections we briefly describe each of the artefacts that are part of the SaCS method.

A. The SaCS process

The SaCS process 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 by selecting: a) SaCS patterns for requirement elicitation; b) SaCS patterns for establishing design basis; and c) SaCS patterns for establishing safety case.
- *Pattern Composition*: The purpose of this activity is to specify the use of the selected patterns by specifying: a) compositions of patterns; and b) instantiations of patterns.
- *Pattern Instantiation*: The purpose of this activity is to instantiate the composite pattern specification by: a) selecting pattern instantiation order; and b) conducting stepwise instantiation.

The SaCS process is exemplified in Section III. The example shows how a pattern selection map is used as support for pattern selection, and how the pattern language supports pattern composition. Pattern instantiation is supported by instantiation rules defined for every basic pattern (fully defined in [6][7]). The definition of one of the basic patterns in SaCS is also presented in Section II-B.

B. The SaCS Library

The SaCS library consists of a set of basic patterns as well as the composite patterns defined by a user on the basis of pre-defined patterns. While a basic pattern is defined by text and illustrations, a composite is defined graphically. In the following, a slightly formatted version of the basic pattern *Establish System Safety Requirements* documented in [6] is reproduced as an example of the content of the library. The pattern is described as a sequence of named sections. The section names are presented in a **bold** font. The section named "Pattern Signature" contains an illustration classifying the pattern as well as its inputs and output parameters according to the syntax of the SaCS pattern langauge [10]. The section "Process Solution" contains a UML activity diagram with SaCS language specific annotations, presented in Fig. 2. 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 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 represent the process solution in terms of a UML activity diagram. The SaCS specific annotations in Fig. 2 indicate in what way inputs are related to the different activities of the process for establishing safety requirements and how the result of one of the activities represents an output of applying the pattern.

Name: Establish System Safety Requirements

Pattern Signature: *Establish System Safety Requirements* is defined with the signature illustrated in Fig. 1. In Fig. 1, 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 1. 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 can 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, performing certain activities, or making use of specific techniques.

Process Solution: Fig. 2 illustrates the *Establish System Safety Requirements* process specified using a UML activity diagram.

The input parameters associated with the activity diagram can 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.

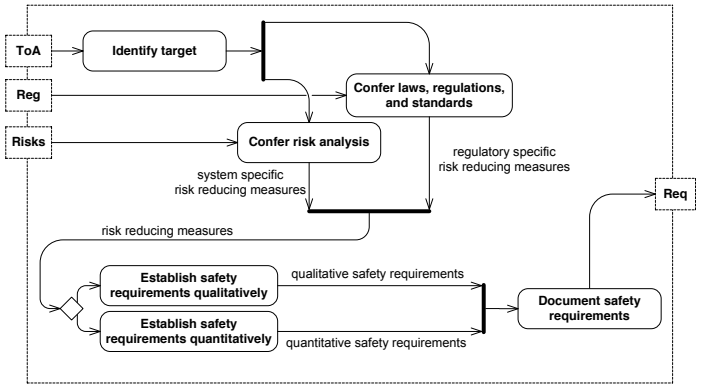

Figure 2. Establish System Safety Requirements – Process Flow

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. 1 and Fig. 2) 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. 2 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:

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

C. The SaCS Pattern Language

Fig. 3 defines a composite pattern according to the syntax of SaCS [10]. The composite described in Fig. 3 is named *Safety Requirements* and consists of the basic patterns *Hazard Analysis*, *Risk Analysis*, and *Establish System Safety Requirements*. The contained patterns of *Safety Requirements* are referenced graphically. The basic patterns are specified separately in a structured manner comparable to what can be found in the literature [2][3][11][12][13][14][15][16] on patterns, e.g., as in the case of the pattern *Establish System Safety Requirements* presented in Section II-B.

In Fig. 3, 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

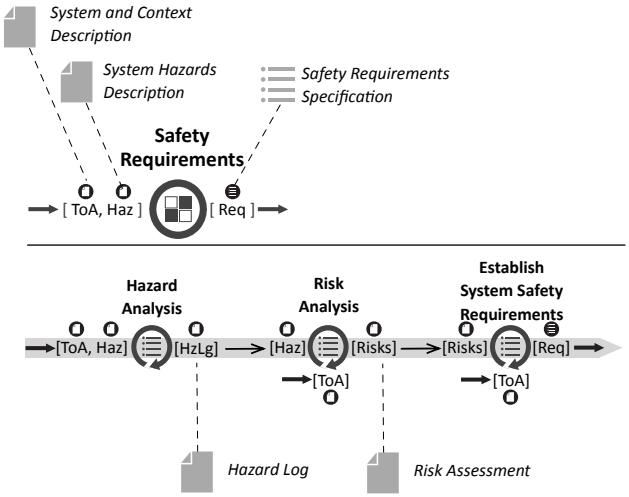

Figure 3. A composite pattern named Safety Requirements

a composite pattern. Every pattern in SaCS is parameterised. 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. 3 are of type *documentation*, while *Req* is of type *requirement*. The inputs and outputs of a composite are always publicly accessible.

A particular instantiation of a parameter is documented by a relation that connects a parameter with its associated development artefact. In Fig. 3, 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. 3 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.

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

Pattern reference: Fig. 4 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.

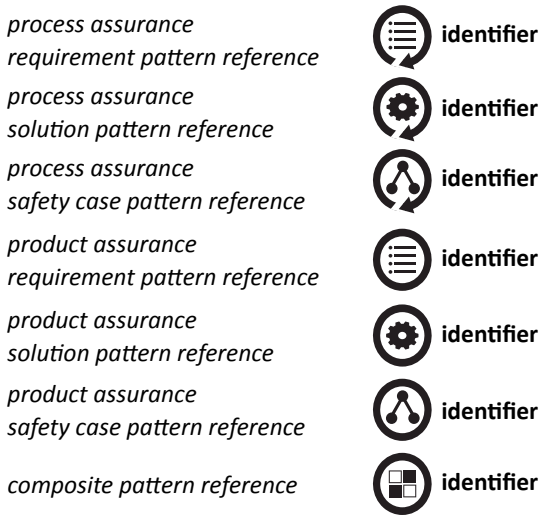

Figure 4. The icons for the different kinds of pattern references in SaCS

Parameter: Fig. 5 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*.

Relation: Fig. 6 presents the symbols for different kinds of relations defined in SaCS. A relation denotes a relationship

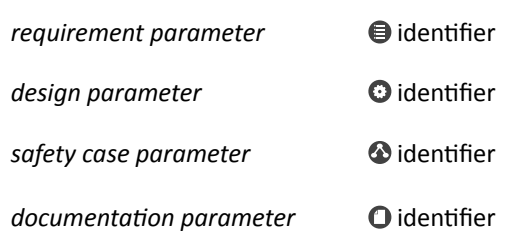

Figure 5. The icons for different kinds of parameters in SaCS

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) is a safety demonstration for the output of a related pattern (typically represented by a design parameter).

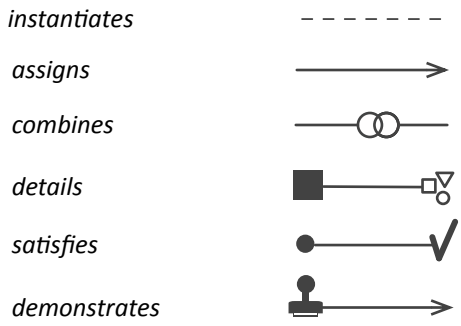

Figure 6. The symbols for the different kinds of relations in SaCS

Artefact reference: Fig. 7 presents the different kinds of artefact references defined in SaCS. An artefact reference consists of an unique identifier in an *italics* font and an icon classifying what kind of artefact that is referenced. Artefact references are used for denoting a specific representation of parameters.

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. 8 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. 8, no specific ordering of the patterns *A* and *B* is assumed and thus the respective pattern references are placed on two separate arrows.

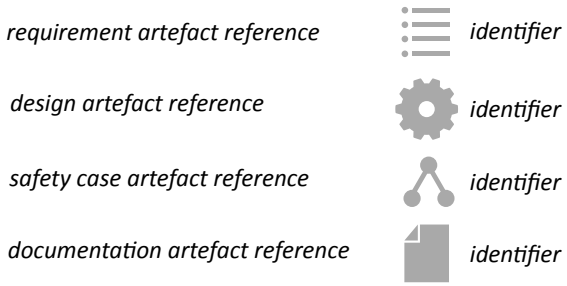

Figure 7. The icons for different kinds of artefact references in SaCS

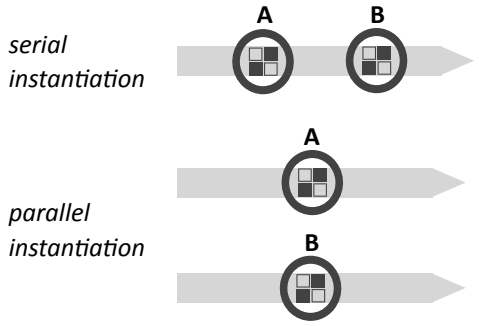

Figure 8. The symbols for the different kinds of instantiation orders in SaCS

III. SaCS EXEMPLIFIED

Fig. 9 exemplifies the integration of the three artefacts into the SaCS method. In Fig. 9, arrows are used to indicate the flow between the use of the SaCS process, the use of the library, and the language 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 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.

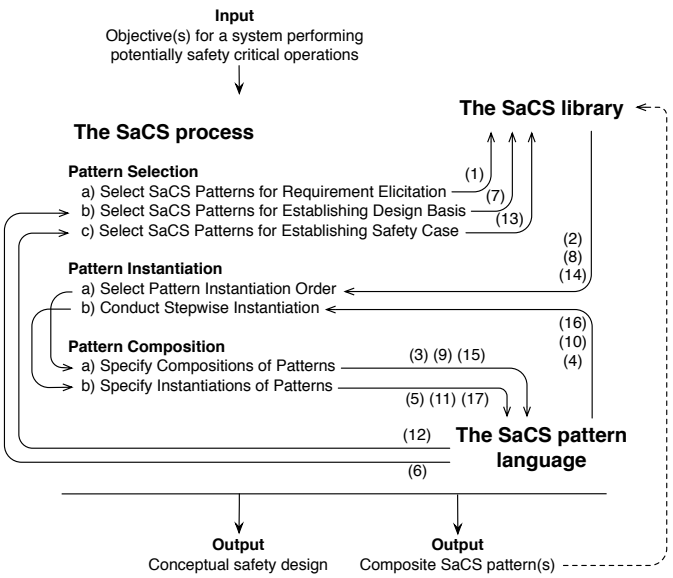

Figure 9. Example integration of the three artefacts into the SaCS method

In the following, the SaCS method is explained by exemplifying the steps (1) to (17) from Fig. 9. The example shows the main steps in the application of the SaCS method for developing a conceptual safety design of a railway interlocking system. A comprehensive explanation can be found in [7]; an outline is given below.

Fig. 10 illustrates a train station with two tracks. The station is connected in both ends of the station area to neighbouring stations with a single track. There are eight train routes possible with the track configuration illustrated in Fig. 10. An interlocking system controls the movements of trains along defined train routes. The interlocking system actuates the different distant signals, main signals, and point equipment according to defined rules in order to enforce safe train movements. A distant signal gives the train driver indication on the signalling to expect from the associated main signal.

Fig. 11 presents references to development documents that are available to an assumed user in the following example. *Interlocking concept description* is a document assumed to define the main functionality of an interlocking system for separating train movements at a train station as illustrated in Fig. 10. Furthermore, *Hazard log* is a document assumed to describe the result from an initial hazard analysis of the interlocking concept. There are patterns in the library that facilitate 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 interlocking system.

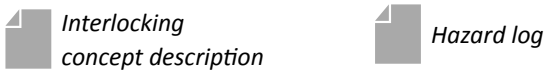

Figure 11. Representation of references to development documentation

Step (1): Fig. 12 illustrates a fragment of a larger pattern selection map 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 indicated by the arrows and considers whether a pattern is relevant for application based on 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.

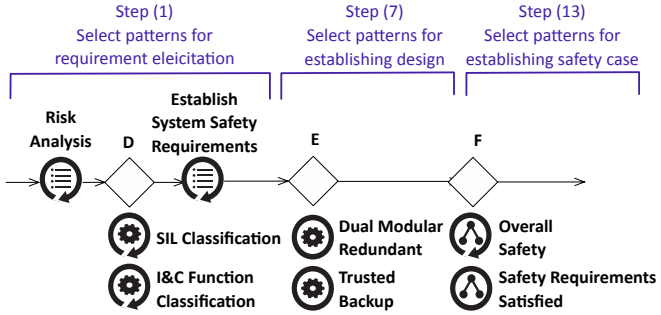

Figure 12. Pattern selection map (simplified)

By the use of Fig. 12, the user starts the pattern selection activity from the left and identify *Risk Analysis* as the first

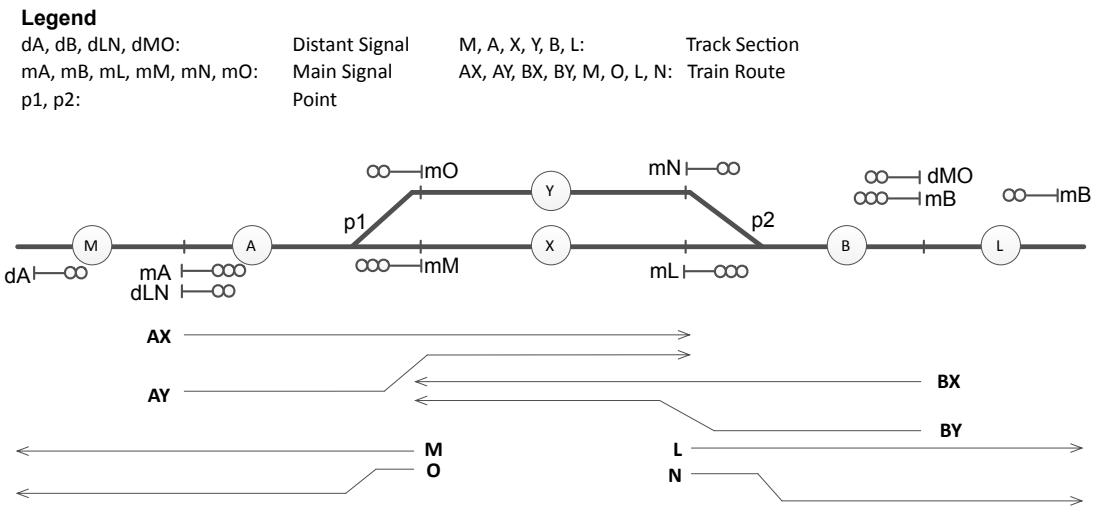

Figure 10. Railway case (adopted from [7])

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 assessment 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 detail earlier. The pattern describes how the result from risk analysis should be used as input to the specification of safety requirements. The user selects these three patterns as support for the elicitation of requirements in Step (1). The user 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. 13 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. 12. The order can also be found by inspecting the "Related Patterns" sections of the pattern definitions. In Fig. 13, the order is specified 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 [ ] embrace a parameter list. 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 the parameters classify the parameters. The thin arrows represents three instances of an *assigns* relation.

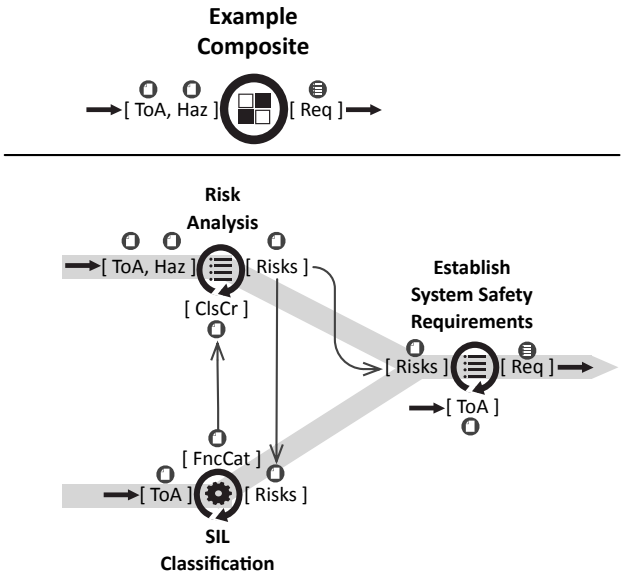

Figure 13. Composite that can be instantiated into a specification of requirements

Step (4)-(5): Fig. 14 is identical to Fig. 13 with the addition of annotations indicating the instantiation of patterns. In Fig. 14, the instantiation of the composite is documented such that *Interlocking 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*. An example finding from the risk analysis is that erroneously positioned points can cause train derailment

or collision. Thus, the interlocking system is clearly a safety critical system. Furthermore, the interlocking system must assure that points are always positioned correctly. An example of a safety requirement that addresses this is *"SR.2: A train route may not be locked unless all points belonging to the train route are positioned correctly."*

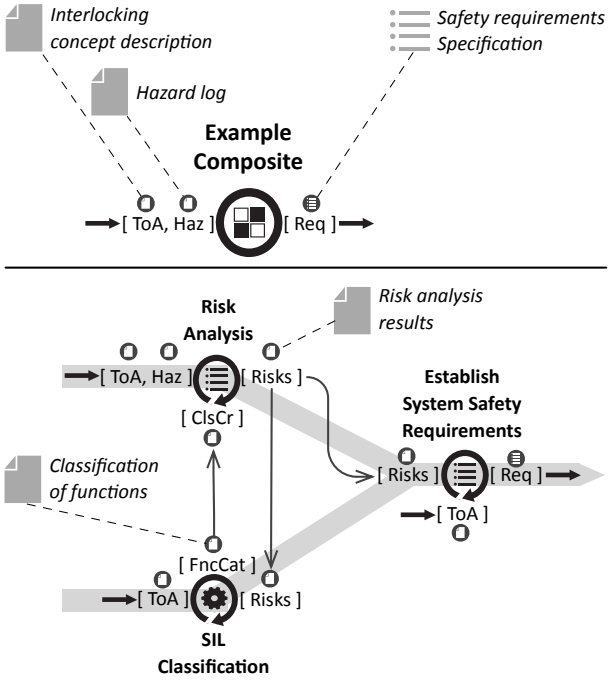

Figure 14. Composite specifying its instantiation into a specification of requirements

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. 12, we have arrived at choice E. In choice E, the design patterns *Dual Modular Redundant* and *Trusted Backup* are indicated as alternatives. We assume that the user finds *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 the relevant requirements. The user selects *Dual Modular Redundant* as support for system design.

Step (9)-(11): Fig. 15 is identical to Fig. 14 with the addition of annotations indicating the instantiation of the design pattern named *Dual Modular Redundant*. The detailing of the application of patterns for establishing design basis is the concern of Step (9) and Step (11). However, the actual instantiation of patterns for system design is the concern of Step (10).

In Step (10), the user is supposed to make use of the requirements derived earlier as input for detailing a system design. We assume that the user instantiate *Dual Modular Redundant* according to its instantiation rule to produce a system design specification. Fig. 16 is a simplified UML component diagram showing an excerpt of the system design specification

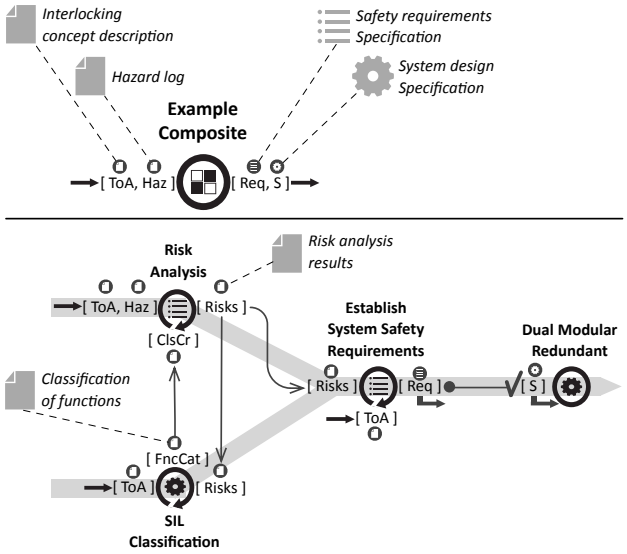

Figure 15. Composite specifying its instantiation into a specification of requirements and system design

that describes the interlocking system. The interlocking system consists of dual controller components (Ctr 1 and Ctr 2) that implements the interlocking rules. The command unit (Cmd) is responsible for handling the interaction with the operator. The interlocking system interacts with equipment like lights, points and train detection equipment through dedicated interfaces. A voter assures fail-safe behaviour (e.g., all lights indicate stop) in the case of disagreement between the redundant controllers.

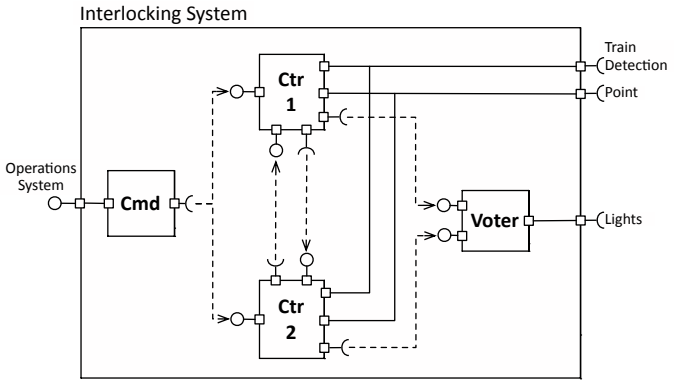

Figure 16. Excerpt of the system design specification – The main components

Fig. 17 is a simplified UML state machine diagram exemplifying the specification of the behaviour of the interlocking system.

The diagram details the interlocking system behaviour given a request for locking a train route AX (see Fig. 10). In Fig. 17, details are only given for the state "Check Points in Correct Position" as this is relevant for the example requirement stated earlier (see SR.2). While Fig. 16 specifies the main components of the system under construction according to the guidance within *Dual Modular Redundant*, Fig. 17 specifies the behaviour of the system in accordance with the requirements derived with the use of *Establish System Safety*

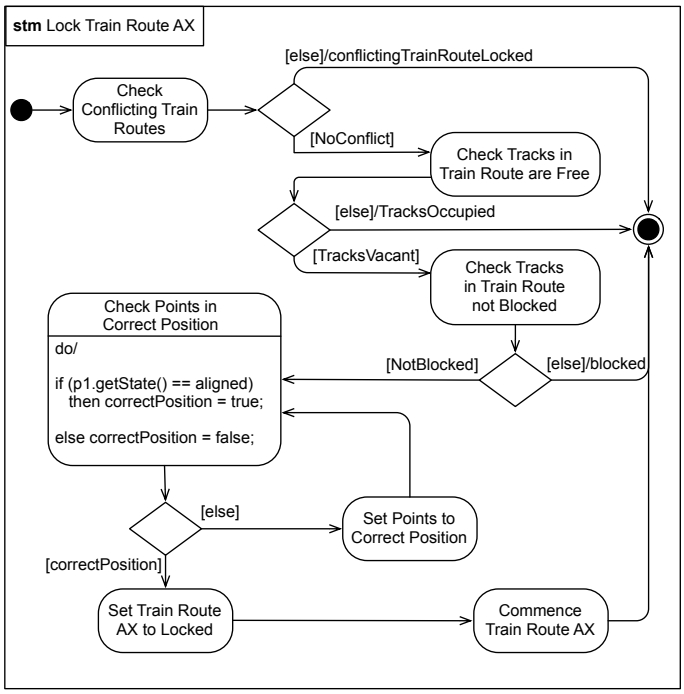

Figure 17. Excerpt of the system design specification – The behaviour exemplified

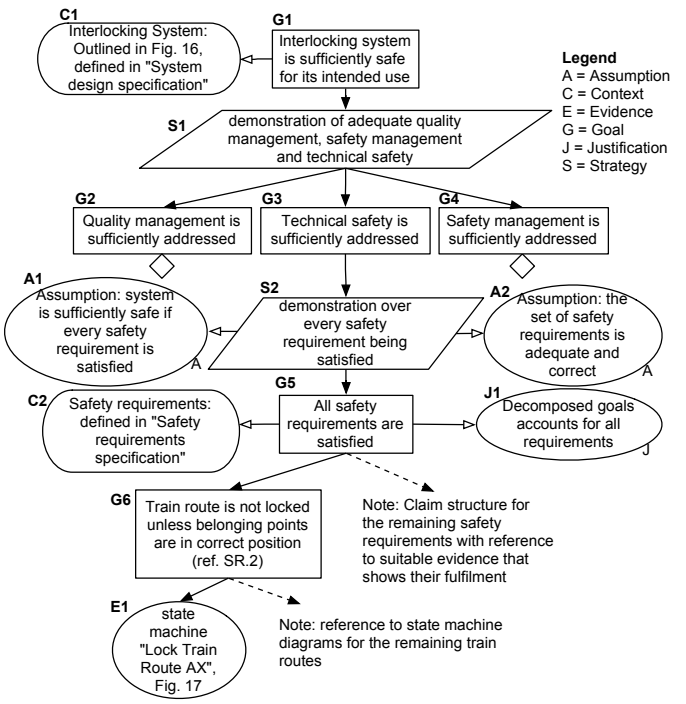

Figure 18. A safety case expressed with the GSN notation [15]

Requirements.

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. 12. 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. This fits well with the railway standard EN 50129 [17], which requires that these 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 of similar pattern compositions (the result of the step can be seen in Fig. 19), 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 Step (13) according to its instantiation rule to produce a safety case as presented in Fig. 18.

The overall claim (expressed in a goal element) in the safety case presented in Fig. 18 states that the proposed "interlocking system is sufficiently safe for its intended use." The overall claim is decomposed into sub-claims, which at some point is supported by evidence (referenced within an evidence element). A diamond symbolises that the goal is not developed. The claim structure is simplified to only account for requirement SR.2, which was defined earlier. Furthermore, the claim structure is also simplified to only reference evidence for the correct specification of required behaviour as is given within this paper. The relevant specification of behaviour with respect to arguing the fulfilment of requirement SR.2 is defined in Fig. 17.

Step (17): In this step the user specifies the end result of pattern composition. Fig. 19 extends Fig. 15 to also specify the use of the pattern *Overall Safety*. Fig. 19 includes the information intended to be specified at Step (15), which was postponed.

In Fig. 19, the parameters of the patterns introduced are abbreviated as follows: *S* is short for System, *ToD* is short for Target of Demonstration, and *Case* is short for Safety Case. The *satisfies* relation (see Fig. 6) 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 conceptual safety design of a railway interlocking system. The result is here partly represented by the specifications in Fig. 16, Fig. 17, and Fig. 18. The conceptual safety design in the example is the result of instantiating the composite pattern expressed in Fig. 19. In the composite, the triple that represents the conceptual safety design is assumed represented by the documentations referred to within Fig. 19 as *Safety requirements specification*, *System design specification*, and *Safety case specification*.

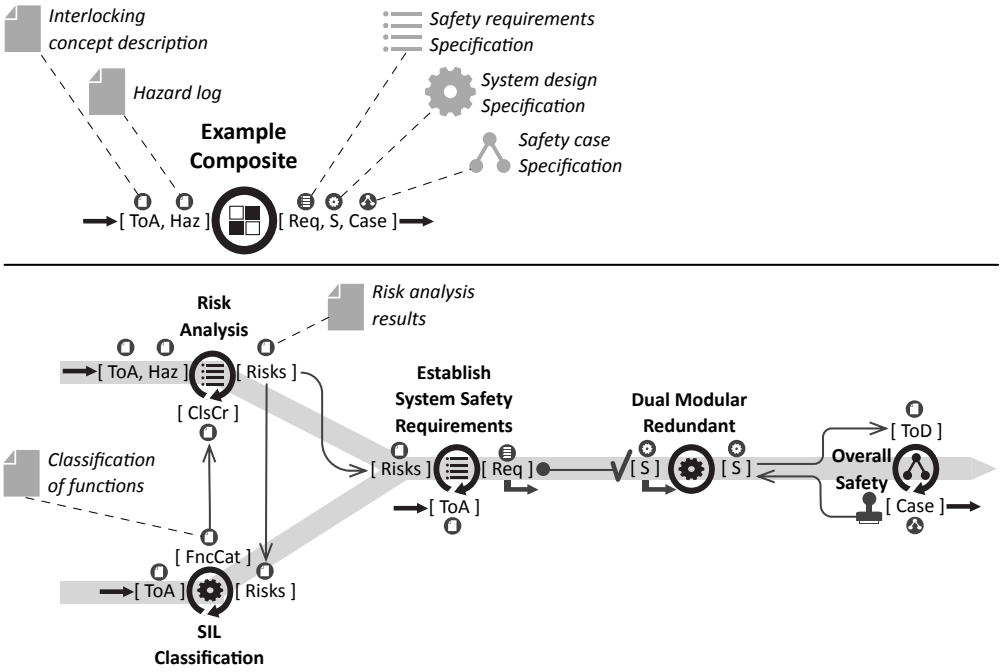

Figure 19. Composite specifying its instantiation into a conceptual safety design

IV. THE FRAMEWORK USED AS SUPPORT FOR THE ANALYTIC EVALUATION

Mendling et al. [18] 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 [18] is SEQUAL [9][19][20]. 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. [21] 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. [22] 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. [18] 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 [9] whose usefulness has been confirmed in experiments [21]. 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 [6] and [7].

According to SEQUAL, the appropriateness of a modelling language for a specific task is 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. 20 is adopted from [23] 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. 20 are related to different properties of the language under evaluation. The appropriateness factors are [9]:

- *Domain appropriateness*: the language should be able to represent all concepts in the domain.
- *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

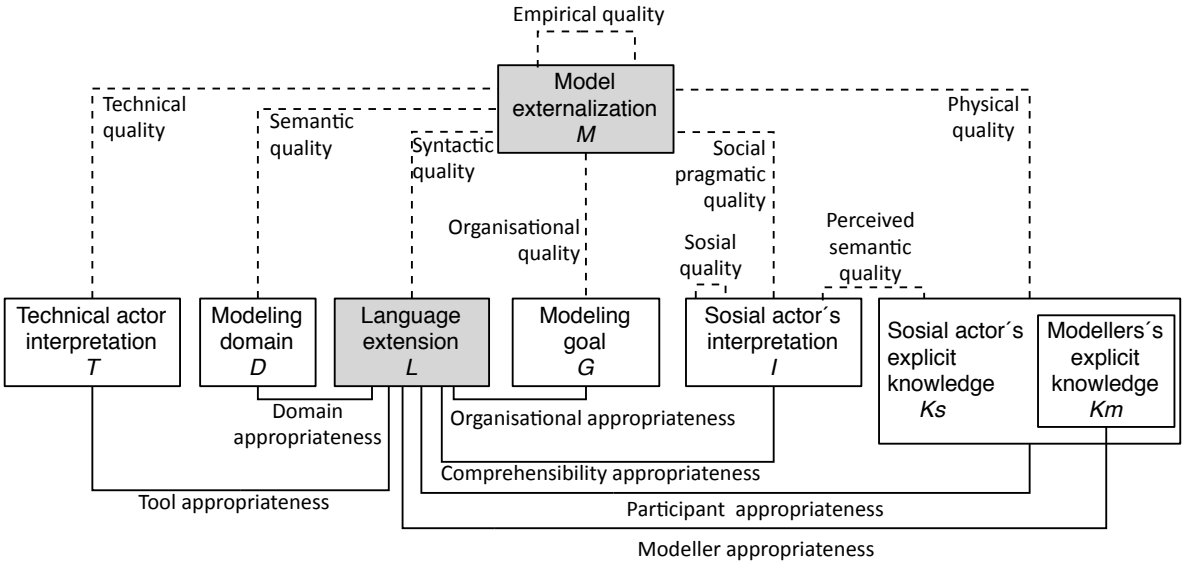

Figure 20. The quality framework (adopted from [23])

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.

V. THE ANALYTIC EVALUATION

A necessary step in the application of SEQUAL [9][19][23] 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.

The different appropriateness factors are addressed successively in Section V-A to Section V-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.

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.

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 [6][7] on different concepts within conceptual safety design. Each pattern can be referenced by its unique name. 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. 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. 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 [24] and in IEC 61508 [25] 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 [17]. The pattern captures the hazard identification part of the phase named "Hazard and risk analysis" of the safety life-cycle presented in IEC 61508 [25]. 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 [17]. The patterns in 2), 3), and 4) captures the intent expressed in the life-cycle phase named "Hazard and risk analysis" in IEC 61508 [25].

4) *Risk Analysis*: The pattern describes a process for assessing risk in accordance with the practises defined in EN 50129 [17] and IEC 61508 [25].

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 [24] named "System Requirements". In EN 50129 [17], 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 [25] 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 described in IEC 60812 [26].

7) *FTA*: The pattern captures the essence of the Fault Tree Analysis (FTA) method as it is described in IEC 61025 [27].

8) *I&C Functions Categorisation*: The pattern captures the method for classifying nuclear Instrumentation and Control (I&C) functions as defined within IEC 61226 [28].

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 [17], applicable for system functions.

10) *Variable Demand for Service*: The pattern is highly specialised to support requirements elicitation for the conceptualisation of a nuclear control system in a case study described in [6]. The case study describes a very specific development challenge and 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 in general.

11) *Station Interlocking Requirements*: The pattern captures the essential requirements for building interlocking systems as defined by the Norwegian Rail Authority in the technical rules JD 550 [30].

12) *Level Crossing Interlocking Requirements*: The pattern captures the essential requirements for building level crossing systems as defined by the Norwegian Rail Authority in the technical rules JD 550 [30].

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 [31]. 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 [32] consisting of two redundant controllers and a voting unit that is implemented in numerous kinds of systems for different kinds of task.

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 [17].

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 [17] 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 expressed in the European Regulation on common safety methods within the railway industry [33] and its associated application guideline [34].

18) *Cross Reference* [6]: 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 [33] and its associated application guideline [34].

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 [33] and its associated application guideline [34].

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 [17] and IEC 61508 [25].

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 suitable assessment techniques has been applied can be seen in Appendix A of EN 50128 [29]. Appendix A of EN 50128 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 is to claim that the software in a given system is developed according to the practices described in a relevant software standard (e.g., [29][35]) 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. Appendix A of EN 50128 [29] 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 [17][24][25][29][36].

The icons and symbols of the SaCS pattern language is presented in Section II-C. Fig. 4 presents the icons used for SaCS patterns within a composite pattern specification and indicates a categorisation. The first three icons are used for categorising patterns providing development guidance with a strong processual focus. The next three icons are used for categorising patterns providing development guidance with a strong product focus. The last icon is used for categorising composite patterns. 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.

Habli and Kelly [37] 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, motivates 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 [36] 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 [36]. 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 [15][36][37][38]. 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 described practices, selected keywords, and designed 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, e.g., IEC 61508 [25], typically demand a number of activities to be performed in which certain activities must be applied in a specific sequence. Furthermore, safety standards can also describe the expected inputs and outputs of different activities and in this sense describe what is the expected content of deliverables that allows a transition from one activity to the next. According to Krogstie [9], 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. 5 presents the icons for the different kinds of parameters and Fig. 7 presents the artefact references in SaCS. The *documentation parameter* and the *documentation artefact reference* types (represented visually by the icons presented in Fig. 5 and Fig. 7) are defined in order to allow a generic classification of parameters and artefacts that cannot be classified as requirement, design, or safety case. An example can 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 [17][25]. 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. 3. 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.

Fig. 6 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 [6][7] define the concepts behind the different relations and exemplify the practical use of all the concepts in different scenarios. Fig. 3 in Section II exemplifies a composite pattern containing five instances of the *instantiates* relation and two instances of the *assigns* relation.

The need for the different relations presented in Fig. 6 is motivated by the practices described in different standards and guidelines, e.g., IEC 61508 [25], 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* relation between patterns. When performing an activity like hazard analysis, the results from the application of a number of methods can be combined and used as input. Two widely used methods are captured in two different basic SaCS patterns known as 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 such as FMEA and FTA are combined into a union consisting of all individual results. 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 [25]. 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 support 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 design.

Fig. 8 illustrates how the intended instantiation order of patterns can 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 can be instantiated in parallel and thus have no specific order; this is visualised by placing pattern references on separate arrows.

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 [6] and [7]. The patterns are defined on the basis of safety engineering best practices as defined in international standards and guidelines [17][25][33][34][36][39] and other sources on safety engineering. The limited number of basic patterns currently available delimits 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 can easily extend the language. A basic pattern, e.g., the pattern *Hazard Analysis* [6] referenced in Fig. 3, is defined in a simple structure of named sections containing text and illustrations according to a common format. The format is thoroughly detailed in [10].

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. [3] 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 [3] is not needed. The format in [3] 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. [13] is also simple and well-known, but tailored specifically for capturing patterns for software design.

All in all, we admit that there can 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 [25] 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 [25], 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 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 can be defined prior to work initiation in order to define a plan for the application of patterns. A composite that is defined as a representation of a work plan can be easily reused for documentation purposes by adding information on the instantiation of parameters. Another use can be to refine a composite throughout the work process. This is exemplified in [6] and [7]. A composite can also be defined once patterns have been applied in order to document the work process.

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* [34], methods such as *fault tree analysis* [27] and *failure mode effects analysis* [26], system design solutions including *redundant modules* and voting mechanisms [32], and practices like arguing safety on the basis of arguing that safety requirements are satisfied [36], are all well known safety engineering practices that can be found in different standards and guidelines [17][25][39]. 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.

TABLE II. PATTERN FORMATS IN THE LITERATURE COMPARED TO BASIC SACS PATTERNS [10]

	[11]	[3]	[13]	[14]	[15]	[40]	[41]	[42]	[10]
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	✓	✓		✓		✓			✓

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 [43], 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 can 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. [44] 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 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. Fig. 4, Fig. 5, Fig. 6, Fig. 7, and Fig. 8 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 not to 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. 3 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. 5, parameters and artefact references as in Fig. 7, relations as in Fig. 6, and instantiation order as in Fig. 8. Guidance to the understanding of the language is provided in [10], where the syntax and the semantics of SaCS patterns are described in detail. The SaCS language comes with a structured semantics [10] 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 [6] and [7]. Although we have not tested SaCS on people unfamiliar with the language, we expect that users familiar with safety engineering can comprehend the concepts and the modelling on the basis of the descriptions in [6][7][10] 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. According to Larkin and Simon [45], the key activities performed by a reader

in order to draw conclusions from a diagram are: searching; and recognising relevant information.

Lidwell et al. [44] 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 [46] provides an extensive overview of the Gestalt principles of perception building upon classic work from Wertheimer [47] 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. [44], 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. 3. Although each referenced pattern has a unique name, their identical icons indicate relatedness. Different kinds of patterns are symbolised by the icons in Fig. 4. The icons are of the same size with some aspects of similarity and some aspects of dissimilarity such that a degree of relatedness can be perceived. An icon for pattern reference is different in shape and shading compared to an icon used for artefact reference, see Fig. 4 and Fig. 7, respectively. 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. [44]: category, time, location, alphabet, and continuum. The *category* refers to the organisation of elements by similarity and relatedness. An example of the application of the principle of categorisation [44] in SaCS is seen in the possibility to reduce the number of relations drawn between patterns when these are similar. Patterns in SaCS can have multiple inputs and multiple outputs as indicated in Fig. 3. 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. 21 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.

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. [44]. 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.

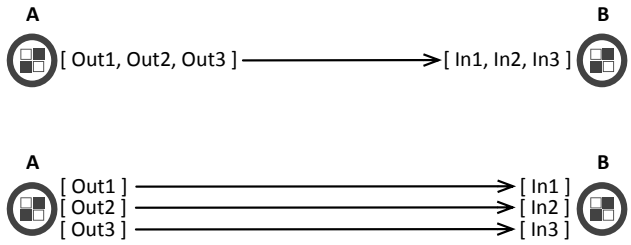

Figure 21. Alternative ways for visualising multiple similar relations

Fig. 22 presents a composite pattern named *Requirements* that reference other composites as part of its definition. The contained pattern *Safety Requirements* is defined in Fig. 3. The contained pattern *Functional Requirements* is not defined and is referenced within Fig. 22 for illustration purposes. *Requirements* can 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. 22. In Fig. 22, 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*.

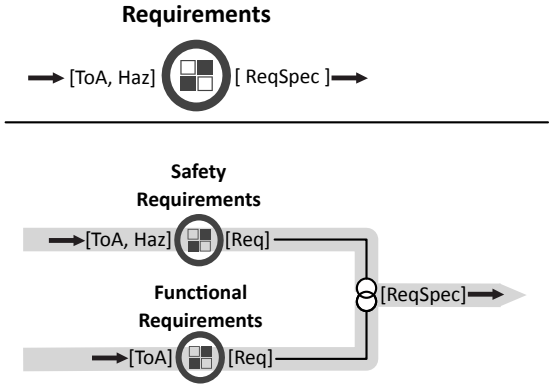

Figure 22. Composition of composites

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. [44] is to use colour with care as it can lead to misconceptions if used inappropriately. The authors points out that there is no universal symbolism for different colours. As colour blindness is common the SaCS language applies different shades of grey in visualisations.

Moody [48] 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 [48].
2) *Perceptual discriminability*: concerns to which extent different symbols are distinguishable [48].
3) *Semantic transparency*: concerns to which extent the meaning of a symbol can be inferred from its appearance [48].
4) *Complexity management*: concerns to which extent the visual notation is able to represent information without overloading the human mind [48].
5) *Cognitive integration*: concerns to which extent there are explicit mechanisms supporting the integration of information from different diagrams [48].
6) *Visual expressiveness*: concerns to which extent the full range of and capacities of visual variables are used [48].
7) *Dual coding*: concerns to which extent text is used to complement graphics [48].
8) *Graphic economy*: concerns to which extent the number of different graphical elements are cognitively manageable [48].
9) *Cognitive fit*: concerns to which extent visual dialects are used to support different tasks and audiences [48].

Fig. 23 presents how the principles 1-9 outlined above are implemented. Fig. 23 also presents how three of the Gestalt principles of visual perception [47][46][44] known as Figure-Ground, Proximity, and Uniform connectedness are implemented. The Gestalt principles express mechanisms for efficient human perception from groups of visual objects. In Fig. 23, 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.

Regarding 1): We have deliberately applied a symbol deficit approach, which influences semiotic clarity [48]. 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 [44]. There is also a limit of human capacity to process information that motivates a small number of symbols. The limit, which was mentioned earlier, is according to Miller [43] seven plus or minus two elements. 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 [48] by using the Gestalt principle of similarity to balance the need for similarity with the need for dissimilarity. The principle of similarity [44] 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. 3. Although each referenced pattern has a unique name, their identical icons indicate relatedness. Different kinds of patterns are symbolised by the icons in Fig. 4. 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. Textual differentiation is approached with different typefaces and font size.

Regarding 3): We described earlier that our aim when designing icons was to achieve a symbolic form according to the classification of Lidwell et al. [44]. The effect is icons that identify a concept at a higher level of abstraction than can be achieved with the preferable similar and example forms, but at least the arbitrary form is avoided where there is little or no relationship between a concept and its associated icon. Section II-C presents the main icons in SaCS and identifies the result of our efforts to achieve semantic transparency [48].

Regarding 4): Several different mechanism are used for complexity management [48] in SaCS. According to Lidwell et al. [44], 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 [49], GSN [50], and Problem Frames [51] 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 [52][44][53] is to balance the need for performance by the importance of preference in designing solutions. However, the choice of several languages challenges cognitive integration [48]. Kim et. al [54] 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 annotations added to the UML activity diagram presented in Fig. 2 represents the SaCS specific mechanism that allows cognitive integration. The annotations facilitates the user in mapping parameters with diagram elements and are briefly described in Section II-C, fully defined in [10].

Regarding 6): The degree of visual expressiveness is defined by the number of visual variables used in a language [48]. Bertin [55] 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 is to use colour with care

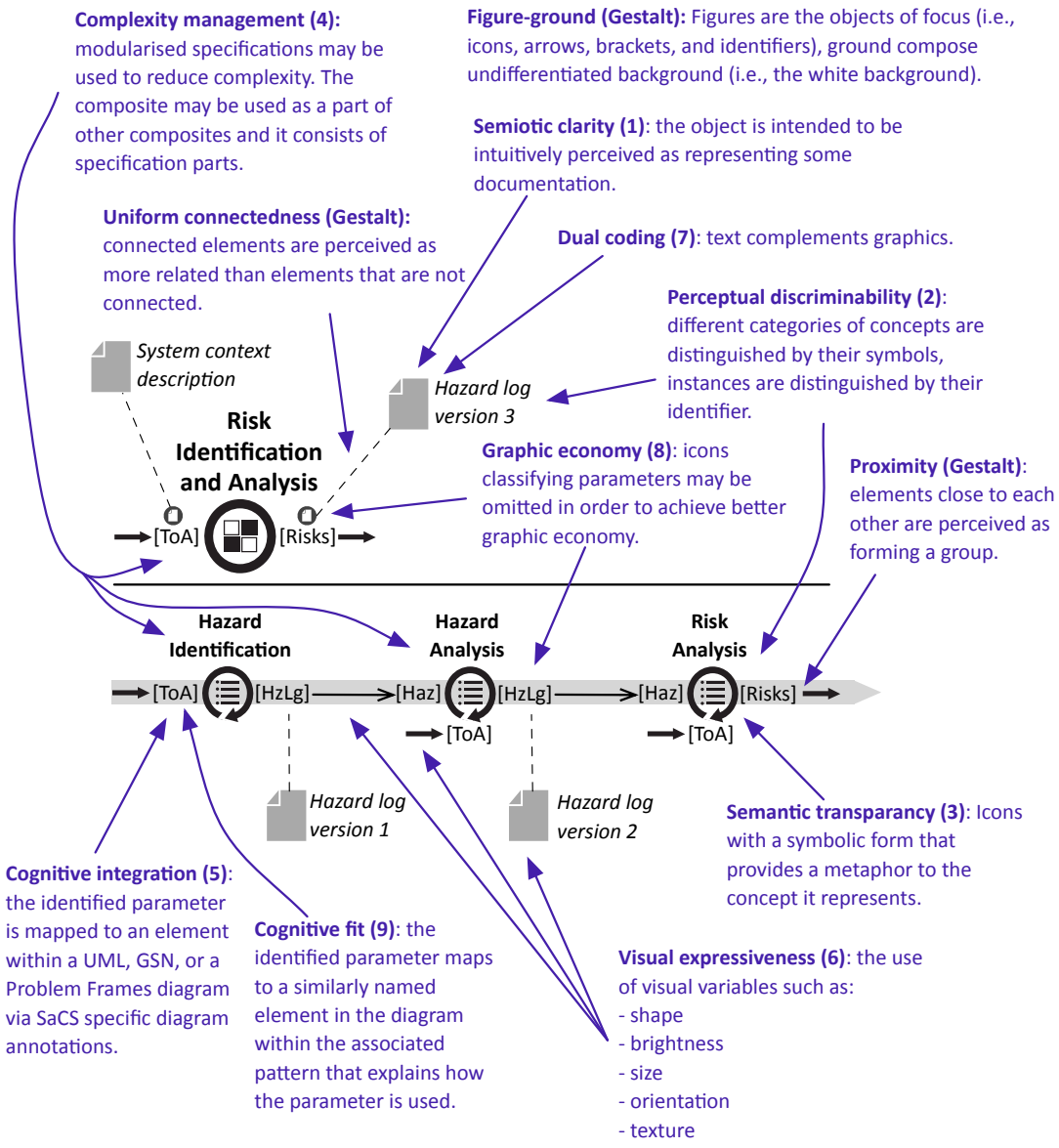

Figure 23. Example composite pattern with annotations showing the implementation of principles for cognitive effective visual notations

as it may lead to misconceptions if used inappropriately [44]. Moody [48] 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 [56] argues that within the dual coding theory, 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 [48]. 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. According to the cognitive fit theory [48], different kinds of information representation should be used depending on task and audience. In order to achieve an overall effective visual representation, visual dialects suited to the individual tasks should be integrated rather than providing one representation for all purposes. A

requirements engineer is expected to be aware of the Problem Frames [51] notation. A system engineer, hardware engineer, or a software engineer is expected to be aware of the UML [49] notation. A safety engineer is expected to be aware of the the GSN [50] notation. The SaCS pattern language integrates these visual dialects and facilitates the communication between users on the development of conceptual safety design.

E. Tool appropriateness

TA.1 The language must have a precise syntax.

The syntax of the SaCS language (see [10]) is defined in the EBNF [57] 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 [10] in the form of a schematic mapping from pattern definitions, via its textual syntax in EBNF [57], to English. The non-formal representation of the semantics supports human interpretation rather than tools, although the translation procedure as described in [10] can 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.

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., [17][25][39]. International safety standards and guidelines describe concepts and practices for development of safety critical systems that can 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 [6] and [7], 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. 3 eases the explanation of how several concepts within conceptual safety design are combined and applied.

Wong et al. [58] reviewed several large development projects and software safety standards from different domains with respect to cost-effectiveness. Their conclusion is that although standards provide useful and effective guidance, safety and cost-effectiveness objectives are met by effective planning and by applying safety engineering best practices evidenced in company best practices throughout the development life cycle. A composite pattern can be easily defined with the SaCS pattern language in order to capture a company specific practice. In order to capture different company practices, different compositions of patterns can be defined.

OA.3 The language must be usable without the need of costly tools.

Every pattern used in the cases described in [6] and [7] 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 [6][7][10] and in this paper is created with a standard drawing tool.

VI. Related Work

In the literature, pattern approaches supporting development of safety critical systems are poorly represented. In the following we briefly discuss some different pattern approaches and their relevancy to the development of conceptual safety designs.

Jackson [51] 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 [51] are defined on a very high level of abstraction.

The use of boilerplates [59][60] 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 can 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 [61] describes 37 requirements patterns for assisting the specification of different types of requirements. The patterns are defined at a low level, i.e., the level of a single requirement. The patterns of Withall can 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. [13] 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. [62] 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 [12] 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 can be relevant for assuring that the development process applied is suitable and leads to well informed design choices, Kelly [15] 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 [63] 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 [64] identifies the need to define relationships between system design patterns in order to efficiently combine them. Noble [65] builds upon the ideas of Zimmer and defines 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 [66] define a formal language for pattern composition. They 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. [13]. 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 [67] presents a catalogue of elementary software design patterns in the tradition of Gamma et al. [13] 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 of Smith [67] is comparable to the UML collaboration notation [49]. The main purpose of a UML collaboration 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 [68][69][70] for expressing patterns graphically use UML as its basis. The notations are simple, but target the specification of software.

VII. 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 can 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 [44][47].
- *Tool*: Tool support can be provided on the basis of the syntax and semantics of the SaCS language [10].
- *Organisational*: Organisations developing safety critical systems are assumed to follow a development process in accordance to what is required by standards. Wong et al. [58] 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 can 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 within the OECD Halden Reactor Project, Institute for Energy Technology, Halden, Norway. The second author has partly been funded by the ARTEMIS project CONCERTO.

REFERENCES

[1] 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). IARIA, 2014, pp. 79–88.

[2] F. Buschmann, K. Henney, and D. C. Schmidt, Pattern-Oriented Software Architecture: On Patterns and Pattern Languages. Wiley, 2007, vol. 5.

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

[4] 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.

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

[6] 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 rev 2, 2014.

[7] A. A. Hauge and K. Stølen, "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 rev 2, 2014.

[8] K. M. Eisenhardt, "Building theories from case study research," The Academy of Management Review, vol. 14, no. 4, 1989, pp. 532–550.

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

[10] 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.

[11] 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.

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

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

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

[15] T. P. Kelly, "Arguing Safety – A Systematic Approach to Managing Safety Cases," Ph.D. dissertation, University of York, United Kingdom, 1998.

[16] 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.

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

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] 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.

[23] I. Hogganvik, "A Graphical Approach to Security Risk Analysis," Ph.D. dissertation, Faculty of Mathematics and Natural Sciences, University of Oslo, 2007.

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

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

[26] IEC, "IEC 60812 Analysis Techniques for System Reliability – Procedure for Failure Mode and Effects Analysis (FMEA), 2nd edition," International Electrotechnical Commission, 2006.

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

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

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

[30] Jernbaneverket, "Teknisk Regelverk, JD550 Prosjektering," https://trv.jbv.no/wiki/Signal/Prosjektering, 2014, [accessed: 2014-08-31].

[31] L. Sha, "Using Simplicity to Control Complexity," IEEE Software, vol. 18, 2001, pp. 20–28.

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

[33] 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.

[34] 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.

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

[36] 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: 2014-08-31].

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

[38] 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.

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

[40] 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.

[41] 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.

[42] 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.

[43] 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.

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

[45] 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.

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

[47] 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.

[48] D. L. Moody, "The "Physics" of Notations: Towards a Scientific Basis for Constructing Visual Notations in Software Engineering," IEEE Transactions on Software Engineering, vol. 35, no. 6, 2009, pp. 756–779.

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

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

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

[52] R. W. Bailey, "Performance versus Preference," in Proceedings of 37th Annual Meeting of the Human Factors and Ergonomics Society, vol. 37, 1993, pp. 282–286.

[53] J. Nilsen and J. Levy, "Measuring Usability: Performance vs. Preference," Communications of the ACM, vol. 37, no. 4, 1994, pp. 66–75.

[54] 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, vol. 11, no. 3, 2000, pp. 284–303.

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

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

[57] ISO/IEC, "14977:1996(E) Information Technology - Syntactic Metalanguage - Extended BNF," International Organization for Standardization / International Electrotechnical Commission, 1996.

[58] 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.

[59] E. Hull, K. Jackson, and J. Dick, Requirements Engineering, 3rd ed. Springer, 2010.

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

[61] S. Withall, Software Requirement Patterns (Best Practices), 1st ed. Microsoft Press, 2007.

[62] 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.

[63] 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.

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

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

[66] 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.

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

[68] 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.

[69] 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.

[70] J. M. Vlissides, "Notation, Notation, Notation," C++ Report, 1998, pp. 48–51.