Argument Patterns for Multi-Concern Assurance of Connected Automated Driving Systems

Fredrik Warg
RISE Research Institutes of Sweden, Borås, Sweden
http://www.ri.se
fredrik.warg@ri.se

Martin Skoglund
RISE Research Institutes of Sweden, Borås, Sweden
martin.skoglund@ri.se

Abstract

Showing that dependable embedded systems fulfil vital quality attributes, e.g. by conforming to relevant standards, can be challenging. For emerging and increasingly complex functions, such as connected automated driving (CAD), there is also a need to ensure that attributes such as safety, cybersecurity, and availability are fulfilled simultaneously. Furthermore, such systems are often designed using existing parts, including 3rd party components, which must be included in the quality assurance. This paper discusses how to structure the argument at the core of an assurance case taking these considerations into account, and proposes patterns to aid in this task. The patterns are applied in a case study with an example automotive function. While the aim has primarily been safety and security assurance of CAD, their generic nature make the patterns relevant for multi-concern assurance in general.

2012 ACM Subject Classification Computer systems organization → Dependable and fault-tolerant systems and networks; Computer systems organization → Embedded and cyber-physical systems

Keywords and phrases Multi-concern assurance, connected automated driving, dependability, functional safety, cybersecurity, cyber-physical systems, critical embedded systems

Digital Object Identifier 10.4230/OASIcs.CERTS.2019.3

Funding This work was supported by Vinnova – project ESPLANADE (2016-04268), and the EU and Vinnova – project ECSEL AMASS (692474), but the contents of the paper only reflect the authors' views.

1 Introduction

When releasing embedded dependability-critical electrical/electronic (E/E) systems on the market it is typically necessary to demonstrate that they are sufficiently safe. This is often done by showing compliance to a functional safety standard. A key design strategy to successfully show the product is safe is to keep the safety-related part of the system as small and simple as possible. While this is still desirable, the technological development is inexorably moving towards higher complexity even for safety-related functionality. One example is automated driving systems (ADS) [17], i.e. functions enabling what is commonly referred to as self-driving or autonomous vehicles. For such functions it is difficult to keep the safety-related part small and isolated as the goal of the function is to drive safely, a task which by necessity involves many of the vehicle's sensory, control and actuator subsystems.

With this complexity, it becomes more difficult to convincingly show that a product is safe. A further complication is that safety cannot be treated in isolation in the presence of other quality attributes (QAs) that may affect safety properties. For instance, it is expected that most ADS equipped vehicles are also connected in order to increase performance of the functionality, e.g. an ADS that exchanges information with surrounding vehicles and roadside infrastructure (traffic lights, signs, roadside sensors) will have a better world model and be able to make better and less defensive driving decisions. However, adding connectivity to enable connected automated driving (CAD) also increases the security risks. A hacker may compromise the ADS remotely to e.g. circumvent its safety mechanisms. Hence, to demonstrate that the function is safe it is also necessary to show that all security risk that may compromise safety have been treated. This extends to arbitrarily many interrelated concerns, e.g. it might be necessary to consider availability of the function to make sure safety and security mechanisms do not lower availability of the function in a way that makes the business case unviable.

Safety for vehicle E/E functions is typically demonstrated by showing conformance to the ISO 26262 standard [8], through a safety case consisting of artefacts resulting from complying to all its normative requirements. The implicit argument is that standard conformance itself is proof of safety. However, making an explicit argument showing the product-specific safety-rationale within the overall framework of the standard can aid both development engineers and safety assessor [3]. For this work our premise is that such an argument is even more important - even if the standards may not mandate it - when the complexity increases, for instance when showing simultaneous conformance to several standards each representing a different QA, also called *multi-concern assurance*, or when including components developed out-of-context by 3$^{\text{rd}}$ party suppliers.

Our contribution in this paper is a discussion and proposed patterns for simultaneously covering the dimensions of *(1) multiple concerns*, *(2) standards conformance*, *(3) element-out-of-context*, and *(4) system lifecycle* in an argument for a multi-concern assurance case. A danger with such multi-dimensional arguments is that it becomes unwieldy and incomprehensible, thus failing to fill its basic purpose, something we attempt to overcome with a clear and regular structure. We also develop such an argument in a case study relevant for CAD.

In work related to ours, others have suggested patterns for ISO 26262 arguments [4, 5, 7, 13, 15, 16]. Most of these provide more detailed patterns that could be combined with what we are proposing, but in some cases also have a somewhat different way of organizing the argument. However these are for safety only. Taguchi et al. [19] discuss and show patterns for different ways of integrating safety and security, which may be done by combining the two concerns, treating them totally separately, or by handling interdependencies in different ways. In this work we use what Taguchi calls a bi-directional reference process pattern as a multi-concern pattern, but also combine it with the other dimensions we discuss. Work focused on co-engineering and how to capture trade-offs between concerns in the argumentation has also been done [1, 12, 6]. In contrast our work is about structuring the assurance case to capture information about inter-concern dependencies together with the other mentioned argument dimensions, not how to handle trade-offs or co-engineering.

2 Argument Dimensions

Here we elaborate on the implication of taking the four dimensions mentioned in Sec. 1 into account. As we focus on dependability aspects and standards conformance, the conceptual V-model used in e.g. many functional safety standards is used to illustrate the lifecycle. In Fig. 1, a triple V-model highlighting the aspects of nominal function, safety and cybersecurity is shown. While some standards and work processes prefer other models, e.g. to highlight iterative aspects more clearly, we note that interpreted as a dependency chain rather than a timeline, the concepts expressed in the V-model are usually applicable, i.e. there is an overall function concept which is refined to implementable components, and tested in several integration levels. When a version of the product is completed, the process is repeated to add new functionality for the next version, while the current version goes into production and maintenance phases. In functional safety standards such as ISO 26262 [8], results from all design and verification steps are collected in a *safety case*, which must be complete and consistent for each product version. When treating several concerns the general term *assurance case* is used instead. In this section we discuss the rationale behind the four dimensions and return to the patterns in the next section.

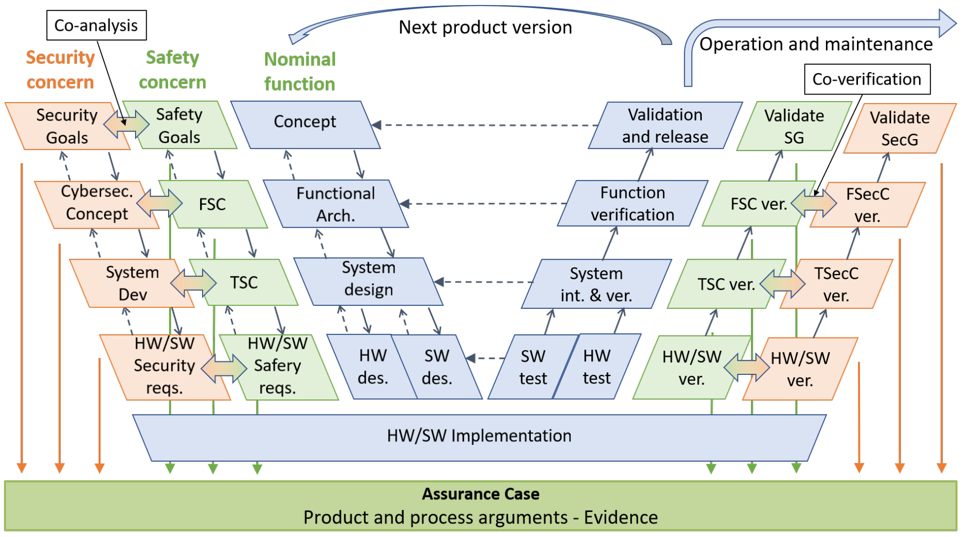


Figure 1 V-model with safety and security attributes and a multi-concern assurance case.

2.1 Lifecycle

As standards and/or company-specific work processes typically prescribe specific development lifecycles, making the lifecycle stages evident in the argument makes it easier to relate the argument to the design process, and thereby show that the risk of introducing systematic faults is sufficiently reduced throughout the lifecycle. In other words, showing the lifecycle in the argument reduces the risks due to misunderstandings and omissions originating from bad mapping between argument and the actual development work. Furthermore there is often a distinction of product and process arguments in standards. Process arguments also include e.g. management practices that are not specifically tied to the product. In our pattern we make a separation of these for increased clarity.

2.2 Standards Conformance

The argument should also preferably reflect if the assurance case shall show conformance to a standard (this is also called a *conformance case*). Our patterns are designed to be compatible with the V-model used in e.g. many functional safety standards. While the patterns create the general structure for the argument, the claims must be instantiated for the specific quality attribute and supported by more low-level claims and supporting evidence. These sub-claims can in many cases be requirements directly from the standard. Thus conformance is not a separate pattern, rather the phases and modules used in our patterns are suitable for combining with standard requirements. Using a tool allowing compliance mapping between standard requirements and the argument, e.g. OpenCert [14], it is even possible to track that all standard requirements have been fulfilled within the framework of the argument.

2.3 Concerns and Their Interplay

If two quality attributes are independent, i.e. fulfilling one can never impact the other, the only aspect of multi-concern assurance compared to separate conformance to the concerns is possible synergies to reduce the assurance work, e.g. joint testing using the same test frameworks (co-verification in Fig. 1). However, typically interplay in the form of potential dependencies, conflicts or synergies between the concerns exist and must be identified and resolved in the multi-concern argument. Again, the analysis of interplay between concerns must cover the entire product lifecycle to make sure conflicts do not appear in any design stage or even after production. For instance, security concerns may make over-the-air (OTA) updates necessary so that security holes uncovered long after the product is released can be fixed, but this makes it necessary to make sure OTA updates cannot compromise safety. We therefore include argument for interplay in our patterns. It should be noted that interplay arguments can become unwieldy if many dependent concerns are treated since the possible combinations quickly grows with number of QAs. Based on [11] as well as own experience, we also claim that specifying well-defined interaction points between concerns (co-analysis in Fig. 1) is preferable to processes integrating several concerns.

2.4 Component Structure

In many domains, including automotive, the most common way to build new features is to integrate parts from suppliers, or reusing existing components. These must be included in the assurance case for the new feature. A supplier may also sell the same part to several customers and even develop it before having any requirements from an OEM. The supplier may then construct their own assurance case for their part, using assumptions on its use, i.e. an assumed context. In ISO 26262, this is called a safety-element-out-of-context. We generalize the concept and use the term element-out-of-context (EooC), which may have an assurance case for multiple concerns. When integrating the EooC in the complete feature, there must be a bridge between the feature and EooC assurance cases explaining how the part developed for the assumed context will also fulfil the requirements for the actual feature in the real context.

3 Putting it Together in Argument Patterns

3.1 Pattern Notation

We use the argumentation structure defined in ISO/IEC 15026-2:2011 [9] which is illustrated on the left hand side of Fig. 2. In this standard, an argument consists of one or more top-level *claims* supported by sub-claims, *evidence*, and/or *assumptions* through an *argument* detailing how these underlying components support the top-level claim. Sub-claims must be supported in the same way; the argument can be arbitrarily many levels with evidence and assumptions as leaf nodes. The choice of top-level claims must be supported by a *justification*, as must an argument (at any level) have a justification for how underlying components support a (sub-)claim. The standard is agnostic as to how the arguments are represented. In this paper we use the GSN notation [18], which provides an illustrative graphical representation, to show our proposed patterns. The GSN notation corresponding to the 15026-2 concepts are shown to the right in Fig. 2; in GSN a claim is called a *goal*, an argument is called *strategy* and evidence is called a *solution*.

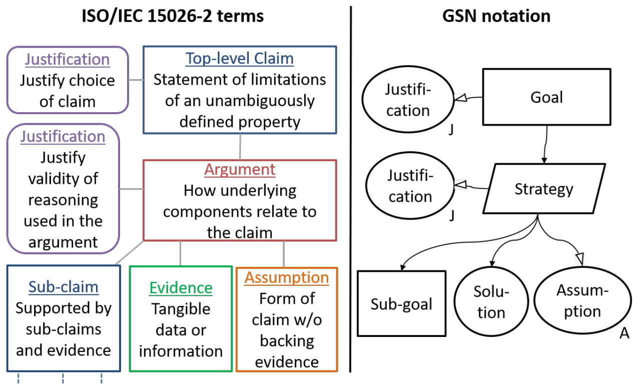

ISO/IEC 15026-2 terms
Justification
Justify choice of claim
Top-level Claim
Statement of limitations of an unambiguously defined property
Justification
Justify validity of reasoning used in the argument
Argument
How underlying components relate to the claim
Sub-claim
Supported by sub-claims and evidence
Evidence
Tangible data or information
Assumption
Form of claim w/o backing evidence
GSN notation
Justifi-cation
J
Goal
Justifi-cation
J
Strategy
Sub-goal
Solu-tion
Assum-ption
A

Figure 2 ISO/IEC 15026-2:2011 argument and corresponding GSN notation.

In addition to the basic notation we use some extensions to GSN. The modular extensions are helpful for managing the complexity of large arguments, and the extensions allowing for abstraction are used to express generic argument patterns. Fig. 3 shows the GSN elements used in this paper. *(a) Module* is used to represent a separate sub-argument which is used either in a *module view* which is a special overview diagram in GSN showing only relationships between such modules, or to show that a goal is supported by an entire argument contained in a separate module. *(b) Contract* is used when a goal will be supported in a yet unspecified module and is used to provide decoupling. The contract module itself is used to provide a glue argument showing how the argument in a module (which might e.g. be provided for a re-usable component) fulfils the goal which was to be supported by the contract. *(c) Away goal* repeats a goal made in another module in the argument of a local module in order to show dependencies between goals in different modules. The away goal also identifies the module where the original goal can be found. The *(d) InContextOf arrow* is used in a way proposed by the AMASS project [2], which is to show that fulfillment of a goal in one concern is dependent on fulfillment of a goal in another concern. *(e) Option* is used to denote alternatives to satisfy a relationship, while *(f) optional arrow* is used for an optional relationship, and *(g) many arrow* denotes a one-to-many relationship with the cardinality shown next to the arrow. A goal can also be *(h) uninstantiated* which means it is an abstract element that needs to be replaced by a concrete instance. Words within {brackets} in the argument are tokens to be instantiated, e.g. *{Goal}* could be instantiated as *LaneKeepingAssist is acceptably safe* as a top-level goal in the safety argument for a lane keeping assist vehicle feature. *(i) Undeveloped* means a goal which is not yet fully supported, i.e. it needs to be developed by completing the argument beneath it. Goals can be both undeveloped and uninstantiated at the same time. Finally, *(j) context* is part of the basic GSN notation and is used to provide contextual information needed for interpreting the goal or strategy it is attached to. These concepts are more fully described in the GSN community standard version 2 [18]. All GSN figures in the rest of the paper are produced with the OpenCert tool [14][1].

[1] Some modifications of the figures produced by the tool have been made for improved readability.

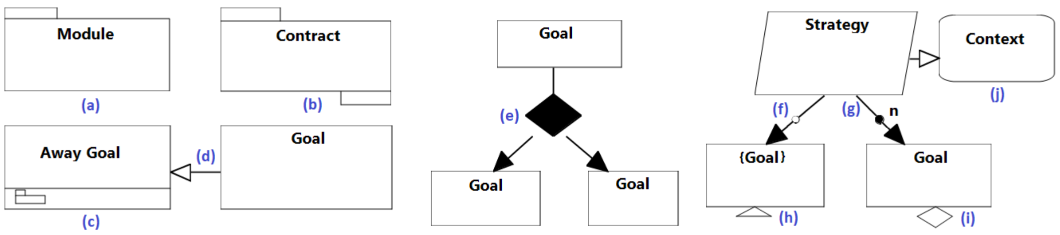


Figure 3 GSN extensions used in the paper.

3.2 Overall Argument Structure and Lifecycle

We organize the overall argument as shown in the GSN modules view in Fig. 4. The top level claim will be that the system (or EooC which we return to in Sec. 3.5) fulfils all quality attributes that have been defined for it. A pattern for the topmost module of Fig. 4 is shown in Fig. 5; this pattern references modules for all QAs and interplay arguments relevant for the product. The concept phase is where initial concept (e.g. item definition in ISO 26262) is defined and risk evaluation is performed (e.g. hazard analysis & risk assessment (HA&RA) and definition of safety goals in ISO 26262). In the concept phase there will be separate modules for each QA and for interplay between all QAs where relevant, e.g. safety and security are not independent and therefore should have an interplay module if they are two of the defined QAs. The rest of the argument is organized according to lifecycle with one module per logical element on the functional concept stage, one module per component on the technical concept/system design stage and modules for software and hardware development for each component. There are separate modules to deal with management and post-development issues such as production, maintenance and decommissioning. These stages are typical for a V-model. The number of abstraction levels may vary but is easy to adapt as the basic structure in each level is the same.

3.3 Concerns

For each quality attribute, a pattern for developing this concern in the concept phase is shown in Fig. 6. The strategy is to use a specified lifecycle, often from a standard, with adequate measures for the QA. The sub-goals are optional depending on the QA, but typical components of the argument is: risk mitigation by using an analysis method and introducing requirements specifying the risks to be avoided (and in many standards the level of risk reduction is quantified with an integrity level [10]); adequate management and operations and maintenance (OaM) practices; and confirmation measures, e.g. review of analysis results.

Following the goal *{QA} requirements introduced to reduce {QA} risks* from one of the leaf nodes in Fig. 6 is a pattern, shown in Fig. 7, for making sure these quality attribute requirements have also been correctly implemented. This pattern provides a way to create a rationale around each QA requirement showing that it has been correctly refined, implemented and verified. The pattern contains goals for refining the QA requirements to the next abstraction level where the pattern will be repeated again for all refined requirements. The refinement goals are optional as they are obviously not applicable on the lowest refinement level while the verified goal is applicable (and mandatory) on all levels.

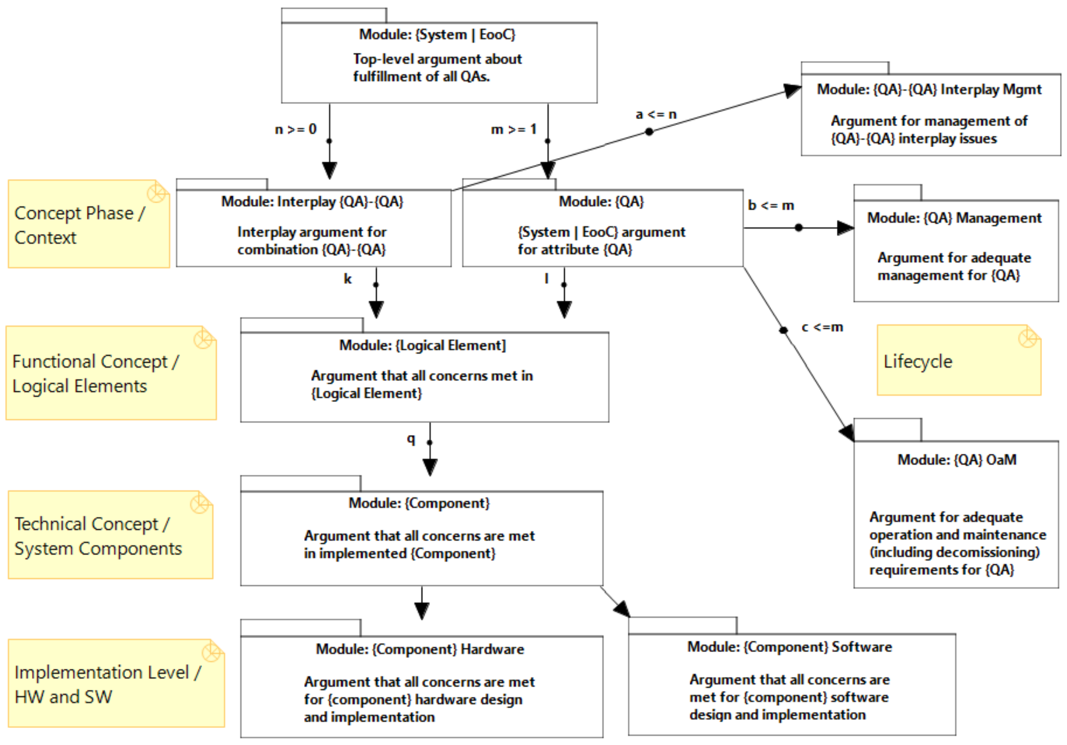

Module: {System | EooC}
Top-level argument about fulfillment of all QAs.
n >= 0
m >= 1
a <= n
Module: {QA}-{QA} Interplay Mgmt
Argument for management of {QA}-{QA} interplay issues
Concept Phase / Context
Module: Interplay {QA}-{QA}
Interplay argument for combination {QA}-{QA}
Module: {QA}
{System | EooC} argument for attribute {QA}
b <= m
Module: {QA} Management
Argument for adequate management for {QA}
k
l
c <=m
Functional Concept / Logical Elements
Module: {Logical Element}
Argument that all concerns met in {Logical Element}
Lifecycle
q
Module: {QA} OaM
Argument for adequate operation and maintenance (including decomissioning) requirements for {QA}
Technical Concept / System Components
Module: {Component}
Argument that all concerns are met in implemented {Component}
Implementation Level / HW and SW
Module: {Component} Hardware
Argument that all concerns are met for {component} hardware design and implementation
Module: {Component} Software
Argument that all concerns are met for {component} software design and implementation

Figure 4 Modules view of system or element-out-of-context.

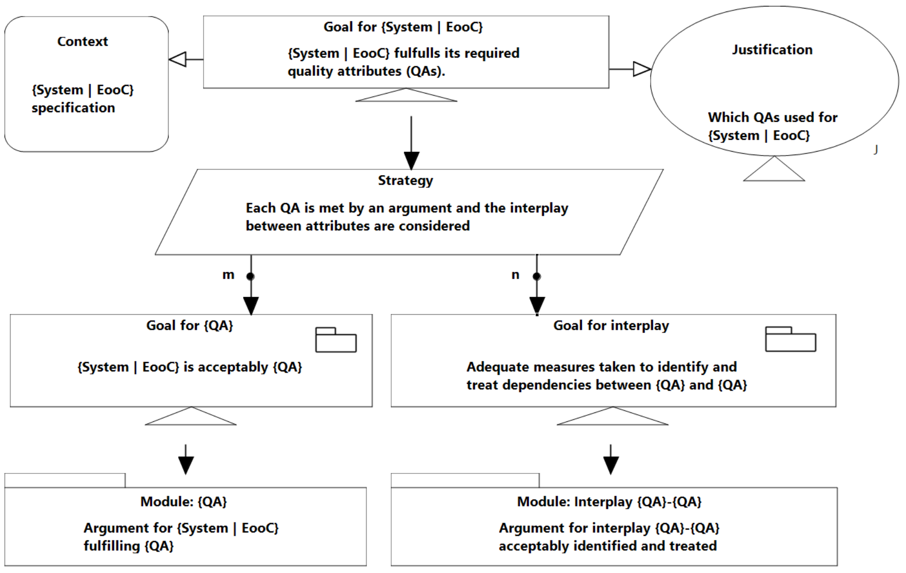
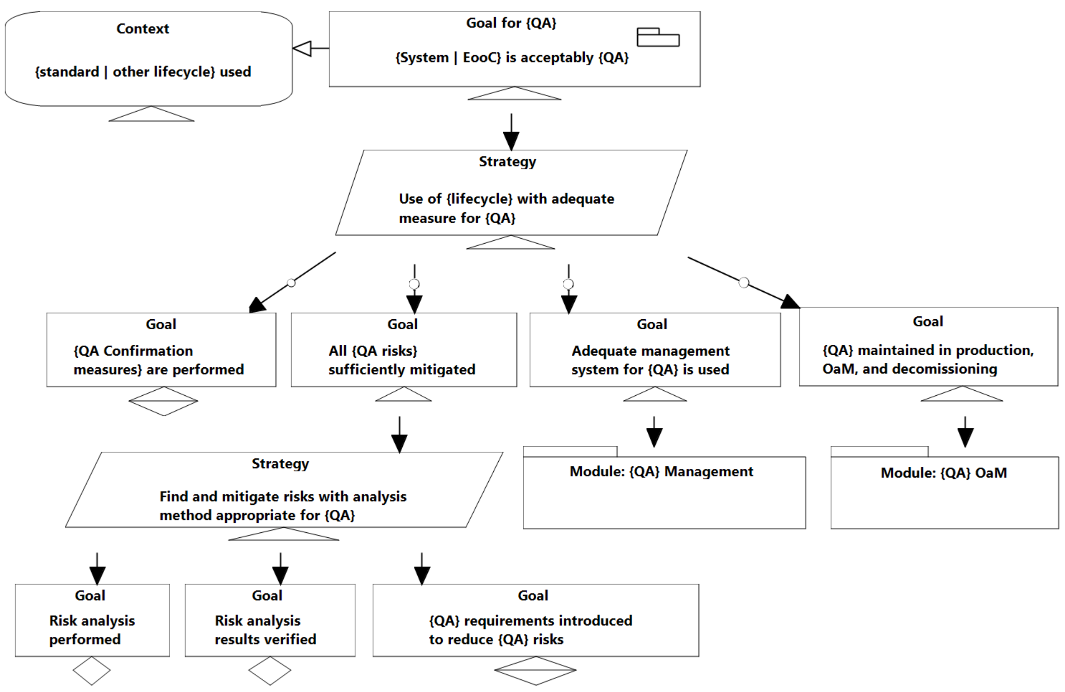
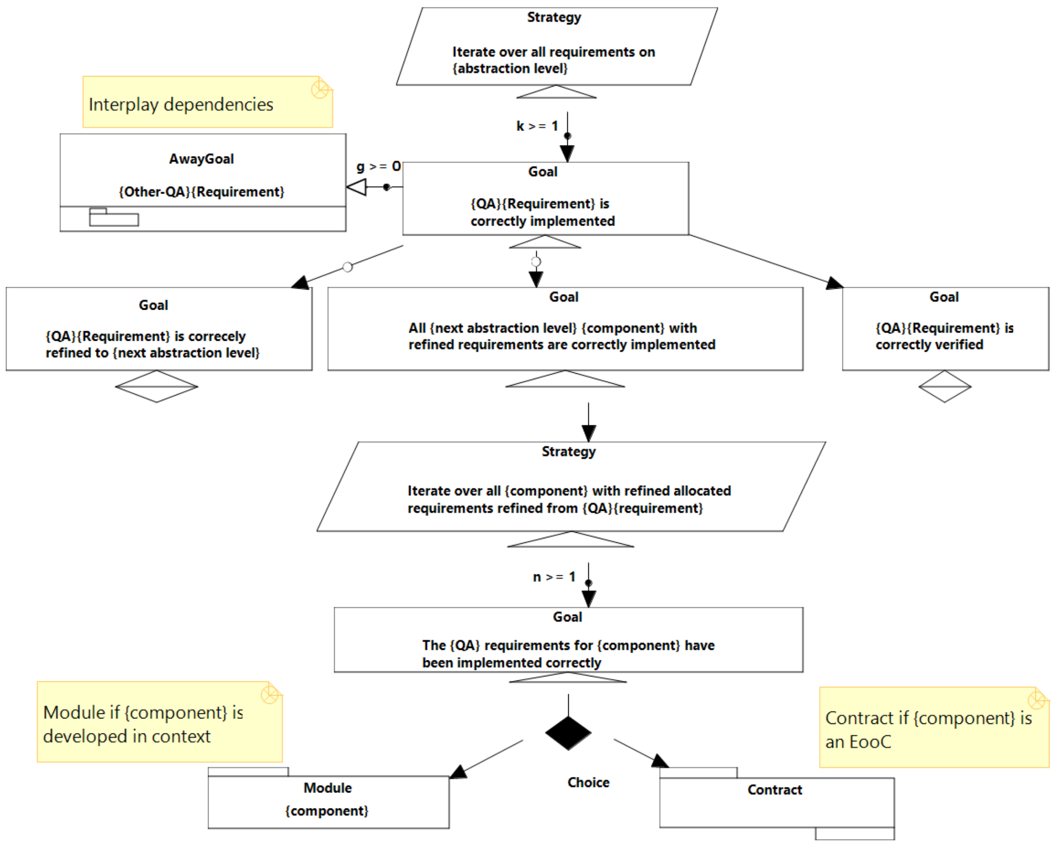

Context
{System | EooC} specification
Goal for {System | EooC}
{System | EooC} fulfulls its required quality attributes (QAs).
Justification
Which QAs used for {System | EooC}
J
Strategy
Each QA is met by an argument and the interplay between attributes are considered
m
n
Goal for {QA}
{System | EooC} is acceptably {QA}
Goal for interplay
Adequate measures taken to identify and treat dependencies between {QA} and {QA}
Module: {QA}
Argument for {System | EooC} fulfilling {QA}
Module: Interplay {QA}-{QA}
Argument for interplay {QA}-{QA} acceptably identified and treated

Figure 5 Pattern for top-level multi-concern argument.



Context
{standard | other lifecycle} used
Goal for {QA}
{System | EooC} is acceptably {QA}
Strategy
Use of {lifecycle} with adequate measure for {QA}
Goal
{QA Confirmation measures} are performed
Goal
All {QA risks} sufficiently mitigated
Goal
Adequate management system for {QA} is used
Goal
{QA} maintained in production, OaM, and decomissioning
Strategy
Find and mitigate risks with analysis method appropriate for {QA}
Module: {QA} Management
Module: {QA} OaM
Goal
Risk analysis performed
Goal
Risk analysis results verified
Goal
{QA} requirements introduced to reduce {QA} risks

Figure 6 Pattern for a quality attribute.



Strategy
Iterate over all requirements on {abstraction level}
Interplay dependencies
k >= 1
AwayGoal
{Other-QA}{Requirement}
g >= 0
Goal
{QA}{Requirement} is correctly implemented
Goal
{QA}{Requirement} is correcely refined to {next abstraction level}
Goal
All {next abstraction level} {component} with refined requirements are correctly implemented
Goal
{QA}{Requirement} is correctly verified
Strategy
Iterate over all {component} with refined allocated requirements refined from {QA}{requirement}
n >= 1
Goal
The {QA} requirements for {component} have been implemented correctly
Module if {component} is developed in context
Contract if {component} is an EooC
Choice
Module
{component}
Contract

Figure 7 Pattern for a QA requirement at any abstraction level.

3.4 Interplay

Interplay can be argued between two or more QAs in each interplay module depending on which methods are found most suitable for interplay in each case. However, it must be evident that all relevant combinations of QAs are taken into account. The pattern, shown in Fig. 8, establishes that management practices for interplay are in place and that dependencies between QAs are found and introduced in the QA requirement hierarchy. This was shown in Fig. 7 as an away goal for a requirement, indicating an interplay dependency. Similar to how QA requirements were handled, the pattern in Fig. 9 then makes sure these interplay dependencies are also refined in lower abstraction levels of the design.

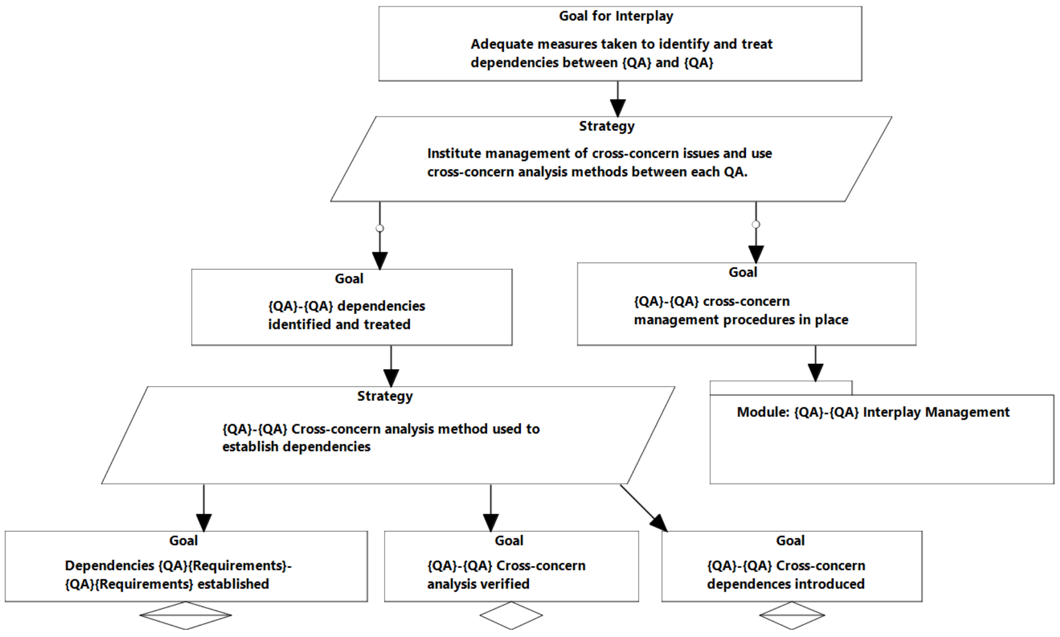

Goal for Interplay
Adequate measures taken to identify and treat dependencies between {QA} and {QA}
Strategy
Institute management of cross-concern issues and use cross-concern analysis methods between each QA.
Goal
{QA}-{QA} dependencies identified and treated
Goal
{QA}-{QA} cross-concern management procedures in place
Strategy
{QA}-{QA} Cross-concern analysis method used to establish dependencies
Module: {QA}-{QA} Interplay Management
Goal
Dependencies {QA}{Requirements}-{QA}{Requirements} established
Goal
{QA}-{QA} Cross-concern analysis verified
Goal
{QA}-{QA} Cross-concern dependences introduced

Figure 8 Pattern for an interplay.

3.5 Element-out-of-Context

The final pattern is the glue between in-context feature and element-out-of-context mentioned in Sec. 2.4. This pattern, shown in Fig. 10, simply connects in-context QAs with the same QA for the EooC, but establishes that an argument showing their compatibility needs to be developed. An analogous pattern (not shown) can be used for the interplay, i.e. it is also necessary to ascertain that the relevant interplay dependencies are covered in the EooC. A component in any abstraction level can be an EooC, which means the EooC will contain the argument from that abstraction level down.

4 Case Study

As a case study we use a positioning element (PE) for CAD which needs to conform to both functional safety and cybersecurity standards. PE is designed as an element-out-of-context (EooC) and can thus be used for various functions. As it is aimed at the automotive domain,

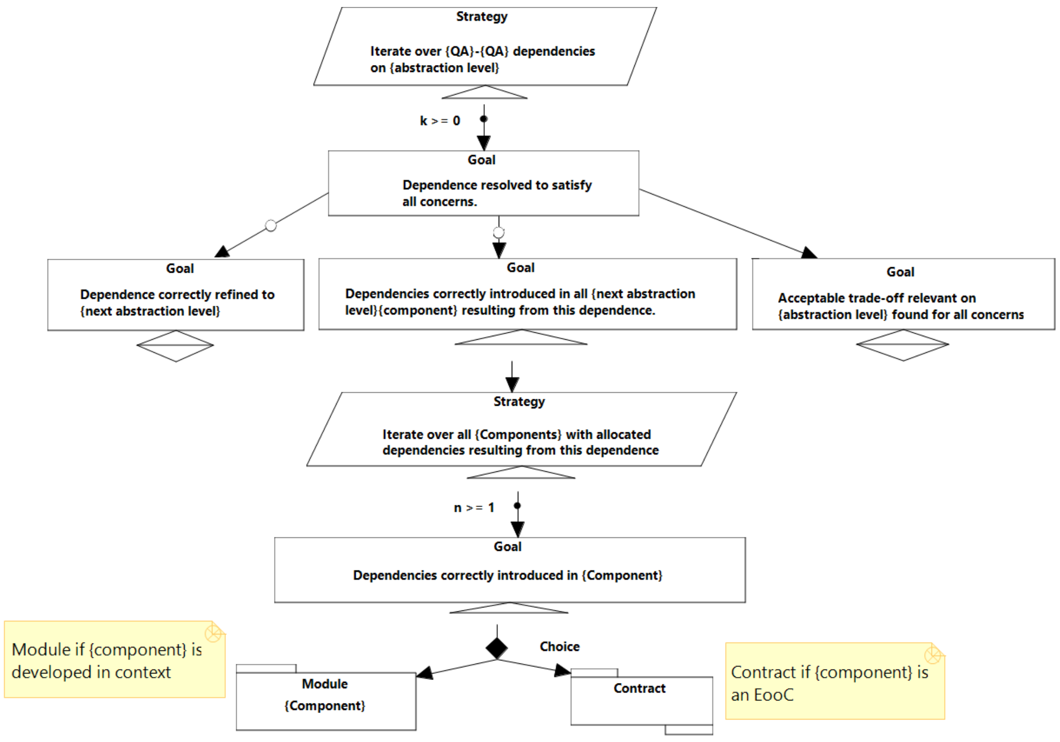

Strategy
Iterate over {QA}-{QA} dependencies on {abstraction level}
k >= 0
Goal
Dependence resolved to satisfy all concerns.
Goal
Dependence correctly refined to {next abstraction level}
Goal
Dependencies correctly introduced in all {next abstraction level}{component} resulting from this dependence.
Goal
Acceptable trade-off relevant on {abstraction level} found for all concerns
Strategy
Iterate over all {Components} with allocated dependencies resulting from this dependence
n >= 1
Goal
Dependencies correctly introduced in {Component}
Module if {component} is developed in context
Choice
Module
{Component}
Contract
Contract if {component} is an EooC

Figure 9 Pattern for an interplay refinement.

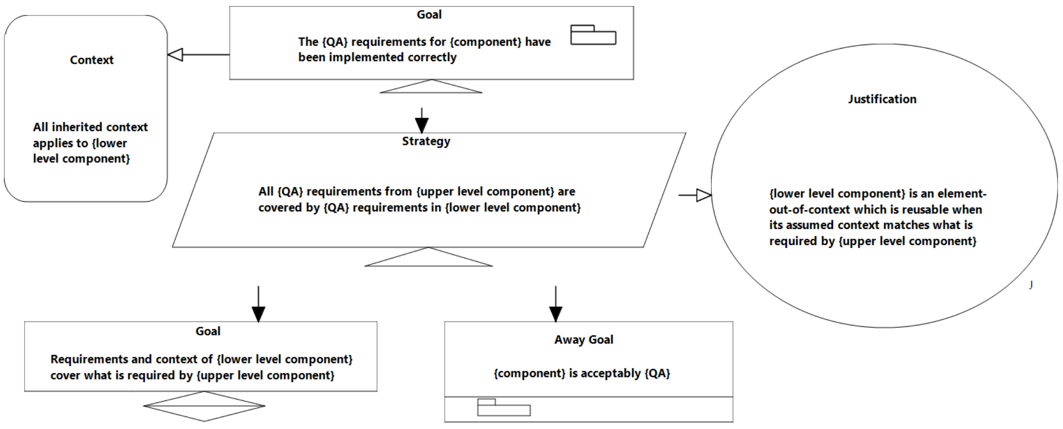

Goal
The {QA} requirements for {component} have been implemented correctly
Context
All inherited context applies to {lower level component}
Strategy
All {QA} requirements from {upper level component} are covered by {QA} requirements in {lower level component}
Justification
{lower level component} is an element-out-of-context which is reusable when its assumed context matches what is required by {upper level component}
J
Goal
Requirements and context of {lower level component} cover what is required by {upper level component}
Away Goal
{component} is acceptably {QA}

Figure 10 Pattern for a contract between system/component and an element-out-of-context.

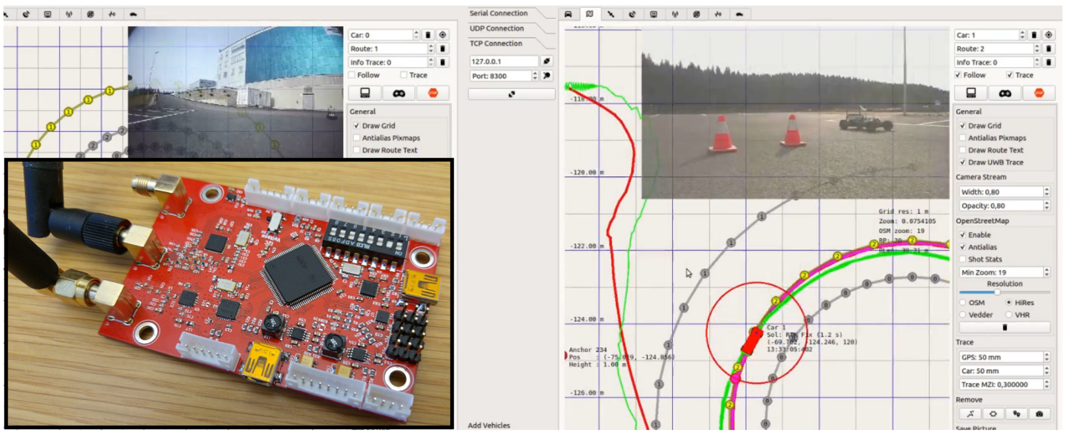


Figure 11 Demonstrator with model cars using the PE (inset) for navigation.

ISO 26262 is used as safety standard and a working draft of ISO/SAE 21434[2] for cybersecurity. Fig. 11 (inset) shows the hardware for PE containing a satellite navigation receiver which is used in conjunction with correction data for enhanced precision. To complete the use case, PE is matched to the hypothetical context of an ADS feature - highway autopilot, where it is used to provide accurate absolute (i.e. on a map) position. Fig. 11 also shows a demonstrator environment for this feature with autonomous model cars.

A detailed description of the function is beyond the scope of this paper; however, it has functional requirements which are analyzed for safety and security risks according to both standards, resulting in safety goals (top-level safety requirements) and security goals. A simplified version of one functional requirements is: *The automated driving mode may only be activated on roads certified for ADS vehicles*. The ISO 26262 HA&RA results in safety goals for the ADC. A safety goal related to the stated requirement is: *ADC may only be activated on certified roads*[3]. Since the function is only designed to work within the parameters given in the functional requirement, its behavior is undefined if enabled anywhere else, thus resulting in high risk of harm. For cybersecurity, a threat analysis and risk assessment (TA&RA) is used to elicit security goals. A security goal with a dependency to the mentioned safety goal is: *Integrity protection against spoofing to fulfil ADC may only be activated on certified roads*.

For space reasons the entire argument for the case study cannot be shown. However, Fig. 12 shows some parts of interest: (a) dependence between the safety and security goals discussed above; (b) the same dependence refined to functional level, (c) example of where requirements from ISO 26262 have been connected to the assurance case (HA&RA forms a tree of its own ending in requirements from the standard, this tree has been automatically generated to a module), and (d) reference to the contract between function and positioning EooC.

5 Conclusions

In this paper, our goal was to propose a structured way to build the argument for a multi-concern assurance case of a complex dependability-critical system such as a CAD function, and demonstrate its feasibility by an example. Our claim is that the complexity can be

[2] A coming cybersecurity standard for the automotive domain.
[3] The actual safety and security goals also have integrity levels but as they are not relevant for the example we have omitted them.

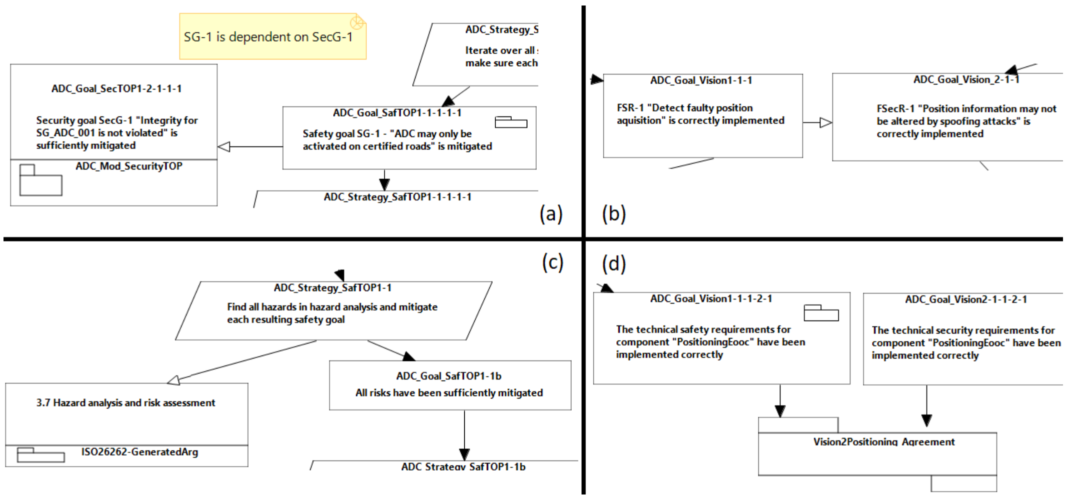

SG-1 is dependent on SecG-1
ADC_Strategy_S
Iterate over all
make sure each
ADC_Goal_SecTOP1-2-1-1-1
Security goal SecG-1 "Integrity for SG_ADC_001 is not violated" is sufficiently mitigated
ADC_Mod_SecurityTOP
ADC_Goal_SafTOP1-1-1-1-1
Safety goal SG-1 - "ADC may only be activated on certified roads" is mitigated
ADC_Strategy_SafTOP1-1-1-1-1
(a)
(b)
ADC_Goal_Vision1-1-1
FSR-1 "Detect faulty position aquisition" is correctly implemented
ADC_Goal_Vision_2-1-1
FSecR-1 "Position information may not be altered by spoofing attacks" is correctly implemented
(c)
ADC_Strategy_SafTOP1-1
Find all hazards in hazard analysis and mitigate each resulting safety goal
3.7 Hazard analysis and risk assessment
ISO26262-GeneratedArg
ADC_Goal_SafTOP1-1b
All risks have been sufficiently mitigated
ADC Strategy SafTOP1-1b
(d)
ADC_Goal_Vision1-1-1-2-1
The technical safety requirements for component "PositioningEooc" have been implemented correctly
ADC_Goal_Vision2-1-1-2-1
The technical security requirements for component "PositioningEooc" have been implemented correctly
Vision2Positioning Agreement

Figure 12 Snippets from argument for case study.

managed by a traceable argument structure, with an attached rationale to every branching in order to keep track of the reasoning behind the design. With this structure the overall design can also be aggregated and contain re-usable components. The structure follows each concern, with dependencies at certain interaction points. The interaction can be predicted because the argument structure also reflects the development lifecycles of the concerns. When the interaction between the concerns is planned and limited, as we propose, there is a good possibility too keep the benefits from co-engineering, without the extra effort of high frequency interaction between different disciplines such as safety and security.

It should be noted that even if a structured approach makes it easier to manage large complex systems, the approach would still require good tool support to be feasible as the arguments can become very large. Traceability and compliance management, management of argument modules, and automation of argument integrity checks are examples where tools are helpful. There is ample opportunity for automation, for instance detecting nodes that have not been developed or instantiated, or solution nodes with no references to actual evidence. Combination with semi-formal notations for goals/requirements to allow for even better control of structure and more checks for possible omissions is yet another possibility to increase automation opportunities. Some tools such as OpenCert already contain many of these features. Another issue we have not discussed in the paper is how to include assurance in the actual development workflow. For instance, today many organizations are adopting agile practices to allow for more frequent product updates. This is an issue we are currently exploring in our continued work.

References

1 AMASS deliverable D4.3: Design of the AMASS tools and methods for multiconcern assurance, 2018. [Accessed 17-April-2019]. URL: `https://www.amass-ecsel.eu/`.

2 AMASS deliverable D4.8: Methodological guide for multiconcern assurance(b), 2018. [Accessed 17-April-2019]. URL: `https://www.amass-ecsel.eu/`.

3 John Birch, Roger Rivett, Ibrahim Habli, Ben Bradshaw, John Botham, Dave Higham, Peter Jesty, Helen Monkhouse, and Robert Palin. Safety cases and their role in ISO 26262 functional safety assessment. In *32nd International Conference on Computer Safety, Reliability, and Security, SAFECOMP*, pages 154–165. Springer, 2013.

4 John Birch, Roger Rivett, Ibrahim Habli, Ben Bradshaw, John Botham, Dave Higham, Helen Monkhouse, and Robert Palin. A Layered Model for Structuring Automotive Safety Arguments (Short Paper). In *10th European Dependable Computing Conference, EDCC*, pages 178–181. IEEE, 2014.

5 Thomas Chowdhury, Chung-Wei Lin, BaekGyu Kim, Mark Lawford, Shinichi Shiraishi, and Alan Wassyng. Principles for systematic development of an assurance case template from ISO 26262. In *2017 IEEE International Symposium on Software Reliability Engineering Workshops, ISSREW*, pages 69–72. IEEE, 2017.

6 Georgios Despotou and Tim Kelly. An Argument-Based Approach for Assessing Design Alternatives and Facilitating Trade-offs in Critical Systems. *Journal of System Safety*, 43(2):22, 2007.

7 Ashlie B Hocking, John Knight, M Anthony Aiello, and Shinichi Shiraishi. Arguing software compliance with ISO 26262. In *2014 IEEE International Symposium on Software Reliability Engineering Workshops, ISSREW*, pages 226–231. IEEE, 2014.

8 ISO. ISO 26262:2018 Road vehicles – Functional safety, 2018.

9 ISO/IEC. ISO/IEC 15026-2:2011 Systems and software engineering – Systems and software assurance – Part 2: Assurance case, 2011.

10 ISO/IEC. ISO/IEC 15026-2:2015 Systems and software engineering – Systems and software assurance – Part 3: System integrity levels, 2015.

11 Nikita Johnson and Tim Kelly. An Assurance Framework for Independent Co-assurance of Safety and Security. In *36th International System Safety Conference, ISSC*, 2018.

12 Helmut Martin, Robert Bramberger, Christoph Schmittner, Zhendong Ma, Thomas Gruber, Alejandra Ruiz, and Georg Macher. Safety and security co-engineering and argumentation framework. In *International Conference on Computer Safety, Reliability, and Security*, pages 286–297. Springer, 2017.

13 Helmut Martin, Martin Krammer, Robert Bramberger, and Eric Armengaud. Process- and product-based lines of argument for automotive safety cases. In *ACM/IEEE 7th International Conference on Cyber-Physical Systems, ICCPS*, 2016.

14 OpecCert contributors. OpenCert. [Accessed 17-April-2019]. URL: `https://www.polarsys.org/projects/polarsys.opencert`.

15 Rob Palin, David Ward, Ibrahim Habli, and Roger Rivett. ISO 26262 safety cases: Compliance and assurance. In *6th IET International Conference on System Safety*. IET, 2011.

16 Robert Palin and Ibrahim Habli. Assurance of Automotive Safety–A Safety Case Approach. In *29th International Conference on Computer Safety, Reliability, and Security, SAFECOMP*, pages 14–17. Springer, 2010.

17 SAE. SAE J3016:201806 - SURFACE VEHICLE RECOMMENDED PRACTICE - Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles, 2018.

18 SCSC Assurance Case Working Group Contributors. GSN Community Standard Version 2, 2018. [Accessed 17-April-2019]. URL: `https://scsc.uk/r141B:1?t=1`.

19 Kenji Taguchi, Daisuke Souma, and Hideaki Nishihara. Safe & sec case patterns. In *33rd International Conference on Computer Safety, Reliability, and Security, SAFECOMP*, pages 27–37. Springer, 2014.