FASTEN.Safe: A Model-Driven Engineering Tool to Experiment with Checkable Assurance Cases

Chapter · July 2020

DOI: 10.1007/978-3-030-54549-9_20

2 authors:

Carmen Carlan
Technische Universität München
19 PUBLICATIONS **56** CITATIONS
SEE PROFILE

Daniel Ratiu
Volkswagen AG
72 PUBLICATIONS **1,158** CITATIONS
SEE PROFILE

FASTEN.Safe: A Model-driven Engineering Tool to Experiment with Checkable Assurance Cases

Carmen Cârlan[1](✉) and Daniel Ratiu[2]

[1] fortiss GmbH, Munich, Germany
carlan@fortiss.org
[2] Siemens Corporate Technology, Munich, Germany
daniel.ratiu@siemens.com

Abstract. The Goal Structuring Notation (GSN) is popular among safety engineers for modeling assurance cases. GSN elements are specified using plain natural language text, this giving safety engineers great flexibility to express their arguments. However, pure textual arguments introduce ambiguities and prevent automation. Currently, assurance cases are verified by manual reviews, which are error prone, time consuming, and not adequate for today's systems complexity and agile development methodologies. In this paper we present our research tool *FASTEN.Safe* , which extends GSN with a set of higher-level modeling language constructs capturing recurring argumentation patterns and integrating formal system models. This allows automatically checking 1) the intrinsic consistency of assurance models, 2) the consistency of arguments with system models and 3) the verification of safety claims themselves by using external verification tools. *FASTEN.Safe* is open source and allows experimenting with language abstractions to bridge the world of GSN-based arguments that are common among safety engineers and the world of formal methods that enable automation. Last but not least, we report on the preliminary experience gained with *FASTEN.Safe*.

Keywords: assurance cases, GSN, NuSMV, language engineering

1 Introduction

Explicit modeling of assurance cases supports engineers in reasoning about the system safety and communicating with third-parties. Assurance cases are central artifacts of the safety assurance engineering process, an explicit model of an assurance case entailing the core of the argument that the system is safe.

The Goal Structuring Notation (GSN) is a compact graphical representation for argumentations and one of the most prominent notations used for modeling assurance cases, containing a small number of constructs that are intuitive to understand and easy to use by practitioners [1]. Using models for assurance arguments brings structure and enables the application of well-formedness rules

– e.g., no circular reasoning is allowed. Claims within GSN elements are specified using plain natural language text, safety engineers having great flexibility to express their arguments. However, pure text-based claims introduce ambiguities and prevent automation. The only validation method of text-based claims within assurance cases are manual reviews, this causing two challenges. First, such reviews prove to be tedious when complex systems are built. Second, manual reviews are not suitable for a more agile development mindset, where change requests of safety critical components are frequent and their impact on the assurance argument should be immediately evaluated.

In our work, we explore the way in which assurance cases can be made checkable, yet easy to understand by practitioners. To this end, we extend GSN language constructs with specialized constructs (DSLs) that reference formally specified system models. This enables automated consistency checks both within assurance cases and with the models linked therein. Further, given the integrated automated verification engines, the verification of the satisfaction of safety claims by the referenced system models can be triggered from the assurance case model and the verification results lifted at the level of the assurance cases. Our long-term vision is to make assurance case models central artefacts for starting verification activities in the context of agile development of safety-critical systems.

Contributions and Structure. The main contribution of this paper is *FASTEN.Safe*[3], a new tooling approach to build extensible and semantically rich assurance cases that are linked with formally defined system models in order to increase the rigour of the assurance cases specification and to enable automated verification of assurance cases. *FASTEN.Safe* is an extension of FASTEN, a *FormAl SpecificaTion ENvironment* described in [9], released under EPL 1.0 license and available on github[4]. The rest of the paper is organized as follows: Section 2 presents a set of GSN extensions that make automated checks possible, Section 3 describes our preliminary experience with using these extensions, in Section 4 we present the related work and Section 5 concludes the paper.

2 Checkable Assurance Cases

To facilitate automation for safety assurance, our approach is to check the assurance case directly by 1) ensuring its intrinsic consistency, 2) enabling automatic consistency checks with system models, and 3) firing up checks of various system aspects that are modeled outside of the assurance case itself. These checks make the assurance case a central engineering artifact that is to be checked and facilitates automation for safety assurance.

Running Example: Airbag of a Car. To facilitate the understanding, we introduce in the following *FASTEN.Safe* features "by example". We use system models of an airbag and integrate them with assurance case elements. The airbag system's main functionality is specified by the following functional requirement: **FR_01**: *The purpose of an airbag is to slow a vehicle occupant's*

[3] `https://sites.google.com/site/fastenroot/home`
[4] `https://github.com/mbeddr/mbeddr.formal`

motion as evenly as possible using a bag designed to inflate extremely quickly, then quickly deflate during a crash scenario. Based on this requirement, a list of hazards is derived. Hazard **H1** is mitigated by implementing, among others, the following derived safety requirement: **SR_01**: *The airbag shall inflate only after a collision.* This requirement is then refined in the system's architecture, as described by Arts et. al. [2]. The architecture comprises a top-level component, named `airbag_system`, containing the following subcomponents: 1) the `Sensor`, which detects collisions and sends messages encoded by an End-2-End (E2E) protection mechanism (specific to the AUTOSAR standard) to the airbag controller, 2) the `Link` that connects the components, and 3) the `Device`, entailing the airbag's controller. All components are specified in a black-box manner, the requirements being specified via formal contracts expressed in LTL. While functional requirement **FR_01** is formalized as *post(1) collision_post* postcondition, postcondition *post(2) no_collision* formalizes safety requirement **SR_01**. The bus guarantees that the airbag system functions correctly even when failures such as message corruption and deletion occur. To express this failure model, we specify precondition `pre(1) collision pre`.

Tool architecture. In Fig. 1, we present an overview over our tooling platform built using the Jetbrains' MPS language workbench[5]. *FASTEN.Safe* is a platform that allows experimentation with different modeling abstractions for the development of safety critical systems. The tool addresses four concerns – 1) hazard and risk analysis (HARA), 2) requirements specification, 3) formal modeling of system architecture [9] and 4) safety argumentation. To enable verification of architecture models, *FASTEN.Safe* integrates the NuSMV verification engine. At its core, *FASTEN.Safe* features an implementation of the Goal Structuring Notation (GSN) language. Additionally, to support automatic checks, we implemented extensions of the GSN language with specific types of goals, strategies or solutions, integrating other safety and system models.

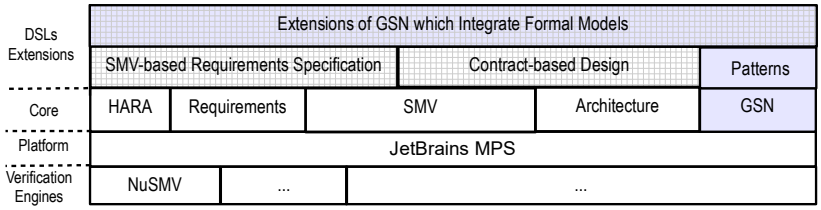

Fig. 1. Overview over the FASTEN DSL-stack developed using JetBrains' MPS. FASTEN integrates verification engines as external tools and provides support for their input languages. In blue we highlight the parts that belong to *FASTEN.Safe* – they comprise an implementation of GSN language and a set of patterns. Specialized GSN entities (blue-hashed) reference formally specified elements of system models (hashed).

Checkable assurance cases are modeled via the instantiation of a special type of GSN-based patterns, based on state-of-the-art patterns, for which we provide

[5] `https://www.jetbrains.com/mps/`

special language constructs integrated with system models, and which come with automated checks. In the following we introduce three categories of such checks.

2.1 Type I Checks: Intrinsic Consistency of the Safety Case

GSN has rules for how to connect GSN elements among each other to obtain a syntactically correct argumentation. However, it does not regard the semantic validity of an argumentation. GSN patterns go a step further and enable a higher intrinsic consistency since they re-use several entities together. To ensure the validity of the arguments created via instantiation, the patterns come with instructions on how to instantiate them. However, unless the instantiation is done automatically, there is no guarantee that the instantiation will generate a valid argumentation. To ensure valid instantiation of patterns, *FASTEN.Safe* provides special types of GSN entities via language extensions, which may only be connected via special types of connections, which extend the GSN *supported by* and *in context of* connections. For example, the *Argument over Hazards Strategy* can only be supported by sub-goals of type *Hazards Mitigation Goal*.

2.2 Type II Checks: Consistency with System Models

The second category of checks ensures the consistency between the assurance case and system models. To this end, we propose specialized entities extending GSN elements, which are integrated with different types of models, specifying the system at different levels of abstraction (e.g., hazards, requirements, architecture). As the system models are created in the same tool as the the safety case model, deeper integration can be achieved.

Examples of patterns that may undergo such checks are arguments that all hazards, or all safety requirements have been addressed, as proposed by Hawkins and Kelly [7]. These patterns entail specialized GSN strategies integrated with hazards and, respectively, requirements models. Consistency checks are then executed on these strategies, ensuring that for each element in the list, an argumentation leg exists. In Fig. 2, on the left hand side, we depict the hazards mitigation checkable pattern; on the right hand side, we depict the patterns' instantiation with hazards from the airbag example. The automated checks are enabled by having specialized goals for claiming hazard mitigation – each goal referencing a specific hazard – and specialized relationships between the strategy and the goals. An error from our checks is triggered as the GSN model does not contain argumentation legs regarding the mitigation of `H3` and `H4` hazards.

Another example for type II checks is a pattern for arguing about the satisfaction of a requirement in a contract-based design setting, entailing specialized GSN entities integrated with the components architecture (see Fig. 3). The structure of this pattern reassembles the structure of the argument on component level pattern proposed by Warg et. al. [10]. Automated checks will trigger errors given any inconsistencies between the components specified in the system architecture and the GSN model (e.g., ensure that there is a correctness implementation claim for each direct sub-component of the top-system).

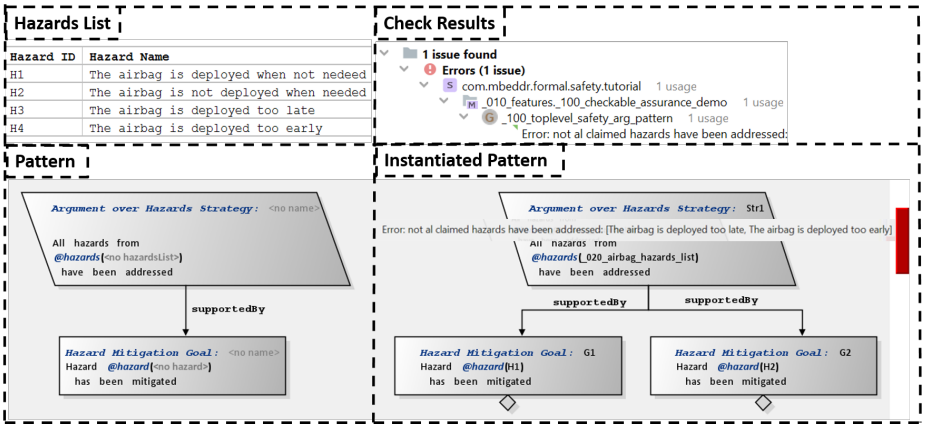

Fig. 2. On top-left a list of hazards of an airbag is presented. On bottom-left the checkable pattern containing specialized GSN strategy and goal is depicted, whereas on bottom-right the instantiation of the pattern for the airbag system is pesented. On top-right the consistency checking results are displayed – in this example a check fails since not all hazards have a corresponding *Hazard Mitigation Goal*.

2.3 Type III Checks: Verification of Safety Claims

The third type of checks that can be executed on an assurance case model are verifications of the satisfaction of claims within the safety case by the referenced system model via external verification tools. GSN elements expressing a system property may be specialized as verifiable entities, by integrating them with the formal specification of the respective system property. Such specialized entities are integrated with external tools capable to verify the respective property type. Verification goals are always supported by specialized solutions, which automatically integrate the verification results whenever the verification is executed.

For example, the satisfaction of a safety requirement formalized as an assume-guarantee contract can be reflected by a specialized strategy supported by three main strands of argument, each comprising a specialized, checkable goal (see Fig. 3). First, an argument about the correct refinement of the contract of the upper-level component by the contracts of the subcomponents should exist. Contract refinement checks ensure that the guarantees of an upper-level component is not weakened by the contracts of its subcomponents and the assumptions of the upper-level component is strong enough to satisfy the assumptions of subcomponents (*Refinement Check* goal).

Second, A/G compatibility checks ensure that the composition of subcomponents is consistent (*Compatibility Check* goal). Third, the correct implementation of each corresponding subcomponent shall be argued (*Implementation Check* goal). In the case when subcomponents are hierarchical, the implementation check is again potentially performed via a CBD strategy. In the case when subcomponents are atomic, a model checker (in our case NuSMV) is used to verify if the implementation (an SMV module) of an atomic subcomponent sat-

isfies its contract (i.e., $A \rightarrow G$). These goals are checkable, meaning that their claim is checked with NuSMV and, if the verification is not successful, an error is triggered in the assurance case editor. Each of the verification goals has a special solution that is automatically updated with the corresponding model checking results as evidence. Specialized solutions enable the automatic integration of NuSMV results as evidence in assurance cases, whenever the verification is (re)executed. The verification results are interpreted and the solution's editor displays whether the verification is successful or not. Thereby, the safety engineer can see at a glance which goals have been invalidated after a change was implemented. Furthermore, a solution is annotated as *outdated* when either the referenced formal models or the claims within the safety case change, thus making the verification results stale.

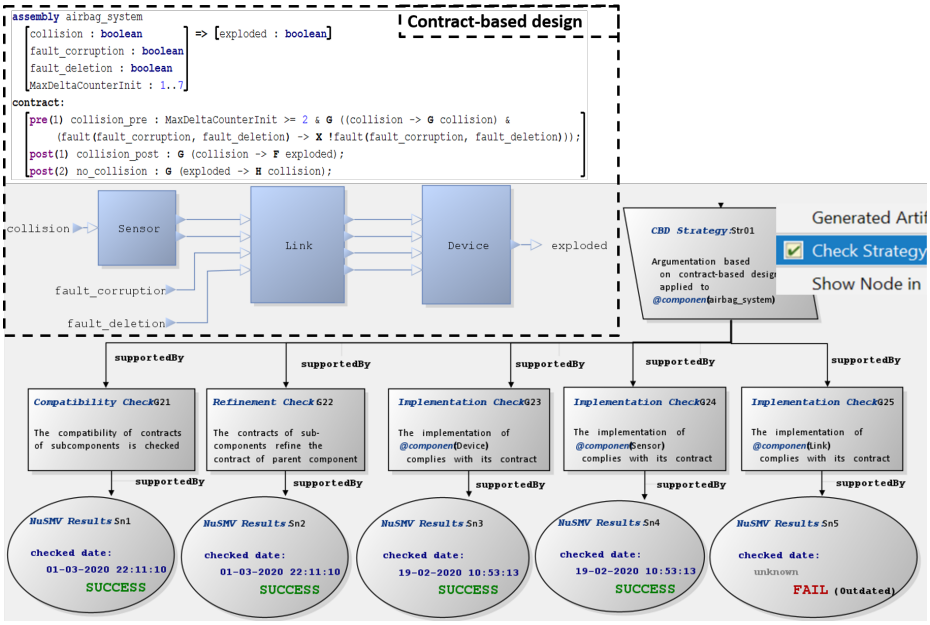

Fig. 3. The top-left side presents the interface definition of the `airbag_system` in terms of ports and contracts (pre-/postconditions) and the architectural decomposition. The bottom side displays the GSN argument about the correct implementation of a requirement based on contract-based design. The argument contains specialized entities (e.g., *CBD Strategy* is a specialization of *Strategy*) that are linked to corresponding system models. NuSMV can be started directly on the *CBD Strategy* node (right) and the results are lifted in and reflected in the corresponding solutions.

3 Experience

Using the Extensions. We are currently evaluating the benefits of using checkable assurance cases by collecting feedback from safety engineering practitioners who used *FASTEN.Safe* in different projects from automotive domain and piloting

the functionality in realistic projects. The most important benefits so far seem to be 1) the immediate feedback in the editor signalizing when assurance cases and system models get desynchronized, and 2) the possibility to start verification activities of claims from the assurance case itself, thereby making the trace between system model changes and assurance cases more transparent. Further, the feedback of practitioners is that, on longer term, based on the management of changes on system models and the assessment of their impact on the assurance case, our approach could support continuous verification of assurance cases, thus enabling the integration of assurance cases in a continuous delivery pipeline.

Developing the Extensions. As presented in Section 2, we have created a set of extensions of GSN constructs covering strategies and claims about hazards, requirements, specification of safety properties via LTL and their verification with NuSMV. These extensions sufficy to express three patterns from the literature [7], [10]. The language workbench MPS allows easy implementation of extensions in a modular fashion. On the conceptual side, we are looking for other argumentation fragments or patterns (e.g., based on safety architecture patterns) that can be made checkable.

4 Related Work

There is a vast amount of tooling approaches for developing assurance cases [8]. To the best of our knowledge, none of the existing tools support all types of checks (I-III). Another distinguishing characteristic of *FASTEN.Safe*, that allows us to implement the checks, is that it features language extensions of GSN that raise the abstraction level at which assurance cases are modeled. In the following, we compare *FASTEN.Safe* with existing assurance case modeling tools.

The most comprehensive automation support is provided by AdvoCATE [6] and AMASS tool platform [5], both tools supporting references from assurance case models to other system models (e.g., hazards, requirements or system architecture). AdvoCATE supports the automated creation and assembly of assurance arguments based on patterns instantiation, hierarchical abstraction for arguments, integration of formal methods in assurance arguments and verification of safety claims. The AMASS platform scopes at supporting assurance activities. For example, similarly to AdvoCATE, AMASS supports automated generation of model-based verification evidence, based on which, via pattern instantiation, assurance case fragments are automatically generated. While AdvoCATE and AMASS use patterns described directly in GSN to increase automation, we lift recurring patterns at language level via DSL abstractions. This allows us to define a richer set of consistency checks both for ensuring the intrinsic consistency of arguments and their consistency with existing system and safety models (checks of Type I and II). Furthermore, similar to our approach, in AdvoCATE analyses performed by external verification tools (e.g. model checkers) can be triggered directly based on the higher level entities.

Another category of approaches aims at automated construction of assurance cases based on system models and existing verification results. The Evidential

Tool Bus (ETB) supports the construction and maintenance of assurance cases by automatic generation of claims and evidence from the outputs of verification tools [4]. ENTRUST supports automatic instantiation of assurance case patterns with information from design-time and runtime system models and verification tools [3]. In contrast to these tools, the automation enabled in *FASTEN.Safe* checks the argumentation constructed by the engineer and does not re-generate entire argumentation fragments. In *FASTEN.Safe* changes in system models are immediately reflected in the assurance case via failed constraints.

5 Conclusions

In this paper we presented *FASTEN.Safe*, which is a platform for modeling assurance cases based on GSN and experimenting with semantically rich extensions for expressing safety arguments. Our work formalizes a subset of published GSN patterns using domain specific constructs and links them to system models that are amenable to automated checks. Our long term goal is to make GSN-based arguments automatically checkable. This would enable incremental safety assurance via (semi-)automated maintenance of safety cases, thereby facilitating the development of safety-critical systems in more agile settings. Next, we plan to identify more semantically rich extensions capturing assurance case patterns and to integrate other verification engines (e.g. for performing quantitative analyses). Furthermore, we intend to use the tool within real-world projects.

References

1. GSN community standard version 1, Nov. 2011.
2. T. Arts, M. Dorigatti, and S. Tonetta. Making implicit safety requirements explicit - an AUTOSAR safety case. In *SAFECOMP*, 2014.
3. R. Calinescu, D. Weyns, S. Gerasimou, M. U. Iftikhar, I. Habli, and T. Kelly. Engineering trustworthy self-adaptive software with dynamic assurance cases. *IEEE Trans. Software Eng.*, 44(11):1039–1069, 2018.
4. S. Cruanes, G. Hamon, S. Owre, and N. Shankar. Tool integration with the evidential tool bus. In *VMCAI*, 2013.
5. J. L. de la Vara, E. Parra, A. Ruiz, and B. Gallina. 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*, volume 2584. CEUR-WS, 2020.
6. E. Denney and G. Pai. Tool support for assurance case development. *Autom. Softw. Eng.*, 25(3):435–499, 2018.
7. R. Hawkins and T. Kelly. A software safety argument pattern catalogue. *The University of York, York*, 30, 2013.
8. M. Maksimov, N. L. S. Fung, S. Kokaly, and M. Chechik. Two decades of assurance case tools: A survey. In *SAFECOMP Workshops*, 2018.
9. D. Ratiu, M. Gario, and H. Schoenhaar. FASTEN: An open extensible framework to experiment with formal specification approaches. In *FormaliSE*, 2019.
10. F. Warg, H. Blom, J. Borg, and R. Johansson. Continuous deployment for dependable systems with continuous assurance cases. In *ISSRE Workshops*, 2019.