Safety.Lab: Model-based Domain Specific Tooling for Safety Argumentation

Daniel Ratiu, Marc Zeller, and Lennart Killian

Siemens Corporate Technology, Munich
name.surname@siemens.com

Abstract. Assurance cases capture the argumentation that a system is safe by putting together pieces of evidence at different levels of abstraction and of different nature. Managing the interdependencies between these artefacts lies at the heart of any safety argument. Keeping the assurance case complete and consistent with the system is a manual and very ressource consuming process. Current tools do not address these challenges in constructing and maintaining safety arguments. In this paper we present a tooling prototype called Safety.Lab which features rich and deeply integrated models to describe requirements, hazards list, fault trees and architecture. We show how Safety.Lab opens opportunities to automate completeness and consistency checks for safety argumentation.

Keywords: model driven engineering, safety-critical systems, assurance cases, tooling

1 Introduction

Product based safety argumentation needs a holistic view over the system and links heterogeneous artefacts from different development stages (requirements specification, system design, implementation, verification & validation) [16]. In the current practice, these artefacts are maintained in different heterogeneous and loosely integrated tools if at all. The content of referenced artefacts from within the argumentation is opaque and the references are only at a high granularity level, e.g. entire documents (see Fig. 1-left).

Developing assurance cases is not new, but is still imature in industrial practice [2]. Building assurance cases is entirely manual and with low tool support. Checking that the safety argumentation is complete and consistent with the system model is expensive and mostly a manual process done through reviews. Moreover, the costs are amplified during the evolution of the system when the safety argumentation needs to be evolved in order to keep up with the changes.

Model-based engineering promotes the use of models in all development phases from requirements to code and deployment. This means that adequate and rich models are used to describe different aspects of the system. Rich models spanning various abstraction levels of dependable systems allow to precisely define the interdependancies of the model artefacts on different level concerning a particular safety feature. Tracing these interdependencies through the model

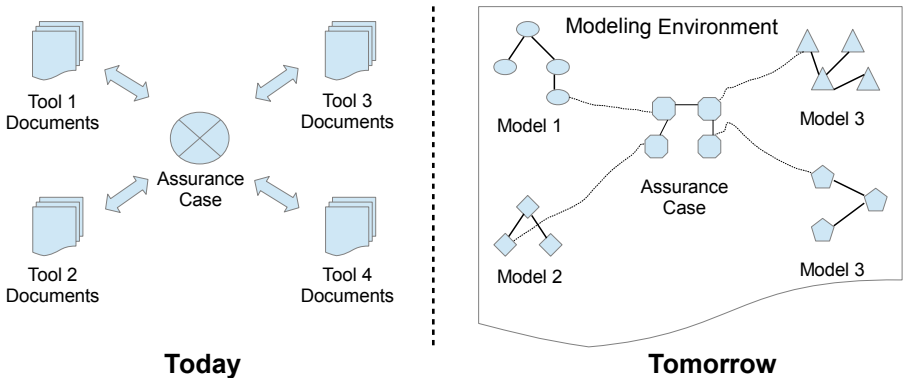


Fig. 1. Safety argumentation is today (left) coarse granular, based on documents managed by different tools. This limits the automation opportunities in building, quality assuring and reviewing assurance cases. Model-based safety argumentation (right) puts together fine granular elements of deep models. This increases the automation and enables advanced consistency checks.

as well as the development process lies at the heart of any safety argument. Such models ease the development, increase the quality and enable a systematic reuse. This opens new possibilities to build and maintain assurance cases which involve different disciplines and directly reference fine granular model elements (see Fig. 1-right). Furthermore, having a deep integration of artefacts in a tool, eases the development and verification of assurance cases by increasing automation for building models and continuously checking their consistency.

However, in industrial practice model-based development is currently only applied to isolated sub-systems (e.g. model-based testing or software modules). The overall system development and safety assessment are mainly based on multiple specification and analysis documents. *Our long term goal is to move away from documents-based development and analysis of safety critical systems to a model-based world in which semantically rich models are used to describe the system characteristics in an appropriate manner.*

In this paper, we present our experiments to build a model-based tool that supports development of safety cases and linking them to models developed in early process phases. Based on these deep models we present a set of automatic consistency checks between the safety argument structure and the structure of the models of the developed product.

Language Engineering as Technological Basis: Our work uses language engineering technologies, which refer to defining, extending and composing languages and their integrated development environments (IDEs). Language workbenches [3] are tools that support efficient language engineering. Our implementation relies on the JetBrains MPS[1] language workbench, which, unlike most other language

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

workbenches, uses projectional editing. Projectional editing makes it possible to easily combine models defined in different languages and notations [4].

One of the biggest projects developed using this technology is mbeddr [1], a stack of extensible domain specific languages for model driven engineering of embedded systems. Siemens is involved in the mbeddr project by building commercial extensions for controls software development. Our work presented here is part of our investigations on how can we extend the mbeddr technology stack to enable deeply integrated safety engineering.

Structure of this paper: In the next section we present our vision for the next generation of model-based tools for the development of assurance cases for safety critical systems. Subsequently we describe tooling for a set of safety-domain specific modeling languages and how are they integrated in order to achieve a gapless landscape of models. We present how we integrate these languages with requirements, architectural design, and safety analyses in order to provide a holistic view over the safety of the system. We conclude the paper with related work and presenting our plans for future work.

2 Long-term Vision for Safety Argumentation Tooling

2.1 Use of rich and domain specific models

The input for safety analyses are models which capture safety relevant characteristics of the system. These models are at different abstraction levels and captured using different notations like text, tables or diagrams. Unfortunately, in practice these models are captured only implicitly in tools providing weak structuring and consistency enforcements mechanisms like spreadsheets in MS Excel, "boxes and lines" drawn in MS Visio or plain natural language text written in MS Word. Hence, constructing and maintaining assurance cases are currently manual tasks which are performed mostly based on documents and with only spare tool support. The structure of artefacts referenced from assurance cases varies from a domain to another. For instance, the certification of trains is different from the certification of cars or medical equipment. Current tools are mostly agnostic with respect to the business domain. Domain specific structures are encoded with the help of modeling conventions if at all.

Safety.Lab targets to use domain specific models in order to fulfill the specific needs of engineers working in specific business units.

2.2 Integration of system design and safety analysis

A lot of safety argumentation relevant information is redundant and replicated across different views, many times in different tools. Common practice today is that weak and high-granular traceability links are present between development artefacts. A deep integration between system design and safety analysis is essential for the development of safety-critical systems because safety conditions

such as hazards and failure modes are often an outcome of unintended system functionalities and behaviours in a given environment.

The goal of Safety.Lab is to work with deeply integrated models in which the artefacts are integrated with each other and referenced from the assurance case. Functional system requirements are used as input in safety assessment and this lead to a set of hazards and failure modes that the system needs to deal with. These hazards must be mapped to the design and checked that indeed they are addressed. Thereby, Safety.Lab aims to minimize redundancies between artefacts and avoid inconsistencies in the safety argumentation and between the safety argumentation and the system model.

2.3 Support the construction of assurance cases

Safety assurance documents gather together heterogeneous information from different development stages. A safety case includes the system's safety concept which is build in an early stage of the development and is based as input for further development. During later development stages a variety of evidence is produced (e.g. test cases, FMEA tables, fault trees). This evidence is used to support safety arguments which explain that the system is sufficiently safe.

Safety.Lab takes advantage of the deep models and modern IDE techniques in order to support engineers building the safety cases. Building of safety cases still remains a manual process but with modern IDE support.

2.4 Support the evolution of assurance cases

Systems are evolving due to refinement and modification during the development and consequently the assurance cases must also evolve. Thereby, it must be enforced that changes in the system design lead to proper adaptions in the safety analyses and assurance cases. Before changes are implemented, we need to estimate the impact of such changes on the safety of the system. Performing change impact analysis is currently a manual process and thereby very time-consuming. Many times the costs are so high that changes are avoided and only work-arounds are provided.

Safety.Lab aims to take advantage of rich models and their integration in order to help engineers to keep their models consistent with the assurance case and assess possible inconsistencies due to changes.

3 Safety.Lab

In this section we, present our tool Safety.Lab as a first step towards realizing our long-term vision of a fully model-based construction of critical systems including their assurance cases. Our tool is based on a deeply integrated set of languages that address early phases in the development process for safety critical systems. Our presentation is illustrated with a running example about the braking function of a car.

System-level functional requirements. Functional requirements represent the input in our process and are documented using a domain specific language (DSL) for describing requirements. The requirements DSL integrates natural language text with model fragments [14] (due to the lack of space, we present in this paper only requirements as prose text and simple meta-data). In Fig. 3 we illustrate an example high-level functional requirement about the braking function of a car. The braking function has two sub-functions: manual brakes triggered by the driver and emergency braking triggered by the onboard-computer in case when an obstacle is detected to be too close to the car.

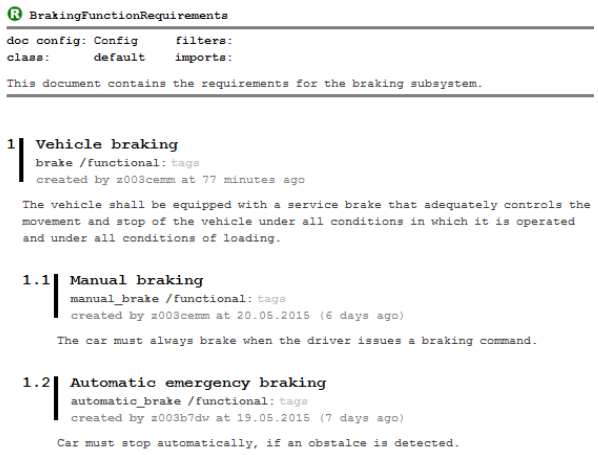
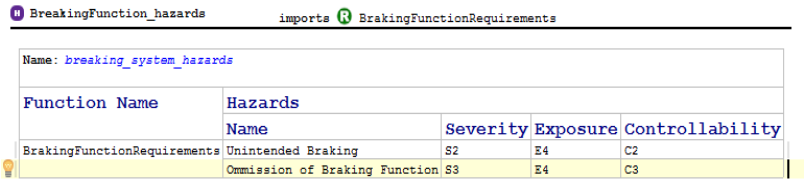


Fig. 2. A fragment of a requirements model.

Hazards analysis. The next process step that we support is hazards analysis. Input for the hazards analysis are the functional requirements. In our example, from the set of functional requirements about braking we identify two hazards: unintended braking and braking omission. The set of hazards along with their attributes (severity, controllability, exposure) are captured using a domain specific language with tabular notation. The reference to the functional requirement that is used as basis for the hazards analysis is a first class modeling construct rather than a trace link.

In Fig. 3 we illustrate how Safety.Lab models a list of hazards. The table field "BrakingFunctionRequirements" is a first-class reference to the requirements module which contains the functional requirements of braking function.



Function Name	Hazards			
	Name	Severity	Exposure	Controllability
BrakingFunctionRequirements	Unintended Braking	S2	E4	C2
	Ommission of Braking Function	S3	E4	C3

Fig. 3. Hazards list for the braking function

High-level safety requirements. Safety requirements are derived based on the hazards analysis. Each hazard leads to one or more safety requirements which are captured using an extension of the requirements DSL. This extension allows each safety requirement to reference the hazard it addresses and contains its integrity level which is automatically derived (based on ISO26262) based on the attributes of the corresponding hazard.

The safety requirements for our braking example are illustrated in Fig. 3. The hazards analysis lead to two safety requirements for avoiding unintended braking and for preventing accidents caused by missing braking function. These requirements have ASIL B and D.

Fig. 4. Safety requirements example

(Sub-)system architecture. The high-level architecture is used to structure the system such that the requirements can be satisfied. During the design phase the integrated safety analysis helps to find tailored solutions the mitigate threats originating from the relevant failure modes within the system. Architectural decisions are thus motivated by the need to address safety requirements. In our example (Fig. 3), we have two channels to implement the braking functionality - this is required (cf. ISO26262) by the ASIL D of one of our safety requirements.

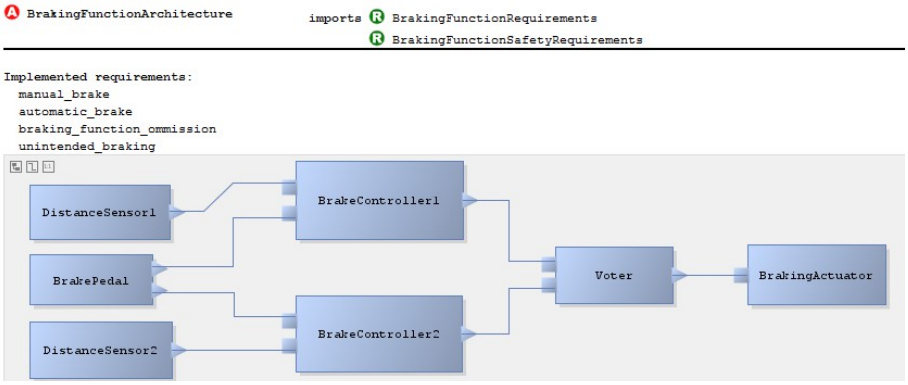


Fig. 5. Braking subsystem architecture. This particular architecture is motivated by the requirements which are directly referenced.

Safety analysis. For example, fault trees are used to analyze possible fault propagation at sub-system level that can lead to a hazard on the system level. The fault tree represents a view (propagation of faults between different sub-systems) on the system architecture. In Fig. 6 we illustrate an example of a fault tree for the "ommision of braking" hazard.

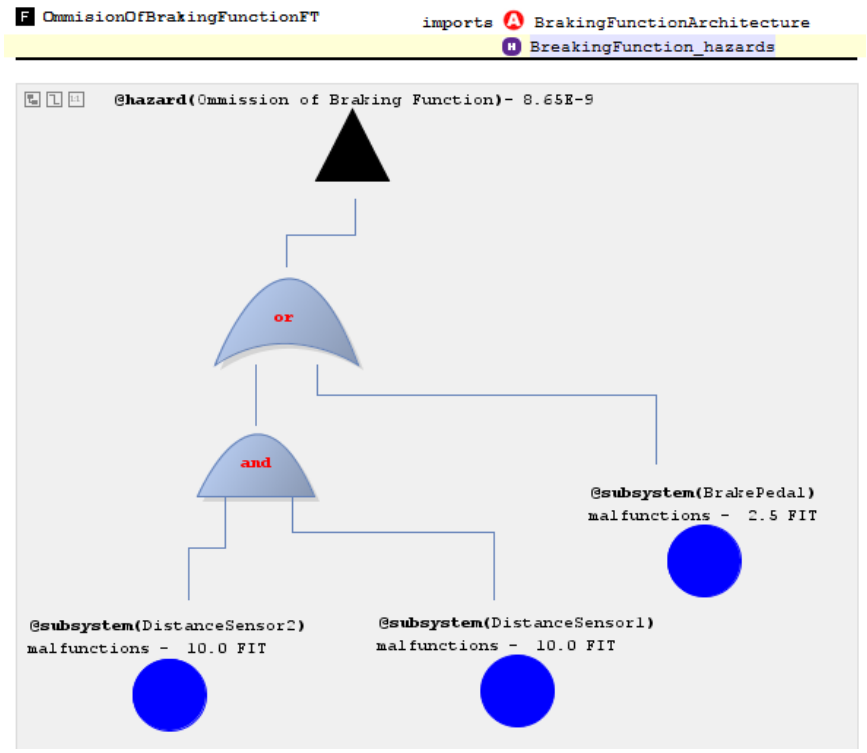


Fig. 6. Example of propagation of faults accross different subsystems. The subsystems are directly referenced as sources for the basic events. The top event is dirrectly linked to one of the hazards.

Safety Case. Goal Structuring Notation (GSN) [7] is a modeling language for capturing safety arguments. GSN has a diagramatic notation; the safety argument is structured by a set of individual elements (claims, evidence and contextual information) as well as the relationships between these elements (i.e. how claims are supported by other claims and by evidences). From the point of view of Safety.Lab, GSN is another modeling language whose elements reference other models from the modeling environment. A GSN represents a view over the system under development which summarizes information already present in the product model. This opens the possibility to define advanced consistency checks for the entire model since the structure of the GSN should be consistent with the structure of the product models whose elements are referenced from within the GSN.

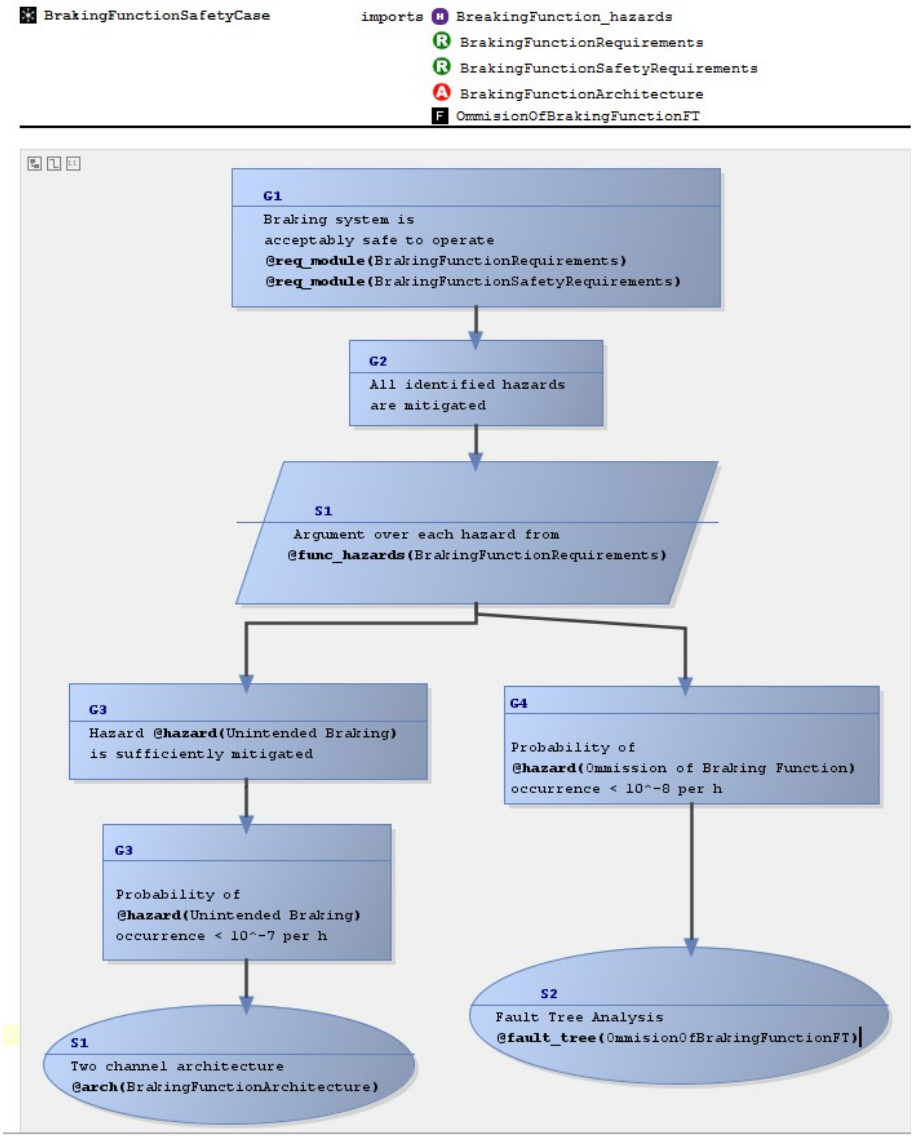


Fig. 7. Each element of the GSN dirrectly references elements from the system model. Thereby, navigating between argumentation and the system model is directly supported. Furthermore, when the system evolves, we can increase the automation of consistency checks that the argumentation is consistent with the implemented changes.

In Fig. 7 we present an example of a safety argumentation fragment for the braking function by using a GSN diagram. This safety case is constructed by the safety engineer manually. Safety.Lab supports deep linking of all needed artefacts which are referenced by the safety argumentation. Based on the fact that the GSN directly references the high-level functional requirement and hazards, we

can define consistency checks such as the fact that all identified hazards have been eliminated or that the fault tree indeed leads to a low enough probability for a hazard.

4 Related Work

Today, there is a number of tools in research and industry which provide an integration of model-based system design and safety analysis, such as QuantUM[8], medini analyze [9] or PREEvison[10]. However, none of the existing commercial tools offer models for structuring the artefacts and link them with assurance case models.

Sophia is a conceptual model and tooling, implemented as UML profiles, for integrating safety analyses with model based systems engineering [13]. Sophia has substantial overlapping with our work when integrating safety with system-level models. The *Eclipse Safety Framework (ESF)* aims at providing a set of tools based on the Eclipse tooling platform *PolarSys* for integrating safety techniques within a model-driven engineering process based on the modeling standards SysML and MARTE. AutoFOCUS3 (AF3) is a model based development tool which provides models for all phases of the system development from requirements to the low-level design. AF3 integrates the GSN notation and allows its users to link elements of the GSN with parts of models [15]. Due to the use of deep models, Safety.Lab has many similarities with Sophia, ESF and AF3. We aim at a deeper integration of fine granular hazards list, fault trees with requirements and architecture and at defining automatic consistency checks between the system model and the safety case.

Various large research project aim at delivering a tooling environment for the model-based development of safety-critical systems. For instance, the CHESS environment the SafeCer tools framework [12], the OPENCOSS platform [11]. Since all these tools are results of research projects only parts of the developed concepts are actually implemented.

A different approach for the model-based development of assurance cases is presented in [5]. In this approach, a weaving model is used, which allows integration between assurance case, design and process models, in order to automatically generate assurance cases. However, a tool to support this approach is not presented yet. Furthermore, Safety.Lab is focused on deep integration of artefacts as basis for advanced consistency checks. Assurance cases are manually created and linked to other artefacts and not automatically generated.

5 Conclusion and Future Work

Our long term goal is to get a holistic and deeply integrated product model that allows mechanized reasoning about safety qualities of software intensive systems. We use models to describe the system across several abstraction layers and to model the safety aspects of the system. In our vision, the safety arguments models will put together fine granular model elements from the system. In this way,

the consistency and completeness of safety cases can be checked automatically by using the information from within development models.

In the future, we plan to work along three directions. Firstly, to extend Safety.Lab in the direction of mbeddr and link safety arguments with code modules or tests. Secondly, we plan to evaluate our tooling with real-world projects from the Siemens business units. Thirdly, we plan to investigate the use of richer models (e.g. state machines, contracts) in order to capture the semantics of requirements and of other artefacts.

References

1. Voelter, M., Ratiu, D., Kolb, B., Schaetz B., mbeddr: Instantiating a Language Workbench in the Embedded Systems Domain, Journal of Automated Software Engineering, 2013
2. Langari, Z., Maibaum, T. Safety Cases: A Review of Challenges, in International Workshop on Assurance Cases for Software-intensive Systems, 2013
3. Fowler, M. Language Workbenches: The Killer-App for Domain Specific Languages?, 2005
4. Voelter, M. Language and IDE Modularization and Composition with MPS, in Generative and Transformational Techniques in Software Engineering IV, 2013
5. Kelly, T., Hawkins, R.D., Habli, I., Kolovos, D., Paige, R.F., Weaving an Assurance Case from Design: A Model-Based Approach., in 16th IEEE International Symposium on High Assurance Systems Engineering, 2015
6. Vesely, W.E., Goldberg, F.F., Roberts, N.H., Haasl, D.F., Fault Tree Handbook, US Nuclear Regulatory Commission, 1981
7. Kelly, T., Weaver, R., The goal structuring notationa safety argument notation, in Workshop on Assurance Cases Dependable Systems and Networks, 2004
8. Beer, A., Kühne, U., Leitner-Fischer, F., Leue, S., Prem, R., Analysis of an Airport Surveillance Radar using the QuantUM approach, Technical Report, University of Konstanz, 2012
9. Kath, O., Schreiner, R., Favaro, J., Safety, security, and software reuse: A model-based approach, in 4th International Workshop in Software Reuse and Safety, 2012
10. Adler, N., Hillenbrand, M., Mueller-Glaser, K.D., Metzker, E., Reichmann, C., Graphically notated fault modeling and safety analysis in the context of electric and electronic architecture development and functional safety, in 23rd IEEE International Symposium on Rapid System Prototyping (RSP), 2012
11. OPENCOSS Consortium, Deliverable D3.3, Integrated OPENCOSS platform, 2015
12. SafeCer Consortium, Deliverables D3.1.3, CTF Platform Prototype, 2012
13. Cancila, D., Terrier, F., Belmonte, F., Dubois, H., et. al., SOPHIA: a Modeling Language for Model-Based Safety Engineering, in 2nd Int.l Workshop On Model Based Architecting and Construction Of Embedded Systems: ACES-MB, 2009
14. Voelter, M., Tomassetti, F., Requirements as First Class Citizens, in Dagstuhl Workshop on Model-based Development of Embedded Systems, 2013
15. Voss, S., and Carlan, C., Schaetz, B., Kelly, T., Safety Case Driven Model-Based Systems Construction, in 2nd International Workshop on Emerging Ideas and Trends in Engineering of Cyber-Physical Systems, 2013
16. Panesar-Walawege, R.K. and Sabetzadeh, M. and Briand, L. and Coq, T., Characterizing the Chain of Evidence for Software Safety Cases: A Conceptual Model Based on the IEC 61508 Standard, in Third International Conference on Software Testing, Verification and Validation, 2010