Checkable Safety Cases: Enabling Automated Consistency Checks between Safety Work Products

1st Carmen Cârlan
fortiss GmbH
Munich, Germany
carlan@fortiss.org

2nd Daniel Petrișor
Continental Automotive Romania
Technical University of Iași
Iași, Romania
daniel.petrisor@continental-corporation.com

3rd Barbara Gallina
Mälardalen University
Västerås, Sweden
barbara.gallina@mdh.se

4th Hannes Schoenhaar
Siemens AG
Munich, Germany
hannes.schoenhaar@siemens.com

***Abstract*—In the automotive domain, the employment of agile development is currently hindered by the fact that the safety lifecycle, which implies the creation and maintenance of safety work products, is manually executed, being a complex and expensive process. Given a change in the system under consideration, ISO 26262 recommends that the impact of that change on the safety case of the system shall be assessed and that the safety case shall be correspondingly updated. To this end, in this paper, while assuming a model-based system and safety engineering context, we propose checkable safety case models, which are semantically rich safety case models integrated with system and safety engineering models (i.e., work products of a model-based safety lifecycle). The semantically rich specification and the model integration allow for automated consistency checks between the safety case and the system, specifically its engineering models. We exemplify our contributions via an in-vehicle driver assistance system for driving through intersections.**

***Index Terms*—safety cases, safety assurance, model-based system engineering, automated checks, maintenance**

I. Introduction

Currently, in the automotive industry, there is an increase in the frequency of development increments motivated, for example, by the need for patching a security threat [9], accommodating changes in the operating context [15], or the addition of new functionality [10]. Such frequent changes call for an agilized execution of the safety lifecycle [7]. However, the maintenance of the safety work products, i.e., artefacts resulting from the execution of the lifecycle, is a time-consuming process, as it is still manually executed [16], hindering the movement to a more agile context.

In ISO 26262-8, the change management process is described as aiming at ensuring the systematic implementation of changes, while maintaining the safety properties of the system under consideration [8]. To this end, there is a need to identify the impacted safety work products. The need for maintenance of safety work products, given certain changes in the system specification, also implies the need for maintenance of safety cases. A safety case is a safety work product providing an argument that functional safety requirements are satisfied, under certain operational conditions, based on evidence compiled from other safety work products [8]. ISO 26262-10 specifically pinpoints the need for checking the consistency of the safety case with the system under development [8].

While state-of-the-art safety cases entail claims in free-form text, allowing flexibility in the argumentation structure, the lack of formalized knowledge in a machine processable form hinders the automatic assessment of the safety case at a semantic level. The safety engineer manually assesses the consistency of the safety case with other safety work products, while understanding the semantics of the respective argumentation structure and based on expert knowledge.

To enable the move towards agilized safety lifecycles, in this paper, assuming a model-based system and safety engineering context, we introduce checkable safety case models (see Fig. 1). Such models entail both checkable and non-checkable safety case fragments. In contrast to the state-of-the-art non-checkable safety case fragments, the claims in checkable fragments are semantically rich and integrated with engineering models (e.g., hazards, requirements or system design models). Further, checkable safety case fragments are based on evidence created by model-based verification engines, which check system design models against safety requirements. As such, given development increments, checkable fragments allow for automated consistency checks between a safety case model and models of other safety work products. For example, when adding a new element to the hazards list, a consistency check warns the safety engineer that the safety case model needs to be updated such that it also argues about the mitigation of the risk associated to the newly added hazard.

Next, in Section II, we provide some insights on notations for modeling safety case arguments. We then present an intersection handling system, together with a selection of engineering models generated during the execution of the lifecycle, which we use throughout the paper to exemplify the proposed concepts (see Section III). Then, in Section IV, we propose specialized safety case constructs integrated with engineering models and model-based verification engines. Further, we define a subset of consistency criteria between safety case and engineering models and we give an example of a checkable safety case fragment. In Section V, we compare our work with state-of-the-art approaches, arguing about the novelty of our contribution. In the final section, we emphasize again our contribution and share the vision on work ahead that will allow further development of the proposed method and concepts (see Section VI).

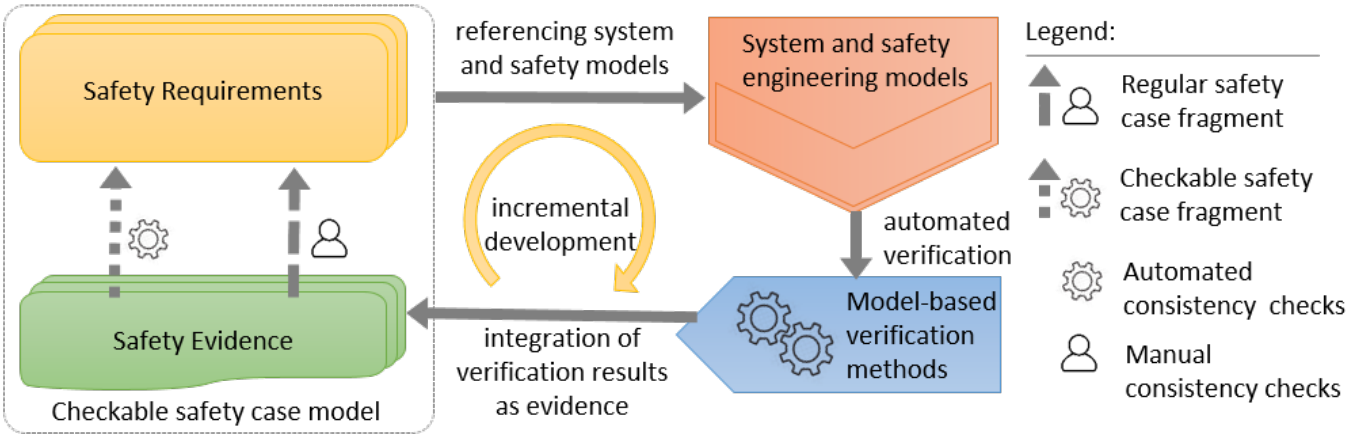

Fig. 1. Overview of checkable safety case models integrated with system and safety engineering models.

II. Foundations

There are several languages for modeling safety case arguments, such as the Structured Assurance Case Metamodel (SACM) syntax [12], Claim-Argument-Evidence (CAE) notation [2], NOR-STA [18] or the Goal Structuring Notation (GSN) [1]. In this paper, we target GSN for modeling safety cases because for GSN there is already a formalization proposed by Denney and Pai [5]. We needed such formalization for specifying the algorithms of our consistency checks. GSN-based models are directed and acyclic graphs (DAGs) with typed nodes specifying an argumentation. *Goals* are claims arguing about the safety of the system that can be further decomposed into sub-goals, given a certain *strategy*. At the bottom of the argumentation, *solution* constructs reference evidence artefacts (e.g., test results), demonstrating the satisfaction of goals. *Context*, *assumption* and *justification* constructs specify the context, scope or rationale of an argumentation.

Denney and Pai propose a mathematical specification for a constrained version of GSN, while considering the following types of safety case constructs: s - strategy, g - goal, e - evidence, a - assumption, j - justification c - context [5]. They define a safety case fragment S as a tuple $< N, l, t, \rightarrow >$ comprising the set of nodes, N, the labeling function $l : N \rightarrow \{s, g, e, a, j, c\}$ setting the type of a construct, the metadata function $t : N \rightarrow E$ specifying the construct contents, where E is a set of expressions for specifying certain attributes of a constructs, and the connector relation $\rightarrow : < N, N >$, specifying the relations between nodes. A safety case fragment shall always be constructed as a directed acyclic graph (DAG). The operation $isroot(r)$ checks if construct r is a root in the DAG. Furthermore, the following structural conditions must be met: (1) Each root of the partial safety case is a goal: $isroot(r) \Rightarrow l(r) = g$; (2) Connectors only leave strategies or goals: $n \rightarrow m \Rightarrow l(n) \in s, g$; (3) Goals cannot connect to other goals: $(n \rightarrow m) \wedge [l(n) = g] \Rightarrow l(m) \in \{s, e, a, j, c\}$; (4) Strategies cannot connect to other strategies or evidence: $(n \rightarrow m) \wedge [l(n) = s] \Rightarrow l(m) \in \{g, a, j, c\}$.

III. An Intersection Handling System

In this section, we introduce a System of Interest (SoI) that handles automated driving (AD) in urban intersection environments, together with a subset of safety work products generated for this system. The SoI shall be compliant with ISO 26262 [8] and ISO 21448 [10]. A more detailed description of the system is accessible online[1].

Functional Requirements. The SoI shall interact with the driver when handling both regular and smart intersections. The driver can turn on the SoI only while the vehicle is in standstill and in-vehicle systems are operational. During operation, the system perceives the operational environment using data from local sensors mounted on the vehicle platform (see Fig. 2-b) and/or from smart intersections (see Fig. 2-a). When approaching an intersection, the driver activates the SoI by indicating the target intersection exit. After a successful activation of the intersection handling, the SoI controls the lateral and longitudinal behavior of the vehicle to successfully exit the intersection. The SoI then requests the driver to take over once the mission is accomplished. The hand over zone is the zone where the transition from manual to AD is done and is depicted in Fig. 2-a.

Hazard Analysis and Functional Safety Concept. A subset of the hazards we identified for the SoI can be seen in the hazards list model presented in Fig. 3-a. To prevent and mitigate the risk associated to the identified hazards, we specify two safety goals (see Fig. 3-b): `SG01`: *AD system shall provide a safe and comfortable driving experience to the passengers*; and `SG02`: *AD system shall detect TPO, VRU, road geometry and topology*, each with the Automotive Safety Integrity Level `ASIL D`. We then derived functional safety requirements. In Fig. 3-c we present the requirements addressing the sensor setup necessary for each relevant sensing zone. For the specification of a functional architecture, we proposed an ASIL decomposition of functionality, as shown in Fig. 3-d. The `ASIL D` compliant system design solution

[1]https://github.com/ccarlan/safe_intersection_handling_system

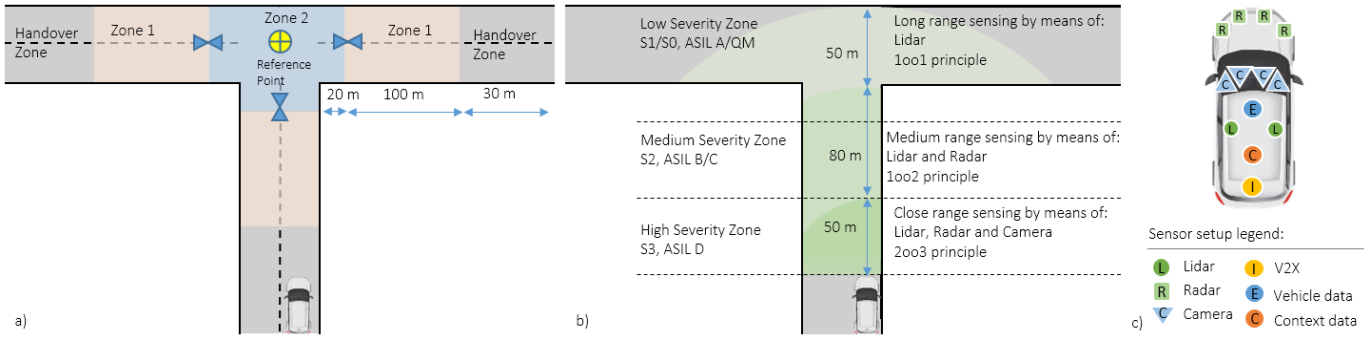

Fig. 2. a) Intersection specific sensing: Zone 1, Zone 2 and Handover zone. b) Vehicle specific sensing: close range high severity, medium range medium severity, long range low severity. c) Vehicle sensor setup that enables a safe control inside ODD.

implements system and safety measures identified by decomposing the `ASIL D` rating into three `ASIL B(D)` sensor components independently implemented. The local sensing setup relies on three different sensing technologies: LIDAR, radar and camera chosen and integrated such that the SoI uses data with sufficient certainty to enable a safe control over the ego vehicle, as shown in Fig. 2-c). `ASIL B(D)` allocation to different sensing components is chosen as a solution considering sensor uncertainty caused by the wide spectrum of ODD and technological limitations of current sensing technologies. The different sensing technologies are used within distinct sensing zones bounded by a decision making process that uses different voting routines that considers sensor limitations, environment and ego vehicle data to output a trustworthy control command. The functional safety concept considers a certain Operational Design Domain of the SoI. The SoI controls the vehicle when enabled within a constrained ODD and requests the driver to take over control of the vehicle when the specified conditions inside the ODD are no longer met.

Change scenarios. The execution of an agilized safety lifecycle would be beneficial for such a system, given the fact that is a new concept, and new hazards and requirements are in scope of the development during the product lifecycle. Furthermore, system reconfigurations may frequently occur. For example, the sensing setup may change, depending on advances in technology.

IV. Checkable Safety Cases

In this section, we present checkable safety case models – our conceptual solution for maintaining safety case models consistent with system and safety engineering models.

A. Semantically Rich Safety Case Constructs

Checkable safety case fragments comprise semantically rich specialized safety case constructs, which extend GSN constructs, as depicted in Fig. 4. We specify these GSN extensions by adding new constructs to the set of GSN node types $\{s, g, e, a, j, c\} \bigcup SpecializedConstructs$. Each such specialized construct specifies a re-occurring safety claim, having certain known semantics. Further, such a construct has placeholders for references to certain types of engineering model elements. As such, for each specialized safety case construct we can specify consistency criteria, i.e., if and how a specific type of change in associated engineering models impacts the validity of the claim. For example, for arguing that the risk associated to a hazard is mitigated, the `Hazard Mitigation` specialized goal may be used. Such a specialized goal will always be associated to an element within the model of a hazards list. Given the deletion or modification of the associated hazard model element, the specialized goal will become inconsistent with the hazards model.

Our proposed specialized constructs are connected with each other via specialized connections, thus constraining the argumentation structure. This constraint not only ensures a semantically correct argumentation structure, but also allows for automated assessment of the argumentation structure given certain changes in the system specification. For example, the `Argument over Hazards` specialized strategy arguing about the fact that the risk of each identified hazard has been addressed shall be associated to a certain hazards list. This strategy may only be supported by a certain type of specialized goals (i.e., `Hazard Referencing Goal Base` goals). Each such specialized goal has a claim about the reduction of the risk associated to a referenced hazard from the respective list. Given the addition of a new hazard to the list, our consistency checks will signal the safety engineer that there should be an additional `Hazard Referencing Goal Base` goal arguing about the mitigation of the newly identified hazard and supporting the `Argument over Hazards` strategy.

B. Exemplary System and Safety Engineering Models

Hazards Models. During the execution of the hazard analysis and risk assessment, the safety engineer can model a list of hazardous events. In Fig. 3-a, we present the such a model for the intersection handling system introduced in Sec III. We specify a hazardous event as a tuple $ha =< id, spec, s, p, c, asil >$, specifying an identifier (id) and a text-based specification. Further, to support risk assessment, the safety engineer may specify for each hazard, given a certain operational situation, severity, exposure and controllability properties to rate the associated ASIL.

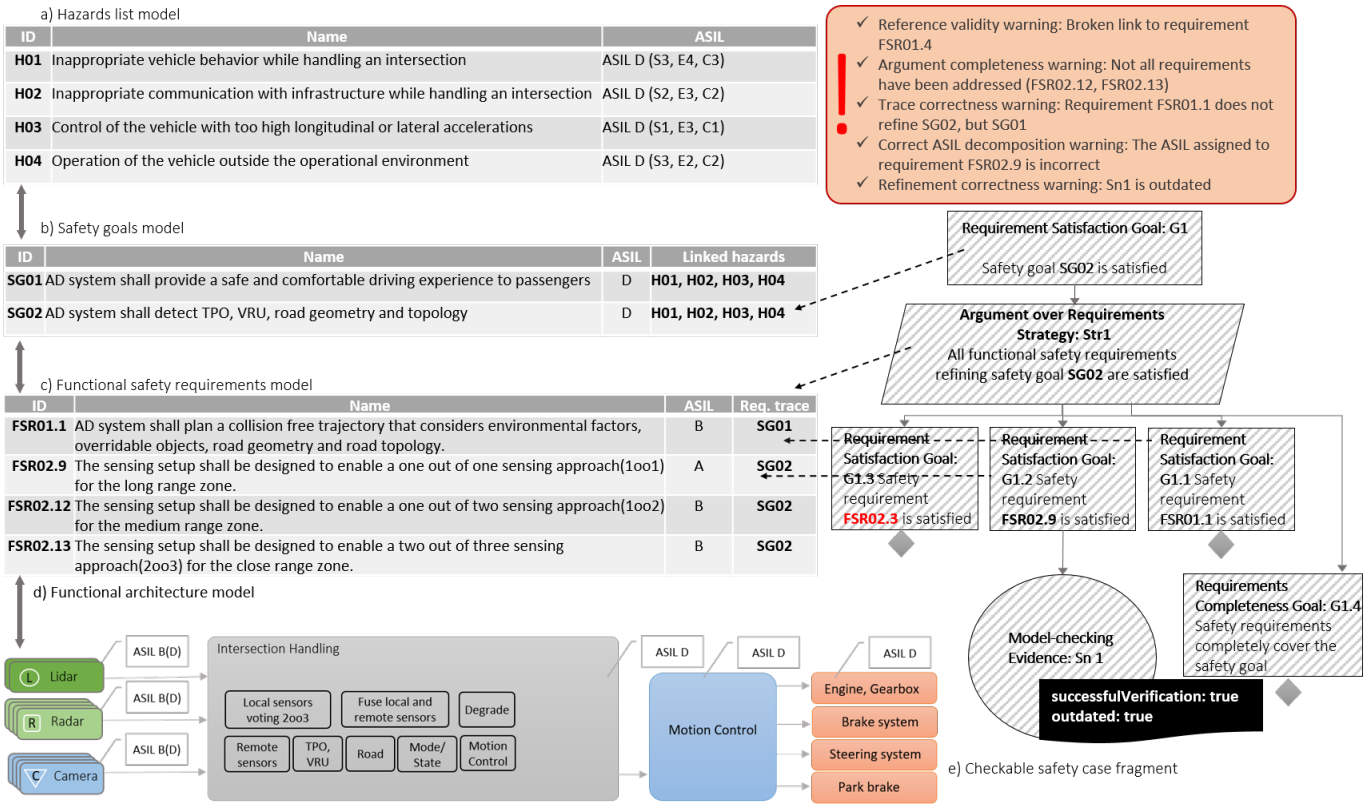

Fig. 3. The *Argument over Requirement Satisfaction* checkable fragment of the safety case model of a intersection handling system, a set of engineering models with which it is integrated and the warnings output by our proposed consistency checks.

Safety Requirements Models. Based on the identified hazardous events, different safety requirements models are specified during the operational, functional and technical safety concept. Examples of such models specifying requirements for the intersection handling system introduced in Sec III can be seen in Fig. 3-b and Fig. 3-c. Each safety requirement is a tuple $sr =< id, spec, author, asil, traces >$, having an id, a specification and an author, an assigned ASIL and a set of tracing relationships. A safety requirement shall be traced to one or more hazards $hazTraces : SafetyRequirements \rightarrow Hazards$ or to a higher-level safety requirement $reqTrace : SafetyRequirements \rightarrow SafetyRequirements$.

System Architecture Models. During the functional and technical safety concept phase in ISO 26262, the safety engineer may model the system architectural design SD as a set of components connected to each other through named channels $SD =< C, CH >$. Fig. 3-d present the functional architecture specified for the system introduced in Sec III.

C. Integration of Safety Case and Engineering Models

Each specialized construct, depending on the type of claim it specifies, has associations to certain types of model elements (e.g., hazards, safety requirements, system design components), this allowing checkable safety case models to be aware of changes in system and safety engineering models. This characteristic is highly needed especially in a change management system. To this end, we define an association between a GSN node to a system model element as an attribute of GSN constructs. The $association$ attribute is actually a link to an engineering model element. While in Fig. 4-a, we illustrate our proposed integration of safety case models with system and safety engineering models, via class association relationships, in Fig. 4-b, we give some examples of semantically rich constructs and their associations to engineering models. We define an association that points to a model element that was deleted as a broken link $brokenLink : Associations \rightarrow Boolean$, where if $g.association = NULL$, then $brokenLink(g.association) = true$.

D. Using Model-based Verification Results as Safety Evidence

One of the objectives of the technical safety concept stated in ISO 26262 is "to verify that the system architectural design and the technical safety concept are suitable to satisfy the safety requirements according to their respective ASIL" [8]. To this end, checkable safety cases integrate as safety evidence the results of engines verifying the correct implementation of safety requirements by system design models, while arguing the confidence in such results. The verification results are communicated at the safety case abstraction level by extending the solution GSN element with metadata specifying whether the verification was successful or not (see Fig. 4-b) and whether the results are outdated or not. We define consistency

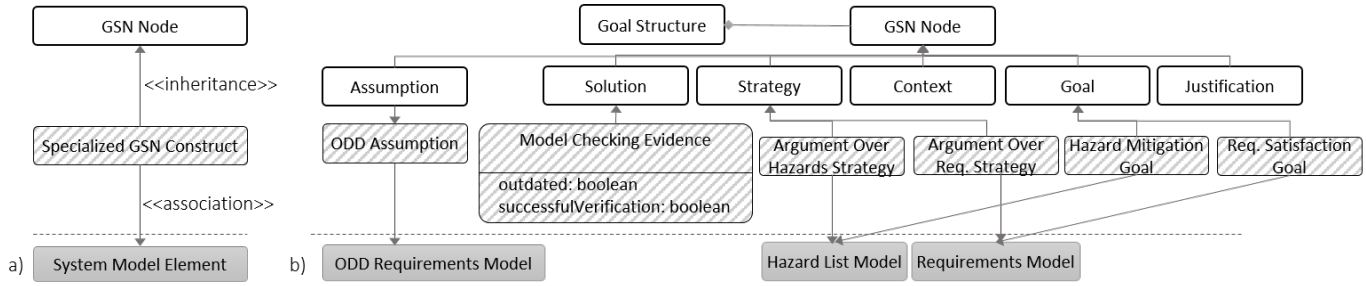

Fig. 4. a) Integration of safety case model elements with system models, via associations. b) A subset of specialized GSN nodes (depicted in hashed-gray) and their associations to system and safety engineering models (depicted in gray).

criteria between the verification evidence and the system specification (i.e., safety requirements and system design). We namely analyse which types of changes in system design or requirements models invalidate current verification evidence, in order to support the reuse of verification evidence.

E. Consistency Criteria

For each argumentation structure entailing specialized constructs, we define certain consistency criteria with respect to specific changes in system and safety engineering models, based on which we then specify consistency checks. The consistency criteria defined for specific claims and argument structures refine the following general consistency criteria between safety case and engineering models:

- **Reference existence.** Evaluation of whether the model elements referenced in the safety case model are available. This check supports the confirmation review objective C.10.3 from ISO 26262-2;
- **Trace correctness.** The tracing relationships specified in safety argumentation structures shall be consistent with tracing relationships specified in system models (see objective C.10.4 from ISO 26262-2 regarding traceability). For example, the claim about the satisfaction of a requirement shall support a higher-level claim about the satisfaction of another requirement only if there is a tracing relationship between the two requirements;
- **Argument completeness.** The safety argumentation shall be sufficient to argue about functional safety (see confirmation review objective C.10.2 from ISO 26262-2). For example, a safety argument shall go over all identified hazards, i.e., for each identified hazard there shall be a specialized goal claiming that the risk associated to the respective hazard has been reduced;
- **Refinement correctness.** Given system specifications at different levels of abstraction, it shall be checked for correct refinement. For example, given certain changes in one requirement from the functional safety requirements model, the functional architecture model shall be re-verified against the changed requirement via model-based verification engines and the new verification results shall be integrated in the safety case as evidence.
- **Correct ASIL decomposition.** Given the available hazards and requirements models, it shall be checked if the ASIL decomposition is done correctly, as assumed in the safety case model, based on ISO 26262-9;
- **ODD assumptions validity.** Given a change in the Operational Design Domain, it shall be checked whether the system functions safely in the new assumed operational context. Further, the assumed operational context properties explicitly specified in the safety case model shall be validated during runtime, in order to recognize when the system under consideration exits the operational design domain, as recommended by ISO 21448.
- **Up-to-date evidence.** Whenever there is a change in a system specification, assurance evidence may become stale [11]. Therefore, we need to check if the evidence on which the argumentation is based reflects the most current state of the system implementation and of the assumed operational context. For example, for manually performed checks, we shall make sure that the timestamp of the review make sure that the evidence is still actual.

F. Argument over Requirement Satisfaction

In Fig. 3, apart from showing models of parts the safety work products generated for this system, as presented in Sec. III, we show an example of a checkable safety fragment, which is integrated with these models. Given certain changes applied to the engineering models, we describe in the figure the outputs of some consistency checks.

The fragment entails a `Requirement Satisfaction` specialized goal, specifying that a certain safety requirement is satisfied. The specialized goal has an association to one requirement model element (i.e., the `SG02` identified for the the intersection handling system introduced in Sec III). The goal is only valid under the assumption that the system operates in the expected ODD. We specify this by `ODD Assumptions` specialized assumption, integrated with a requirements list model specifying the ODD. The requirements in that list may be checked at runtime. The `Requirement Satisfaction` goal is supported by an argument about the satisfaction of all the requirements derived from the respective requirement (see `Argument over Requirements` specialized strategy).

The `Argument over Requirements` strategy has an association to a model entailing the functional safety requirements derived from the respective safety goal, as shown in Fig. 3. The strategy is further supported by `Requirement Satisfaction` specialized sub-goals arguing about the satisfaction of one particular requirement in the list associated to the strategy. These sub-goals can be supported by `Model-checking Evidence` specialized solutions, integrated with model-based verification results. Each such specialized solution entails metadata in form of attributes regarding the status of the verification, namely whether the verification has been successful and whether the results are up-to-date or not. Given these specialized GSN constructs, we add new nodes to the set of GSN node types $\{s, g, e, a, j, c\} \bigcup \{argumentOverRequirementsStrategy, requirementSatisfactionGoal, modelCheckingEvidence\}$. The extensions can be seen in Fig. 4-b.

Regarded changes and proposed checks. Given the general consistency criteria defined in Section IV, we present in the following a set of consistency checks between the checkable safety case fragment presented before and the models with which it is integrated. The checks are triggered by certain types of changes in engineering models:

- **Reference existence check.** When an element from the requirements model is deleted, it is checked if the deletion causes a broken link in goals of type `Requirement Satisfaction`: $\forall rsg\ l(rsg) = requirementSatisfactionGoal\ CHECK\ brokenLink(rsg.associatation) = true$. For example, in Fig. 3, we show the output of the consistency checks when `FSR02.3`: *AD system shall detect non-overridable obstacles (small objects, pot holes) that may affect the dynamic of the ego vehicle if driven over* defined for the intersection handling system was deleted from the model. The requirement was deleted because the detection of pot holes and small objects hinders the performance of the system;
- **Argument completeness check.** When a new requirement is added to the requirements model associated to the `Argument Over Requirements` specialized strategy, it is checked if, for each requirement in the list, there is an argumentation leg about its satisfaction. This means that it is checked if, for each requirement, there is a `Requirement Satisfaction` specialized goal having an association to the respective requirement: $\forall req \in reqList$, where $\exists l(str) = argumentOverRequirements \bigwedge str.association = reqList\ CHECK\ \exists g$, where $l(g) = requirementSatisfaction \bigwedge str \rightarrow g \bigwedge g.association = req$. For example, considering our intersection handling system example, after deciding during the functional safety concept to use cameras for short-range sensing, we added a requirement `FSR02.12`. Given the newly added requirement to the requirements model associated to the `Argument over Requirements` specialized strategy, the consistency check alerted the safety engineer that the satisfaction of the newly added safety requirement has not been regarded in the argumentation (see Fig. 3);
- **Trace correctness check.** When a new requirement is added to the requirements model associated to the `Argument Over Requirements` specialized strategy, it is checked if the respective requirement has a trace to the requirement associated to the supported `Requirement Satisfaction` specialized goal: $\forall req \in reqList$, where $\exists l(str) = argumentOverRequirements \bigwedge str.association = reqList \bigwedge \exists g \rightarrow str \bigwedge l(g) = hazardMitigation\ CHECK\ req.reqTrace = g.association$. For example, in the intersection handling system, when added, requirement `FSR01.1` will not satisfy the trace correctness consistency criterion because, unlike the other requirements in the list, it does not trace to safety goal `SG02`, and therefore it cannot contribute at the argument regarding the satisfaction of `SG02` (see Fig. 3);
- **Correct ASIL decomposition check.** Given a newly added requirement in the requirements model associated to the `Argument Over Requirements` specialized strategy, it is checked if the ASIL of the respective requirement correctly decomposes the ASIL assigned to the requirement from which the newly added requirement was derived: $\forall reqList$, where $\exists l(str) = argumentOverRequirements \bigwedge str.association = reqList \bigwedge \exists g \rightarrow str \bigwedge l(g) = hazardMitigation\ CHECK\ correctASILDecomposition(g.association, reqList)$, where $correctASILDecomposition$ is a function that, given a hazard or a higher-level requirement and a set of derived requirements, it assess if the ASIL decomposition rules have been followed. For example, in our intersection handling system, `FSR02.09`, together with the other requirements defined at the same abstraction level do not correctly decompose `SG02` safety goal (see Fig. 3);
- **Refinement correctness check.** Given a modification in either the verified system component in the system architecture or the claim of the requirement against which the component is verified, the verification results are to be annotated as *outdated* and should be re-generated. For example, when modifying the claim of requirement `FSR02.9` by writing "long range zone", instead of "medium range zone", the safety engineer shall reverify the functional architecture model implementing this requirement (see Fig. 3), as different sensing zones require different sensing capabilities.

V. Related Work

The state of the art proposes a series of works on creating and manipulating safety cases models integrated with models of work products generated during the safety lifecycle.

AdvoCATE is a safety case modeling approach supporting the automated generation of safety case models consistent with engineering models generated during system development and safety analysis [6]. The automated generation of safety case models is based on automatic instantiation of safety case patterns. Similar to patterns in software design, a safety case pattern is a template for arguing the satisfaction of a reoccurring type of safety claim with placeholders for system-specific information. The automated instantiation of such patterns in AdvoCATE is done via a table that maps placeholders within the claims of the pattern with string-based identifiers of elements within system and safety engineering models, such as hazards, requirements or system design elements. While the instantiation is automated, the safety engineer still needs to manually assess when the safety argumentation needs to be regenerated, in order to be consistent with engineering models. Further, in AdvoCATE model-based verification results are integrated as evidence in safety case models via a reference from solution elements to the path verification results files.

ENTRUST is a methodology for the systematic ENgineering of TRUstworthy Self-adaptive sofTware, which proposes assurance processes to develop trustworthy self-adaptive software, while combining design-time and runtime modeling and verification activities [3]. Furthermore, ENTRUST proposes a safety case pattern arguing about the satisfaction of safety properties, based on both design-time and runtime model-based verification evidence. After each system reconfiguration, the verification is re-executed and the safety argumentation is regenerated, by automatic pattern instantiation.

Support for automatic detection of consistency problems. Our consistency checks could be used in AdvoCATE or ENTRUST to trigger the need for safety case regeneration, as currently the decision of regeneration is manually taken by the safety engineer. Unlike AdvoCATE and ENTRUST, which propose safety case models referencing engineering models by specification of string-based identifiers of model elements as metadata, we propose that the safety case model is integrated with other engineering models. To this end, our proposed specialized safety case constructs have direct associations to other model elements (see Fig. 4-a). One advantage of the integration between safety case and other engineering models is that it allows safety case models to be aware of changes in system and safety engineering models.

The AMASS platform supports engineering and assurance activities, including the creation of safety case models integrated with system architecture models [17]. In the AMASS platform, safety case fragments are automatically generated via pattern instantiation with associations to elements within the system design model [13]. While the safety case model is integrated with the system architecture model, thus enabling traceability, there are no guidelines for maintaining the two models consistent. Given a change in the system specification, the decision of regenerating the safety case is taken by the safety engineer, after assessing the impact of a change.

Support for reuse of safety evidence and argumentation. Instead of completely regenerating the safety case, as it is done in the approaches presented before, it may be sufficient to change part of the argumentation, thus facilitating the reuse of verification evidence and safety argumentation. Our consistency checks provide feedback to the safety engineer on the impact a certain change has on the safety case model and on how to update the safety case model in order to achieve consistency with the other work products.

MMINT-A is a model management approach for maintaining GSN-based safety case models consistent with system models based on model evolution concepts, with the scope of identifying the minimum set of reusable argumentation elements after a change in the system specification [11]. Given any change (i.e., addition, deletion, modification) of a referenced system model element, the approach first annotates the safety case elements referencing the respective system model element as to be revised. Then, each element in the safety case model having a connection to the element to be revised and referencing to a model element having a trace to the changed model element is annotated as to be rechecked. However, the MMINT-A impact analysis is too conservative, hence the need for more precise impact analysis. Still, MMINT-A can be used as complementary to our approach - when a change for which we have not define any consistency criteria occurs, the change impact analysis proposed by MMINT-A may be used.

More precise impact analysis. One advantage of our approach over MMINT-A is that, whereas in MMINT-A any change in a system element causes its associated elements in the safety case model to be marked for revision, our consistency checks trigger warnings only for specific types of system changes affecting specific types of claims. As such, only actually impacted elements are identified.

Guiding recovery actions. We not only identify the impacted elements, but also the type of impact, supporting the safety engineer in deciding the recovery actions meant to re-establish the consistency between safety work products. For example, given the modification of the ASIL attribute of a requirement, the consistency checks provide feedback on whether the ASIL decomposition is still correct. If the ASIL decomposition is incorrect, then the safety engineer knows that the recovery action is to reconsider the ASIL assignment.

VI. Conclusion and Next Steps

Enabled automations in consistency assessment. Based on the general consistency criteria introduced in Sec. IV, we can define for specific checkable safety case fragments, specific consistency checks considering certain types of changes in referenced engineering models. Currently, we automated the following consistency checks:

1) Given the deletion of any element in engineering models, warnings regarding broken links in safety case models will be triggered;
2) Given the addition of new items in hazards list or requirements models, warnings regarding the fact that the argumentation does not consider the mitigation of the newly identified hazard or the satisfaction of the newly specified requirement are output;

3) Given the modification of a system-level requirement or of the system design model, model-based verification evidence is re-generated;
4) Given the modification of ASIL attributes, the correctness of the ASIL decomposition is checked;
5) Given the modification of tracing relationships in engineering models, it is checked if the safety argumentation structure is invalidated.

Next, we plan on investigating the automation of other checks. For example, we want to look into how to check the consistency between safety case models and models of Operational Design Domain (ODD). Further, given process models, we could check the consistency of safety arguments with organizational changes. Also, we plan on investigating what types of changes in requirements or system design models do not trigger the need for re-verification, scoping at supporting the reuse of verification evidence.

Towards agilized model-based system and safety engineering in automotive. In this paper, we introduce checkable safety case models, which may support safety managers and engineers in developing automotive systems in an agilized manner (i.e., allowing frequent changes in system specifications) through frequent and automated consistency checks between safety case and engineering models. Given both the integration of safety and system engineering models and the integration of replayable and maintainable model-based verification results as evidence in safety case models, whenever a change occurs in an engineering model, a set of checks for consistency with the safety case model will automatically be triggered. To this end, we implemented a tool – FASTEN.Safe, which supports the modeling of checkable safety cases and of other safety work products and the automated execution of consistency checks, given certain changes in engineering models [4]. While we present our concepts by extending GSN, the conceptual solution presented in this paper for GSN is translatable to for SACM [14] or any other safety case modeling approach.

Limitations and next steps. Automation of consistency checks is only achievable for certain types of changes, depending on the capabilities of formalizing both the safety claims and the work products. In Sec. IV, we already presented a set of change types for which automated checks are feasible. Next, we plan to investigate for which other types of changes the automation of consistency checks is achievable. We envision that agilized safety lifecycles will only allow a constrained set of changes. Some changes, such as changes in the addressed operational context, are usually undesirable due to their impact on the entire safety argumentation. Further, we identified three aspects hindering the usage of our proposed approach in practice. First, while we assume the existence of models for all safety work products, in practice, especially for legacy systems, the specification of work products is done in simple, unstructured ways (e.g., text-based format). As such, to foster adoption of our approach, one line of future work could be to propose methods for generating such models from already existing artefacts. Second, while our approach relies on the integration between the safety case and other engineering models, such artefacts are typically dispatched in various tools, sometimes without open interface to access them. However, there already exists dedicated work for that, the most well-known being Open Services for Lifecycle Collaboration (OSLC)[2]. Third, even given a tool-chain implementing our proposed approach, tool qualification could be another challenge to be addressed.

Acknowledgments. The author B. Gallina is partially supported by by Sweden's Knowledge Foundation via the SACSys (Safe and Secure Adaptive Collaborative Systems) project.

REFERENCES

[1] GSN community standard version 2 (2018), http://www.goalstructuringnotation.info/
[2] Bloomfield, R.E., Netkachova, K.: Building blocks for assurance cases. In: Proceedings of the 25th International Symposium on Software Reliability Engineering Workshops (ISSRE Workshops). pp. 186–191. IEEE Computer Society (2014)
[3] Calinescu, R., Weyns, D., Gerasimou, S., Iftikhar, M.U., Habli, I., Kelly, T.: Engineering trustworthy self-adaptive software with dynamic assurance cases. IEEE Trans. Software Eng. 44(11), 1039–1069 (2018)
[4] Cârlan, C., Ratiu, D.: Fasten.safe: A model-driven engineering tool to experiment with checkable assurance cases. In: Proceeding of the 39th International Conference on Computer Safety, Reliability, and Security - SAFECOMP. LNCS, vol. 12234, pp. 298–306. Springer (2020)
[5] Denney, E., Pai, G.: A formal basis for safety case patterns. In: Proceedings of the 32nd International Conference on Computer Safety, Reliability, and Security - SAFECOMP. LNCS, vol. 8153, pp. 21–32. Springer (2013)
[6] Denney, E., Pai, G.: Tool support for assurance case development. Automated Software Engineering 25(3), 435–499 (2018)
[7] Gallina, B., Muram, F.U., Ardila, J.P.C.: Compliance of agilized (software) development processes with safety standards: a vision. In: 4th international workshop on Agile Development of Safety-Critical Software. Association for Computing Machinery (2018)
[8] International Organisation for Standardization (ISO): 26262: Road vehicles - functional safety. Tech. rep. (2018)
[9] International Organization for Standardizatio: Road vehicles cybersecurity engineering. Tech. rep. (2019)
[10] International Organization for Standardization (ISO): Road vehicles safety of the intended functionality. Tech. rep. (2019)
[11] Kokaly, S., Salay, R., Chechik, M., Lawford, M., Maibaum, T.: Safety case impact assessment in automotive software systems: An improved model-based approach. In: Proceedings of the 36th International Conference on Computer Safety, Reliability, and Security (SAFECOMP). LNCS, vol. 10488, pp. 69–85. Springer (2017)
[12] Object Managment Group: Structured Assurance Case Metamodel - SACM, version 2.1. Tech. rep. (2020), https://www.omg.org/spec/SACM/About-SACM/
[13] Sljivo, I., Uriagereka, G.J., Puri, S., Gallina, B.: Guiding assurance of architectural design patterns for critical applications. Journal of Systems Architecture 110, 101765 (2020)
[14] The GSN Working Group: Mapping between gsn and sacm 2.0. Tech. rep. (2020), http://www.goalstructuringnotation.info/gsn-metamodel
[15] US National Highway Traffic Safety Administration (NHTSA): A framework for automated driving system testable cases and scenarios. Tech. rep. (2018)
[16] US National Highway Traffic Safety Administration (NHTSA): The safer affordable fuel efficient (safe) vehicles final rule for model years 2021-2026. Tech. rep. (2020)
[17] de la Vara, J.L., Parra, E., Ruiz, A., Gallina, B.: The amass tool platform: An innovative solution for assurance and certification of cyber-physical systems. In: Proceedings of the 26th International Conference on Requirements Engineering: Foundation for Software Quality. vol. 2584. CEUR-WS (2020)
[18] Wardziński, A., Jones, P.: Uniform model interface for assurance case integration with system models. In: Proceedings of the 36th International Conference on Computer Safety, Reliability, and Security (SAFECOMP). vol. 10488, pp. 39–51. Springer (2017)

[2]https://en.wikipedia.org/wiki/Open_Services_for_Lifecycle_Collaboration