This is the submitted version of a paper presented at ACM/SIGAPP Symposium on Applied Computing.

Citation for the original published paper:

Constructing Product-Line Safety Cases from Contract-Based Specifications
In: Proceedings of the Symposium on Applied Computing (pp. 2022-2031). Association for Computing Machinery (ACM)

N.B. When citing this work, cite the original published paper.

Permanent link to this version:
http://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-250230
Constructing Product-Line Safety Cases from Contract-Based Specifications

ABSTRACT

Safety cases are used to argue that safety-critical systems satisfy the properties determined to mitigate the potential hazards in the systems deployment environment. Although primarily a manual task, safety cases have been successfully created for single systems. However, when systems with a high number of configurations are considered, typically developed as a Product Line (PL), considering each possible configuration and constructing sound and complete safety-case argumentation is challenging. This paper presents a novel and general approach for the construction of a safety case for an arbitrary PL that is based on Contract-Based Specification (CBS) of the PL. Starting from a general CBS framework, a PL extension of the CBS framework is presented and it is shown that the extension preserves the properties of the original framework. Then, given a CBS specification of a PL, a set of transformation rules for the construction of the safety case argumentation-structure is defined. 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

1 INTRODUCTION

Many software-intensive systems of systems are also safety critical. Indeed, failures of systems developed from domains like automotive, aerospace, or medical, can cause harm to material property or humans. Consequently, these domains are regulated and are required to comply with general functional safety standards such as IEC 61508 [27] and with domain-specific standards such as ISO 26262 [28] in automotive, or DO 178C [34] in aerospace. These standards emphasize structured and traceable requirements engineering and require the creation of a safety case [3] to show compliance. A safety case is a structured argument, supported by a body of evidence, showing that a system is designed and implemented to be acceptably safe in the environment in which it will be deployed.

As acknowledged in the literature [23, 29, 31], constructing and maintaining a safety case is a notoriously difficult task. First and foremost, safety cases are huge artifacts whose construction and maintenance is primarily manual [31]. For a reasonably sized system, just listing all the evidence such as safety analyses for different functional breakdown levels, requirements obtained from these analyses, test cases and test results verifying these requirements, results in a huge amount of information. Secondly, because different parts of the system are usually specified, designed, implemented, and verified using a variety of methods with different underlying semantics, it is especially difficult to create a sound argumentation structure which combines the available evidences into a safety case. Also, the lack of a unified semantics hinders automation and the creation of the argumentation structure is also most often manual.

Despite the difficulties, safety cases have been successfully developed in some domains, e.g. aerospace, but in other domains, e.g. automotive, they still represent a research challenge. The caveat separating these domains is the number of possible product configurations. A frequently used paradigm for designing and implementing a family of products is Product Line Engineering (PLE) [5, 30], where the commonalities and differences between product configurations are systematically managed, thus allowing a high level of reuse and quick creation of new product configurations. From the certification viewpoint, if the number of product configurations is low, like in aerospace domain, then it is feasible to create a safety case for each product configuration while reusing parts of safety argumentation between different safety cases. On the contrary, when the number of product configurations reaches hundreds of thousands [9], like in the automotive domain [42], this is not feasible both because it is impossible to produce the required evidence for each product configuration and because of the amount of manual work required to construct the argumentation structure for each safety case.

In this paper we propose a general and novel approach for the construction of a safety case for the complete Product Line (PL), instead of constructing a safety case per product configuration. The contribution of the paper is divided into two sub-contributions. The first sub-contribution is a PL extension of a general Contract-Based Specification (CBS) framework [6, 8, 11, 33, 40] which is then used to create a unified representation of the PL design and implementation. There are multiple reasons for relying on the CBS framework. Firstly, contemporary functional safety standards, for which PL safety cases are developed, emphasize structured and traceable requirements engineering which is one of the applications of CBS frameworks [8, 40]. Secondly, CBS frameworks are general enough to represent arbitrary systems and this removes the difficulty with the creation of a sound argumentation structure due to the need to consider artifacts based on different methods and languages. Thirdly, because CBS frameworks are compositional, i.e. the
properties of all product configurations specified as requirements can be inferred from the properties of the constituent components, the need to analyze the properties of each system configuration in isolation is also avoided. Consequently the effort of creating the safety case is lowered. The second sub-contribution is the definition of a set of transformation rules from the CBS specification of a PL to a well-established safety-case notation, the Goal Structuring Notation (GSN) [2]. The result is, from a formal standpoint, a sound argumentation structure, and a set of verification obligations that if evidences showing that these obligations hold are provided, then the safety case complete-by-construction.

There are several notable approaches for safety case construction in the PLE context [14, 26, 35, 37]. Work in [26] presents a method for assurance of configurable architectures, based on established safety case patterns [15], and it enables explicit management of the argumentation belonging to each system configuration. Although the variability of the safety case argumentation is explicitly managed, the soundness and the completeness of the overall argumentation can be verified only by analyzing safety case argumentation for each system configuration. As discussed earlier, this is not feasible in domains with a high number of configurations and the approach in the present paper is purposely developed to avoid this issue. The approach in [14] is building on top results in [26] and automates the construction of a safety case for Simulink-based [12] specification of a PL. Although the automation reduces the effort of constructing a safety case for particular class of PLs, because of the underlying framework it is still necessary to perform per-configuration analysis of safety case soundness and completeness. An approach based on safety contracts, can be found in [35, 37], and its implementation is part of the AMASS Tool Platform [4]. Although potentially scalable to PL safety cases, work in [35] and [37] primarily focuses on a concept from ISO 26262, the so-called Safety Element out of Context (SEooC). In contrast, the approach in the present paper considers safety cases for arbitrary PLs and leverages the CBS which is more formal and more expressive. Furthermore, unlike [35] and [37], the obtained safety case argues for the safety of the overall system in each of its configurations, compared to arguing only about the SEooC. A detailed discussion on related work, can be found in Section 6.

The paper is structured as follows. Section 2 summarizes the basics of PLE and CBS. Section 3 presents the CBS extension for specification of PLs. 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 development paradigm [5, 30], 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 \( \mathcal{V} = (F, \mathcal{C}) \) where where \( F = \{ f_1, \ldots, f_n \} \) is a set of declared features which are either Boolean or Integer variables, and \( \mathcal{C} \) is a set of Boolean constraints over the features in \( F \).

An example of a constraint is \( f_1 \land f_2 > 100 \Rightarrow f_3 \). 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 \( \gamma \) is a set feature-value assignments \( \gamma = \{ f_i = \text{value}_i \}_{i=1}^n \). A product configuration for which each constraint in \( \mathcal{C} \) evaluates to true is valid.

From hereon, terms product configuration and configuration will be used interchangeably and we will consider only valid configurations. For intuition, real industrial PLs often contain feature models with thousands of features and consequently define hundreds of thousands of valid configurations [9, 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 \( \varphi \). They are written using the standard logical operators \( \land, \lor, \neg \), 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 the features representing the characteristics of the product.

2.2 Basic CBS framework

In order to support rigorous design of complex and heterogeneous systems, several lines of research have presented CBS frameworks [6, 8, 11, 33, 40, 41]. As noted in [7], one of the main strengths of CBS is that it formally captures two central systems engineering principles: vertical design refinement across the development process, and horizontal composition of logical or physical components at a given abstraction level. Although with a slightly different notation, and with an emphasis on the use of CBS for requirements engineering, this section summarizes the CBS concepts presented in [39] and [8] that are relevant for safety case construction.

In order to illustrate the CBS concepts we use fragments from an application example (cf. Section 5) which is the Fuel Level Display (FLD) system that is a part of each Scania CV AB vehicle. The safety requirement on the FLD system 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 fuel volume higher than the actual one can lead to running out of fuel which in turn leads to sudden engine stop and loss of servo-steering which is essential for heavy vehicles due to heavy loads.
2.2.1 CBS concepts. Basic concepts of CBS are:

i) component \( C \),

ii) specification \( S \),

iii) component implements a specification, denoted as \( C \triangleright S \),

iv) specification \( S_i \) fulfills specification \( S_j \), denoted \( \text{fulf}(S_i, S_j) \),

v) n-ary component composition that results in new components, denoted as \( C' = C_1 \otimes ... \otimes C_n \).

A specification corresponds to a requirement and it specifies 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 specified in a specification. The notion of fulfillment between \( S_i \) and \( S_j \) represents the intention that if a component implements specification \( S_j \) then it will also implement specification \( S_i \). In [6], 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 \( C \triangleright (A, G) \), if \( \forall C_e, C_e \triangleright A \Rightarrow C_e \triangleright C \triangleright G \).

The main idea behind contracts is the separation of responsibilities. Given the contract \((A, G)\), the component \( C \) can be developed independently of other components but when it is composed with a component \( C_e \), often referred as the environment of \( C \), which implements \( A \), then the composition implements \( G' \). We consider that each contract is intended to be satisfied by one and only one component. This intention is expressed by saying that a contract \( K_i \) is allocated to a component \( C_j \), and denoted \( \text{allTo}(K_i, C_j) \).

The intended semantics of the statement \( \text{fulf}(S_i, S_j) \) is that the property expressed in \( S_i \) logically entails the property expressed in \( S_j \), denoted as \( S_i \models S_j \). Given that specifications are implemented by components, the fulfills relation between specifications \( S_i \) and \( S_j \) expresses the intention that \( \forall C \models S_i \Rightarrow C \models S_j \).

**Assumption 1.** The composition operator is commutative and associative.

The composition operator must be commutative and associative in order to enable that independently developed components can be composed in any order. 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 like SW, HW, or mechanical components, but depending on the system’s level of abstraction being considered, 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 functions. From hereon, we introduce the shorthand \( C' = \bigotimes_{i=1}^{n} C_i \) that replaces \( C' = C_1 \otimes ... \otimes C_n \) notation.

Figure 1 shows the graphical representation of the CBS concepts introduced so far. Components are represented as rectangles and the composite component \( C_{FLD} \) has subcomponents \( C_{COO} \) and \( C_{BMS} \) that has further subcomponents \( C_{B-EST} \) and \( C_{B-CAN} \). Components 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. allTo((\( A' \), \( G' \)), \( C_{FLD} \)). Each fulfills relation is represented by a line ending in an arrow, while the line ending with a small filled circle represents 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 \((A_6, G_5)\).

### 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 forms a specification structure.

**Definition 2.5 (Specification structure)***. Let \( C \) be a possibly empty set of components organized into a rooted tree, \( \mathcal{K} \) be a possibly empty set of contracts such that each contract \((A_i, G_i)\) in \( \mathcal{K} \) is allocated to a single component in \( C_i \in C \), and the root component \( C' \) in \( C \) is such that \( C' = \bigotimes_{j=1}^{m} C_j \) where \( C_j \in C \setminus \{C'\} \). Then, a specification structure \( \mathcal{D} \) for \( C \) is a directed graph, i.e. \( \mathcal{D} = (N, E) \), if

i) each node \( n \in N \) is a specification \( A_i \) or \( G_i \),

ii) each edge \( e \in E \) corresponds to a single fulfill or assumptionOf relation between two specifications, and is denoted as \((S_i, S_j)_{A} \) or \((S_i, S_j)_{G} \),

iii) for each \((S_i, S_j)_{A} \) and \((S_i, S_j)_{G} \) it holds that \( S_i \neq S_j \),

iv) for each edge \((S_i, S_j)_{A} \) it holds that \( S_i \) is an assumption, \( S_j \) is a guarantee, and \( S_i \) and \( S_j \) is a contract \((S_i, S_j)\),

v) for each edge \((S_i, S_j)_{A} \) it holds that

a) if \( S_i \) is an assumption then it holds that if \( \text{allTo}(S_i, G_1, C_1) \), and \( \text{allTo}(A_i, S_j, C_j) \) then \( C_j \) is a subcomponent of \( C_i \),

b) if \( S_i \) is a guarantee then it holds that if \( \text{allTo}(A_i, S_i, C_i) \), and \( \text{allTo}(A_i, S_j, C_j) \), then there exists a component \( C_x \) whose subcomponents are \( C_i \) and \( C_j \).

vi) for each edge \((S_i, G_j)_{A} \) it holds that \( S_i \) is a guarantee such that \( \text{allTo}(A_i, S_i, C_i) \), \( \text{allTo}(A_i, G_j, C_j) \), and \( C_i \) is a subcomponent of \( C_j \).

Def. 2.5 corresponds to the concept of contract structure in [40] and it provides an intuitive way to visualize how the property in the
A specification structure \( \mathcal{D} \) is proper if

i) for each assumption \( A_i \) of a contract \( K_i \) such that \( \text{allTo}(K_i, C_i) \), there exists a specification \( S_k \) and an edge \( (S_k, A_i) \), where \( S_k \) is either an assumption such that \( \text{allTo}(S_k, G_k), C_{\text{EST}} \) and \( C_i \) is a subcomponent of \( C_k \) or \( S_k \) is a guarantee such that \( \text{allTo}(A_k, S_k), C_{k+i} \) and there exists a component \( C_{k+i} \) whose subcomponents are \( C_i \) and \( C_k \).

ii) for each guarantee \( G_j \) of a contract \( K_j \) such that \( \text{allTo}(K_j, C_j) \) where \( C_i \) is a composite component \( C_i = \bigotimes_j C_j \), there exits a guarantee \( G_j \) and an edge \( (G_j, G_i) \), where \( \text{allTo}(A_j, G_j), C_j \).

iii) \( \mathcal{D} \) is acyclic.

Going back to the example in Figure 1, even if the relation \( \text{fulfill}(G_{10}, A_6) \) would be removed, the example would still not be a proper specification structure because of all the relations labeled \((++)\). Relation \( \text{fulfill}(G_5, A_6) \) 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.

The following theorem shows the key idea of CBS, i.e. the compositional reasoning about system properties and it corresponds to the dominance relation defined in [22].

**Theorem 2.7.** Given a set of components \( C_i \in \mathcal{C} \), a set of contracts \( K_i \in \mathcal{K} \) allocated to components from \( \mathcal{C} \) 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 \( \text{allTo}(K', C') \), where \( C' \) is the root component, if

i) \( \forall K_i \in \mathcal{K}_A \exists C_i \ni K_i \),

ii) contracts \( K_j \in \mathcal{K} \) form a proper specification structure for \( \mathcal{C} \),

iii) \( \forall \text{fulfill}(S_i, S_j). S_i \ni S_j \),

then it holds that: \( C' \ni K' \).

The theory that has been presented 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 part of the contribution, which is a PLE-oriented extension of the theory presented in Section 2.2 and Section 2.3. Before the formal definitions of the extension, 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 system configuration would have to be specified as a specification structure \( \mathcal{D} \), as indicated by the upper part of Figure 2, and the reasoning if each system 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 a merge of the two configurations from the upper part of the figure where the dashed line of component \( C_{B-EST} \) indicates that this component exists only in some configurations of the \( C_{FLD} \).

The applicability of a particular component in the merged model to a particular configuration is defined using presence conditions. As depicted by the bottom part of Figure 2, each component and each specification is labeled with a presence condition denoted \( \varphi_i \).

Contrary to the approach in [36] where presence conditions are assumptions of contracts, the presence conditions in the present paper are orthogonal to components and specifications because presence conditions of an artifacts and the artifact can evolve separately.
3.1 CBS model of a configurable system

Assume that a feature model of a PL defines \( n \) configurations \( \gamma_1, \ldots, \gamma_n \). In order to avoid creating \( n \) different specification structures, we introduce the \emph{PL specification structure} \( \mathcal{E} \) which represents each of the \( n \) configurations.

**Definition 3.1 (PL Specification Structure).** Let \( C \) be a possibly empty set of components organized into a rooted tree, \( \mathcal{K} \) be a possibly empty set of contracts such that each contract \((A_i, G_i) \in \mathcal{K}\) is allocated to a single component in \( C_i \in C \), and the root component \( C' \in C \) is such that \( C' = \bigotimes_j C_j \) where \( C_j \in C \setminus \{C'\} \).

Then, a PL specification structure \( \mathcal{E} \) for \( C \) is a directed graph \( \mathcal{E} = (N, E) \) such that

(i) conditions (i)-(vi) of Def. 2.5 hold,

(ii) a labeling function \( P \) labels each assumption \( A_i \), guarantee \( G_i \), and component \( C_i \) with a presence condition \( \phi_i \).

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

**Definition 3.2 (Proper PL specification structure).** A PL specification structure \( \mathcal{E} \) 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 \( \mathcal{E} \) contains cycles and therefore the condition (iii) from Def. 2.6 does not apply to Def. 3.2. However, because allowing circular PL specification structures does not bring any conceptual benefits but increases the complexity of the following exposition, we introduce the following assumption.

**Assumption 2.** A PL specification structure \( \mathcal{E} \) 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 \( \gamma \). From here on, we consider that each presence condition evaluates to true or false for any given configuration \( \gamma \).

3.2 Constraints on presence conditions

As outlined in section 2.1, artifacts are labeled with presence conditions in order to be able to automatically obtain the set of artifacts that apply to a certain configuration \( \gamma \) by evaluating each presence condition and selecting the ones whose presence conditions evaluate to true.

We say that a PL specification structure \( \mathcal{E} \) is \emph{instantiated} for a configuration \( \gamma \), if each \( A_i, G_i \), and \( C_i \) whose presence condition evaluate to false for a given \( \gamma \), is removed from \( \mathcal{E} \). The result of instantiation, denoted as \( \mathcal{E}_\gamma \), 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 \emph{invariance with respect to configurations}, hereinafter only \emph{invariant}. In the following definitions, the symbol \( \models \) represents logical entailment with respect to the set of constraint in the set \( \mathcal{E} \) of a feature model \( V \). More formally, \( f_i \models f_j \) is equivalent to \( f_i, \mathcal{E} \models f_j \). For example, \( f_i \land f_2 \models f_3 \) but if the set \( \mathcal{E} \) contains the constraint \( f_1 \rightarrow f_3 \) then it holds that \( f_1 \land f_2 \models f_3 \).

**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 \((S_1, S_1) \ldots (S_n, S_n)\), assumption \( S \) is \emph{invariant} if \( P(S) \models \bigvee_{k=(S_i, S)} P(S_i) \).

**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 \emph{invariant} if \( P(S) \models \bigvee_{k=(S_i, S)} P(S_i) \).

**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 \emph{invariant} if \( P(G) \models P(C) \) and \( P(A) \models P(C) \).

**Definition 3.6 (Invariant Composition).** Composition of atomic components \( C_1, \ldots, C_n \) into a composite component \( C' = \bigotimes_j C_j \) is invariant if it holds that \( \forall C_i, P(C_i) \models P(C') \).

Given a PL specification structure \( \mathcal{E} \) with invariant assumptions, guarantees, allocation, and composition, instantiating \( \mathcal{E} \) for an arbitrary configuration \( \gamma \), 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 \( \mathcal{E} \), if

i) each assumption is invariant,

ii) each guarantee is invariant,

iii) each allocation of a contract \((A_i, G_i)\) is invariant,

iv) component composition is invariant,

then each instantiation \( \mathcal{E}_{\gamma_i} \) is a specification structure.

**Proof.** The proof can be found in [1].

Theorem 3.7 established the conditions under which instantiating the PL specification structure \( \mathcal{E} \) for any configuration \( \gamma \) will result in a specification structure. However, as shown by Theo- rem 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 PL specification structure \( \mathcal{E} \), such that each instantiation \( \mathcal{E}_{\gamma_i} \) is a specification structure, if

i) for each assumption \( A_i \) of a contract \((K_i, C_i)\) it holds that \( P(A_i) \models \bigvee_{f \in \text{AllTo}(K_i, C_i)} P(S_k) \) where \( S_k \) is either a guarantee of a contract \((K_k, C_k)\) such that \( \text{allTo}(K_k, C_k) \) and exists a component \( C_x \) whose subcomponents are \( C_i \) and \( C_j \), or \( S_k \) is an assumption of a contract \((K_k, C_k)\) such that \( \text{allTo}(K_k, C_k) \) and \( C_j \) is a subcomponent of \( C_i \),

ii) for each guarantee \( G_j \) of a contract \((K_j, C_j)\) and \( C_j \) is a composite component, it holds that \( P(G_j) \models \bigvee_{f \in \text{AllTo}(G_j, C_j)} P(G_i) \) where each \( G_i \) of a contract \((K_i, C_i)\) is such that \( \text{allTo}(K_i, C_i) \) and \( C_i \) is a subcomponent of \( C_j \),

then each \( \mathcal{E}_{\gamma_i} \) is proper.

**Proof.** The proof can be found in [1].
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 \( \{y_i\}_{i \in \mathbb{N}} \), and a PL specification structure \( \mathcal{E} \), if
i) premises of Theorem 3.7 hold, ii) premises of Theorem 3.8 hold, iii) \( \forall u \in \mathcal{E}(S_i, S_j), S_i \models S_j \)
iv) for each contract \( K_i \) allocated to an atomic component \( C_i \) it holds that \( C_i \triangleright K_i \),
then it holds that \( C' \triangleright (A', G') \) for each \( y_i \).

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 evidences 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 \( \mathcal{E} \)**

This section presents the second part of the contribution, which is a method for the construction of a safety case which is expressed using the GSN notation and 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 [2] 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" [2]. More specifically, GSN models form graphs called goal-structures. Figure 3 shows the GSN elements considered in the present paper. For a complete description of GSN notation, the reader is referred to [2].

![Figure 3: GSN elements used in the present paper](image)

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 holds, v) **Context** explicitly declares the scope of a claim, and vi) **Module** is a container for self-contained piece of argumentation. The presented elements can be connected via the **supportedBy** link, which allows decomposing goals to subgoals and connecting them to solutions, or via **inContextOf** links, which allow relating goals and evidence with assumptions and contexts.

In order to present a formal transformation of a CBS model to GSN notation, and in accordance with definitions from [15], a GSN model is defined as follows.

**Definition 4.1 (GSN model).** A GSN model \( G \) is a tuple, \( G = (B, A, T_B, T_A, d) \), where:

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

**4.2 Constructing a safety case**

A safety case can only be developed with respect to a particular standard with which it should show compliance. 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.

In summary, product-based argumentation must show that the system 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 pieces of argumentation 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 assigned property \( G' \) under the assumption \( A' \), 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 fu11(S_i, S_j) it should be verified that S_j ⊨ S_i, and for each contract-atomic component pair a11To(K_i, C_j) it should be verified that C_j ⊢ K_i by using SW or HW verification techniques such as testing, formal verification etc. Aggregating argumentation, supported by evidence, that all of the previously mentioned conditions hold, corresponds to an argument claiming that the system satisfies the contract (A', G'), where from a formal standpoint, and because it is based on a CBS model, this argument is complete-by-construction.

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 ∈ 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 TGR(b) = Goal,

ii) for each node b_i where TGR(b_i) = Goal, there exists a path to a node b_j where TGR(b_j) = Solution.

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 ̂Ω, the corresponding GSN safety case argumentation-structure is constructed as follows:

i) for each contract (A', G') such that a11To(A', G'), C') where C' is the composite component representing the system, create a GSN node b_i such that TGR(b_i) = Goal and d(b_i) = "C' satisfies (A', G') for each configuration y'";

ii) create three GSN nodes b_1, b_2, b_3 such that TGR(b_1) = TGR(b_2) = TGR(b_3) = Strategy, and TGR((b_1, b_2)) = TGR((b_1, b_3)) = TGR((b_2, b_3)) = 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", 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 TGR(b_j) = Goal where TGR(b_j) = 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_jx where TGR(b_jx) = Solution, TGR((b_j, b_jx)) = supportedBy and d(b_jx) refers to the evidence about presence conditions entailment.

iv) for each pair fu11(S_i, S_j) create a GSN node b_f such that TGR(b_f) = Goal, where TGR((b_f, b_j)) = supportedBy and d(b_j) = "S_i entails S_j". Also, for each b_f create a GSN node b_f_i where TGR(b_f_i) = Solution, TGR((b_f, b_f_i)) = supportedBy and d(b_f_i) refers to the evidence of specification entailment.

v) for each pair a11To((A_i, G_i), C_j) where C_j is an atomic component, create a GSN node b_v such that TGR(b_v) = Goal, where TGR((b_v, b_j)) = supportedBy and d(b_v) = "C_j satisfies (A_i, G_i)." Also, for each b_v create a GSN node b_v_x where TGR(b_v_x) = Solution, TGR((b_v, b_v_x)) = supportedBy and d(b_v_x) refers to the evidence of contract (A_i, G_i) being satisfied.

The presented transformation rules can be easily transformed 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 construction of FLD system safety case.

5 EXAMPLE APPLICATION

Figure 5 depicts the PL specification structure for the subset of all configurations of the FLD system, and the corresponding CBS-based part of the safety case for the FLD system. This example is a simplification of the complete FLD technical architecture presented in [39], but unlike in [39], the example in Figure 5 considers several 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 the COOrdinator Electronic Control Unit (ECU), and C_BMS representing 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_4-SENS and a digital one represented by C_4-SENS. Component C_COO has subcomponents C_C-EST which is C-function performing the fuel level estimation based on the sensor readings, and the subcomponent C_C-CAN which is a C-function responsible for transmitting the estimated value over Controller Area Network (CAN) to the component C_ICL which represents the vehicle Instrumentation Cluster (ICL). Similarly, the component C_BMS has the subcomponent C_B-EST also performing the fuel level estimation but only in particular FLD configurations in which C_COO does not have this responsibility, and component C_B-CAN which transmits the estimated value to C_COO which in turn passes it further to C_ICL. It should be noted that the FLD system architecture contains a single logical component representing the fuel
volume estimator but because this component can be allocated to two different ECUs, components $C_{C-EST}$ and $C_{B-EST}$ capture the two different SW implementations of the same logical component.

Each component with a dashed border is used only in a subset of all $C_{FLD}$ configurations. Actually, 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 and 8 of $C_{COO}$. Representing each of these is beyond the scope of the paper, and except $C_{a-SENS}$ and $C_{d-SENS}$ which explicitly represent two variants of the fuel volume sensor, all other components 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 expressed through presence conditions\(^1\). The presence conditions $\phi_1 - \phi_33$ were manually extracted from tools that manage requirements, SW and other artifacts. 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]. However, because work in [39] considered only a single FLD configuration, the PL specification structure in Figure 5, contains additional contracts such as ($A_a, G_5$) 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 that guarantee 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 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 and ultimately $C_{COO}$ can transmit the estimated value over CAN to $C_{ICL}$ where the estimated value is indicated.

\(^1\)Feature names have been modified due to confidentiality reasons.

---

**Table 1: Examples of presence conditions and contracts**

<table>
<thead>
<tr>
<th>Presence conditions from the FLD system</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\varphi_{20}$ = $(F1.1 \land F1323.5) \lor (F1.2 \land F1323.5) \lor (F1.1 \land F1323.6 \land RD \geq 201706)$</td>
</tr>
<tr>
<td>$\varphi_{21}$ = $[F2482.1 \lor (F2482.2 \land F1940.1)] \lor [F1837.1 \lor (F1837.19 \land F379.26)] \lor [(F37.1 \land F2719.1 \land 201404 \leq RD \leq 201801)] \lor (F37.3 \land F2719.1)$</td>
</tr>
<tr>
<td>$\varphi_{35}$ = $(F1.1 \land F1.2) \lor F.1323.5$</td>
</tr>
<tr>
<td>$\varphi_{37}$ = $F1.1 \land F1323.6 \land RD \geq 201705$</td>
</tr>
<tr>
<td>$\varphi_1$ = $\varphi_2 = F1.1 \lor F1.2$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Contracts from the FLD system</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A'$: The ignition state equals true.</td>
</tr>
<tr>
<td>$G'$: Indicated fuel volume deviates at most 7% compared to actual fuel volume.</td>
</tr>
<tr>
<td>$A_{12}$: Estimated fuel volume update is updated every 1s.</td>
</tr>
<tr>
<td>$G_{11}$: Indicated fuel volume deviates at most 7% compared to actual fuel volume.</td>
</tr>
<tr>
<td>$A_{31}$: Measured fuel volume corresponds to the actual fuel volume.</td>
</tr>
<tr>
<td>$A_{9}$: Estimated fuel volume is received over CAN every 60ms.</td>
</tr>
<tr>
<td>$G_{9}$: Estimated fuel volume is transmitted over CAN every 900ms.</td>
</tr>
</tbody>
</table>

---

**5.1 Safety Case Argumentation-Structure**

Given the PL specification structure of the FLD system, the right-hand side of Figure 5 depicts CBS-based part of the FLD safety case. According to the transformation rules from Section 4.3, the goal root node, arguing that "$C_{FLD}$ satisfies $A'(x'), G'$ in each $y'$", is broken down through three strategy nodes into sub-goals arguing that the premises of Corollary 3.9 hold.

While the construction of the GSN model on the right-hand side of Figure 5 is straightforward, judging if the evidence for each sub-goal can be obtained without disrupting enterprises processes and tools is the true test of the approach feasibility. The first set of evidences relates to proving that premises of Theorem 3.7 and Theorem 3.8 hold, i.e. proving logical entailment between Boolean

---

**Figure 5:** PL specification structure of the FLD system (left), and the corresponding safety case argumentation-structure (right)
expressions with respect to a set of Boolean constraints. Although the number of features is high, and the number of constraints in $C$ is high, because the features and the constraints are primarily Boolean, encoding and checking the satisfiability of various entailments using efficient SMT solvers such as Z3 [13] is simple and feasible.

The second set of evidences relates to proving that for each pair of specification $full(S_i, S_j)$ it holds that $S_i \models S_j$. In the general case, this is a hard problem because specifications $S_i$ and $S_j$ can be such that it is not possible to formalize them and check entailment using a single technology. 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 as with $G_{11}$ and $G'$ or even without formal analysis, it is obvious that $S_j$ fulfills $S_i$ as with $G_6$ and $A_6$. The reason for this is that when specifying a component in terms of a contract over component interface, not matching assumptions and guarantees of components is immediately counterintuitive. The third set of evidences relates to proving that for each contract and atomic component such that $allTo(K_i, C_i)$ it holds that $C_i$ satisfies $K_i$. Providing this type of evidences is the most difficult of the three because proving that $C_i \supset K_i$, means formally verifying the behavior of $C_i$. On the one hand, functional safety standards recommend that safety-critical software should be of low complexity and in addition formally verified. Here, formal verification approaches of SW against CBS specifications produce good results [24]. On the other hand, introducing large-scale formal verification into an enterprise as a part of everyday processes is still not realistic which means that majority of evidences for arguments of type $C_i \supset K_i$ will be based on testing as exemplified by the evidence discovered by analyzing the artifacts related to the atomic components of the $C_{FLD}$ system. Even without formal verification, verifying SW behavior using unit-testing is a de-facto standard in SW development practice and the creation of the third type of evidences is certain. The important thing to note is that by basing the PL safety case on the CBS framework which is compositional, the need to perform integration testing is removed and the already existing unit-testing process is sufficient for the production of evidences needed for the safety cases. Because testing is a technique that does not provide any guarantees, existing methods for the assessment of the confidence [16, 38] in safety case arguments can maybe be lifted to assessing confidence in a PL safety case.

From the safety case construction-process viewpoint, the presented method significantly reduces the amount of manual work because it defines a general argumentation structure applying to any PL of any size and complexity. Because each of the evidences is a precise verification obligation, detecting if the PL specification structure evolution leads to a violation of either of the obligations can be used for continuous analysis of the PL consistency and correctness. Furthermore, the approach allows iterative creation of the safety case because except the evidence needed to show that $C_i \supset K_i$, the premises of Corollary 3.9 can be verified stepwise in parallel with PL development. A relevant issue hindering 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 lightweight data-integration [17] 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 [32].

6 RELATED WORK

Two main approaches have been proposed for construction of safety case argumentation. The first line of research has focused on capturing safety case patterns that have been successfully used to argue for different properties [10, 15] while the second line of research has focused on generating safety case arguments from various types of artifacts [14, 26, 37]. In the area of PLE, most notable contributions [14, 18, 19, 21, 26, 36, 37] come from the latter group.

Work in [25] uses safety-analysis artifacts to construct a safety case in the PL context which shows that contributions of identified faults to potential failures is acceptable. Work in [26] applies the method from [25] for creating a safety case in a PL context which argues that a configurable PL architecture is acceptably safe. However, unlike our approach, the approach in [25] does not consider the requirements that are defined for each identified fault and whether the system actually implements these requirements. The tool-supported method presented in [14] provides an automated way to construct a modular safety case based on the meta-models presented in [25] given input information from a variability management tool, architecture specification tool, and a hazard and safety-analysis tool. Work in [37] provides an approach for the creation of reusable safety cases fragments based on system specification in terms of weak and strong contracts that incorporate assume-guarantee reasoning. The motivation for the approach is based on the concept of Safety Element out of Context from the standard ISO 26262 and the essence of the approach is to perform hazard analysis and risk assessment using a variant of Fault Propagation and Transformation Calculus (FPTC) [20] and then map the output of FPTC onto introduced safety contracts. As argued in Section 1, the present paper considers development and assurance of arbitrary PLs and consequently relies on CBS paradigm where the constructed assurance case argues for system level properties instead of only SEooC. The line of research in [18, 19, 21] introduces approaches for constructing 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 aim is a general-purpose framework which enables constructing safety cases for arbitrary specifications allocated to configurations of a PL in the form of contracts.

7 CONCLUSION

As the criticality of the tasks performed by the systems from domains such as aerospace or automotive increases, the ability to construct safety cases for these systems is of outmost importance. Unfortunately, current methods for safety case construction fall short in scenarios when it is required to construct a safety case for a system with a huge number of configurations, i.e. for a PL.

In order to alleviate this problem, the present paper introduces a general and novel approach for the construction of a safety case for the complete PL, instead for each product configuration individually. In order to establish a unified representation of a PL, the first part of the contribution is an extension of a general CBS framework with the ability to specify a PL by using a mix of established CBS and PLE principles. The result of this extension is Corollary 3.9, which summarizes the constraints on CBS-based specification of PLs that
preserve the compositional reasoning of the basic CBS framework. This allows reasoning about the properties of all PL configurations by inspecting only their constituent components. The second part of the contribution defines a set of transformation rules from a CBS-based specification of a PL to a PL safety case argumentation-structure expressed using the GSN notation. Because of the formal CBS foundations, the PL safety case argumentation-structure is sound and complete-by-construction. Moreover, the approach facilitates iterative safety case construction in parallel with the PL development, and a way to detect errors in the CBS model of a PL that lead to the inability to construct a PL safety case using the provided transformation rules.

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

ACKNOWLEDGMENT

- and - are funded by the ECSEL PRYSTINE project and supported by Scania CV AB. - is funded by EU and VINNOVA via the ECSEL Joint Undertaking, agreement No. 692474, AMASS project.

REFERENCES

[1] [n. d.]. Theorem 3.7 and Theorem 3.8 proofs. Separate document created in order to preserve the double-blind process. https://www.dropbox.com/sh/wxjox1e1ef9p91uc/AADcLWUNqX9KqZGyVu34q24N?dl=0