Using Safety Contracts to Guide the Integration of Reusable Safety Elements within ISO 26262

Irfan Šljivo Barbara Gallina Jan Carlson
Hans Hansson
Mälardalen Real-Time Research Centre, Mälardalen University,
Västerås, Sweden
{irfan.sljivo, barbara.gallina, jan.carlson, hans.hansson}@mdh.se

Abstract

Safety-critical systems usually need to be compliant with a domain-specific safety standard, which in turn requires an explained and well-founded body of evidence to show that the system is acceptably safe. To reduce the cost and time needed to achieve the standard compliance, reuse of safety elements is not sufficient without the reuse of the accompanying evidence. The difficulties with reuse of safety elements within safety-critical systems lie mainly in the nature of safety being a system property and the lack of support for systematic reuse of safety elements and their accompanying artefacts. While safety standards provide requirements and recommendations on what should be subject to reuse, guidelines on how to perform reuse are typically lacking.

We have developed a concept of strong and weak safety contracts that can be used to facilitate systematic reuse of safety elements and their accompanying artefacts. In this report we define a safety contracts development process and provide guidelines to bridge the gap between reuse and integration of reusable safety elements in the ISO 26262 safety standard. We use a real-world case for demonstration of the process, in which a safety element is developed out-of-context and reused together with its accompanying safety artefacts within two products of a construction equipment product-line.

1 Introduction

The basis for building modern safety-critical systems often lies in reusing existing components [2]. Most of these systems need to comply with a domain specific safety standard that often requires a safety case in form of a clear and comprehensible argument supported by evidence to show why the system is acceptably safe. The safety standards typically do not provide detailed guidelines for reusing safety elements and the accompanying artefacts, which makes the integration of the elements and the provided evidence challenging [3]. For example, the automotive safety standard ISO 26262 [12] supports reuse through the notion of Safety Elements out of Context (SEooC), which are elements explicitly developed for reuse according to ISO 26262. While the standard provides

requirements and recommendations on which information is needed for the integration of SEooC, guidance on performing systematic reuse is missing.

Since safety is a system property, traditional safety analyses such as Fault Tree Analysis (FTA) and other safety artefacts such as safety arguments are made on the system level. Reusing such artefacts is difficult since what is safety relevant in one system is not necessarily safety relevant in another system. Non-systematic reuse of safety artefacts has shown to be dangerous [13], hence there is a need to fill the gap created by the safety standards' lack of guidelines for systematic reuse and integration of safety elements and their safety artefacts.

Systematic reuse of safety artefacts can be achieved by generative reuse. The term "generative reuse" is used to indicate indirect reuse of artefacts [6], be it the code itself, results of a failure analysis such as FTA [14] or parts of safety arguments [10], where a customised artefact is generated for a specific context from specification written in a domain specific specification language. For example, an out-of-context component with a pre-developed safety argument is reused in a particular system. Such safety argument, produced out-of-context, might contain irrelevant information for the particular system. Instead of trying to reuse and integrate pre-developed safety arguments, the relevant information for the particular system is first identified from the provided artefacts, and then the corresponding system safety argument is generated from the identified information. In our work we use safety contracts for capturing safety-related information and for identifying the relevant information for a particular environment.

A contract is an assumption/guarantee pair, where a component offers guarantees about its own behaviour if the assumptions on its environment are met. Safety contracts are a specific types of contracts that deal specifically with component behaviours that are deemed relevant from the perspective of hazard analysis. In our previous work we showed how safety contracts can be used to support generative reuse of safety artefacts [18]. Since reusable elements can exhibit different behaviours in different environments, contracts are characterised as either strong or weak to allow capturing these different behaviours in a more flexible manner [17]. Furthermore, since the safety contracts deal with some of the information used in the safety arguments, we can use the contracts to semi-automatically generate the argument-fragments related to components [16].

In this report we complement the safety guidelines provided by ISO 26262 to include contract-specific activities and facilitate systematic reuse that aims at easing integration of safety-relevant components and the provided evidence within ISO 26262 systems. We first define the safety contracts development process and the corresponding contract-specific activities. Then we provide guidelines on how and when to use the contract-specific activities in the case of SEooC.

To demonstrate the proposed process we use a product-line scenario as a common real-world case. The product-line is composed of two construction machines whose compliance to ISO 26262 will be required in the near future. Both machines are equipped with lifting arms, whose software controller in both cases includes a component for automatic positioning of the arm in a predefined position. We develop this component as a SEooC and then reuse it within the two products. On this real-world case we demonstrate how safety contracts can be used for SEooC development. Moreover, we illustrate the benefit of generative reuse of safety arguments on the the SEooC integration within the

two products.

The contributions of this work are (1) the guidelines in form of safety contracts development process describing the role of the safety contracts within the development and integration of reusable components within safety-critical systems, (2) alignment of the proposed process with the ISO 26262 safety process, and (3) its demonstration in a real-world case. In contrast to existing works that focus on facilitating reuse of safety artefacts within safety-critical systems [7, 15, 11, 9], we focus on detailing the guidelines for development and integration of reusable safety elements within safety-critical systems via safety contracts. More specifically, we align the proposed process with ISO 26262 to facilitate generative reuse of safety artefacts, primarily safety arguments. We focus on providing means for capturing the SEooC assumptions recommended by the standard and support their validation during integration of the SEooC in an ISO 26262 compliant system. We assume that for the integration to work, both the SEooC and the target ISO 26262 system have safety contracts established.

The rest of the paper is structured as follows: In Section 2 we provide background information. We present the safety contracts development process and align it with the SEooC development process recommended by ISO 26262 in Section 3. In Section 4 we demonstrate the proposed process and the related guidelines on a real-world case. We provide discussion in Section 5 and related work in Section 6. Finally, conclusions and future work are presented in Section 7.

2 Background

In this section we provide background information on the ISO 26262 safety process and the processes recommended for development and integration of Safety Elements out of Context. Furthermore, we provide essential information on the strong and weak safety contracts as well as graphical argumentation notation for representing safety arguments.

2.1 ISO 26262

ISO 26262 [12] has been developed as a guidance to provide assurance that any unreasonable residual risks due to malfunctioning of E/E systems have been avoided. The standard requires a safety case in form of a clear and comprehensible argument to show that the safety requirements allocated to an item are complete and satisfied by the evidence generated during the system development. An *item* within ISO 26262 is composed of at least a sensor, controller and an actuator, which together implement a function at the vehicle level.

Central part of Fig. 3 shows the safety process of the ISO 26262 standard. The process starts with the *Concept phase* (Part 3 of the standard) that is initiated with the *item definition* activity where the main objective is to define and describe the item by capturing its dependencies on, and interactions with, its environment. In the subsequent activities of this phase the hazards related to the item are identified and classified according to Automotive Safety Integrity Levels (ASILs), safety goals are established and further refined into functional safety requirements that are allocated to the architectural elements.

In the first part of the *Product development at system level* phase, the technical safety requirements are derived from the functional safety concept, and the system is designed to comply with both the technical and functional safety requirements. Based on the system design, development and testing of both the hardware (HW) and software (SW) elements is performed. During *Product development at HW/SW level* (Parts 5&6 shown in Fig. 3) the corresponding HW/SW safety requirements are derived with consideration of environmental/operational constrains identified during the concept phase. The process continues with integration and testing of the HW/SW elements, followed by integration of elements that compose an item to form a complete system, and then the item is integrated with other systems and tested on the vehicle level. The *Product development at system level* is finalised with safety validation and an assurance case is presented to show that the safety goals are sufficient and that they have been achieved.

We provide additional information on the concept and system design phases as they play an important role in reuse of safety elements. Based on the ISO 26262 development process, the information that needs to be gathered during these phases includes the following: (1) purpose and functionality of the item, (2) operating modes and states of the item (including the configuration parameters), (3) law, regulation and standard requirements, (4) operational and environmental constraints, (5) interface definition, (6) hazard analysis results, including the known hazards, their ASILs and the associated safety goals.

To ease the development of ISO 26262 compliant systems, the standard acknowledges different reuse scenarios: (1) elements that have been developed for reuse according to ISO 26262 in form of SEooC, (2) pre-existing elements not necessarily developed for reuse or according to ISO 26262 that have to be qualified for integration, and (3) elements that qualify for reuse as proven-in-use. In this report we focus on the SEooC reuse scenario.

2.1.1 Safety Element out of Context

SEooC can be an element used to compose an item, but it cannot be an item since item implements functions at vehicle level, while a reusable elements such as SEooC are not developed in the context of a particular vehicle. The development of SEooC follows the ISO 26262 safety process, but since SEooC is developed out-of-context, the information related to the system context (gathered during the concept and system design phases) first needs to be assumed. The assumptions are made to the functional safety concept as the main output of the concept phase and the external design (system-level assumptions; the interactions with, and dependencies on the elements in the environment are assumed). After assuming the relevant system design, the development of the SEooC follows the product development at SW/HW level.

2.2 Safety Contracts

In our previous work [17], we have proposed a contract-based formalism with strong $\langle A, G\rangle$ and weak $\langle B, H\rangle$ contracts to distinguish between context-specific properties and those that must hold for all contexts. A traditional component contract $C = \langle A, G\rangle$ is composed of assumptions (A) on the environment of the component and guarantees (G) that are offered by the component if the

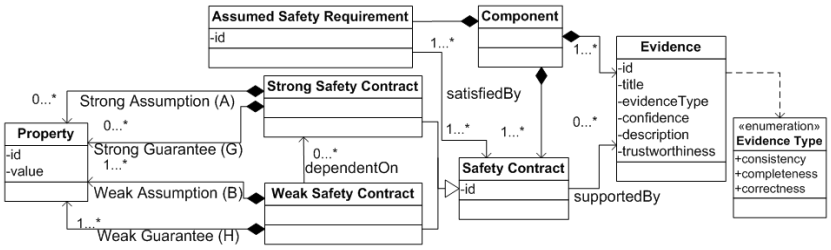

Figure 1: Component and safety contract meta-model

assumptions are met. The strong contracts capture behaviours of components that should always be guaranteed (strong guarantees G) and the corresponding strong assumptions (A) that should always be met. The weak contracts handle behaviours that are not required to hold in every environment (weak guarantees H), but only when besides all the strong assumptions, the corresponding weak assumptions (B) are satisfied as well. For example, strong contracts can be used to prevent misuse of configuration parameters of the component by requiring parameters scope and guaranteeing interaction of the different parameters, while weak contracts could be used to describe distinct component behaviours achieved by the different configurable parameter values. The *related contracts* of a contract C are those contracts that either assume the guaranteed properties of C or the ones which guarantee properties that are assumed by the contract C.

As introduced in Section 1, we call a contract capturing safety-relevant behaviour a *safety contract*. Since not all safety-relevant information can be captured in formal contracts, we recognise that contracts consist of both formal and informal assumptions and guarantees. The formal part of the contracts can for example be captured in form of Failure Propagation and Transformation Calculus (FPTC) syntax [18].

The component meta-model (Fig. 2) that connects safety contracts with supporting evidence provides a base for evidence reuse together with the contracts [16, 18]. The component meta-model specifies a component in an out-of-context setting, composed of safety-contracts, evidence and the assumed safety requirements. Each safety requirement is satisfied by at least one safety contract, and each contract can be supported by one or more evidence. This component meta-model is used as the basis for semi-automatic generation of safety case argument-fragments [16]. For example, if we assume that late output failure of the component can be hazardous, then we define an assumed safety requirement that specifies that late failure should be appropriately handled. This requirement is addressed by a contract that captures in its assumptions the identified properties that need to hold for the component to guarantee that the late failure is appropriately handled. The evidence that supports the contract includes the contract consistency report and analysis results used to derive the contract.

2.3 Overview of Goal Structuring Notation

The Goal Structuring Notation (GSN) [1] is a graphical argumentation notation that can be used to represent the individual elements (e.g., goals/claims,

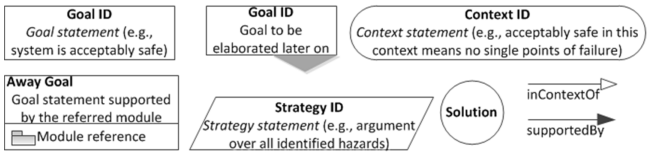

Figure 2: A subset of GSN symbols used within this article

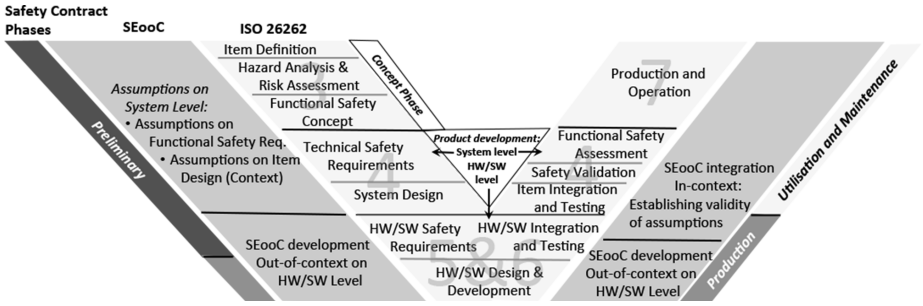

Figure 3: ISO 26262, SEooC and safety contract development phases mapping

evidence, context) of any safety argument. More importantlynly, GSN can be used to capture the relationships that exist between the individual elements by using the two relationships *inContext* and *supportedBy*. The *inContext* relationship connects claims with the contexts that are used as the clarifications of the related claims, while the *supportedBy* relationship is used for connecting goals with its subgoals, backing up goals with evidence and specifying the decomposition strategies used to decompose a goal to a set of subgoals. Basic symbols of GSN used in this article are shown in Figure 1 (with examples for each of the elements).

3 ISO 26262 Safety Process Supported by Safety Contracts Development Process

In this section we present the guidelines for using the strong and weak safety contracts for development and integration of reusable safety elements within safety-critical systems. Moreover, we align the guidelines with the ISO 26262 safety process. More specifically, we first define the safety contracts development process and the contract-specific activities, and then we detail how and when these activities can be aligned with the SEooC development process.

3.1 Safety Contracts Development Process

As mentioned in Section 1, the nature of safety being system property and the dangers of non-systematic reuse hinder reuse of safety elements within safety-critical systems. To alleviate these issues a clear process and guidelines on how to perform reuse should be provided to promote systematic reuse of safety elements. To integrate the systematic reuse approach based on strong and weak safety contracts within a safety process, a safety contracts development process

needs to be defined. We propose such a process divided into three phases: (1) *Preliminary* safety contracts, (2) Safety contracts *production*, and (3) Safety Contract *utilisation and maintenance*. The alignment of the safety contract, SEooC and ISO 26262 development phases is shown in Fig. 3. In the reminder of this subsection we provide more details about the corresponding contract-specific activities each phase is constituted of.

3.1.1 Preliminary Safety Contracts Phase

- *Establishing strong and weak contracts*: The strong contracts are established by considering behaviours such as nominal functional or safety mitigation behaviours not bound to context-specific configuration parameters. In contrast, weak contracts are established by considering behaviours bound to context-specific configuration parameters (e.g., accuracy of an algorithm may depend on the physical properties of the system in which it is used).

- *Enriching assumptions with environmental/operational constrains*: The different types of properties that should be captured by safety contracts include nominal functional behaviour, failure logic behaviour, resource usage behaviour and timing behaviour [9]. Upon establishing the strong and weak contracts, the contract assumptions need to be enriched to achieve sufficient level of completeness by including environmental properties such as platform properties, HW/SW interface and/or dependencies to other elements.

- *Preliminary matching of contracts to HW/SW safety requirements*: As mentioned in Section 2.2, the safety contracts should capture information needed to satisfy the safety requirements allocated to the corresponding safety element. For example, supporting each derived SW safety requirement allocated to a software element with at least one preliminary contract is the final goal in completing the set of the preliminary safety contracts. If the contract to satisfy a particular requirement has not been previously developed, a preliminary contract should be established with its guarantee reflecting the corresponding requirement.

3.1.2 Safety Contracts Production Phase

- *Actualisation of the contracts with implementation-specific properties*: Since not all information is fully known during the preliminary safety contracts phase, certain preliminary contracts (e.g., on resource usage) can only be captured with speculative targeted behaviour. After the product development stage, such contracts need to be finalised once the actual behaviour of the element (or a more accurate approximation) can be established. For example, when more accurate information about the actual accuracy of an algorithm, actual timing behaviour, or actual memory footprint of the element is available, then we can actualise the contracts capturing such behaviours with the actual implementation-specific values.

- *Supporting contracts with evidence*: The final step in producing the safety contracts for reuse is to support such contracts with the evidence supporting the information captured by the contracts. For example, in case

that information captured within a safety contract is based on simulation or testing results, the corresponding guarantee of the contract should be based on the results while the assumptions should capture the environmental parameters under which the simulation/testing has been performed. The artefacts related to the simulation/testing are then attached to the particular safety contract with a description in which way they are related. Further trustworthiness evidence can be attached to the artefacts [18]. Since each safety requirement is associated with an ASIL, which in turn influences the stringency of evidence that needs to be provided to assure that the particular requirement is satisfied, the achieved ASIL information is attached to the evidence rather than to the contracts themselves. In this way the safety requirements are connected to the achieved ASILs through the connection of the safety contracts with the associated evidence.

3.1.3 Utilisation and Maintenance Phase

- *Contract-based verification*: The results of unsuccessful verification can be interpreted as follows:
  - *Contracts are contradicting each other* (e.g., strong and weak contracts of the same component make contradicting assumptions on the same property). To address this result, either the existing contract assumptions and guarantees should be re-established or a replacement component should be used instead.
  - *Not all strong or relevant weak contract assumptions are satisfied*, which means that the component is either not compatible with the particular context or cannot satisfy a particular safety requirement allocated to it. In either case, the system can be re-designed to handle the incompatibility (e.g., by adding component wrappers to convert the input/output signals to a compatible format) or the reused component itself can be replaced or modified.
  - *Contract assumptions are incomplete*, e.g., safety contract on timing behaviour of component X guarantees the timing behaviour based on platform assumptions (including compiler configuration), while a related contract on timing behaviour of component Y does not include assumptions on compiler configuration. In this case the contracts with missing assumptions should be re-evaluated and either enriched with the appropriate assumptions or a new contract should be established to capture the newly identified behaviour.
- *Contract maintenance*: In case of changes to the existing contracts, all contracts of the corresponding component should be revisited, while when updating contracts with additional assumptions, only contracts capturing the same type of behaviour (e.g., timing) should be reassessed. Modifications of a component or system design requires that all its contracts are reassessed and reestablished if required.
- *Contract-based artefacts generation*: In this article we focus on the utilisation of contracts for safety case argument generation. The generative reuse potential of the contracts can be utilised for generating other artefacts, but that is out of scope of this article.

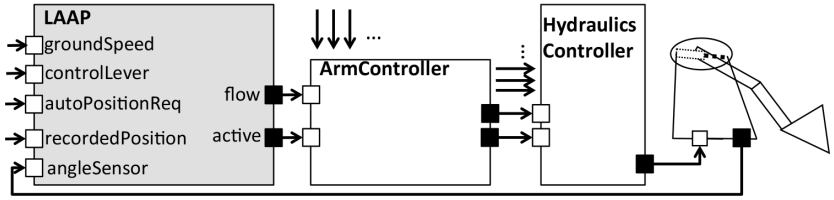

Figure 4: The assumed structure of the lifting arm unit context

3.2 SEooC Development with Safety Contracts

SEooC development starts by capturing the system-level assumptions as shown in Fig. 3. Simultaneously, the preliminary safety contracts phase is initiated, as described in the Section 2.1. All relevant assumed properties should be covered by the established preliminary contract assumptions. Once the HW/SW safety requirements are derived, each requirement is associated with at least one contract such that the behaviour achieved by the associated contracts satisfies the required behaviour by the corresponding requirement. After the safety contracts are established and associated with the safety requirements, the safety contract production phase and the corresponding ISO 26262 product development at HW/SW level are continued to develop the SEooC and its safety contracts. At this point the development of the SEooC out-of-context is completed.

Once the SEooC is used in a particular system (in-context), the assumed requirements are compared and matched to the actual safety requirements allocated to the element, and contracts are used to verify that the assumptions captured during the SEooC development are satisfied. The contract production phase continues in-context of a particular system to capture the behaviours of the SEooC that could not be established out-of-context. In case of assumptions mismatch, ISO 26262 impact analysis can be assisted by the contract maintenance activity. Once all the relevant safety contracts are satisfied for the reused SEooC, an argument for the element is generated to show the satisfaction of the safety requirements through the satisfaction of the associated safety contracts [16].

4 Real-world Case

In this section we demonstrate the application of the guidelines introduced in Section 3. We use a product-line based case for our demonstration as it is a common scenario found in industry. The aim of the case is to develop a component out of context of a particular product and reuse the component and its accompanying artefacts in two different products of a product-line. This can be challenging even for two similar products such as those in a product-line since the hazards related to the products can differ, as discussed in Section 1. The SEooC we develop is a Lifting Arm Automatic Positioning (LAAP) component commonly used within wheel-loaders and other construction machines. We first present the LAAP and its SEooC development, and then we discuss the LAAP integration within two different products of a wheel-loader product-line.

4.1 SEooC definition and development

As discussed in Section 3.2, the development of a SEooC starts by making assumptions on the item in which the component is intended to be used. The assumed structure of the lifting arm unit context for a wheel-loader is shown in Fig. 4. Wheel loaders are equipped with a lifting arm, which can perform up and down movements that are directly controlled by a hydraulic controller. The operator controls of interest for the development of the LAAP consist of a control lever that is used to lift/lower the arm and an automatic position request button that positions the arm in a predefined position. Once the automatic positioning is started, it can be stopped by moving the control lever and switching automatically to manual mode. Besides the operator controls, the LAAP uses an arm angle sensor to determine the current arm position, recorded position to which the arm should be moved and the ground speed of the vehicle for tracking the vehicle movements. The assumptions include only information deemed relevant to the SEooC development, hence the full interface of the arm controller is not assumed at this stage.

Before specifying the assumed software safety requirements that the LAAP will implement, we need to assume safety implications of the component and its relation to possible hazards. We identified contributions of LAAP to two possible vehicle-level hazards: *(H1) unintended movement of the lifting arm*, and *(H2) hydraulic leakage*. We consider the hazards in the following operational situations:

- high speed (the vehicle is moving with varying speeds that can go up to the maximum available speed)
- short cycle (a combination of load lifting and low speed transportation)
- load and carry (the vehicle is moving with varying ground speed with bucket fully loaded)

Hazard H1 can be dangerous during high speed due to e.g., heavy traffic when driving on a public road, during the short cycle and load and carry phases it can be dangerous to bystanders present in the area while high precision movement is required from the machine. LAAP can contribute to hazard H1 by e.g., value failure of the flow command that can be caused by value failures of the angle sensor and the recorded position variable. Furthermore, the unintended arm movement can occur in case of omission of the autoPositionReq signal. Omission or late failure of the control lever signal can cause LAAP to continue its operation when not intended.

The high-pressure hydraulic leakage could produce a highly flammable oil/air mixture spray mist that might ignite in contact with hot surface, hence the leakage should be identified as soon as possible. One way in which LAAP can contribute to this situation is when the component starts operating but due to the leakage the arm either never reaches the recorded position or it moves much slower than usual, which contributes to increasing the leakage. The occurrence of the hazard H2 in either of the operational situations can be danger to the driver, other participants in traffic and bystanders present in the area. To address these possible hazardous events related to both hazards, functional safety concept is assumed and the corresponding software safety requirements are derived (Table 1).

Table 1: SW Safety Requirements

SWSR1	Safe state shall be applied during high-speed	ASIL B
SWSR2	The stop position of the arm shall not deviate more than ± 0.04 rad	ASIL B
SWSR3	Safe state shall be applied if erroneous input (ground speed, angle sensor, control lever or recorded position) is detected	ASIL B
SWSR4	Safe state shall be applied if the operational time of the LAAP is taking more than the maximum raise time of the lifting arm	ASIL A
SWSR5	LAAP shall not start inadvertently	ASIL B
SWSR6	Safe state shall be applied when manual arm movement is in progress (i.e., when control lever value not 0)	ASIL B

The strong and weak contracts of the LAAP, initially captured during the *Preliminary Safety Contracts* phase to address the SW safety requirements, are shown in Table 2. The strong contract *LAAP-1* requires that the groundSpeedLimit is set below 20km/h and guarantees that LAAP will be disabled when the ground speed of the vehicle is greater than the groundSpeedLimit parameter. Disabling of the LAAP is the safe state achieved by setting the active flag to false and the flow value to 0.

The strong contract *LAAP-2* specifies the assumed value ranges of the input signals and guarantees that the safe state shall be applied when either of the inputs is out of bounds. In case of *controlLever* signal, the LAAP can be active only when the lever is inactive (i.e., when the lever is 0), hence the contract specifies that when *controlLever* is different than 0, the safe state shall be applied.

The strong contract *LAAP-3* describes a SW watchdog timer implemented as a part of the component that disables LAAP if its operation time is longer than expected. To detect possible hydraulic leakage, the timer is set within the interval bound by *raiseTime* parameter that represents the maximum lifting time of the arm under full load from lowest to highest position.

The weak contracts *LAAP-4* and *LAAP-5* capture failure propagation behaviour of the LAAP such that they state which conditions should the environment of the LAAP fulfil to mitigate a potentially hazardous failure propagation. The *LAAP-4* contract specifies that in order to avoid the flow command value failure, the environment of the LAAP should guarantee that the angle sensor signal and recorded position value do not exhibit value failure. The *LAAP-5* contract contract specifies that in order to mitigate inadvertent commands sent from the LAAP (in form of commission failures of the *flow* and *active* output ports), the environment should ensure that commission of the *autoPositionReq* signal and omission of the *controlLever* signal do not occur.

The weak contract *LAAP-6* relates the guaranteed *flow* accuracy and the lifting arm stop position based on the assumptions on the accuracy of the angle sensor, recorded position and the actuation.

The matching of the established contracts and the SW safety requirements is presented in Table 3. The contract *LAAP-4* is not fully addressing the requirement *SWSR2*, but it only establishes that the accuracy of the flow command is dependent on the accuracy of the angle sensor and the recorded position value. Hence a more concrete contract *LAAP-6* is established to fully address

Table 2: LAAP Safety Contracts

$\mathbf{A}_{LAAP-1}$:	*groundSpeedLimit* within [0, 20] km/h AND *groundSpeed* within [0, 200] km/h;
$\mathbf{G}_{LAAP-1}$:	*groundSpeed* > *groundSpeedLimit* **implies** (*active* = false and *flow* = 0)
$\mathbf{A}_{LAAP-2}$:	*groundSpeed* within [0, 200] km/h AND *angleSensor* within [0,3] rad AND *controlLever* within ± 1 rad AND *recordedPosition* within [0,3] rad;
$\mathbf{G}_{LAAP-2}$:	(*groundSpeed* not within [0, 200] km/h OR *angleSensor* not within [0,3] rad OR *controlLever* not 0 rad OR *recordedPosition* not within [0,3] rad;) **implies** (*active* = false and *flow* = 0)
$\mathbf{A}_{LAAP-3}$:	*watchdogTimerInterval* within [*raiseTime*, 1.2**raiseTime*] AND *raiseTime* > 0;
$\mathbf{G}_{LAAP-3}$:	(not (*active* = false and *flow* = 0) **implies** watchdogTimer start) AND (*LAAP-OperationalTime* > *watchdogTimerInterval* **implies** (*active* = false and *flow* = 0 and watchdogTimer reset));
$\mathbf{B}_{LAAP-4}$:	not *angleSensor.valueFailure* AND not *recordedPosition.valueFailure*;
$\mathbf{H}_{LAAP-4}$:	not *flow.valueFailure*;
$\mathbf{B}_{LAAP-5}$:	not *autoPositionReq.comission* AND not *controlLever.omission*;
$\mathbf{H}_{LAAP-5}$:	not *flow.comission* AND not *active.comission*;
$\mathbf{B}_{LAAP-6}$:	*angleSensor* accuracy is 0.02 rad AND actuation deviation is within ±0.01 rad AND *recordedPosition* does not introduce deviation;
$\mathbf{H}_{LAAP-6}$:	*flow* accuracy is 0.01 rad **implies** stop position is within ±0.04 rad from the *recordedPosition*

Table 3: SW Safety Requirements and safety contracts mapping

SWSR1	*LAAP-1*
SWSR2	*LAAP-4*, *LAAP-6*
SWSR3	*LAAP-2*
SWSR4	*LAAP-3*
SWSR5	*LAAP-5*
SWSR6	*LAAP-2*, *LAAP-5*

the requirement *SWSR2*. During the *Safety Contracts Production* phase, the contract *LAAP-6* is updated with the actual accuracy of the *flow* command.

As mentioned in Section 3, the SW safety requirements addressed by the safety contracts are supported with evidence through the connection of the contracts and the supporting evidence. Since requirements are categorised with ASILs, the stringency of the evidence supporting the contracts should be appropriate for the corresponding integrity level. Since the assumed requirements are associated with at most ASIL B, to support the contracts associated with the requirements we use inspection and testing as verification means recommended by ISO 26262 for the specified ASILs. The context statements that provide clarifications of the contracts and the supporting evidence attached during Safety Contract Production phase are shown in Figure 4. The context statements are denoted with *LAAP-x_Cy* and evidence with *LAAP-x_Cy.*, where x is the number of the related contract and y the number of the evidence/context statement.

Table 4: The context statements and evidence of the LAAP safety contracts

LAAP-1_C1:	The contract is based on the specification of the Input validation and error handling of LAAP;
LAAP-1_E1	*name*: Unit testing results *description*: The evidence satisfies ASIL B requirements. *supporting argument*: -;
LAAP-2_C1:	The contract is based on the specification of the Input validation and error handling of LAAP;
LAAP-2_E1	*name*: Unit testing results *description*: The evidence satisfies ASIL B requirements. *supporting argument*: -;
LAAP-3_C1:	The contract is based on the LAAP watchdog timer configuration;
LAAP-3_E1	*name*: Watchdog inspection report *description*: The evidence satisfies ASIL A requirements. *supporting argument*: -;
LAAP-3_E2	*name*: Unit testing results *description*: The evidence satisfies ASIL B requirements. *supporting argument*: -;
LAAP-4_C1:	The contract is derived from the FPTC analysis results for the LAAP component;
LAAP-4_E1	*name*: LAAP FPTC analysis report *description*: The evidence satisfies ASIL B requirements. *supporting argument*: FPTC_analysis_conf;
LAAP-5_C1:	The contract is derived from the FPTC analysis results for the LAAP component;
LAAP-5_E1	*name*: LAAP FPTC analysis report *description*: The evidence satisfies ASIL B requirements. *supporting argument*: FPTC_analysis_conf;
LAAP-6_C1:	The contract is derived from the FPTC analysis results for the LAAP component;
LAAP-6_E1	*name*: LAAP FPTC analysis report *description*:The evidence satisfies ASIL B requirements. *supporting argument*: FPTC_analysis_conf;
LAAP-6_E2	*name*: Unit testing results *description*: The evidence satisfies ASIL B requirements. *supporting argument*: -;

4.2 SEooC Integration

The two products in which we reuse the developed SEooC are a part of the same wheel-loader product line. First product is a Gigant Wheel-loader (GWL) used within closed construction sites. Due to its size, both the GWL itself and its arm move slower than other machines. Time needed to raise the arm under full load from minimum to maximum position is around 10 seconds. The second product is a Small Wheel-loader (SWL) used for less intensive tasks and often outside of construction sites (e.g., public service). It is much more compact than the GWL and it has two times faster lifting arm raise time.

Due to the differences between the two products, what is hazardous in one product is not necessarily hazardous in the other. Since the GWL is used in a controlled environment and its tasks do not require high precision, the value failure of the LAAPs' flow port is not considered hazardous in that case. Hence, the

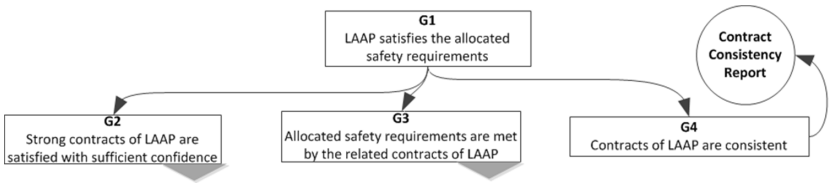

Figure 5: Top goals of the LAAP safety argument for both the GWL and SWL

requirement *SWSR2* is not considered safety-relevant in context of the GWL, but is regarded as quality management. Moreover, the weak contracts *LAAP-4* and *LAAP-6* are not satisfied in the context of GWL, as integrity of the sensor data and recorded position is not ensured for the *LAAP-4* contract, and the assumption on actuation accuracy for the *LAAP-6* contract.

In contrast to the GWL, since the SWL is used in less controlled environments for tasks that usually require precision, the LAAP accuracy is much more critical. Besides a higher quality angle sensor to ensure high confidence in sufficient accuracy of the *angleSensor* input to the LAAP, an error-detecting code is used to ensure that the stored *recordedPosition* has not been accidentally changed (e.g., due to bit flip). Contracts of the corresponding components guarantee these properties of the angle sensor and the *recordedPosition* variable which satisfies the contract *LAAP-6*, while the contract *LAAP-4* is not satisfied in the SWL system either as it would be too expensive to achieve it.

Since the strong contract *LAAP-1* requires *groundSpeedLimit* to be set in every vehicle to a value below 20 km/h, both products must set the appropriate values. In the GWL the limit is 20 km/h, since the arm moves much slower and in a controlled environment, while the limit is 10 km/h for the SWL.

Once the reused contracts are checked and new contracts established during the *Utilisation and Maintenance* phase, we utilise the contracts for the generation of safety argument-fragments. Based on the satisfied contracts we can identify safety artefacts related to such contracts (e.g., test cases) that can be useful in the current context.

4.3 Generated Safety Arguments

Figure 5 shows the top level goals of the LAAP safety argument for the two systems. Both argument-fragments are generated in the same way, hence share the similar structure. To support the top-level goal that the component satisfies the allocated safety requirements, we decompose the top-level goal to argue over the following: the LAAP strong contracts are satisfied, all satisfied contracts are consistent, and the relevant weak contracts are satisfied. As all strong contracts must be satisfied in both context, the argument related to the strong contract satisfaction (Figure 6) is the same for both cases.

The top level goals are further decomposed to argue over satisfaction of each allocated safety requirement. As discussed in Section 4.2, some of the contracts are not satisfied in the GWL and in the same time some of the requirements are discarded as quality management, hence not included in the LAAP safety argument in context of GWL. In case of the GWL, SWSR2 and SWSR4 are not included in the corresponding argument (Figure 7), while for the SWL, all six requirements are included in the corresponding argument (Figure 8).

As most of the requirements are addressed by the strong contracts that are argued in a separate argument branch, the away goals are used to relate to those arguments, while the weak contracts that are used to support a requirement for the first time in the argument are further developed (e.g., the contracts *LAAP-5* and *LAAP-6* for requirements SWSR2 and SWSR5). Establishing that the safety contracts are sufficient to support a certain requirement is done by inspection.

5 Discussion

As described in Section 2.1, ISO 26262 requires certain information to be gathered during the concept phase. The standard states that software safety requirements should consider this information. In case of SEooC, this information should be assumed out-of-context and validated in-context. In the case of other reusable elements such as qualified software elements, this information should be made available and validated prior to the integration of the element into an ISO 26262 compliant system. The guidelines provided by the standard do not go into further detail but stop at the message that this information should be considered, assumed and validated. As described in Section 3 and demonstrated in Section 4, the generative reuse approach based on safety contracts provides means to assume, consider and validate this information. When developing SEooC, the required information is assumed within safety contracts, by associating these contracts with SW safety requirements, the requirements are related and consider this information. Upon integration of a reusable component together with its safety contracts, the assumed information or information that should be made available is validated by checking that the reused safety contracts assumptions are satisfied in the particular system.

As demonstrated in Section 4.2, what is safety relevant in one system can sometimes be regarded as quality management in another system. This is the main reason why reusing safety artefacts (such as product-based safety argument-fragments) first needs a phase of identifying what is relevant, which is supported by the safety contracts, and after that the relevant information can be composed and the artefact reused. In the scope of our work we use the safety contracts to generate safety case argument-fragments, while there is potential to use the contracts to generate different types of safety analyses (e.g., FTA) [8] through the connection of the safety contracts with the FPTC analysis [18]. The generation of the specific safety argument-fragments is still semi-automatically performed since the integrator needs to align the assumed with the actual safety requirements. Although methods could be developed to ease the matching of the safety requirements and the associated contracts, and matching assumed and actual safety requirements, the step towards developing such methods would be formalisation of the requirements, which faces different challenges [5]. Safety contracts share some of these challenges as well. Hence we recognise the need for capturing both formal and informal aspects in the safety contracts. While the formally specified parts of the assumptions and guarantees are used for both contract-based verification and argument-fragment generation, the informal parts are only used for the arguments generation where they can be further reviewed manually.

6 Related Work

The ISO 26262 lack of detailed guidelines for systematic reuse has triggered researchers to align different reuse engineering methods with the standard, e.g., Product-line Engineering (PLE) and Component Based Software Engineering (CBSE). PLE can be aligned with the ISO 26262 to facilitate reuse of artefacts [7]. The proposed approach provides means to specify, manage and trace commonalities and variabilities at different parts of the ISO 26262 safety process.

Reusing safety artefacts requires that variability within them is managed. A PLE-based approach shows how variability can be integrated into the functional safety models by combining functional safety and variability modelling tools [15]. Another approach focuses on Trusted Product Lines by forming a framework for demonstrating that the derived products are fit for purpose in high-integrity civil airspace systems [11]. The work aligns PLE with civil airspace safety standard recommendations on development and integration of reusable elements.

An approach that distinguishes between component types as out-of-context components and component implementations as in-context instantiations of the component types explores use of assume/guarantee contracts to facilitate reuse [9]. The work provides an incremental certification lifecycle for CBSE and outlines the role of contracts in the proposed lifecycle.

In contrast to these works we focus on providing detailed guidelines for development and integration of reusable safety elements within the safety-critical systems. Moreover, we focus on the automotive industry by aligning the provided guidelines with the ISO 26262 safety process. More specifically, we support generative reuse of safety argument-fragments since that increases the reusability of the efforts invested in capturing safety rationale within the safety contracts. We are not aware of other works with this specific focus.

7 Conclusion and Future Work

Safety standards, particularly ISO 26262, lack support for reuse and integration of safety elements, although modern safety-critical systems highly rely on reuse. In this paper we have presented a safety contracts development process that bases reuse of safety elements around the notion of safety contracts. We have shown on a real-world case that the safety contracts can be successfully used to complement and augment ISO 26262 safety process to provide support for reuse and integration of safety elements. Moreover, safety contracts provide a platform for generative reuse of safety artefacts by facilitating generation of safety case argument-fragments and potentially other safety analyses.

As future work we plan to develop the real-world case further and conduct series of studies to evaluate different techniques related to safety contracts. Furthermore, we intend to fully utilise the generative reuse potential of the safety contracts by looking into generation of different safety analyses. While currently only partially supported by CHESS-toolset [4], we plan to continue extension of the tool and further optimisations of the implemented contract formalism.

Acknowledgements

This work is supported by the Swedish Foundation for Strategic Research (SSF) project SYNOPSIS and the EU Artemis-funded nSafeCer project.

References

[1] GSN Community Standard Version 1. Technical report, Origin Consulting (York) Limited, Nov. 2011.

[2] J.-M. Astruc and N. Becker. Toward the Application of ISO 26262 for Real-life Embedded Mechatronic Systems. In *International Conference on Embedded Real Time Software and Systems*. ERTS2, 2010.

[3] S. Baumgart, J. Fröberg, and S. Punnekkat. Industrial Challenges to Achieve Functional Safety Compliance in Product Lines. In *The 40th Euromicro Conference on Software Engineering and Advanced Applications*, August 2014.

[4] CHESS-toolset, `http://www.chess-project.org/page/download`.

[5] P. Filipovikj, M. Nyberg, and G. Rodriguez-Navas. Reassessing the Pattern-Based Approach for Formalizing Requirements in the Automotive Domain. In *22nd International Requirements Engineering Conference*. IEEE, August 2014.

[6] W. B. Frakes and K. Kang. Software Reuse Research: Status and Future. *Transactions on Software Engineering*, 31(7):529–536, 2005.

[7] B. Gallina, A. Gallucci, K. Lundqvist, and M. Nyberg. VROOM & cC: a Method to Build Safety Cases for ISO 26262-compliant Product Lines. In *SAFECOMP Workshop on Next Generation of System Assurance Approaches for Safety-Critical Systems*. HAL / CNRS report, Sept. 2013.

[8] B. Gallina, M. A. Javed, F. U. Muram, and S. Punnekkat. Model-driven Dependability Analysis Method for Component-based Architectures. In *Euromicro Conference on Software Engineering and Advanced Applications*. IEEE, 2012.

[9] P. Graydon and I. Bate. The Nature and Content of Safety Contracts: Challenges and Suggestions for a Way Forward. In *The 20th Pacific Rim International Symposium on Dependable Computing*. IEEE, November 2014.

[10] R. Hawkins, I. Habli, D. Kolovos, R. Paige, and T. Kelly. Weaving an Assurance Case from Design: A Model-Based Approach. In *16th International Symposium on High Assurance Systems Engineering*, pages 110–117. IEEE, Jan. 2015.

[11] S. Hutchesson and J. A. McDermid. Trusted Product Lines. *Information & Software Technology*, 55(3):525–540, 2013.

[12] ISO 26262:2011. *Road vehicles — Functional safety*. ISO, 2011.

[13] B. Meyer. The next software breakthrough. *Computer*, 30(7):113–114, 1997.

[14] Y. Papadopoulos, M. Walker, D. Parker, E. Rüde, R. Hamann, A. Uhlig, U. Grätz, and R. Lien. Engineering Failure Analysis and Design Optimisation With HiP-HOPS. *Engineering Failure Analysis*, 18(2):590–608, 2011.

[15] M. Schulze, J. Mauersberger, and D. Beuche. Functional Safety and Variability: Can It Be Brought Together? In *17th International Software Product Line Conference*, pages 236–243. ACM, 2013.

[16] I. Sljivo, B. Gallina, J. Carlson, and Hansson. Generation of Safety Case Argument-Fragments from Safety Contracts. In A. Bondavalli and F. Di Giandomenico, editors, *33rd International Conference on Computer Safety, Reliability, and Security*, volume 8666 of *LNCS*, pages 170–185. Springer, Heidelberg, Sept. 2014.

[17] I. Sljivo, B. Gallina, J. Carlson, and H. Hansson. Strong and weak contract formalism for third-party component reuse. In *International Workshop on Software Certification*. IEEE Computer Society, Nov. 2013.

[18] I. Sljivo, B. Gallina, J. Carlson, H. Hansson, and S. Puri. A Method to Generate Reusable Safety Case Fragments from Compositional Safety Analysis. In *14th International Conference on Software Reuse*. Springer-Verlag, Jan. 2015.

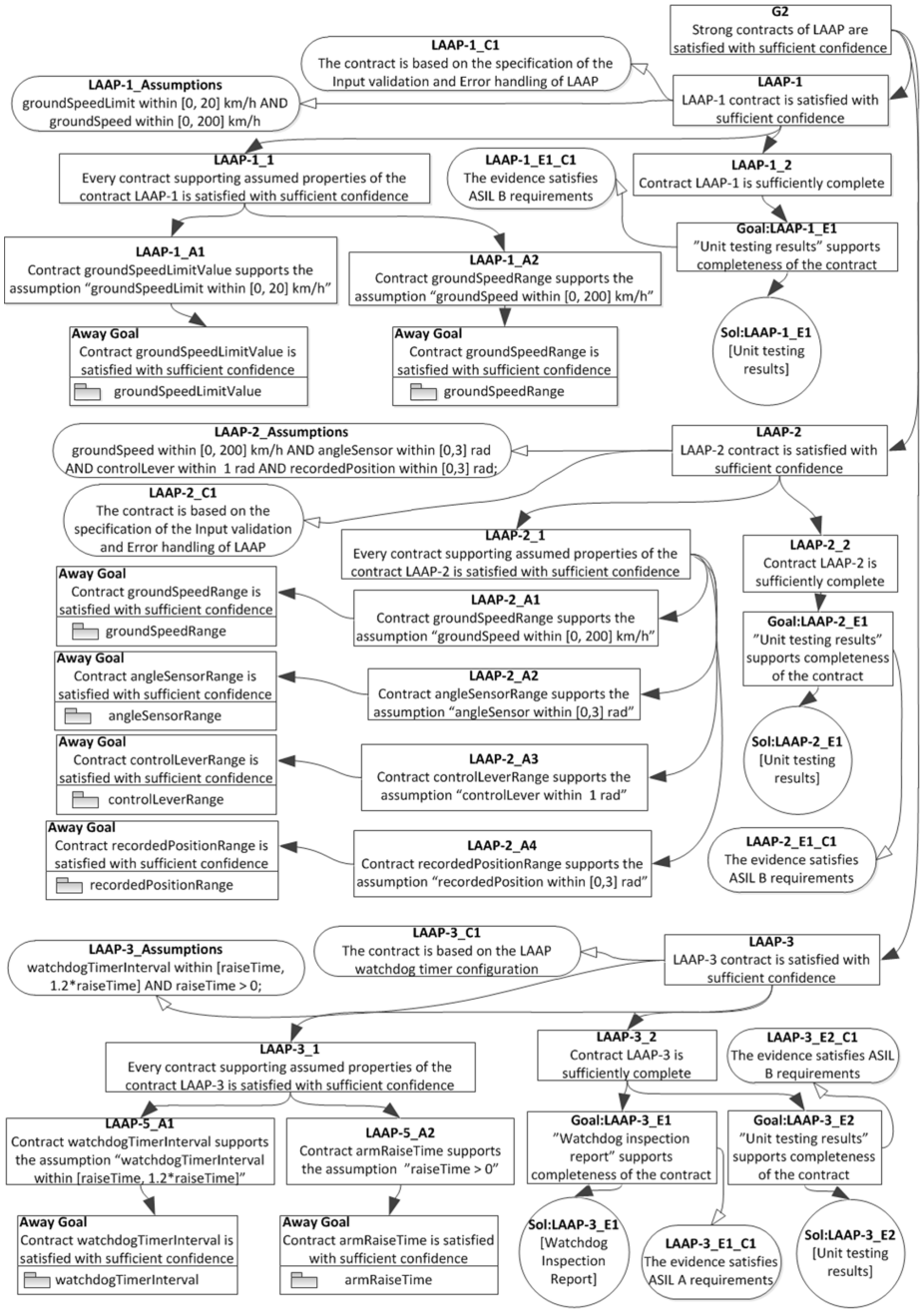

Figure 6: Safety argument-fragment for the strong contracts (the same for both systems)

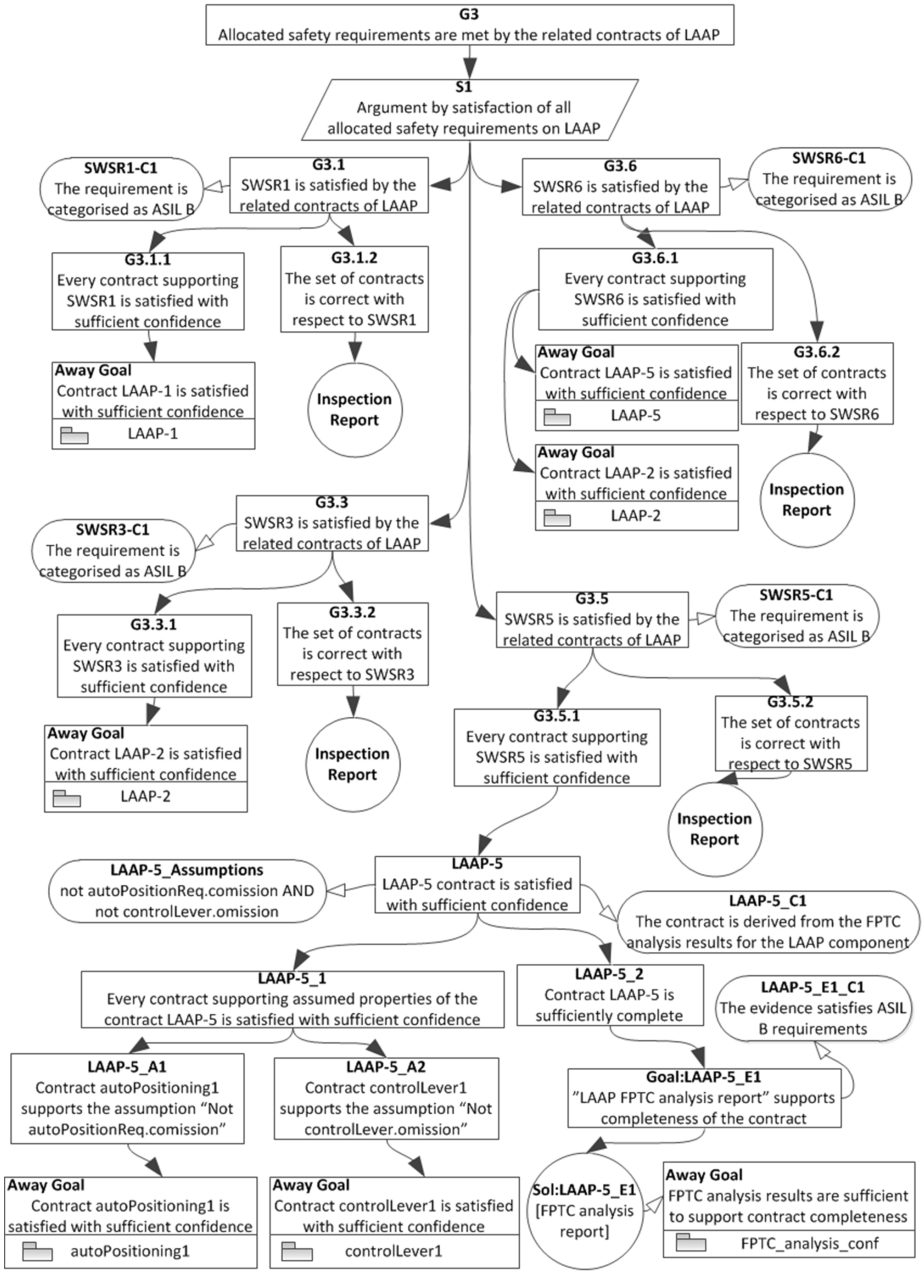

Figure 7: Safety argument-fragment for the safety requirements allocated on the LAAP in context of GWL

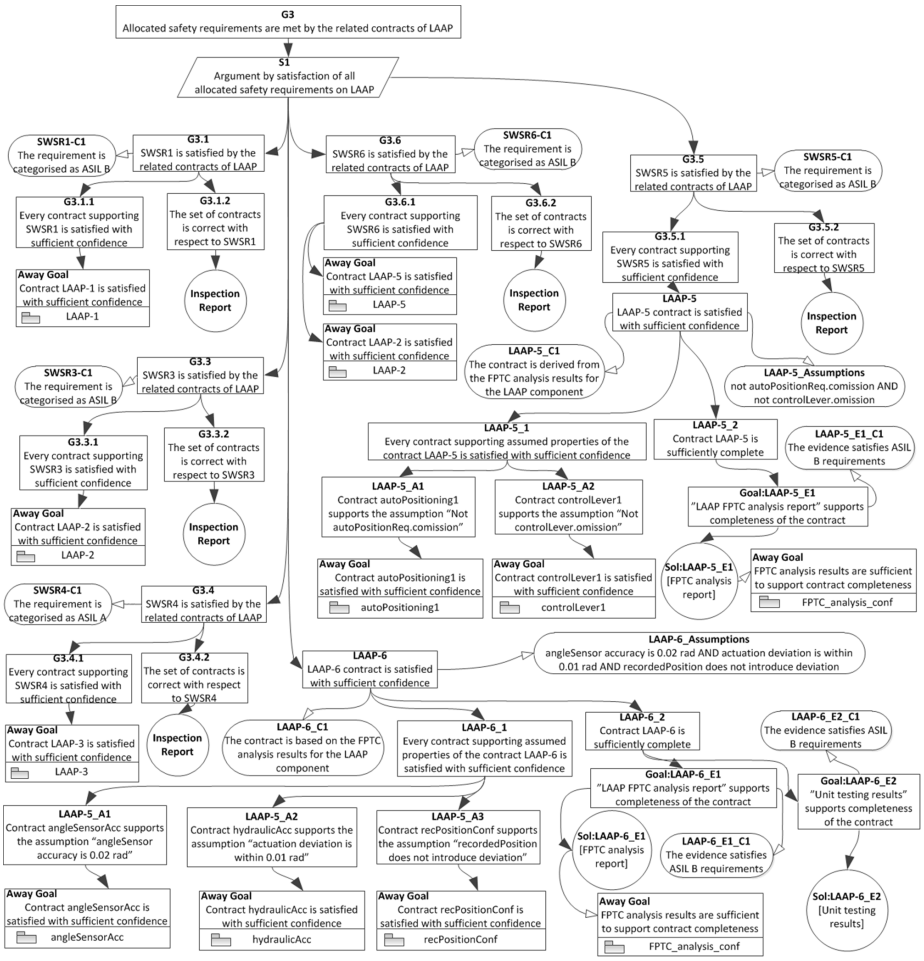

Figure 8: Safety argument-fragment for the safety requirements allocated on the LAAP in context of SWL)