Constructing Product-Line Safety Cases from Contract-Based Specifications

Damir Nešić
Royal Institute of Technology
Stockholm, Sweden
damirn@kth.se

Mattias Nyberg
Royal Institute of Technology
Stockholm, Sweden
matny@kth.se

Barbara Gallina
Mälardalen University
Västerås, Sweden
barbara.gallina@mdh.se

ABSTRACT

Safety cases are used to argue that safety-critical systems satisfy the requirements that are determined to mitigate the potential hazards in the systems operating environment. Although typically a manual task, safety cases have been successfully created for systems without many configuration options. However, in highly configurable systems, typically developed as a *Product Line* (PL), arguing about each possible configuration, and ensuring the completeness of the safety case are still open research problems. This paper presents a novel and general approach, based on *Contract-Based Specification* (CBS), for the construction of a safety case for an arbitrary PL. Starting from a general CBS framework, we present a PL extensions that allows expressing configurable systems and preserves the properties of the original CBS framework. Then, we define the transformation from arbitrary PL models, created using extended CBS framework, to a safety case argumentation-structure, expressed using the *Goal Structuring Notation*. Finally, the approach is exemplified on a simplified, but real, and currently produced system by Scania CV AB.

CCS CONCEPTS

• **Software and its engineering → Software product lines**; *Software safety*;

KEYWORDS

Safety case, Product line engineering, Contract-based specification

ACM Reference Format:
Damir Nešić, Mattias Nyberg, and Barbara Gallina. 2019. Constructing Product-Line Safety Cases from Contract-Based Specifications. In *The 34th ACM/SIGAPP Symposium on Applied Computing (SAC '19), April 8–12, 2019, Limassol, Cyprus.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3297280.3297479

1 INTRODUCTION

Many *software-intensive systems of systems* are also safety critical. Failures of systems from domains like automotive, aerospace, or medical, can cause harm to property or humans. Consequently, these domains are regulated with domain-independent *functional safety* standards such as IEC 61508 [8], or with domain-specific standards such as ISO 26262 [15] in automotive, or DO 178C [14] in aerospace. The standards emphasize structured and traceable requirements engineering as the backbone of the development process, and require the creation of a *safety case* [34] as a proof of compliance with the standard. A safety case is a structured argument, supported by a body of evidence, which should argue about two claims. Firstly that the defined safety requirements are *complete* with respect to the potential *hazards* in the intended operating environment, and secondly that the system satisfies the defined safety requirements. The phrase *structured* refers to the breakdown of more abstract arguments, e.g. *"the system is safe"*, into more detailed *sub-arguments* about the system properties, thus forming an *argumentation structure*.

Because safety cases are large and complex artifacts, their construction and maintenance are notoriously difficult activities, which are typically performed manually [22, 27, 30]. For any reasonably sized system, just listing all potential evidence, e.g. safety analyses, or testing results, leads to a huge amount of information. Moreover, because a variety of methods with different semantics are used to specify, design, implement, and verify different parts of a system, it is especially difficult to create a *complete* argumentation structure. In other words, ensuring that each argument is true if the supporting sub-arguments are true is challenging because typically a holistic model of a system against which all arguments can be evaluated does not exist.

Despite the difficulties, safety cases have been created successfully in some domains, e.g. aerospace, but in other domains, e.g. automotive, their creation is still a research problem. The caveat separating these domains is the number of possible product configurations. Highly configurable systems are typically developed using the *Product Line Engineering* (PLE) paradigm [1, 29], which puts emphasis on high levels of reuse and quick creation of new product configurations. From the point of safety case construction, if the number of product configurations is low, like in the aerospace domain, then it is feasible to create a safety case for each product configuration. On the contrary, when the number of product configurations reaches hundreds of thousands [6], like in the automotive domain [42], this is not feasible because of two reasons. Firstly, it is impossible to produce verification evidence for each product configuration, and secondly the amount of manual work required to construct the argumentation structure for each safety case is overwhelming [26].

In this paper we propose a general and novel approach for the construction of a safety case for a complete *Product Line* (PL), instead of constructing a safety case per product configuration. The

SAC '19, April 8–12, 2019, Limassol, Cyprus

ACM ISBN 978-1-4503-5933-7/19/04…$15.00
https://doi.org/10.1145/3297280.3297479

safety case construction is based on the *Contract-Based Specification* (CBS) [5, 7, 40] model of a PL, where the CBS model captures the technical architecture, and the corresponding safety requirements, of each product configuration. In order to support such modeling of PLs, the first contribution is an extension of a general *assume-guarantee* CBS framework with PL concept of *presence conditions* [38]. Then, as a second contribution, the paper defines a set of transformation rules that convert an arbitrary CBS model of a PL, into a PL safety case expressed in a well-established safety-case notation, the *Goal Structuring Notation* (GSN) [28].

There are several reasons for relying on the CBS framework. From the methodological view, CBS frameworks match well with the requirement engineering approach emphasized by functional safety standards [5, 40], and thus stand a realistic chance for industrial adoption. Furthermore, CBS frameworks are *formal* and *general-purpose*, i.e. they can be used to rigorously and holistically model arbitrary systems at arbitrary abstraction levels thus removing the need to use a variety of methods and languages. Finally, CBS frameworks support *sound compositional reasoning*, i.e. properties of the modeled systems follow from the properties of their constituent components. In the PL context this means that the effort of verifying the behavior of all product configurations is reduced to verifying the behavior of the components that comprise them.

Several previous approaches aim at constructing a safety case for highly configurable system [10, 25, 35, 37]. The approach in the present paper differs in two main aspects. Firstly, because the CBS framework defines sufficient conditions for a system property to hold, e.g. a safety property, and because the safety case is based on a CBS model of a PL, the argumentation structure obtained from a CBS model is *complete-by-construction*, i.e. it is ensured that each argument holds if the supporting sub-arguments hold. Previous approaches typically start from a *safety case pattern* [11] and try to populate the pattern with system-specific information. However, they do not offer a method to reason about the completeness of the obtained argumentation structure. Secondly, we leverage the *sound compositional reasoning* of the CBS framework in order to construct an argumentation structure that directly argues about the safety of all product configurations of a PL, but only requires verifying the behavior of the components comprising the product configurations. In contrast, previous approaches construct argumentation structures on a per-configuration basis, and typically require physical creation and verification of each product configuration behavior. As discussed earlier, this is not feasible when the number of product configurations is high. A detailed discussion about related work, can be found in Section 6.

Paper Structure. Section 2 summarizes the basics of PLE and CBS. Section 3 presents the proposed PL extension of the general CBS framework. Section 4 defines the overall safety case argumentation-structure and defines the CBS to GSN transformation rules. Section 5 presents a real example which is modeled using CBS and for which a safety case is constructed. Section 6 discusses the related work while Section 7 concludes the paper.

2 BACKGROUND

This section summarizes the basic concepts of the PLE paradigm and CBS frameworks from existing literature.

2.1 Product Line Engineering

Product line engineering [1, 29], facilitates the development of a family of systems that are jointly referred to as a *product line*. The central idea of PLE is to declare all functional and non-functional characteristics of each system in the PL, commonly referred to as *features*, and express these, together with any mutual dependencies, in a model where the most commonly used model is a *feature model*.

Definition 2.1 (Feature model). A *feature model* is a pair $\mathcal{V} = (F, \mathfrak{C})$ where $F = \{f_1, \ldots, f_n\}$ is a set of Boolean and Real variables, called *features*, and $\mathfrak{C}$ is a set of Boolean constraints over the features in F. □

An example of a constraint is $f_1 \wedge f_2 > 100 \rightarrow f_3$. A value assignment to each of the features in F is referred to as a product configuration.

Definition 2.2 (Product configuration). Given a feature model $\mathcal{V}$, a *product configuration* γ is a set feature-value assignments $\gamma = \{f_i = value_k\}_{i=1}^{n}$. A product configuration for which each constraint in $\mathfrak{C}$ evaluates to true is *valid*. □

From hereon, terms *product configuration* and *configuration* will be used interchangeably and only *valid* configurations will be considered. For some intuition, real-world PLs often contain feature models with thousands of features and consequently define hundreds of thousands of valid configurations [6, 42].

Given a feature model, development artifacts are labeled with formulas expressed in terms of features and these formulas are known as *presence conditions*, denoted φ. They are written using the standard logical operators $\wedge, \vee, \neg$, arithmetic relations $>, <, =, \leq, \geq$, and their combinations. The purpose of presence conditions is to define the configurations to which a development artifact applies. By selecting a particular configuration, the artifacts that describe or implement the selected configuration are those whose presence conditions evaluate to true for the given feature value-assignment. In this way, a real-world product of a particular configuration can be *derived* automatically by selecting a product configuration.

2.2 Basic CBS framework

In order to support rigorous design of complex and heterogeneous systems, several lines of research have presented CBS frameworks [4, 5, 7, 40, 41]. As noted in [4], the main strength of CBS is that it formally captures two central systems engineering principles: vertical design *refinement* during the development process, and horizontal *composition* of logical or physical components at a given abstraction level. This section summarizes the CBS concepts from [5, 39], but with a slightly different notation, and with an emphasis on the use of CBS for requirements engineering.

The CBS concepts will be illustrated on a fragment from the *Fuel Level Display* (FLD) system (cf. Section 5), which is a part of each Scania CV AB vehicle and which is later used as the application example. The overall FLD system safety-requirement is to ensure that the *fuel volume* indicated on the vehicle's *instrumentation cluster*, with some tolerances, corresponds to the actual fuel volume in the *fuel tank*. This is a safety requirement because indicating higher fuel volume that the actual can lead to running out of fuel in traffic. This in turn leads to engine stop and the loss of servo-steering which is essential for heavy vehicles due to heavy loads.

2.2.1 CBS concepts. Basic concepts of a CBS framework are:

i) *component* C,
ii) *specification* S,
iii) component *implements* a specification, denoted as $C \rhd S$
iv) specification S_i *fulfills* specification S_j, denoted $\texttt{full}(S_i, S_j)$,
v) n-ary component *composition* that results in new components, denoted as $C' = C_1 \otimes \cdots \otimes C_n$.

A *specification* corresponds to a requirement and it defines the intended behavior of a component which represents any logical or physical component. While here we refer to requirements as *"specifications"*, other approaches refer to them as *"assertions"*. The notion *implements* is similar across the majority of CBS frameworks and it corresponds to the expectation that the implementation of a component will exhibit the behavior defined in a specification. The notion of *fulfillment* between S_i and S_j represents the intention that the property expressed in S_i logically *entails* the property expressed in S_j. From this it follows that if a C component implements specification S_i then it will also implement specification S_j, i.e. $\forall C.C \rhd S_i \rightarrow C \rhd S_j$. In [3], the term *refinement* is used to describe the same concept. Finally, *composition* correspond to the process of integrating existing components into new components.

Given the above concepts, we introduce the following definitions.

Definition 2.3 (Contract). A contract K is an ordered pair of specifications, denoted (A, G), where A is called an *assumption* and G is called a *guarantee*. □

Definition 2.4 (Satisfy Contract). Component C satisfies a contract (A, G), denoted as $C \blacktriangleright (A, G)$, if $\forall C_e.C_e \rhd A \rightarrow C_e \otimes C \rhd G$. □

The main idea behind contracts is separating responsibilities. Given a contract (A, G), a component C can be developed independently of other components, with respect to the contract. Whenever C is composed with a component C_e, often referred as the *environment* of C, which implements A, then the composition of C and C_e implements G. In order to ease the following exposition, we assume that each contract is intended to be satisfied by a single component. This intention is expressed by saying that a contract K_i is *allocated to* a component C_j, denoted $\texttt{allTo}(K_i, C_j)$.

In order to enable that independently developed components can be composed in any order, we assume that the composition operator is commutative and associative.

Assumption 1. *The $\otimes$ operator is commutative and associative.* □

We refer to components that are not composed of other components as *atomic components* and we refer to components composed of other components as *composite components* and say that a composite component has *subcomponents*. Atomic components usually represent low-level implementation such as SW, HW, or mechanical components, but depending on the considered level of abstraction, a SW module can be an atomic component while in other cases a single SW function can be an atomic component. On the other hand, composite components can represent everything from a high level system's functionality, down to a composition of two SW functions. From hereon, we introduce the shorthand $C' = \bigotimes_{i=1}^{n} C_i$ that replaces $C' = C_1 \otimes \cdots \otimes C_n$ notation.

Figure 1 graphically illustrates the CBS concepts introduced so far. Components are represented as rectangles, e.g. composite component C_{FLD} has subcomponents C_{COO} and C_{BMS} that has further subcomponents C_{B-EST} and C_{B-CAN}. Contracts are depicted as white oval-like shapes and if a contract is overlaid over a component, that corresponds to the *allocated to* relation, e.g. $\texttt{allTo}((A', G'), C_{\text{FLD}})$. Each *fulfills* relation is represented by a line ending with an arrow, while the lines ending with a filled circle represent the *assumption of* relation which is *intrinsic* in each contract. Finally, the same specification can belong to multiple contracts, e.g. guarantee G_5 in contracts (A_5, G_5) and (A_6, G_5).

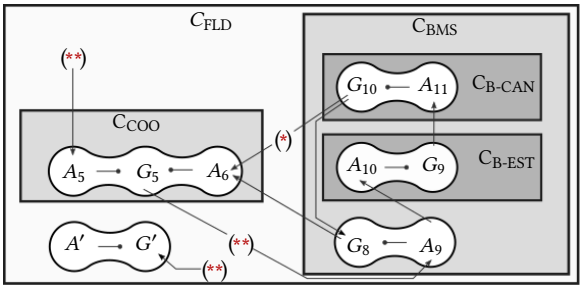

Figure 1: Graphical representation of CBS concepts.

2.3 Specification structure

The assumptions and guarantees connected with *fulfill* and *assumptionOf* relations in Figure 1 form a graph that will be referred to as a *specification structure*. We will also say that a set of contracts and a set of components form a specification structure. The intuition behind the specification structure is to capture the idea that assumptions can be fulfilled only by guarantees of sibling components or assumptions of parent components while guarantees can only be fulfilled by guarantees of subcomponents.

Definition 2.5 (Specification structure). Let $\mathcal{C}$ be a possibly empty set of components organized into a rooted tree with the root component $C' \in \mathcal{C}$ such that $C' = \bigotimes_k C_k$ where $C_k \in \mathcal{C} \setminus \{C'\}$. Let $\mathcal{K}$ be a possibly empty set of contracts such that each contract $(A_i, G_j) \in \mathcal{K}$ is allocated to a single component from $\mathcal{C}$. Then, a *specification structure* $\mathfrak{D}$ for $\mathcal{C}$ and $\mathcal{K}$ is a directed graph, i.e. $\mathfrak{D} = (\mathcal{N}, \mathcal{E})$, if

i) each node $n \in \mathcal{N}$ is a specification A_i or G_j,
ii) each edge $e \in \mathcal{E}$ corresponds to a single *fulfill* or *assumptionOf* relation between two specifications, and is denoted as $\langle S_i, S_j \rangle_f$ or $\langle S_i, S_j \rangle_a$, respectively,
iii) for each $\langle S_i, S_j \rangle_a$ and $\langle S_i, S_j \rangle_f$ it holds that $S_i \neq S_j$,
iv) for each edge $\langle S_i, S_j \rangle_a$ it holds that the S_i and S_j form a contract $(S_i, S_j) \in \mathcal{K}$,
v) for each edge $\langle S_i, A_j \rangle_f$ it holds that
  a) if S_i is an assumption then it holds that $\texttt{allTo}((S_i, G_l), C_m)$, $\texttt{allTo}((A_j, S_u), C_p)$ and C_p is a subcomponent of C_m,
  b) if S_i is a guarantee then it holds that $\texttt{allTo}((A_l, S_i), C_m)$, $\texttt{allTo}((A_j, S_u), C_p)$, and there exists a component C_x whose subcomponents are C_m and C_p.
vi) for each edge $\langle S_i, G_j \rangle_f$ it holds that S_i is a guarantee such that $\texttt{allTo}((A_l, S_i), C_m)$, $\texttt{allTo}((A_u, G_j), C_p)$, and C_m is a subcomponent of C_p. □

Def. 2.5 corresponds to *contract structure* [40] and it provides an intuitive way to visualize how the property in the form of a

guarantee G' allocated to the component representing the system C', is partitioned across the subcomponents of C'. Moreover, the specification structure matches well with the ideas in functional safety standards, such as ISO 26262, which require the decomposition of higher level requirements into lower level requirements with explicitly managed traceability links. The example in Figure 1 is not a specification structure because relation $\texttt{full}(G_{10}, A_6)$ violates condition (v)-b of Def. 2.5, i.e. C_{COO} and C_{B-CAN} are not subcomponents of the same composite component. In a real system, this correspond to the scenario where the specification of SW component C_{B-EST} defines that C_{B-EST} and C_{COO} can communicate directly, thus circumventing the interface of C_{BMS}.

The following conditions are necessary to ensure that satisfying the contract allocated to the system is a consequence of satisfying the contracts allocated to the atomic components.

Definition 2.6 (Proper specification structure). A specification structure $\mathfrak{D}$ is *proper* if

i) for each A_i of a contract K_l such that $\texttt{allTo}(K_l, C_m)$, there exists a specification S_k and an edge $\langle S_k, A_i \rangle_f$, where S_k is either an assumption such that $\texttt{allTo}((S_k, G_j), C_p)$ and C_m is a subcomponent of C_p or S_k is a guarantee such that $\texttt{allTo}((A_j, S_k), C_p)$ and there exists a component C_x whose subcomponents are C_m and C_p,
ii) for each guarantee G_i of a contract K_l such that $\texttt{allTo}(K_l, C_m)$, there exits a guarantee G_k and an edge $\langle G_k, G_i \rangle_f$, where $\texttt{allTo}((A_u, G_k), C_j)$ and C_j is a subcomponent of C_m,
iii) $\mathfrak{D}$ is acyclic. □

Going back to the example in Figure 1, even if the relation $\texttt{full}(G_{10}, A_6)$ would be removed, the example would still not be a proper specification structure because of all the relations labeled (∗∗). Relation $\texttt{full}(G_5, A_9)$ introduces cycles, which violates condition (iii) of Def. 2.6, while the absence of fulfill relation to guarantees A_5 and G' violates conditions (i) and (ii), respectively.

Assuming that the component composition is monotonic [41], the following theorem shows the key idea of CBS, i.e. the *sound compositional reasoning* about system properties and it corresponds to the *dominance relation* defined in [21].

Theorem 2.7. *Given a set of components $C_i \in \mathbf{C}$, a set of contracts $K_i \in \mathcal{K}$ allocated to components from $\mathbf{C}$ where $\mathcal{K}_A \subseteq \mathcal{K}$ is the set of contracts allocated to atomic components C_i, a contract $K' \in \mathcal{K}$ such that $\texttt{allTo}(K', C')$ where C' is the root component, if*

i) $\forall K_i \in \mathcal{K}_A . \exists C_i \in \mathbf{C} . C_i \blacktriangleright K_i$,
ii) contracts in $\mathcal{K}$ and components in $\mathbf{C}$ form a proper specification structure,
iii) $\forall \texttt{full}(S_i, S_j) . S_i \models S_j$,

then it holds that: $C' \blacktriangleright K'$. □

The theory presented so far does not support specifying systems, i.e. composite components, that are configurable. In the next section, the presented theory is extended with PLE constructs.

3 CBS EXTENSION FOR PLE

This section presents the first contribution, which is a PL-oriented extension of the theory presented in Section 2.2 and Section 2.3. Before the formal definitions we provide some intuition.

Consider using the CBS theory presented in the previous section for the development of a system with multiple configurations. This implies that each configuration would have to be specified as a specification structure $\mathfrak{D}$, as indicated by the upper part of Figure 2, and the reasoning if each configuration satisfies the allocated contract would be per-configuration. In order to avoid this, specification structures of individual system configurations can be merged into a common *PL specification structure* and then the reasoning about all system configurations can be performed simultaneously.

The CBS model in the lower part of Figure 2 is obtained by merging the two configurations from the upper part of the figure where the dashed component borders indicate that such components belong only to some configurations of C_{FLD}. As indicated in Figure 2, presence conditions φ_i define the set of configurations to which a particular component or a specification belongs to.

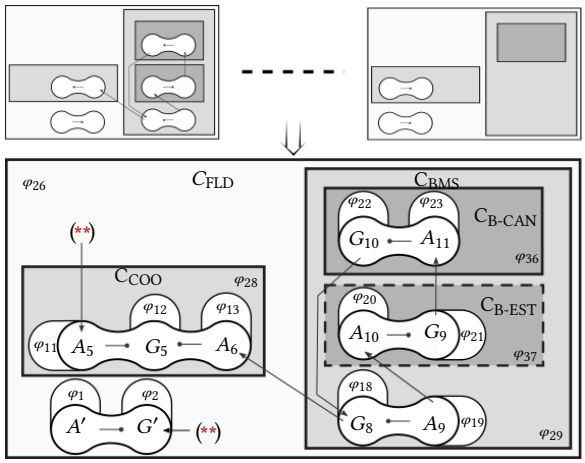

Figure 2: Example of a CBS model representing several configurations of the C_{FLD} system.

In order to ensure that each configuration represented by the merged CBS model conforms to Def. 2.5 and Def. 2.6, different mismatches between presence conditions, and consequently between the corresponding artifacts, must be prevented. For example, in Figure 2, guarantee G_8 fulfills assumption A_6, and because component C_{BMS} is intended to satisfy the contract (A_9, G_8), presence conditions $\varphi_{18}, \varphi_{19}, \varphi_{29}$ and φ_{13} must simultaneously evaluate to *true* or *false* for each configuration.

In the following two subsections, we formally define the PL specification structure and we define the constraints to which presence conditions must conform.

3.1 CBS model of a configurable system

Assume that a feature model of a PL defines n configurations $\gamma_1, ..., \gamma_n$. In order to avoid creating n different specification structures, we introduce the *PL specification structure* $\widehat{\mathfrak{D}}$, which represents each of the n configurations.

Definition 3.1 (PL Specification Structure). Let $\mathbf{C}$ be a possibly empty set of components organized into a rooted tree with the root component $C' \in \mathbf{C}$ such that $C' = \bigotimes_k C_k$ where $C_k \in \mathbf{C} \setminus \{C'\}$.

Let $\mathcal{K}$ be a possibly empty set of contracts such that each contract $(A_i, G_j) \in \mathcal{K}$ is allocated to a single component from C. Then, a *specification structure* $\mathfrak{D}$ for C and $\mathcal{K}$ is a directed graph, i.e. $\mathfrak{D} = (\mathcal{N}, \mathcal{E})$, if

i) conditions (i)-(vi) of Def 2.5 hold,
ii) a labeling function P labels each assumption A_i, guarantee G_j, and component C_k with a presence condition φ_i, φ_j, φ_k, respectively. □

Similarly to the basic CBS framework, we define a *proper* PL specification structure.

Definition 3.2 (Proper PL specification structure). A PL specification structure $\widehat{\mathfrak{D}}$ is *proper* if conditions (i) and (ii) of Def. 2.6 hold. □

Because a proper PL specification structure is a superimposition of several configuration-specific specification structures, in the general case, it is possible that the graph of $\widehat{\mathfrak{D}}$ contains cycles and therefore the condition (iii) from Def. 2.6 does not apply to Def. 3.2. However, because allowing circularity in proper PL specification structures does not bring conceptual benefits, but increases the complexity of the following exposition, we introduce the following assumption.

Assumption 2. *A proper PL specification structure $\widehat{\mathfrak{D}}$ is* acyclic. □

Before the constraints on presence conditions are defined, we introduce additional notation. $P(S)$ and $P(C)$ will be used to denote the presence condition of specification S or a component C while the notation $P_\gamma(S)$ will be used to denote the *true* or *false* value of the presence condition of S or C for a configuration γ.

3.2 Constraints on presence conditions

As outlined in Section 2.1, artifacts are labeled with presence conditions in order to be able to determine the set of artifacts that apply to a certain configuration γ by evaluating each presence condition and selecting the ones whose presence conditions evaluate to *true*.

We say that a PL specification structure $\widehat{\mathfrak{D}}$ is *instantiated* for a configuration γ, if each A, G, and C whose presence condition evaluate to *false* for a given γ, is removed from $\widehat{\mathfrak{D}}$. Moreover, if the assumption or a guarantee of a contract K is removed, then the contract K is also removed. The result of *instantiation*, denoted as $\mathfrak{D}_\gamma$ is a directed graph which is not necessarily a specification structure according to Def. 2.5 because of the previously described presence condition mismatches. In order to ensure that instantiation results in a specification structure, we introduce the following definitions and we refer to them as *invariance with respect to configurations*, hereinafter only *invariant*. In the following definitions, the symbol $\models_{\mathfrak{C}}$ represents logical entailment with respect to the set of constraints in the set $\mathfrak{C}$ of a feature model $\mathcal{V}$. More formally, $f_i \models_{\mathfrak{C}} f_j$ is equivalent to $f_i, \mathfrak{C} \models f_j$. For example, it holds that $f_1 \wedge f_2 \not\models f_3$ but if the set $\mathfrak{C}$ contains the constraint $f_1 \rightarrow f_3$ then it holds that $f_1 \wedge f_2 \models_{\mathfrak{C}} f_3$.

Definition 3.3 (Invariant Assumption). Given a specification S and set of specification $S_1, \ldots, S_n$ where S is the assumption of contracts $(S, S_1) \ldots (S, S_n)$, assumption S is *invariant* if $P(S) \models_{\mathfrak{C}} \bigvee_{K=(S,S_i)} P(S_i)$. □

Definition 3.4 (Invariant Guarantee). Given a specification S and set of specification $S_1, \ldots, S_n$ where S is the guarantee of contracts $(S_1, S) \ldots (S_n, S)$, guarantee S is *invariant* if $P(S) \models_{\mathfrak{C}} \bigvee_{K=(S_i,S)} P(S_i)$. □

Definition 3.5 (Invariant Allocation). An *allocation* of a contract (A, G) to a component C where A and G are an invariant assumption and guarantee, respectively, is *invariant* if $P(G) \models_{\mathfrak{C}} P(C)$ and equally if $P(A) \models_{\mathfrak{C}} P(C)$. □

Definition 3.6 (Invariant Composition). Composition of atomic components $C_1, ..., C_n$ into a composite component $C' = \bigotimes_i C_i$ is *invariant* if it holds that $\forall C_i . P(C_i) \models_{\mathfrak{C}} P(C')$. □

Given a PL specification structure $\widehat{\mathfrak{D}}$ with invariant assumptions, guarantees, allocation, and composition, instantiating $\widehat{\mathfrak{D}}$ for an arbitrary configuration γ, results in a specification structure according to Def. 2.5.

Theorem 3.7. *Given a set of configurations $\{\gamma_i\}_{i\in\mathbb{N}}$ and a PL specification structure $\widehat{\mathfrak{D}}$, if*

i) each assumption is invariant,
ii) each guarantee is invariant,
iii) each allocation of a contract (A_i, G_j) is invariant,
iv) component composition is invariant,

then each instantiation $\mathfrak{D}_{\gamma_i}$ is a specification structure. □

Theorem 3.7 established the conditions under which instantiating the PL specification structure $\widehat{\mathfrak{D}}$ for any configuration γ will result in a specification structure. However, as shown by Theorem 2.7, proving that a property, in the form of a guarantee of a contract allocated to the system is satisfied, instantiating a PL specification structure must result in a *proper* specification structure.

Theorem 3.8. *Given a set of configurations $\{\gamma_i\}_{i\in\mathbb{N}}$ and a proper PL specification structure $\widehat{\mathfrak{D}}$ such that each instantiation $\mathfrak{D}_{\gamma_i}$ is a specification structure, if*

i) for each A_i it holds that $P(A_i) \models_{\mathfrak{C}} \bigvee_{\mathtt{full}(S_j,A_i)} P(S_j)$,
ii) for each G_i it holds that $P(G_i) \models_{\mathfrak{C}} \bigvee_{\mathtt{full}(G_j,G_i)} P(G_j)$,

then each $\mathfrak{D}_{\gamma_i}$ is proper. □

A direct consequence of Theorem 3.7 and Theorem 3.8 in conjunction with Theorem 2.7 is the following corollary.

Corollary 3.9. *Given a set of configurations $\{\gamma_i\}_{i\in\mathbb{N}}$, and a PL specification structure $\widehat{\mathfrak{D}}$, if*

i) premises of Theorem 3.7 hold,
ii) premises of Theorem 3.8 hold,
iii) $\forall \mathtt{full}(S_i, S_j) . S_i \models S_j$,
iv) for each contract K_i allocated to an atomic component C_i it holds that $C_i \blacktriangleright K_i$,

then it holds that $C' \blacktriangleright (A', G')$ for each γ_i. □

If a PL specification structure is considered to be the PL design, then Corollary 3.9 summarizes all conditions that the design must satisfy (i-iii), and the condition that the implementation must satisfy (iv) in order to be able to claim that each configuration satisfies the contract (A', G'). Given a particular PL specification structure, providing evidence that the premises of Corollary 3.9 hold, is the basis for construing a PL safety case in the next section.

4 CONSTRUCTING A SAFETY CASE FROM $\widehat{\mathfrak{D}}$

This section presents the second contribution, which is a method for the construction of a safety case, expressed using the GSN notation, which is based on a PL specification structure of a configurable system. Before presenting the safety-case construction method, the GSN notation is briefly introduced.

4.1 Goal-Structuring Notation

GSN notation [28] is one of the most prominent formalism for safety case representation. GSN is *"a graphical argument notation which can be used to document explicitly the elements and structure of an argument and the argument's relationship to evidence"* [28]. More specifically, GSN models form graphs called *goal-structures* which capture the safety-case argumentation structure. Figure 3 shows the GSN elements considered in the present paper. A complete description of the GSN notation, can be found in [28].

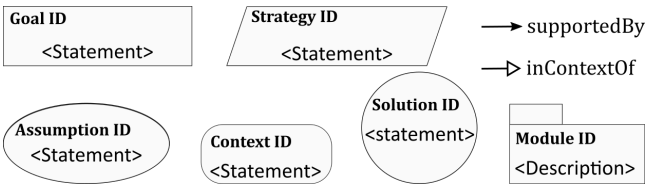

Figure 3: GSN elements used in the present paper

The elements in Figure 3 have the following meaning: i) *Goal* represents a *claim* about the system, ii) *Solution* represents the evidence expected to show that a goal has been met, iii) *Strategy* represents the rationale for decomposing a goal into sub-goals, iv) *Assumption* is the context in which a claim should hold, v) *Context* explicitly declares the scope of a claim, and vi) *Module* is a container for smaller, coherent goal-structures. The presented elements can be related via the *supportedBy* link, which allows decomposing goals to subgoals and connecting them to solutions, or via *inContextOf* links, which allow relating goals with assumptions and contexts.

In order to present a formal transformation of a CBS model to GSN notation, and in accordance with definitions from [11], a GSN model is defined as follows.

Definition 4.1 (GSN model). A GSN model $\mathcal{G}$ is a tuple, $\mathcal{G} = (\mathcal{B}, \mathcal{A}, T_{\mathcal{B}}, T_{\mathcal{A}}, d)$, where:

i) $\mathcal{B}$ is a set of nodes, where each node is denoted as b_i,
ii) $\mathcal{A} \subseteq \mathcal{B} \times \mathcal{B}$ is a set of edges, where each edge is denoted as $\langle b_i, b_j \rangle$,
iii) nodes and edges form a *rooted, directed acyclic-graph*,
iv) $T_{\mathcal{B}}$ is a labeling function $T_{\mathcal{B}} : \mathcal{B} \rightarrow \{Goal, Strategy, Solution, Assumption, Module\}$,
v) $T_{\mathcal{A}}$ is a labeling function $T_{\mathcal{A}} : \mathcal{A} \rightarrow \{supportedBy, inContextOf\}$,
vi) d is a labeling function $d : \mathcal{B} \rightarrow String$ which labels each node with a textual description,
vii) for each edge $T_{\mathcal{A}}(\langle b_i, b_j \rangle) = supportedBy$ it holds that $T_{\mathcal{B}}(b_i) \in \{Goal, Strategy\}$. If $T_{\mathcal{B}}(b_i) = Goal$ then $T_{\mathcal{B}}(b_j) \in \{Strategy, Solution\}$ and if $T_{\mathcal{B}}(b_i) = Strategy$ then $T_{\mathcal{B}}(b_j) = Goal$,
viii) for each edge $T_{\mathcal{A}}(\langle b_i, b_j \rangle) = inContextOf$ it holds that $T_{\mathcal{B}}(b_i) = Goal$ and $T_{\mathcal{B}}(b_j) = \{Assumption, Context\}$. □

4.2 Constructing a safety case

A safety case can only be developed with respect to a particular standard for which compliance should be shown. In what follows, we consider the ISO 26262 functional safety standard for the automotive domain but the approach is general and it can be reused in other domains. Note that ISO 26262 contains nearly 800 requirements that are both process-based and product-based. Process-based requirements relate to the use of appropriate tools, the existence of appropriate expertise among the developers etc. This section focuses on product-based requirements. Product-based argumentation must show that the system-level requirements are sufficient to *mitigate* the potential hazards in the systems intended operating environment and that the system, in each of its configurations, implements these requirements. With this overall goal in mind, we propose the GSN model in Figure 4 as the overall argumentation-structure of the PL safety case.

The root node of the PL safety case is the claim that the *"Product Line is safe"*, and the supporting argumentation is partitioned using the GSN *modules* in order to separate goal-structures related to different systems. The argument that a particular system is safe is achieved by decomposing this claim into arguments about system safety with respect to each identified hazard, as exemplified in *Module-System 1*. One part of showing that a system is safe with respect to a certain hazard corresponds to arguing that overall system requirement, G' in Figure 4, *mitigates* the identified hazard. The evidence for this claim, as indicated by Figure 4, is in the majority of cases a manual review. As support for this review process, the GSN *context* nodes are used to refer to the artifacts that define the system and that have been produced while analyzing potential hazards based on which systems requirements were defined. On the contrary, the argumentation about the fact that the system implements the requirement in the form of a guarantee G' can be fully developed from the PL specification structure of the system.

Arguing that the system satisfies the contract (A', G') corresponds to arguing that the premises of Corollary 3.9 hold for the given system. In other words, besides the invariance conditions on the PL specification structure, for each pair of specifications $\texttt{full}(S_i, S_j)$ it should be verified that $S_i \models S_j$, and for each contract K_i and component C_i such that $\texttt{allTo}(K_i, C_i)$, it should be verified that $C_i \rhd K_i$ by using techniques such as testing or formal verification. Aggregating claims, supported by evidence, that all of the previously mentioned conditions hold, corresponds to a claim that the system satisfies the contract (A', G'). Because such argumentation-structure is based on a CBS model of the PL, from a formal point of view, the argumentation-structure is *complete*.

Before presenting the transformation rules from a PL specification structure to a GSN model, additional constraints must be enforced on a GSN model. For example, a single node $b_i \in \mathcal{B}$ is a valid GSN model but it does not represent a valid safety case.

Definition 4.2 (GSN safety case). A GSN model is a *GSN safety case* if

i) the root node b is such that $T_{\mathcal{B}}(b) = Goal$,
ii) for each node b_i where $T_{\mathcal{B}}(b_i) = Goal$, there exists a path to a node b_j where $T_{\mathcal{B}}(b_j) = Solution$. □

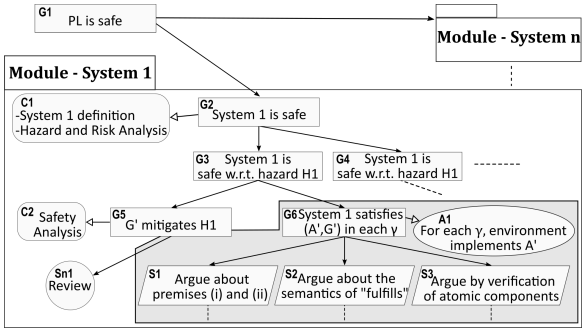

Figure 4: Overview of the PL safety case where the shaded part is constructed from a CBS model of the PL.

The reason for introducing these conditions is that a safety case must argue that a certain claim holds, and each claim must be eventually supported by evidence.

4.3 PL specification structure to GSN transformation rules

Given an arbitrary PL specification structure $\widehat{\mathfrak{D}}$, the corresponding GSN safety case argumentation-structure is constructed as follows:

i) for each contract (A', G') such that $\texttt{allTo}((A', G'), C')$ where C' is the composite component representing the system, create a GSN node b_i such that $T_{\mathcal{B}}(b_i) = \textit{Goal}$ and $d(b_i) =$ "C' *satisfies* (A', G') *for each configuration* γ",

ii) create three GSN nodes b_1, b_2, b_3 such that $T_{\mathcal{B}}(b_1) = T_{\mathcal{B}}(b_2) = T_{\mathcal{B}}(b_3) = \textit{Strategy}$, $T_{\mathcal{A}}(\langle b_i, b_1\rangle) = T_{\mathcal{A}}(\langle b_i, b_2\rangle) = T_{\mathcal{A}}(\langle b_i, b_3\rangle) = \textit{supportedBy}$, and $d(b_1) =$*"Argue that premises (i) and (ii) of Corollary 1 hold"*, $d(b_2) =$*Argue that semantics of each fulfills relation holds"*, and $d(b_3) =$*Argue by verification that contracts allocated to atomic components are satisfied"*.

iii) for each premise of Theorem 3.7 and Theorem 3.8 create a GSN node b_j such that $T_{\mathcal{B}}(b_j) = \textit{Goal}$ where $T_{\mathcal{A}}(\langle b_1, b_j\rangle) = \textit{supportedBy}$ and $d(b_j) =$*"Premise x holds"* where x is one of the premises. Also, for each S_i or C_j that is in the scope of a premise x, create a GSN node b_{js} where $T_{\mathcal{B}}(b_{js}) = \textit{Solution}$, $T_{\mathcal{A}}(\langle b_j, b_{js}\rangle) = \textit{supportedBy}$ and b_{js} refers to the evidence about presence conditions entailment.

iv) for each pair $\texttt{full}(S_i, S_j)$ create a GSN node b_f such that $T_{\mathcal{B}}(b_f) = \textit{Goal}$, where $T_{\mathcal{A}}(\langle b_2, b_f\rangle) = \textit{supportedBy}$ and $d(b_f) =$ "S_i *entails* S_j". Also, for each b_f create a GSN node b_{fs} where $T_{\mathcal{B}}(b_{fs}) = \textit{Solution}$, $T_{\mathcal{A}}(\langle b_f, b_{fs}\rangle) = \textit{supportedBy}$ and b_{fs} refers to the evidence of specification entailment.

v) for each pair $\texttt{allTo}((A_i, G_i), C_i)$ where C_i is an atomic component, create a GSN node b_v such that $T_{\mathcal{B}}(b_v) = \textit{Goal}$, where $T_{\mathcal{A}}(\langle b_3, b_v\rangle) = \textit{supportedBy}$ and $d(b_v) =$ "C_i *satisfies* (A_i, G_i). Also, for each b_v create a GSN node b_{vs} where $T_{\mathcal{B}}(b_{vs}) = \textit{Solution}$, $T_{\mathcal{A}}(\langle b_v, b_{vs}\rangle) = \textit{supportedBy}$ and b_{vs} refers to the evidence of contract (A_i, G_i) being satisfied.

The presented transformation rules can be easily converted into an algorithm that can be used to automate the creation of the safety case argumentation-structure. Given these transformation rules, and Def. 4.1 and Def. 4.2, next section presents an example application of the extended CBS framework and the transformation rules for the construction of FLD system safety case.

5 EXAMPLE APPLICATION

Figure 5 depicts the PL specification structure for the subset of all possible configurations of Scania's FLD system, and the corresponding part of the constructed safety case. The example is a simplification of the complete FLD technical architecture presented in [39], but unlike in [39], the example in Figure 5 considers multiple configurations of the FLD system in order to validate the PLE extension of the basic CBS framework.

The FLD system, represented by the component C_{FLD}, has subcomponents $C_{TANK-ASSY}$ representing the fuel tank assembly, C_{COO} representing a complete system, the *COOrdinator Electronic Control Unit* (ECU), and C_{BMS} representing another system, the *Body Management System* ECU. Component $C_{TANK-ASSY}$ has further subcomponents C_{TANK} representing the actual fuel tank and two fuel level sensors; an analog one represented by C_{a-SENS} and a digital one represented by C_{d-SENS}. Component C_{COO} has subcomponents C_{C-EST} which is a C-code function performing the fuel level estimation for all types of fuel except for diesel, and the subcomponent C_{C-CAN} which is a C-code function responsible for transmitting the estimated value over *Controller Area Network* (CAN) to the vehicle *Instrumentation CLuster* represented by component C_{ICL}. Similarly, the component C_{BMS} has the subcomponent C_{B-EST} which performs the fuel level estimation in FLD configurations with diesel fuel, and subcomponent C_{B-CAN} which transmits the estimated value to C_{COO}.

Note that each SW component exists in several production versions, each HW component exists in several variants, e.g. *round* or *square* C_{TANK}, and each ECU exists in several variants depending on the ECU generation, e.g. generation 7 of C_{COO}. Due to space limitations, representing each of these is beyond the scope of the paper, i.e. each component in Figure 5 represents a single version, variant, or a generation of a component. As described, the applicability of a particular component or a specification to a particular configuration is defined through presence conditions[1]. The presence conditions $\varphi_1 - \varphi_{33}$ were manually extracted from various tools in Scania CV AB. Due to space limitations, not all presence conditions are shown but for illustration purposes, several are presented in Table 1. Each term $FX.Y$ represents one of more than 40000 Boolean features, e.g. $F1.1$ is the *Truck* feature while $F1.2$ is the *Bus* feature. *RD* stands for *release date* which is a point in time when a certain component is released for production.

The contracts in the PL specification structure of the FLD, are based on the ones presented in [39] but because the present paper considers multiple configurations, the PL specification structure in Figure 5, contains additional contracts such as (A_6, G_5) or all contracts allocated to C_{BMS} and its constituent components. These contracts were manually extracted from FLD system documentation by relying on principles presented in [39]. Besides the contract examples in Table 1, the PL specification structure of the FLD system contains contracts guaranteeing that if the position of floater in C_{TANK} corresponds to the actual fuel volume, then either of the sensors can guarantee that the measured value corresponds to the

[1]Feature names have been modified due to confidentiality reasons.

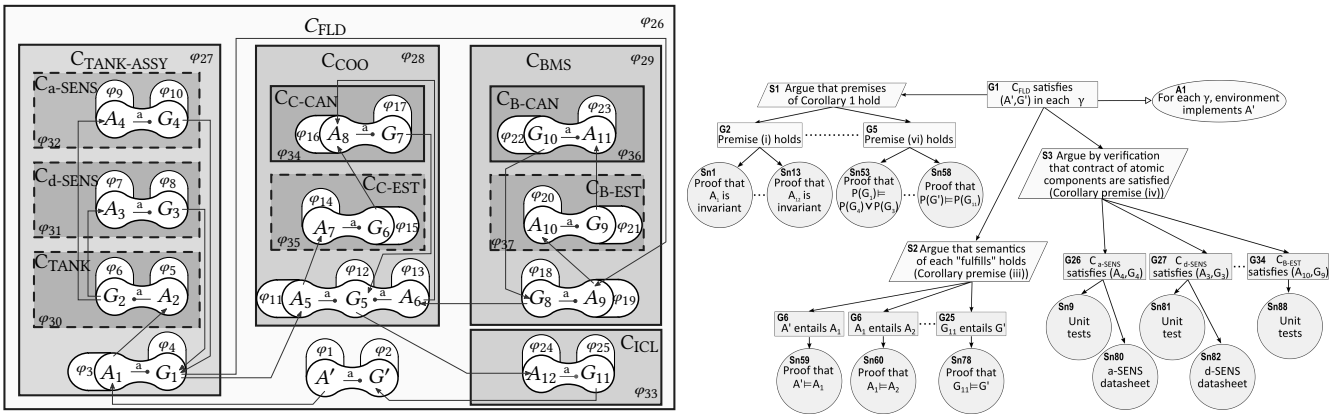

Figure 5: PL specification structure of the FLD system (left), and the corresponding safety case argumentation-structure (right)

Table 1: Examples of presence conditions and contracts

Presence conditions from the FLD system
$\varphi_{26} = (F1.1 \wedge F1323.5) \vee (F1.2 \wedge F1323.5) \vee (F1.1 \wedge F1323.6 \wedge RD \geq 201706)$
$\varphi_{28} = [F2482.1 \vee (F2482.2 \wedge F1940.1)] \wedge [F1837.1 \vee (F1837.19 \wedge F579.26)] \wedge [(F37.1 \wedge F2719.1 \wedge 201404 \leq RD \leq 201801) \vee (F37.3 \wedge F2719.1)]$
$\varphi_{35} = (F1.1 \vee F1.2) \wedge F1323.5$
$\varphi_1 = \varphi 2 = F1.1 \vee F1.2$
...

Contracts from the FLD system	
$\mathbf{A'}$:	The *ignition state* equals *true*.
$\mathbf{G'}$:	*Indicated fuel volume* deviates at most 7% compared to *actual fuel volume*.
$\mathbf{A_{12}}$:	*Estimated fuel volume* value is updated every 1s.
$\mathbf{G_{11}}$:	*Indicated fuel volume* deviates at most 7% compared to *actual fuel volume*.
$\mathbf{A_5}$:	*Measured fuel volume* corresponds to the *actual fuel volume*.
$\mathbf{A_6}$:	*Estimated fuel volume* is received over CAN every 60ms.
$\mathbf{G_6}$:	*Estimated fuel volume* is transmitted over CAN every 90ms.
...	

actual fuel volume and consequently, the C_{C-EST} and C_{B-EST} can in different configurations guarantee that the estimated value will corresponds to the actual fuel volume. Then C_{COO} guarantees to transmit the estimated value C_{ICL} which displays it.

It should be noted that the development of assemblies and ECUs represented by components C_{TANK}, C_{COO}, C_{BMS} and their subcomponents is performed by dedicated engineering groups, while the overall system, represented by component C_{FLD} with subcomponents C_{TANK}, C_{COO}, and C_{BMS} is managed by the architecture group. In other words, the CBS model in Figure 5 is a holistic interpretation of the FLD system that spans across several engineering teams and its purpose is to enable holistic analyses of the FLD system, e.g. for the purpose of the safety case.

5.1 Obtained Argumentation-Structure

The right-hand side of Figure 5 shows the argumentation structure corresponding to the CBS model on the left-hand side. Application of the transformation rules from Section 4.3 is straightforward and it results in the argumentation structure with the *goal* root-node with the claim "C_{FLD} *satisfies* (A', G') *in each* γ", followed by further *strategy*, *sub-goals*, and expected *solution* nodes. In this section we discuss the feasibility, benefits and limitations of the presented approach.

5.1.1 Feasibility of producing the required evidence. The first set of required evidence relates to proving that premises of Theorem 3.7 and Theorem 3.8 hold. As shown in [32], verifying these premises can be automated using *Description Logic* [2] or in general, using SMT solvers [9]. The second set of evidence relates to proving that for each pair of specifications $\texttt{full}(S_i, S_j)$ it holds that $S_i \models S_j$. In the general case, independently of the approach, this is a hard problem because specifications S_i and S_j can be such that it is not possible to formalize them and verify entailment. However, the observation from the analysis of the FLD system is that most often, as exemplified in Table 1, S_i and S_j are either identical, e.g. G_{11} and G', or even without formal analysis it is obvious that S_j fulfills S_i, e.g. G_6 and A_6. The reason for this is that when specifying components in terms of contracts over component's interfaces, not matching assumptions and guarantees of components is immediately counter-intuitive. The third set of evidence relates to proving that for each contract K_i and atomic component C_i, such that $\texttt{allTo}(K_i, C_i)$, it holds that C_i *satisfies* K_i. Providing this type of evidence is the most difficult because it involves verifying the behavior of C_i. On the one hand, because functional safety standards recommend that safety-critical software should be of low complexity, formal SW verification approaches against contract-based specifications produce good results [23]. On the other hand, large-scale formal verification is still not adopted in industrial settings which means that majority of evidence for arguments of type $C_i \rhd K_i$ will be based on testing, as observed during the analysis of the FLD system. Since *unit-testing* is a *de-facto* standard in SW development practice, the third type of evidence will certainly be created. However, because testing is a technique that does prove absence of faults, the confidence in claims supported by testing evidence should be assessed [13]. Note that because a compositional reasoning framework is the basis of the PL safety case, the need to perform *integration testing* is removed and *unit testing* is sufficient to produce the third type of evidence.

5.1.2 Benefits of the approach. The primary aim of the presented approach is the reduction in the amount of manual work. Most importantly, the approach enables automated construction of safety case argumentation-structure for an arbitrary PL. Also, in the worst case, at least a half of the required evidence can be produced using automated techniques. The presented approach also facilitates the decoupling of the responsibilities among the different engineering groups, i.e. it allows *modular* safety case development. In the example from Figure 5, the engineering group responsible for vehicle level safety should monitor if the premises of Corollary 3.9 hold for the composition of components C_{FLD}, C_{TANK}, C_{COO}, C_{BMS} while the group managing subcomponents of, for example C_{COO}, should monitor if the Corollary 3.9 holds for C_{COO}. Because the constructed safety case argumentation-structure is directly based on the PL design and implementation, expressed as a CBS model, the approach enables stepwise creation of the safety case in parallel with PL development process. In other words, except the evidence needed to show that $C_i \rhd K_i$, the premises of Corollary 3.9 can be verified at any stage of the PL development. The tight coupling of the PL safety case with the PL design and implementation also facilitates *change-impact* analysis. Elements of the safety case affected by a change in the CBS model of the PL can be precisely determined, but more importantly, it is possible to verify if a change in the CBS model of PL is such that a premise of Corollary 3.9 no longer holds, and consequently the safety case cannot be created.

5.1.3 Limitations of the approach. The described benefits require that an enterprise adopts a sufficiently formal approach to requirements engineering. While in general this cannot be expected, in cases of highly critical safety-critical systems, the relevant standards typically highly recommend semi-formal and formal methods for requirements engineering and verification. An issue that hinders complete automation of the presented approach is the need to access various types of information about the PL. Because such information is typically maintained by different tools, a *data-integration* [12] process that would merge and clean the necessary information is required. A promising approach for such data-integration are *semantic web* technologies as suggested in [20, 31, 33]. Finally, in order to simplify the exposition, but primarily due to the lack of space, the presented CBS framework is simplistic in some aspects. For example, there are scenarios where it would be beneficial to declare a contract without assumptions or perhaps with a set of assumptions. Since such generalizations are non-conceptual and mostly trivial, they are left as future work.

6 RELATED WORK

Two main approaches have been proposed for the construction of argumentation structures. The first line of research has focused on capturing *safety case patterns* that have been used in real world safety cases or in case studies performed by researchers [11]. The second line of research has focused on the generation of safety case argumentation-structures from various types of artifacts [10, 25, 37]. In the area of PLE, most notable contributions [10, 16, 17, 19, 25, 36, 37] come from the latter group.

Work in [24] uses artifacts resulting from safety analyses to construct a safety case for a PL by showing that the contributions of identified faults to potential failures is acceptable. Work in [25] uses the method from [24] for the creation of a safety case for a configurable PL architecture. Unlike the presented approach which directly argues about the safety of all product configurations, the approach in [24] defines a GSN extension that allows merging safety cases of individual product configurations. However, there are no criteria for assessing the completeness of individual safety cases or of the merged safety case. Also, unlike the presented approach, which uses the identified requirements as the basis of the argumentation structure, work in [24] uses the identified faults as the basis of the argumentation structure. The tool-supported method presented in [10] provides an automated way to construct a modular safety case based on the meta-models presented in [24] given the information from a variability management tool, architecture specification tool, and a hazard and safety-analysis tool. Work in [36, 37] provides an approach for the creation of reusable safety cases fragments based on a semi-formal notion of *weak* and *strong safety contracts*. The approach considers the concept of *Safety Element out of Context* (SEooC) from the standard ISO 26262, which is used to represent the components that OEMs procure from external suppliers. The essence of the approach is to perform hazard analysis and risk assessment for a SEooC using a variant of *Fault Propagation and Transformation Calculus* (FPTC) [18] and then transform the analysis results into safety contracts. The present paper considers development and assurance of arbitrary PLs from the OEM perspective, and because of that relies on a more expressive contract-based framework which subsumes the *weak* and *strong* safety contracts. Consequently the approach in the present paper argues about system-level safety instead of only SEooC. Furthermore, unlike the approach in [36] where presence conditions are assumptions of contracts, the presence conditions in the present paper are orthogonal to components and specifications thus allowing presence conditions of artifacts and the artifact themselves to evolve separately. The line of research in [16, 17, 19] introduces approaches for constructing reusable safety case fragments that target particular requirements from various domain-specific safety standards. A notable aspect of these approaches is the attention to processes and their assurance with respect to standardized practices. Unlike this work, our approach is oriented towards product-based arguments for arbitrary PLs.

7 CONCLUSION

To satisfy the needs of as many different customers as possible, enterprises are increasing the allowed levels of customization, even for safety-critical systems. Making safety-critical systems highly configurable, e.g. creating *Product Lines* (PLs), prevents the use of existing, primarily manual, safety assurance practices because the number of configuration is simply too high.

The present paper has introduced a general and novel approach for the construction of a safety case that argues about the safety of the complete PL, instead about each product configuration individually. The first contribution of the paper is the PL-oriented extension of a general-purpose *Contract-Based Specification* (CBS) framework, in order to create a rigorous, holistic model of all product configurations. The result of this extension is Corollary 3.9, which summarizes the constraints on CBS models of PLs that preserve

the *sound compositional reasoning* of the original CBS framework. This ability allows the reasoning about the properties of all PL configurations by analyzing only their constituent components. The second contribution defines a set of transformation rules from an arbitrary CBS model of a PL into a PL safety case expressed using GSN. Because the compositional reasoning in the CBS framework is *sound*, it is ensured that the argumentation structure of the PL safety case is *complete-by-construction*.

Future work includes generalizations of the approach, and development of tool support that will enable large-scale case studies.

ACKNOWLEDGMENT

D. Nešić and M. Nyberg are funded by the ECSEL PRYSTINE project (No 783190) and supported by Scania CV AB. B. Gallina is funded by ECSEL AMASS project (No 692474).

REFERENCES

[1] Sven Apel, Don Batory, Christian Kästner, and Gunter Saake. 2016. *Feature-oriented software product lines.* Springer-Verlag, Berlin-Heidelberg, Germany.

[2] F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider (Eds.). 2003. *The Description Logic Handbook: Theory, Implementation, and Applications.* Cambridge University Press.

[3] Albert Benveniste, Benoît Caillaud, Alberto Ferrari, Leonardo Mangeruca, Roberto Passerone, and Christos Sofronis. 2008. Multiple Viewpoint Contract-Based Specification and Design. In *Proceedings International Symposium on Formal Methods for Components and Objects.* Springer, Berlin, Heidelberg, 200–225.

[4] Albert Benveniste, Benoît Caillaud, Dejan Nickovic, Roberto Passerone, Jean-Baptiste Raclet, Philipp Reinkemeier, Alberto Sangiovanni-Vincentelli, Werner Damm, Tom Henzinger, and Kim Guldstrand Larsen. 2015. *Contracts for Systems Design: Theory.* Research Report RR-8759. Inria Rennes Bretagne Atlantique.

[5] Albert Benveniste, Benoît Caillaud, Dejan Nickovic, Roberto Passerone, Jean-Baptiste Raclet, Philipp Reinkemeier, Alberto Sangiovanni-Vincentelli, Werner Damm, Thomas A. Henzinger, and Kim G. Larsen. 2018. Contracts for System Design. *Foundations and Trends in Electronic Design Automation* (2018), 124–400.

[6] Thorsten Berger, Ralf Rublack, Divya Nair, Joanne M. Atlee, Martin Becker, Krzysztof Czarnecki, and Andrzej Wasowski. 2013. A Survey of Variability Modeling in Industrial Practice. In *Proceedings of the Seventh International Workshop on Variability Modelling of Software-intensive Systems (VaMoS '13).* ACM, New York, NY, USA, 7:1–7:8.

[7] A. Cimatti and S. Tonetta. 2012. A Property-Based Proof System for Contract-Based Design. In *Proceedings Euromicro Conference on Software Engineering and Advanced Applications '12.* 21–28.

[8] The International Electrotechnical Commission. 2010. ISO 61508: Functional Safety. (2010).

[9] Leonardo De Moura and Nikolaj Bjørner. 2008. Z3: An efficient SMT solver. In *International conference on Tools and Algorithms for the Construction and Analysis of Systems.* Springer Berlin Heidelberg, Berlin, Heidelberg, 337–340.

[10] André L. de Oliveira, Rosana T. V. Braga, Paulo C. Masiero, Yiannis Papadopoulos, Ibrahim Habli, and Tim Kelly. 2015. Supporting the Automated Generation of Modular Product Line Safety Cases. In *Proceedings International Conference on Dependability and Complex Systems.* Springer-Verlag, Berlin-Heidelberg, 319–330.

[11] Ewen Denney and Ganesh Pai. 2015. *Safety Case Patterns: Theory and Applications.* Technical Report NASA/TM-2015-218492. NASA.

[12] AnHai Doan, Alon Halevy, and Zachary Ives. 2012. *Principles of Data Integration.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[13] Lian Duan, Sanjai Rayadurgam, Mats P. E. Heimdahl, Anaheed Ayoub, Oleg Sokolsky, and Insup Lee. 2017. Reasoning About Confidence and Uncertainty in Assurance Cases: A Survey. In *Software Engineering in Health Care*, Michaela Huhn and Laurie Williams (Eds.). Springer International Publishing, Cham.

[14] Radio Technical Commission for Aeronautics. 2011. Software Considerations in Airborne Systems and Equipment Certification. (2011).

[15] International Organization for Standardization. 2011. ISO 26262: Road vehicles - Functional safety. (Nov 2011).

[16] Barbara Gallina. 2015. Towards Enabling Reuse in the Context of Safety-critical Product Lines. In *Proceedings International Workshop on Product LinE Approaches in Software Engineering '15.* IEEE Press, Piscataway, NJ, USA, 15–18.

[17] Barbara Gallina, Antonio Gallucci, Kristina Lundqvist, and Mattias Nyberg. 2013. VROOM & cC: a Method to Build Safety Cases for ISO 26262-compliant Product Lines. In *Proceedings International Conference on Computer Safety, Reliability and Security: SASSUR Workshop '13.* HAL, Toulouse, France, 13.

[18] B. Gallina, M. A. Javed, F. U. Muram, and S. Punnekkat. 2012. A Model-Driven Dependability Analysis Method for Component-Based Architectures. In *Proceedings Euromicro Conference on Software Engineering and Advanced Applications '12.* 233–240.

[19] Barbara Gallina, Shaghayegh Kashiyarandi, Karlheinz Zugsbratl, and Arjan Geven. 2014. Enabling Cross-Domain Reuse of Tool Qualification Certification Artefacts. In *Proceedings International Conference on Computer Safety, Reliability and Security '14.* Springer International Publishing, Cham, 255–266.

[20] B. Gallina and M. Nyberg. 2017. Pioneering the Creation of ISO 26262-Compliant OSLC-Based Safety Cases. In *2017 IEEE International Symposium on Software Reliability Engineering Workshops.* 325–330.

[21] Susanne Graf and Sophie Quinton. 2007. Contracts for BIP: Hierarchical Interaction Models for Compositional Verification. In *Proceedings Forum for Fundamental Research on Theory, Models, and Application for distributed systems '07.* Springer Berlin Heidelberg, Berlin, Heidelberg, 1–18.

[22] William S. Greenwell. 2006. A taxonomy of fallacies in system safety arguments. In *Proceedings International System Safety Conference '06.*

[23] Dilian Gurov, Christian Lidström, Mattias Nyberg, and Jonas Westman. 2017. Deductive Functional Verification of Safety-Critical Embedded C-Code: An Experience Report. In *Proceedings Critical Systems: Formal Methods and Automated Verification Workshop '17.* Springer International Publishing, Cham, 3–18.

[24] Ibrahim Habli. 2009. *Model-based assurance of safety-critical product lines.* Ph.D. Dissertation. Department of Computer Science, University of York, York, UK.

[25] Ibrahim Habli and Tim Kelly. 2010. A Safety Case Approach to Assuring Configurable Architectures of Safety-Critical Product Lines. In *Proceedings International Symposium on Architecting Critical Systems '10.* Springer, Berlin, Heidelberg, 142–160.

[26] Stuart Hutchesson and John McDermid. 2013. Trusted Product Lines. *Information and Software Technology* (2013), 525 – 540.

[27] Zarrin Langari and Tom Maibaum. 2013. Safety Cases: A Review of Challenges. In *Proceedings International Workshop on Assurance Cases for Software-Intensive Systems '15.* 1–6.

[28] Origin Consulting (York) Limited. 2018. GSN community standard version 2. (Jan 2018).

[29] Frank J. van der Linden, Klaus Schmid, and Eelco Rommes. 2007. *Software Product Lines in Action: The Best Industrial Practice in Product Line Engineering.* Springer-Verlag, Secaucus, NJ, USA.

[30] Sunil Nair, Jose Luis de la Vara, Mehrdad Sabetzadeh, and Lionel Briand. 2014. An extended systematic literature review on provision of evidence for safety certification. *Information and Software Technology* 56, 7 (2014), 689 – 717.

[31] Damir Nešić and Mattias Nyberg. 2017. Applying Multi Level Modeling to Data Integration in Product Line Engineering. In *Proceedings MODELS Satelite events: International Workshop on Multi-Level Modeling '17.* 235–242.

[32] Damir Nešić and Mattias Nyberg. 2018. Verifying Contract-Based Specifications of Product Lines using Description Logic. In *Proceedings 31st International International Workshop on Description Logic.* 13. http://ceur-ws.org/Vol-2211/#paper-26

[33] Damir Nešić, Mattias Nyberg, and Barbara Gallina. 2017. Modeling Product-Line Legacy Assets Using Multi-Level Theory. In *Proceedings of the 21st International Systems and Software Product Line Conference - Volume B (SPLC '17).* ACM, New York, NY, USA, 89–96.

[34] UK Ministry of Defence. 1996. 00-56: Safety Management Requirements for Defence Systems. (1996).

[35] Irfan Sljivo, Barbara Gallina, Jan Carlson, and Hans Hansson. 2014. Generation of safety case argument-fragments from safety contracts. In *Proceedings International Conference on Computer Safety, Reliability, and Security '14.* Springer International Publishing, Cham, 170–185.

[36] Irfan Sljivo, Barbara Gallina, Jan Carlson, and Hans Hansson. 2016. Configuration-Aware Contracts. In *Proceedings International Conference on Computer Safety, Reliability, and Security '16.* Springer International Publishing, Cham, 43–54.

[37] Irfan Sljivo, Barbara Gallina, Jan Carlson, Hans Hansson, and Stefano Puri. 2017. A Method to Generate Reusable Safety Case Argument-fragments from Compositional Safety Analysis. *Journal of Systems and Software* 131 (2017), 570–590.

[38] A. v. Rhein, A. Grebhahn, S. Apel, N. Siegmund, D. Beyer, and T. Berger. 2015. Presence-Condition Simplification in Highly Configurable Systems. In *Proceedings IEEE International Conference on Software Engineering '15.* 178–188.

[39] Jonas Westman. 2014. *Specifying and structuring requirements on cyber-physical systems using contracts.* Technical Report urn:nbn:se:kth:diva-159599. KTH.

[40] Jonas Westman and Mattias Nyberg. 2018. Conditions of contracts for separating responsibilities in heterogeneous systems. *Formal Methods in System Design* 52, 2 (01 Apr 2018), 147–192.

[41] Jonas Westman and Mattias Nyberg. 2018. Preserving Contract Satisfiability under Non-Monotonic Composition. In *Formal Techniques for Distributed Objects, Components, and Systems.* Springer International Publishing, Cham, 181–195.

[42] Len Wozniak and Paul Clements. 2015. How Automotive Engineering is Taking Product Line Engineering to the Extreme. In *Proceedings of the 19th International Conference on Software Product Line (SPLC '15).* ACM, New York, USA, 327–336.