Software and Systems Modeling (2020) 19:889–910
https://doi.org/10.1007/s10270-020-00782-w

SPECIAL SECTION PAPER

Model-based safety assessment with SysML and component fault trees: application and lessons learned

Peter Munk[1] · **Arne Nordmann**[1]

Received: 19 July 2019 / Revised: 30 October 2019 / Accepted: 7 February 2020 / Published online: 26 February 2020

Abstract
Mastering the complexity of safety assurance for modern, software-intensive systems is challenging in several domains, such as automotive, robotics, and avionics. Model-based safety analysis techniques show promising results to handle this challenge by automating the generation of required artifacts for an assurance case. In this work, we adapt prominent approaches and propose to augment of SysML models with component fault trees (CFTs) to support the fault tree analysis and the failure mode and effects analysis. While most existing approaches based on CFTs are only targeting the system topology, e. g., UML class diagrams, we propose an integration of CFTs with SysML internal block diagrams as well as SysML activity diagrams. We realized our approach in a prototypical tool. We conclude with best practices and lessons learned that emerged from our case studies with an electronic power steering system and a boost recuperation system.

Keywords Model-based systems engineering · MBSE · Model-based safety analysis · MBSA · Fault trees · Fault tree analysis · FTA · Component fault tree · CFT · Failure mode and effects analysis · FMEA · Safety · Reliability · Dependability

1 Introduction

The complexity of modern, software-intensive systems continues to increase due to their rising number of features and functionalities. This trend exists in several domains, including automotive, robotics, and avionics. Model-based systems engineering (MBSE) methods are suitable to tackle the complexity of such systems, since they support engineers with different, yet consistent perspectives on a holistic model of the system and provide different layers of abstraction.

Systems in the mentioned domains are often safety-critical since their malfunction might lead to damages or even loss of live. Hence, these systems have to be developed according to mandatory safety standards such as the IEC 61508 and the ISO 26262 in the automotive domain. These standards require safety analysis methods such as fault tree analysis (FTA) or failure mode and effects analysis (FMEA), which inherently include a model of the system as well.

However, as of today, these safety analysis methods are typically not linked with MBSE methods and artifacts. Thus, without strict sequential processes, there is a great risk of inconsistency between the evolving system models and their safety analyses.

Several existing approaches combine MBSE with safety analyses methods, e. g., Hierarchically Performed Hazard Origin and Propagation Studies (HiP-HOPS) [33], modularized component fault trees (CFTs) [22,23], and the Safe Component Model (SCM) [11]. Extensive overviews of these model-based safety analysis (MBSA) methods are given by Aizpurua and Muxika [2], Sharvia et al. [39], and Lisagor et al. [25]. The main benefits of these MBSA approaches are that the implications of changes in the system model on the system's safety are directly visible and that they enable the divide and conquer paradigm, since the error propagation of each component is specified separately. Höfig et al. [15] present two case studies that show why these benefits are especially useful for complex industrial use cases.

MBSA methods were introduced for various metamodels, e. g., hyper-graphs [13], MathWorks' MATLAB/Simulink [3,34], Architecture Analysis & Design Language (AADL) [29], EAST-ADL [35,45], and the universal modeling language

Communicated by Richard Paige, Andrzej Wasowski, and Oystein Haugen.

✉ Peter Munk
peter.munk@de.bosch.com

Arne Nordmann
arne.nordmann@de.bosch.com

[1] Corporate Sector Research and Advance Engineering, Robert Bosch GmbH, 71272 Renningen, Germany

(UML) [1]. However, in our industrial experience, MBSE methods mainly use SysML [32] as metamodel. While a recent activity of the Object Management Group (OMG) to extend the SysML by a safety profile [7] is certainly beneficial, to the best of our knowledge this profile is not yet applied in the industry. Clegg et al. [9] use a similar profile to extend SysML with the ability to model fault trees and link basic event and failure modes to SysML blocks. The link between elements from safety analysis and system architecture improves a common understanding of the system and allows validation checks, especially on updates. However, the approach of Clegg et al. does not leverage the system architecture and the error propagation information contained within it.

In [30], we present our MBSA approach that extends SysML with CFTs, we show how it supports safety experts when performing FTA and design FMEA, and we outline our design optimization approach with respect to safety. In [24], we extended our MBSA approach to support Hazard and Operability (HAZOP) studies, too. In [31], we presented our lessons learned from MBSA with SysML and CFTs. We showed how we extend SysML with CFTs. Our approach is not limited to the static system topology, e. g., represented in SysML internal block diagrams (IBDs), but also supports the architectural aspects of SysML activity diagrams by combining CFTs with call operations. Please note that in contrast to existing MBSA approaches, we do not propose a profile that extends the SysML, but leverage JetBrains' Meta Programming System (MPS) to combine the languages of SysML and CFT in a prototypical tool. We discuss our conceptual considerations based on a set of design drivers and the suggestions of Kaiser et al. [23].

In this paper, we significantly extend our work [31] by (1) combining it with our work [30] to support extending the methodology not only to support FTA but also FMEA and by (2) providing a detailed case study with an electronic power steering unit that provides valuable insights of applying MBSA with industrial models.

The remainder of this paper is structured as follows: Sect. 2 discusses the state of the art in MBSA; Sect. 3 presents our adaptations to the state of the art, which include a proposal for integrating CFTs with SysML internal block diagrams and activity diagrams. Section 4 shows the realization of our approach integrated in a prototypical tool based on the language workbench MPS. Section 5 presents two industrial case studies in two Bosch automotive products before Sect. 6 discusses the lessons learned and remaining challenges.

2 State of the art

Existing MBSA approaches, e. g., HiP-HOPS [33], CFTs [22,23], or the SCM [11], all depend on a model of the system architecture with components that are interconnected by ports and connections. Several metamodels and languages, e. g., MATLAB/Simulink, AADL, EAST-ADL, UML, and SysML, support such kind of system architecture models.

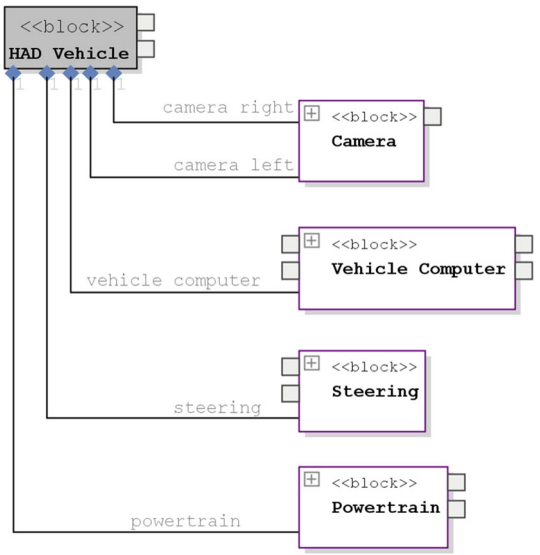


Fig. 1 Simple SysML block definition diagram (BDD) of the "**HAD vehicle**" block with four internal blocks. Please note that this figure and most of the following figures are screenshots from our tool, which is described in Sect. 4

We focus on SysML since it is the most widely used systems (not software) modeling language in the automotive domain according to our industrial experience.

In SysML, a block definition diagram (BDD) defines features of blocks and relationships between blocks such as associations, generalizations, and dependencies [32]. Figure 1 depicts a simple example of an highly automated driving (HAD) vehicle as a BDD. The SysML IBD captures the internal structure of a block in terms of properties and connectors between properties. A block can include properties to specify its values, parts, and references to other blocks [32]. The IBD of our simple HAD vehicle example is shown in Fig. 2. It specifies two instances of the camera block, "**camera left**" and "**camera right**". The "**wheel movement**" port located on the top-left corner of the block "**powertrain**" is the **proxy port** "**longitudinal movement**" of the "**HAD vehicle**".

According to the MBSA approaches introduced in Sect. 1, the potential combinations of internal faults and incoming errors from the input ports are defined and their propagation to outgoing errors at the output ports is specified for each block. This block-internal error propagation can be modeled either by an interface focused-FMEA (IF-FMEA) in a tabular manner as done in HiP-HOPS [33], by a set of analytical expressions as proposed by Yakymets et al. [46], or by CFTs [22,23]. CFTs use a graphical notation based on Boolean algebra similar to the notation of classical fault

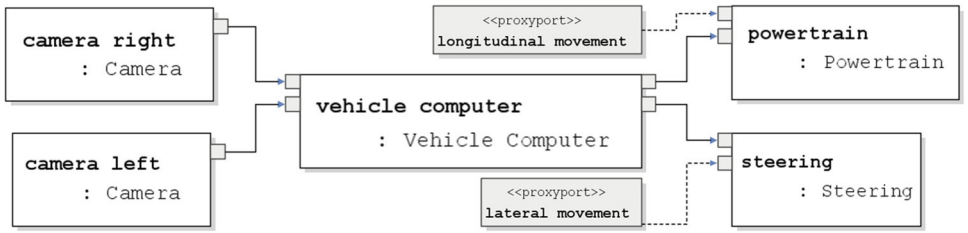


Fig. 2 Simple SysML internal block diagram (IBD) of the "**HAD vehicle**" block with five connected parts

trees. As we detailed in our previous work [30], we propose a deductive method to model the error propagation, i. e., we start modeling the block-internal error propagation of those blocks where errors at the output ports can directly violate safety goals. Since this deductive approach fits better to CFTs than to IF-FMEAs, we select CFTs to model the block-internal error propagation. Furthermore, we consider the graphical notation of CFTs easier to read than the tabular manner of IF-FMEAs.

An exemplary CFT for the "**vehicle computer**" block is shown in Fig. 3. The incoming errors "**blurry image left**" and "**blurry image right**" at the **ports** "**camera image left**" and "**camera image right**" are ANDed. The resulting intermediate error "**blurry image**" is ORed with the internal fault "**internal fault**" and results in the outgoing errors "**too high torque**" and "**too low torque**" at the **port** "**torque**". The "**internal fault**" directly leads to the outgoing error "**wrong angle**" at the "**angle**" port. Note that the probability "**p**" of the "**internal fault**" is chosen arbitrarily and it might not possible to determine such probability for CFTs of functional architectures.

With the connections specified in the IBD and the error propagation defined in the CFTs of the blocks, the fault tree of each error of the system can automatically be generated by following the connections between the parts and the block-internal error propagation information. The composability property of CFTs is formally proven [12]. In the following, we refer to this generated fault tree of the entire system as *system fault tree*. Note that the system fault tree can be generated for each output event of interest. Such system fault trees can be mandatory artifacts in the safety case for the respective system. They can be analyzed by well-established FTA algorithms, e. g., based on binary decision diagrams (BDDs) [10], and the respective certified software tools. Note that interpretation of the FTA results, i. e., the minimum cut sets and the probabilities of top events, is still to be done by safety experts. A cut set consists of basic events that in combination trigger the corresponding top event; a cut set is minimal if none of its real subsets triggers the top event.

Choley et al. [8] propose to generate the FTA and FMEA by analyzing the functional and component-based topologies of the system enclosed in activity diagrams and internal block diagrams without any additional information. Mhenni

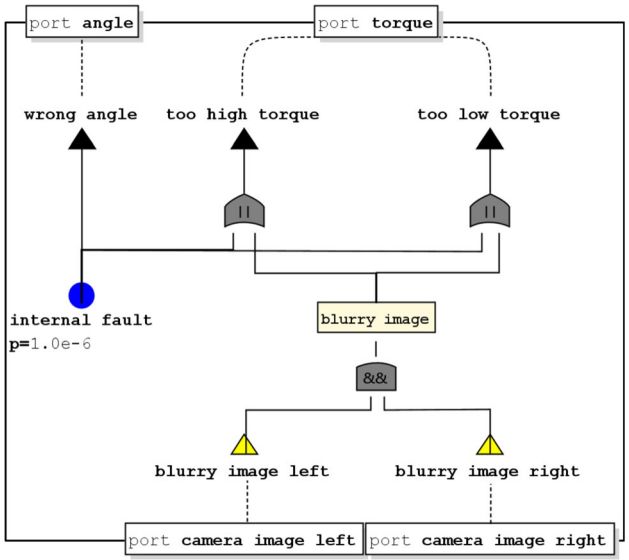


Fig. 3 Exemplary component fault tree of the "**vehicle computer**" block. Incoming errors are shown as yellow triangles, AND (&&) and OR (||) gates are shown in gray with respective text, internal faults are shown as blue circles, and outgoing errors are shown as black triangles

et al. [27] automatically generate the system fault tree by traversing SysML structural diagrams and identifying block design patterns. In a later work, Mhenni et al. [26] propose SysML extensions that allow to specify different failure modes of individual blocks. Helle [14] proposes a SysML profile to allow modeling the reliability of individual blocks and presents a software tool that evaluates the resulting reliability block diagram. Alshboul and Petriu [4] annotate transitions to component failures in SysML state machines together with the architectural information in SysML BDDs and IBDs to automatically generate fault trees of the system. In contrast to these approaches, our approach extends the blocks of the SysML model with individual error propagation information. This allows safety experts to model complex block-internal error propagation with AND or OR relationships of errors, hence encoding their knowledge into the MBSE process.

In our previous work [30], we also showed how the CFTs can support the failure mode and effects analysis (FMEA). The FTA is a deductive analysis and FMEA is an inductive method, so instead of finding the faults that lead to a failure,

the FMEA lists all faults (causes) and for each fault (cause) finds the failure (effects) the fault leads to. Safety norms such as the ISO 26262 [18] recommend both, inductive and deductive analysis methods, since both methods complement each other. As mentioned above, a minimum cut set describes the relationship between the faults (causes) and the failures (effects). Some authors [27,33,44] use this relationship to generate an FMEA table from the extended system model as well. However, similar to leaving the interpretation and analysis of the minimal cut sets of the system fault tree to the safety experts, we argue that instead of generating the resulting artifact of an FMEA, i. e., FMEA table, it is more beneficial for the safety experts to be supported in the process of creating the FMEA. By supporting the FMEA process, it is also possible to keep the inductive nature of the FMEA and complement it with the deductive way CFTs are created. In the automotive domain, the FMEA processes typically follow the VDA [40] and the IEC [17] specification and have the following five steps [40]:

1. Structure Analysis: the system is split up into the *structure tree*, which contains the individual structural elements (SE) hierarchically arranged for the description of the structural connections.
2. Function Analysis: each SE is analyzed with regard to its *functions* in the system. The interaction of the functions of several SEs are described by a function network.
3. Failure Analysis: possible *failures* are deduced from the functions of each SE. Failures are linked according to cause and effect in failure networks.
4. Actions Analysis: existing mitigation actions are assigned to each failure and the risk connected to each cause of failure is evaluated.
5. Optimization: if the result of the previous step is not satisfactory, new actions are identified and the risks are reassessed.

The FMEA process is an established and widely accepted method. The insights generated during the FMEA with its system-wide view might extend the information contained in the component-based MBSA approaches. Hence, in order to convince the FMEA experts in the industry of the benefits of MBSA concepts, we argue that it is important to support their work by automatically generating the structure tree, the function network and the failure network from the system model instead of generating only the resulting FMEA table.

Kaiser et al. [23] reflect the last 10 years of research on CFTs and describe future ideas toward "CFT 2.0," explicitly suggesting the combination of CFTs with SysML. Kaiser et al. suggest the following set of rules:

(K1) "for each component in the SysML architecture, there is exactly one corresponding CFT with the same name"

(K2) "the nesting hierarchy in the SysML IBD and the CFT is the same"

(K3) "cause-consequence edges in the CFT exist wherever and only where corresponding signal or service flows exist in the SysML IBD architecture"

(K4) "for each incoming flow port, at least one failure input port shall be assigned in the CFT (the same applies for outgoing flow ports and output ports)"

(K5) "for each required or provided service port (lollipop symbol), several failure input and output ports can be assigned, since failure consequences can be passed from service provider to requester and vice versa. Safety engineers are responsible for avoiding cycles in the causal chains."

(K6) "all CFT failure ports are bound to one architectural port; graphically represented either by drawing the rectangular architectural port around its triangular failure ports, or by drawing edges between them."

In the following, we will show how we included the suggestions of Kaiser et al. [23] in our approach.

3 Approach

While some of the related work discussed in Sect. 2 has been evaluated in industrial use cases already [15], we still need to adapt the presented approaches to our industrial setting. In the following, we discuss the drivers that impact our metamodel, domain-specific languages and tool choice, most of them discussed by Amarnath et al. [5]:

(D1) For a valid assurance case, safety argumentation needs to be consistent with the actual system at all times. This is a huge effort when done manually that we expect to become virtually impossible as complexity of software-intensive systems such as highly automated driving vehicles continues to increase. Therefore, a main driver of our approach is to provide immediate feedback about the safety properties of the system while designing and modeling the system. This requires a tight coupling and traceability between the system model and the safety artifacts of the assurance case so that changes in the architecture are immediately reflected in the safety analysis [11,23].

(D2) Since safety is a system property and tightly interlinked with further technical and architectural aspects, e. g., software and hardware concerns, non-functional aspects such as resource demands, timing, security, etc., we need to be extensible to model a system with all these aspects and be extensible for further aspects that need to be integrated. To be able to make a consistent assurance case covering all these aspects, we

target a *single-source-of-truth* approach where there is no—potentially lossy—conversion between different tools only being able to assess isolated aspects.

(D3) We conducted an extensive survey of available metamodels, languages, and tools that cover the aspects to perform model-based safety assessment on automotive systems, i. e., specification of architectures, error behavior, and safety argumentation. While one of the surveyed languages, EAST-ADL [45], covered all required aspects, it is not a feasible choice in our industrial setting. Most Bosch business units and Bosch customers use SysML to model automotive system architectures, therefore our approach needs to support SysML as base language. Part of the popularity of SysML is due to the large support by commercial tools such as Rational Rhapsody and Enterprise Architect, as well as the amount of modeling experts familiar with SysML.

(D4) While the base language for our Bosch use cases is SysML, we see quite different SysML *flavors* that we need to support: modeling style and conventions, different SysML profiles, and the actually used SysML modeling tools differ a lot between Bosch business units and Bosch customers. In order to support cross-divisional system engineering and safety assessment, we need to *assess* those different flavors and be able to integrate them into our approach. As system models of some of the supported industrial use cases heavily rely on the modeling of SysML activities, we need to provide the approach that related work bases on system topology, e. g., SysML IBDs, also on models of dynamic architecture models, i. e., SysML activity diagrams.

(D5) Due to strict resource constraints in the automotive domain, the deployment of functions to (embedded) control units (ECUs) or, at a lower level of abstraction, the technical mapping of software components to hardware blocks is a multi-objective optimization problem. One of the objectives is the dependability of the resulting system. Hence, it is vital to support an efficient modeling of deployment information and analysis of the resulting dependability.

(D6) System design and safety assessment involves different stakeholders, that come with a different set of competencies and—being domain experts—different sets of familiar tools and notations. While the use of domain-specific languages already allows to provide an efficient way of expressing certain concerns, different stakeholders expect different—textual or graphical—notations and visualizations of the same concerns, as detailed in Sect. 4.1.

In the following, we describe our design decisions and the adoptions to the state of the art in model-based safety analysis while complying to **(K1)**–**(K6)** and **(D1)**–**(D6)**.

3.1 Language modularization

Our language modularization was mainly driven by **(D2)** and **(D3)**. A complex, software-intensive system usually does not have one single notation that fits all relevant aspects, especially since different aspects may be specified by different roles at different times. Often, it is desirable to define separate DSLs for each viewpoint or provide different notations for different viewpoints [41].

Figure 4 shows our main language modules, i. e., the language modules that enable the model-based safety assessment of the targeted highly automated driving architectures. In conformance with **(D3)**, the main language is SysML. However, we introduce three *domain-specific* language modules that extend SysML for the more explicit description of the functional/logical architecture, the software architecture, and the hardware architecture. All these languages conform to the SysML metamodel, but allow to enforce additional constraints, e. g., that only software blocks, i. e., SysML blocks from the more specific *software* language, can be deployed to hardware blocks, i. e., SysML blocks from the more specific *hardware* language. This is beneficial as efficient handling of deployment was a major driver of our approach **(D5)** and has to be explicitly handled during system fault tree construction, see Sect. 3.2.

One of the language modules that we integrate with SysML languages is the *CFT* language that allows efficient expression of component fault trees. The CFT language references the SysML base language, see Fig. 4 (left), to be able to map `events` (input and output errors) of the CFT language to `ports` from the SysML language. This allows the *models* of the CFT language, the component fault trees, to reference *models* of the SysML language, e. g., SysML blocks and parts, see Fig. 4 (right). This direct (unidirectional) tie between the failure model and the system model is an important feature to satisfy **(D1)**and fulfills the suggestions **(K1)** and **(K2)**.

3.2 SysML internal block diagrams and CFTs

In order to keep the system model and the safety artifact—the system fault tree—consistent at all times **(D1)**, it is necessary to link the elements of the system model to the CFTs, as these are eventually the building blocks of the system fault tree. When specifying the topology of a system in an SysML internal block diagram (IBD), the main building blocks of the system are SysML `parts`, so these need to be linked to the CFTs.

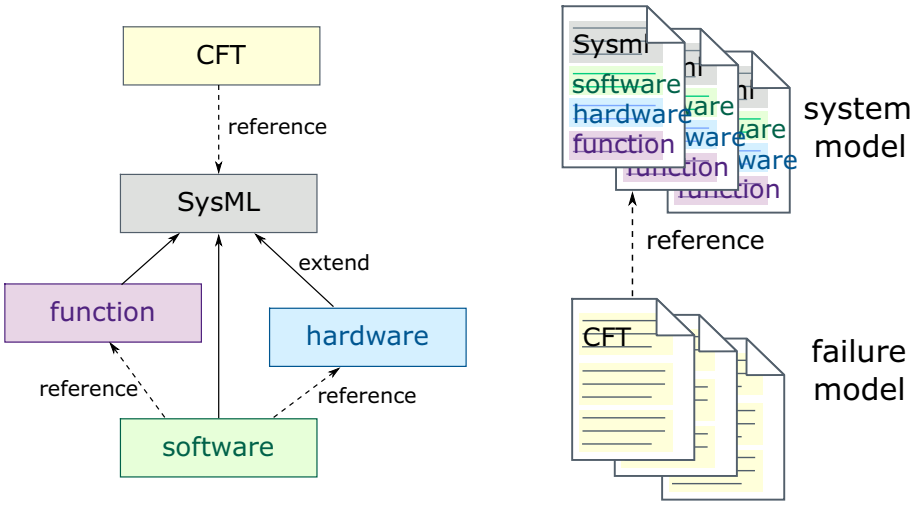


Fig. 4 Language modules and their dependencies

3.2.1 CFTs for blocks

While we need CFTs for every SysML *part* in order to build a system fault tree from an IBD, we attach CFTs to all SysML *blocks* instead. **`Blocks`** specify the *type* of SysML **`parts`**, they therefore correspond to UML classes. Every SysML **`part`** is *typed* by a **`block`**, so that the corresponding CFT for a part is known and can be used for construction of the system fault tree from all **`parts`** of an IBD.

In the CFT language, we differentiate between input errors, intermediate errors, and output errors. Input and output errors have to be connected to a port of the respective block. This way we implement the suggestions **(K4)** and **(K6)** of Kaiser et al. [23]. Intermediate errors can be used to denote the result of a logical combination, i. e., the result of an AND or OR gate. For example, in Fig. 3, the result of the input error "**`blurry image left`**" AND the input error "**`blurry image right`**" is the intermediate error "**`blurry image`**".

Since SysML **`blocks`** may consist of **`parts`**, it is possible to express hierarchy in the model. The SysML concept of **`proxy ports`** allows to express relation between parent **`ports`** and **`ports`** of its **`parts`**. In the example shown as IBD in Fig. 2, the **`ports`** "**`longitudinal movement`**" and "**`lateral movement`**" of the **`block`** "**`HAD vehicle`**" are proxies for ports of the "**`powertrain`**" and "**`steering`**" **`blocks`**. In general, there are two ways to model the error propagation of a **`block`** that contains **`parts`**: First, use the **`proxy ports`** to descend and ascend the hierarchy and create CFTs for each **`parts`** or respective **`block`**. Second, model the error propagation only at the higher level of hierarchy, neglecting the internal propagation of a **`block`**'s **`parts`**. Kaiser et al. [23] request that in the second case, the error propagation of a **`block`** has to follow the **`connections`** of its **`parts`** **(K3)**. So far, we do not enforce this suggestion in our approach, instead we generally prefer the former way of modeling, i. e., explicitly modeling CFTs down to the finest component granularity.

3.2.2 System fault tree construction

As introduced in Sect. 2, the system fault tree is constructed from an IBD and determined by (i) the CFTs of all parts of the IBD and (ii) the connectors between all parts of the IBD. Figure 5 shows the construction of the system fault tree from an IBD in a simple example:

For every SysML **`part`** in the IBD (Fig. 5, upper left), the corresponding SysML **`block`** is found and the corresponding CFT (Fig. 5, lower left) is *instantiated* in the system fault tree (Fig. 5, right). Notice that the example IBD contains two "**`camera`**" **`parts`** of the same **`block`** type, so that the corresponding CFT of the "**`camera`**" is instantiated twice in the system fault tree. Connecting the CFT instances in the system fault tree is done in accordance with the connections in the IBD, leading to a fully connected system fault tree. The connection between the CFTs propagates output events from the output ports to the input events of the input ports of connected parts.

Note that a complete FTA for a system modeled in SysML requires to construct fault trees for each of the *top events* of the system. As the fault tree construction and a succeeding FTA can be automated as detailed in Sect. 4.2, fast and early feedback can be provided to system architects and safety managers after each system change **(D1)**.

3.2.3 FMEA support

The SysML expresses a system by **`blocks`** that contain **`parts`** of other **`blocks`**. Based on this information the FMEA structure tree can be automatically generated. For typical systems, it is likely that there are more hierarchical

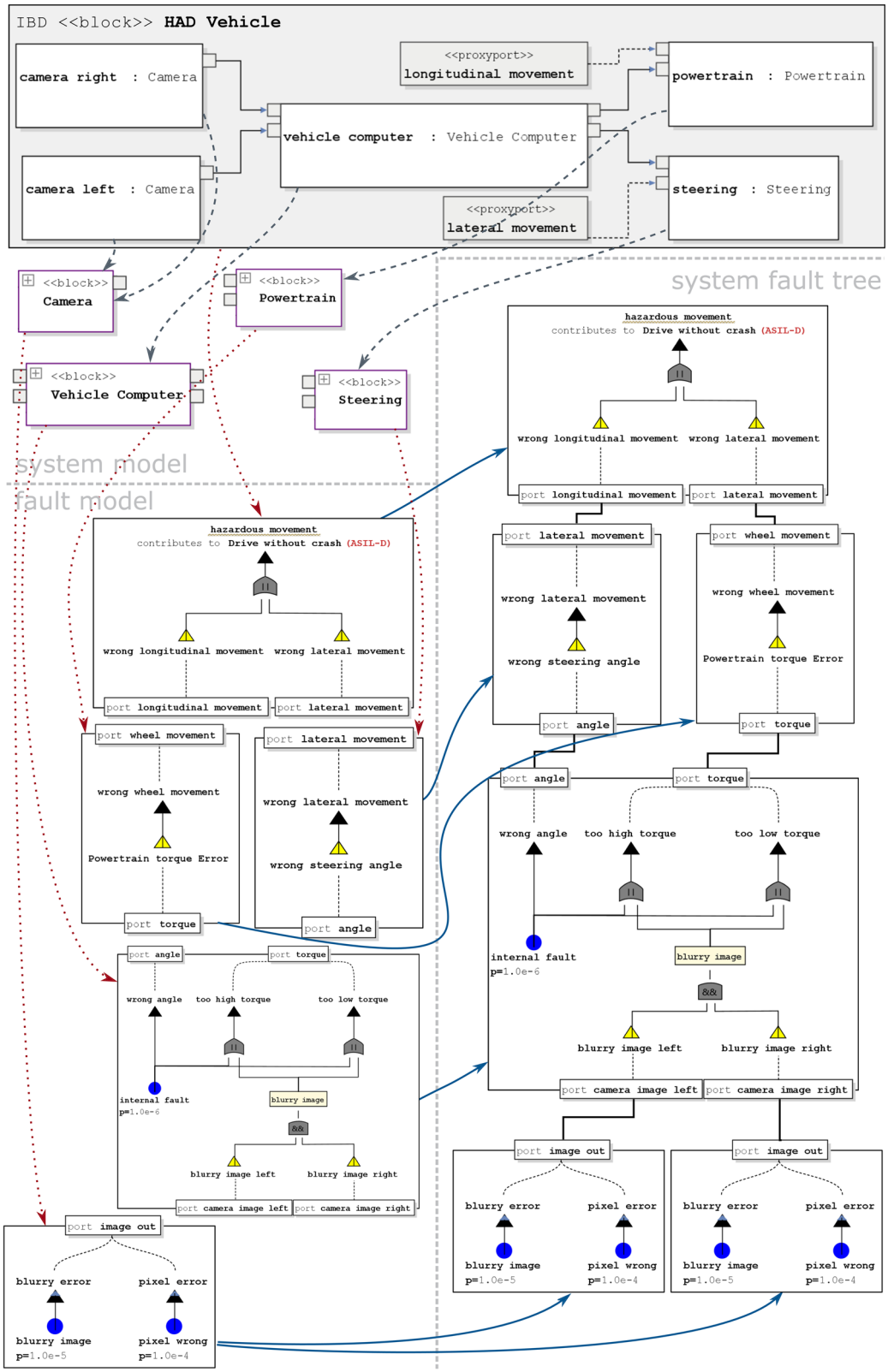


Fig. 5 Construction (solid blue arrows) of the system fault tree (lower right) for the HAD vehicle output error. It is based on the IBD (top), their corresponding **`blocks`** (linked by gray dashed arrows), and the CFTs of all **`parts`** (linked by red dotted arrows). Connectivity of the CFTs in the fault tree is determined by the **`connectors`** in the IBD (color figure online)

layers in the functional architecture than the typical number of layers (3–4) in one FMEA structure tree. In this case, the generated structure tree is automatically split after a specific hierarchical depth, which results in one top level FMEA and structure tree that is linked to multiple FMEAs and structure trees of subcomponents.

We assume that requirements are available and annotated to the **blocks** of the model, e. g., using SysML's «satisfy» relationship. Furthermore, we assume that the requirements linked to a **blocks** represent the functions in the sense of an FMEA that this **blocks** implements. We are aware that this does not hold for all requirements in general. However, we argue that some higher-level requirements or a composition of requirements linked to a component of the functional architecture give a good indication of the (FMEA) functions implemented by the component. Thus, the generated function network is still beneficial compared to a complete manual creation. If these two assumptions hold, the FMEA function network can be automatically generated from the requirements. The connections between the function of the function network stem from the connections between the requirements, e. g., SysML's «deriveReq» or containment relationships.

The CFTs specify all output events that potentially propagate through the **ports** of **block**. Based on this information and the previously generated structure tree, the FMEA failure networks can be generated automatically.

The FMEA failure network typically only connects failures that have a cause and effect relationship across different hierarchical layers (vertical error propagation). However, the components of the functional architecture are often connected to components inside the same hierarchical layer (horizontal error propagation). Therefore, while automatically generating the FMEA failure network, our approach follows the connections between the output ports and input ports of the components and generates a relation only if the respective components are at a different hierarchical level. For example, see the failure network in Fig. 9 which has been automatically generated from the SysML model shown in previous figures.

It is important to note that in contrast to AND gates in fault trees and CFTs, the FMEA failure network typically considers only single failures and not their combined occurrence. However, we conservatively generate relations between failures that are connected with one or multiple intermediary AND gates, since there might be common causes not contained in the functional architecture and the CFTs.

In general, the FMEA can also be used to detect cause–effect relationships that do not follow the effect chain and communication paths between components, e. g., unwanted side effects such as noise or vibration. These relationships are most likely not captured in the CFTs and are therefore not present in the generated failure network. Hence, the automatically generated structure tree, function network, and failure network have to be reviewed and checked for correctness and completeness by safety experts and function owners. Insights and findings of this review process are used to improve the FMEA as well as to enhance the CFTs and the SysML model itself. Furthermore, the safety experts and function owners have to specify manually which failures belong to which functions, since this information is not available from the CFTs nor the functional architecture. In order to retain this information in the failure networks generated in future, the specified failure and function relations are then added to the CFTs by linking requirements to outgoing errors.

3.2.4 Error type system

In the example shown in Fig. 5, the CFT of the "**camera** **block** shows that two different output errors can occur at the same **port**. This **port** is connected to the block "**vehicle computer**", where the respective CFT only specifies *one* input error to the respective **port**, though. In general, we support such kind of models. When constructing the system fault tree, we generate an OR gate before the input event that pessimistically assumes any of the connected output events is propagated. The same effect is produced when an input has multiple in-going connections from multiple output ports, each with associated output events. It is especially useful for the *deployment port*, which is defined in the next section, since it always captures all output events independent of which hardware block the software component is deployed to.

However, in the example shown in Fig. 5 it might make sense to differentiate between the two different output errors (blurry image or wrong pixel) in the CFT of the "**vehicle computer**", e. g., when one of the errors can be mitigated while the other leads to an error. To support this, we employ the hierarchical error type system proposed by Möhrle et al. [28]. The root of the error hierarchy is a generic error, refined by provision and content errors. Provision errors are omission and commission errors while content errors are **under-performance** (*too late*), **over-performance** (*too early*), and **non-code performance** (*wrong values*). Below this level of refinement, we allow additional definition of domain-specific subtypes, e. g., "**blurry image**" as **non-code performance** subtype.

The error type system allows to connect the matching output events and input events even if multiple ports are connected to one port. Furthermore, it is possible to model output error types at a higher refinement level than the input events and still OR the ones that belong to the same super-type.

In case the output error types do not match the input error types, it is possible to detect and notify the designer about inconsistencies in the model.

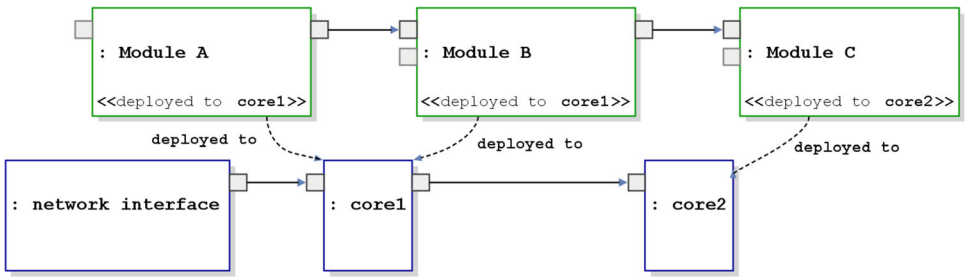


Fig. 6 IBD with three software components deployed to two different hardware blocks

3.2.5 Deployment ports

The previous sections illustrate system fault tree construction in a fairly simple functional architecture. This, however, neglects additional complexity that is introduced with the technical realization, i. e., implementation of the functions in software and hardware as well as the mapping of software components to hardware computation units. This is, however, a major source of complexity in automotive and therefore an important aspect to be supported by our approach **(D5)**.

Deployment in SysML is often modeled in terms of *allocation*, for example: a software component might be *allocated to* a hardware computation unit; a function may be *allocated to* a software component that realizes the function. This is an issue for the construction of the system fault tree as shown in Fig. 6 as there is no direct connection between the **`ports`** of the corresponding hardware and software **`parts`** and therefore no clear notion of error propagation between them.

Taking the example of *deployment*, i. e., allocation of a software component to a hardware component, hardware errors need to be distinguished that propagate *horizontally* along regular SysML connectors between the hardware components or propagate *vertically* to the software component. While a bit flip in a hardware component might propagate to the allocated software component, a hardware communication error might propagate to the connected hardware component. An example model is shown in Fig. 6. If a fault in the CFT of the "**`network interface`**" is propagated horizontally to "**`core 1`**" but not to "**`core 2`**" and if it is filtered by the software components "**`Module A`**" and "**`Module B`**", it does not affect "**`Module C`**".

Höfig et al. [16] assume in their Architecture Layer FailuRE Dependency (ALFRED) methodology that in this case all failure modes of the hardware component propagate to the software component. While this is a *safe* pessimistic assumption, it neglects the possibility to (a) model that a hardware fault only propagates *horizontally* and (b) to explicitly model the effects of a hardware fault on a software component, e. g., the fault is *benign*, i. e., having no effect.

To circumvent these problems, we propose explicit **`deployment ports`** that are automatically instantiated in a deployed software component, i. e., a **`block`** of our software language, and in the corresponding hardware component, i. e., a **`block`** of our hardware language. By default all failure modes of the hardware component are propagated to the **`deployment port`** of the deployed software component, similar to Höfig et al. [16]. However, the **`deployment ports`** are also available in the CFT of the software and hardware component, allowing to handle failure modes explicitly.

Figure 6 shows an example of three software components (green) "**`Module A`**", "**`B`**", and "**`C`**" that are deployed to two hardware blocks (blue) "**`core 1`**" and "**`core 2`**". The software components are connected with ports and connections. The hardware blocks are connected, too. A third hardware block "**`network interface`**" is connected only to "**`core 1`**". If the deployment of software component "**`Module C`**" is changed to hardware block "**`core 1`**", a fault in the "**`network interface`**" that propagates through "**`core 1`**" would affect it as well. This effect can be analyzed without having to change the CFT of any of the blocks, since faults are propagated through the deployment ports. At the same time, if a software component is benign to certain faults of hardware blocks, it can be explicitly modeled in the CFT of the software block.

3.3 SysML activity diagrams and CFTs

To the best of our knowledge, generation of fault trees based on SysML [14,26,27] and UML [1] is based on the static system topology as represented in UML class diagrams or SysML IBDs. However, in automotive and other domains, SysML activity diagrams and alike get necessary as systems become more complex and dynamic architectural aspects become more vital. Since safety is a system property, all system aspects need to be covered [23]. Hence, one of our drivers is to use the information modeled in SysML activity diagrams and use them for automated generation of system fault trees as well **(D4)**. We therefore propose a scheme similar but orthogonal to the system fault tree generation from IBDs as detailed in Sect. 3.2.

3.3.1 CFTs for operations

SysML activity diagrams represent the control flow and data flow of step-wise actions. System models that we evaluated mainly used **`call behavior actions`** and **`call`**

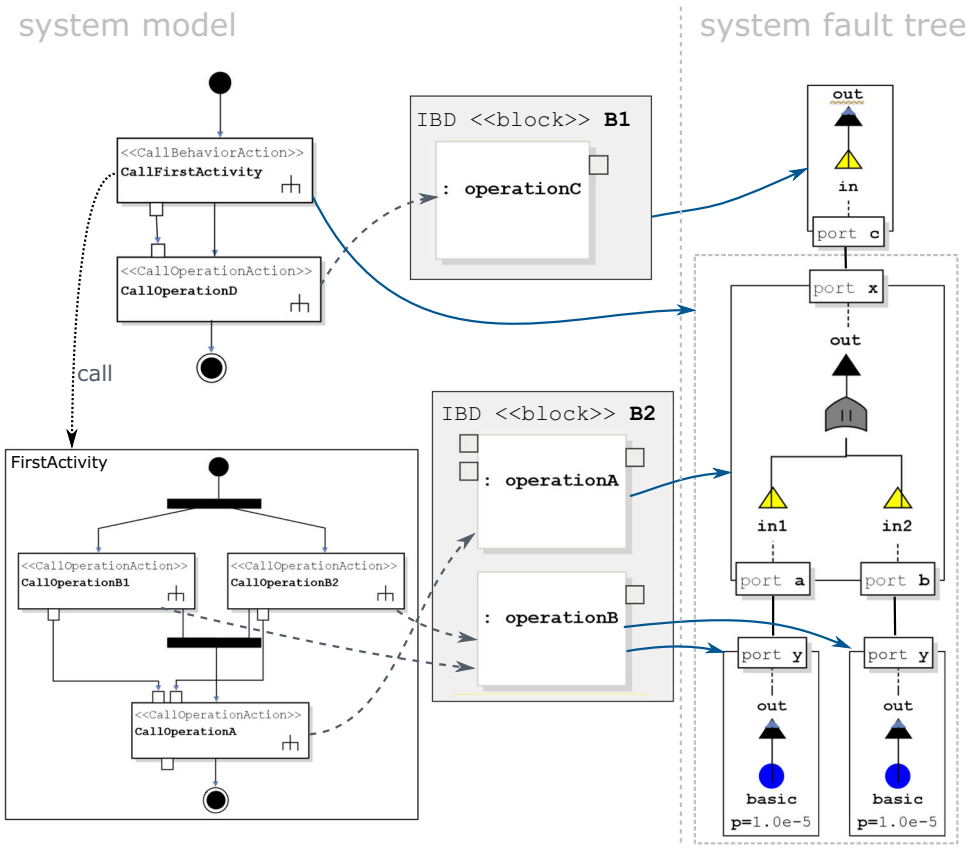


Fig. 7 Creation of a system fault tree from an activity diagram, **`blocks`**, **`operations`**, and their respective CFTs (analogous to Fig. 5, repetition of CFTs omitted for the sake of brevity)

`operation actions`, as well as the flow between them. **`Call operation actions`** directly call **`operations`** of SysML **`blocks`**; **`call behavior actions`** call other **`activities`**. While **`call behavior actions`** are primarily a means of reuse, **`call operation actions`** are where computation is done, thus, where errors can occur.

Instead of attaching CFTs to **`call operation actions`**, though, we attach CFTs to the **`operations`** itself to prevent duplication of CFTs, since arbitrarily many **`call operation actions`** can call the same **`operation`**. This is consistent with the scheme introduced in Sect. 3.2 as the **`operation`** is the *type* that determines the error propagation.

3.3.2 System fault tree construction

Figure 7 shows how a system fault tree is constructed for a main activity (upper left). The top event in this case is associated to the pin that leads to the **`final state`** of the main activity. For each **`call operation action`**, the corresponding **`operation`** and therefore the respective CFT can be found and instantiated in the system fault tree. Each **`call behavior action`** is followed to the corresponding **`activity`** where the same scheme is applied. **`Edges`** between actions, i. e., control flow and data flow, determine how the instantiated CFTs are connected to get the full system fault tree as shown in Fig. 7.

Currently, we do not harvest the dynamics that are expressed in SysML activity diagrams, i. e., order or sequence, but translate activity diagrams to (inherently static) fault trees. A natural next step would be to generate safety artifacts that are able to consider the dynamics, e. g., *dynamic fault trees* [19,38] as discussed in Sect. 6. Furthermore, we leave the FMEA support for activity diagrams as future work.

3.3.3 Object flow versus object pins

The above scheme to generate a fault tree relies on the assumption that activity flow is modeled with SysML's **`object pins`**, as these can be associated with the corresponding **`arguments`** of the **`operations`** and therefore the respective input and output errors in the CFTs.

However, SysML also allows to model flows between **`activities`** instead of **`pins`**, which we currently cannot use for generation of the system fault tree.

Additionally, general-purpose SysML modeling tools often do not enforce consistency between **`parameters`** of **`operations`** and **`pins`** in the corresponding **`call operation action`**. Inconsistencies between these might, however, lead to mistakes in the system fault tree generation.

4 Realization

The presented approach was realized in a tool for evaluation in two real-world automotive use cases. This section introduces our tool and discusses the realization of particular features that correspond to our drivers **(D1)**–**(D6)**.

The modeling tool is based on the language workbench *JetBrains Meta Programming System* (MPS)[1] that provides rich features for language design, language modularization, textual and graphical editors, as well as artifact generation.

Usage of language workbenches such as MPS in safety-critical environments have been proposed by Völter et al. [42], who justify the use of DSLs in order to introduce rigor, consistency, and traceability into development processes.

Particularly, the strong language modularization support provides the necessary means to allow integration of all relevant aspects into one language family and allow modeling these aspects in a single-source-of-truth model **(D2)**. Only a very limited number of approaches we found in the literature are model-based or leverage the benefits of a DSL.

Projectional editing in MPS allows tailoring of our languages and editors to the needs of our stakeholders, e. g., different textual and graphical editors **(D6)**. The general-purpose language Java as base language in MPS allows us to integrate with the domain tools that our stakeholders are used to or required to use for creation of assurance cases **(D6)**.

4.1 SysML

A complex, software-intensive system usually does not have one single notation that fits all relevant aspects. Particularly, since different aspects may be specified by different roles at different times. Often you want to define separate DSLs for each viewpoint, or provide different notations for different viewpoints [41].

We implemented the subset of the SysML 1.4 metamodel that is relevant for our stakeholders and use cases (see Sect. 5) in MPS' meta-metamodel, i. e., essentially **`blocks`** [32, Ch. 8], **`ports`** and **`flows`** [32, Ch. 8], **`activities`** [32, Ch. 11], **`allocations`** [32, Ch. 15], and **`requirements`** [32, Ch. 16]. MPS provides features that allow to provide different DSLs for different aspects of the system. The *projectional editing* feature of MPS allows to provide different concrete notations by providing different *projections* of the underlying abstract syntax tree. Since all projections directly manipulate the same abstract syntax tree, no conversion needs to be done between the different notations as it would be necessary in parser-based approaches [43].

For certain viewpoints, a graphical notation is better suited, especially when dealing with relationships between system entities or some kind of data flow as often specified with SysML. In these cases, graphical notations make it easier to derive a mental model of a system structure [36].

Textual models, on the other hand, integrate more easily with source code management and build infrastructures, e. g., merging of models.

We provide one textual DSL and two graphical DSLs for editing of **`blocks`**, **`ports`**, and **`connectors`**, corresponding to the SysML block definition diagram and internal block diagram. Component fault trees can be edited in a textual or graphical DSL as well, see Fig. 8. For editing of activities, we provide one textual and one graphical DSL corresponding to the SysML activity diagram.

From our experience, textual notation is faster and more efficient in the initial creation of a system, while graphical notation is better suited for understanding and maintaining a system. Additionally, different stakeholders prefer a textual or a graphical notation. In our use case, software developers tend to prefer textual notation for its efficiency, while safety experts tend to prefer graphical notation that is closer to the notation they are used from domain-typical tools, e. g., Isograph's Reliability Workbench[2]. For our approach, we want to provide a graphical and a textual notation for the component fault trees language. While the graphical notation is more accustomed to the safety expert reviewing it, the textual notation is closer to the component developer, in the future potentially providing the component fault tree together with the component.

An additional view that proved extremely efficient to discuss about error propagation is an integrated view that shows the graphical CFTs inside the blocks of the SysML IBDs, easing the comprehension of the error propagation through the system.

4.2 Automated analysis

We decided for use existing, potentially certified tools for performing the fault tree analysis, instead of performing it on our own. Advantages are the use of certified tools as well as providing the domain experts the possibility to review the analysis results in the tools of their choice and expertise, potentially integrated in established certification processes.

4.2.1 Integration of third-party tools

In Sects. 3.2.2 and 3.3.2, respectively, we detail how the system fault tree and the FMEA artifacts are generated from the SysML model and the CFTs.

In order to analyze the generated fault tree, we transform it into the OpenPSA[3] format, an XML-based open format for

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

[2] https://www.isograph.com/software/reliability-workbench/.

[3] https://open-psa.github.io/.

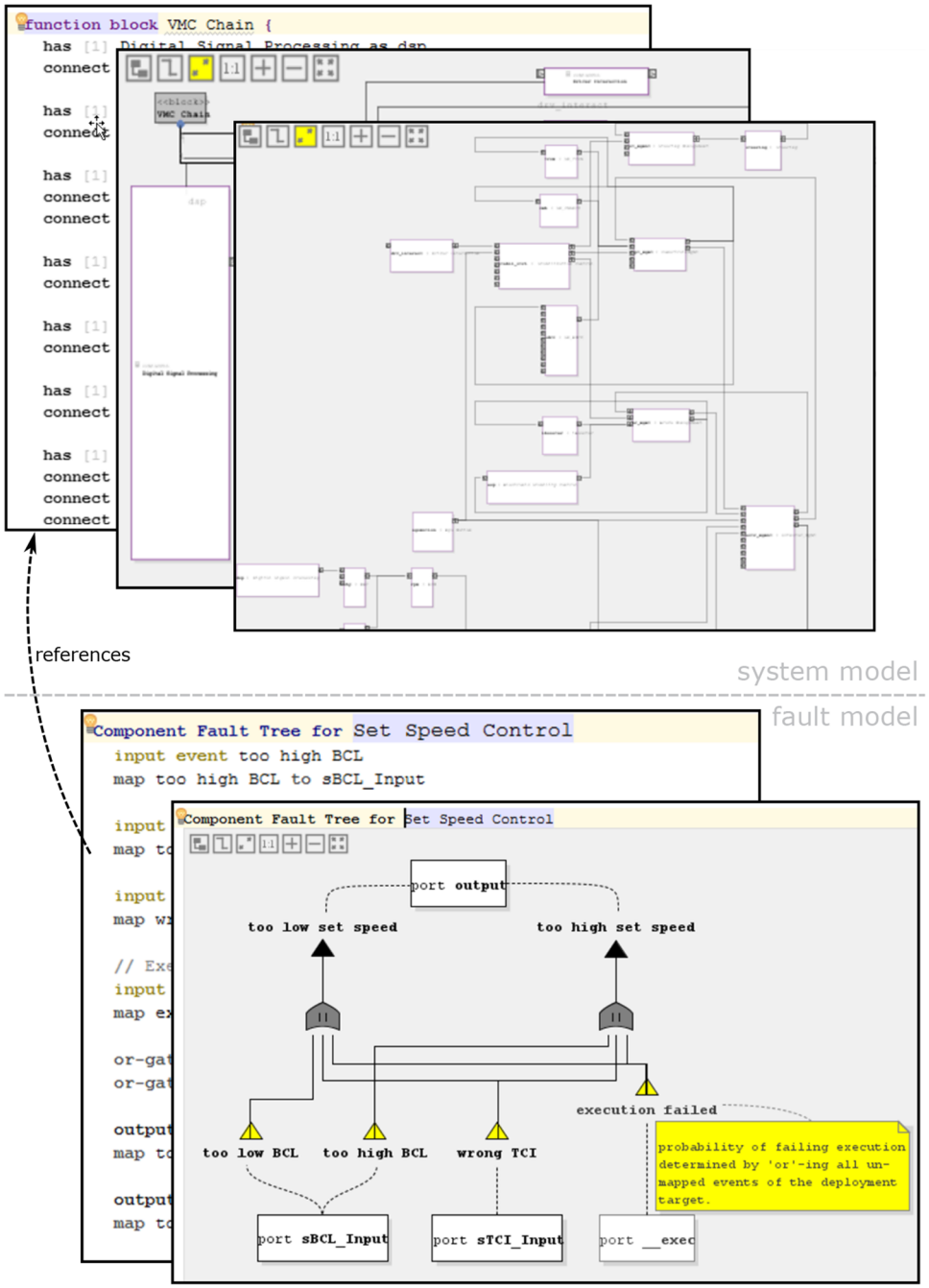


Fig. 8 Graphical and textual notations (projections) of the SysML languages (top) and the CFT languages

fault trees, as well as the respective configuration files for an FTA call to the open-source command-line tool XFTA[4]. After the FTA was performed, we read the generated results file and display the computed reliability value at the respective output error.

The experience with our business units shows that the resulting reliability is often not as important as the minimum cut sets or the generated fault tree itself. Arbre-Analyste[5] is a free graphical viewer for the OpenPSA format that allows to compute the minimum cut sets on the generated system fault tree. However, in order to use the resulting fault tree in a safety case of a product, the respective safety standards demand for qualified software tools. For this reason, we extended our tool to not only generate the system fault tree as OpenPSA files, but additionally transform the fault tree in CSV files that can be imported by Isograph's Reliability Workbench, which is a qualified and established FTA tool.

The automatically generated FMEA structure tree, function network, and failure network are transformed into the

[4] http://altarica-association.org/contents/xfta.html.

[5] http://www.arbre-analyste.fr/en.html.

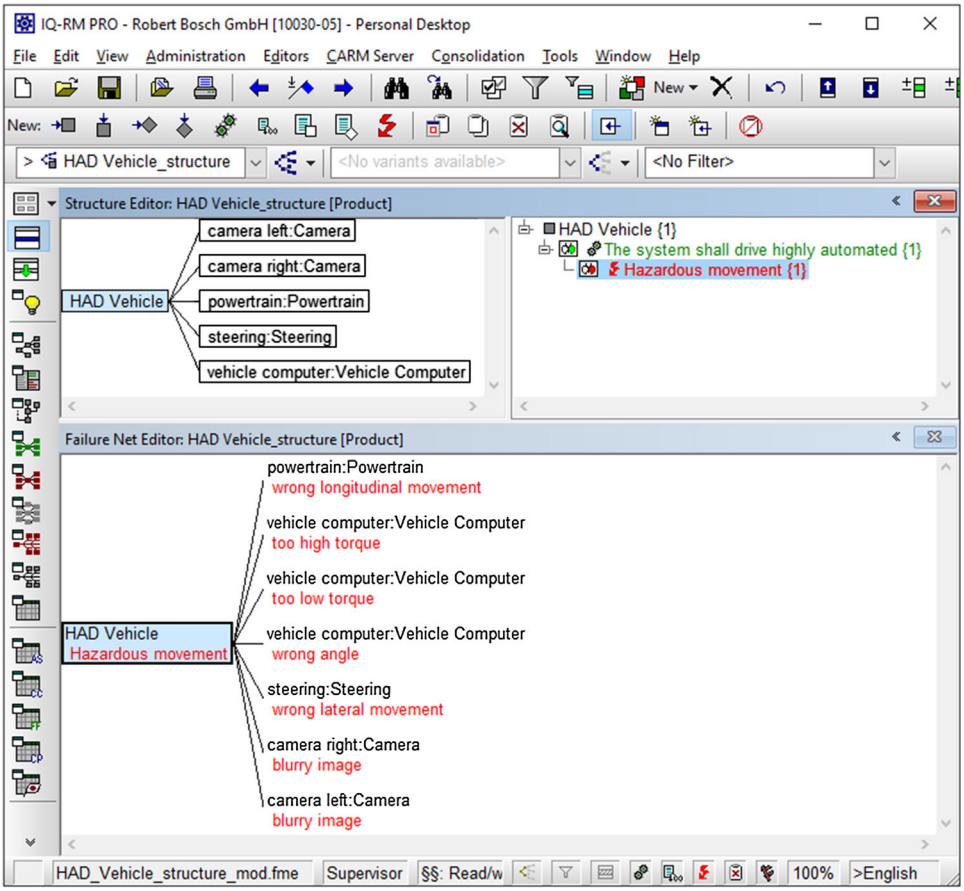


Fig. 9 The resulting FMEA structure tree and failure network generated from the sample system shown in APIS IQ-RM tool. Note that the function network is not shown

XML-based format of APIS IQ-RM, which is a qualified and established FMEA tool. Figure 9 shows a screenshot of the generated FMEA artifacts in IQ-RM.

4.2.2 Lifting analysis results to the model

While the usage of external tools for the analysis has its advantages, we loose the immediate feedback inside our modeling environment, which is one of our main drivers (**D1**). As a countermeasure, we want to lift the analysis results from the domain-specific tools up to the level of our models. In case of the quantitative FTA, this means annotating the error probabilities calculated by the domain-specific FTA tools to the respective errors in the CFTs.

CFTs of **`blocks`** and **`activities`** may be instantiated multiple times in a system fault tree, though, so that we potentially get different probabilities for the **`output`** errors for each instantiation of the **`blocks`** or **`activities`**, respectively. When reading back the results we therefore annotate a *table* of probabilities to the **`errors`** that shows the probability of the **`error`** for each instantiation of the CFT. Figure 10 shows the CFT of the **`operation`** "**`ADCTrigger`**" that is called by four different **`call operation actions`**. Thus, it is instantiated four times in the system fault tree,

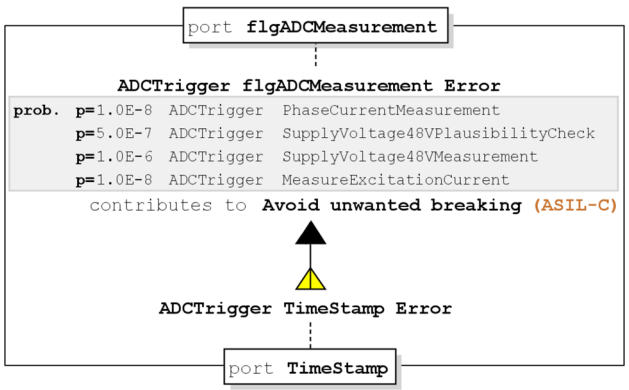


Fig. 10 Quantitative FTA results, lifted to the model level (gray box)

potentially leading to four different probabilities of the "**`flgADCMeasurement`**" **`output error`**. The probabilities table attached to the output shows the probabilities in the context of the four **`call operation actions`**.

4.3 Usability features

We have implemented several features that ease the creation and handling of SysML models and CFTs.

4.3.1 Import and export

Most Bosch business units use either IBM's Rhapsody[6] or Sparx's Enterprise Architect[7] to model their systems in SysML. In order to not manually regenerate the models in our MPS-based prototypical tool, we created importers that are able to read the exported XML Metadata Interchange (XMI) file from Rhapsody or use the Java API of Enterprise Architect.

Due to the different modeling ways, we found it necessary to adapt and extend these importers for each use case we considered **(D4)**. We also provide exporters to XMI in order to export modifications back to general-purpose SysML modeling tools.

However, while we are able to import a model itself, its graphical aspects and layout information are not yet considered by our importers. As a result, the graphical representation often differs considerably from the representation in Rhapsody or Enterprise Architect. Unless these aspects are covered properly by the importers and exporters as well, including the FTA and FMEA tools in the loop, fluent round-trip engineering is not possible, which hinders adoption by end-users. Additionally, the current version of our graphical representations only shows one layer of hierarchy. We found this to be hindering discussions with the domain experts, since additional time is needed to check and adapt to the novel representation.

As long as these issues remain, we assume changes of the model are done within the general-purpose modeling tool in order to ensure the single-source-of-truth principle as well as the graphics and layouting decisions are kept.

4.3.2 Pessimistic CFT generation

Since the imported SysML models often consist of a large number of blocks without an explicit failure model, we created a CFT generator that adds a pessimistic CFT to each block. The generator assumes that blocks without any input ports contain one basic event that is forwarded to each output port. Blocks with input ports have one OR gate that collects all input errors and propagates the result to each output port. A first analysis of the output events of top level blocks often results in quick insights into the system. It is especially helpful in detecting unconnected input ports or missing connections.

Since this feature allows automatic generation of the error model from a SysML IBD or activity diagram, it allows together with the analysis introduced in Sect. 4.2 a pessimistic system fault tree generation and FTA from a SysML IBD or activity diagram without additional modeling required, cf. [27].

Note, however, that this feature is only intended to accelerate the manual generation of CFTs. It might not be helpful without manual adaptation of the CFTs and must not be used without manual review of the result.

4.3.3 Design optimization patterns

In addition to the analysis of the system, we started developing a library of model transformations that optimize the system design with respect to dependability. We follow existing approaches discussed in Sect. 2 and leverage the possibility of generating fault trees to automatically optimize the cost of the system with respect to the specified safety requirements. In the following, we summarize this design optimization approach that was originally published by Munk et al. [30].

We propose to automatically improve an architecture model such that the probability of the top events remains below a given upper bound while the costs of that system are minimized. As costs we consider not only the monetary costs of the hardware components but also the number of components, the size of required memory, the runtime load, the chip area, etc.; costs are annotated to the components of the architecture model. Note that for functional architecture models the probability of basic events might be hard to define. In this case, the functional architecture can still be automatically improved by removing minimum cut sets with only one basic event.

Our approach uses a catalog of safety-aware design optimization patterns of which each pattern is described as a model transformation on the architecture model. For example, one optimization pattern adds dual modular redundancy (DMR) or triple modular redundancy (TMR), respectively, to the system. Another pattern replaces a component with a more reliable but more expensive, or a less reliable but cheaper variant. The catalog of safety-aware design optimization patterns can then be shared among different projects and used for different products.

Due to the multi-objective nature of the underlying optimization problem, we use metaheuristics such as evolutionary algorithms (EAs) as most suitable to find good architecture models [6]. An EA evaluates the fitness of a set of solutions (represented as a so-called population). For the next generation of that population, only a specific number of the fittest solutions are kept while the rest of the population is replaced by mutations of solutions or crossovers between two solutions. In our case, the fitness of a specific solution is derived from the costs annotated to the architecture model as well as the probability of the top events. Mutations are represented by model transformations while the minimum cut sets give a good indication which components of the architecture

[6] https://www.ibm.com/us-en/marketplace/systems-design-rhapsody.

[7] https://www.sparxsystems.de/.

model are best to transform. So far, we have not implemented the crossover operator.

Since different model transformations affect different cost metrics and can lower the probability of one top event but increase the probability of another top event, their result is a set of Pareto-optimal architecture models. Thus, the presented cost optimization approach can merely indicate a set of potential architectures to a human expert who then manually chooses the final architecture from that set.

5 Case study

We applied the proposed approach on two case studies within Bosch, an electronic power steering (EPS) system for highly automated driving and a boost recuperation system (BRS). Applying the approach to concrete automotive system unearthed practical challenges that needed to be dealt with and design decision to be taken in order to make generation of safety artifacts from SysML and CFTs feasible.

5.1 Electronic power steering (EPS)

The motivation of the first case study is to evaluate the system fault tree generation based on a SysML model of the EPS system and the extensive documentation of a manually created FTA of the EPS system that was performed using Isograph's Reliability Workbench. This FTA was created with knowledge of a previous version of the SysML model, but mainly in exchange with the responsible persons of the individual components. Nevertheless, the abstraction level of the manually created FTA and the SysML model matches.

The SysML model of the EPS system is created in Enterprise Architect and contains 9082 elements with 1530 ports and 8234 connections in 656 packages. Note that the elements are of various types, e. g., textNode, ActionPin, or ActivityParameter, but also contain 786 classes, i. e., blocks, and 2679 objects, i. e., parts. This model was imported into our prototypical tool.

Based on an existing FTA documentation, we identified the relevant components in the imported SysML model and extended them with CFTs. Note that in general, all components in the model need to have a corresponding CFT **(K1)**. However, to evaluate our approach, we considered it sufficient to select only one top event and stopped at a specific level of abstraction, so not all components of the system level are relevant. With the model extended by CFTs, we used our tool to generate the fault tree of the entire system. Since connections in the imported SysML file were considered to be bidirectional, we considered them in both directions when generation the system's fault tree. Finally, we compared the generated fault tree with the manually created fault tree and its documentation.

5.1.1 Findings

During the process described above, several findings in the SysML model itself, the prototypical tool or its metamodel, and the FTA documentation were identified.

Findings in the original SysML model

- We found parts defining own ports and these ports not always matched the ports of their blocks in number and name.
- We found blocks that instantiate themselves as part or aggregation.
- We found connections to a wrong part of the correct block, i. e., the part that was connected is of the right type but the wrongly located instance.
- Some blocks were defined twice, i. e., with the same name and in the same package.
- We found blocks with parts where connections and identically named ports were used instead of proxy ports. A specialization of the same block used proxy ports correctly.
- We found deprecated ports without a name or with strange names, e. g., "??".
- We found parts with names like "x:X," where instead the name "x" and an instantiation relation to the block "X" would be correct.
- While some blocks for software components are part of the model, we had to add the deployment relationship to the hardware block executing them **(D5)**, as presented in Sect. 3.2.5.

In our opinion, many if not all of these findings have to following origins:

- Multiple engineers are working in parallel on the same model, potentially distributed over the world.
- These engineers are not educated sufficiently or do not have the time model soundly.
- There is a lack of appropriate modeling guidelines.
- The model is not checked automatically.

Please note that we used a model that was not reviewed yet and the modeling activities were still in progress. These issues have been reported and solved.

However, we also see these findings as an indication of how important it is to perform these modeling activities not only because it is requested by the development process guidelines, but rather use and leverage the model for various analyses, including fault tree generation, also in early stage of the development to show the benefits of modeling and motivate the developers.

Findings and adjustments in the prototypical tool

- We have to differentiate between unidirectional and bidirectional connectors in the metamodel and in the generation of the fault tree.
- We have to support basic events without a failure model or probability, when the FTA is only performed qualitatively.
- We have to support basic events that are common causes for parts of the same block and for parts with a common ancestor block. For example, a construction fault in the drive side block that is instantiated twice for left and right needs to be indicated as common cause. Furthermore, it has to be possible to mark basic events as a common cause for completely unrelated parts in the model, e. g., due to an event that is below the level of abstraction in the model such as coupling between circuit paths in a system model.
- Besides connections between ports, connections between provided and required interfaces have to be supported **(K4)**.
- Specializing blocks have to inherit CFTs from their generalized block, so CFTs are inherited but can be overwritten.

In our opinion, these findings show the importance to evaluate scientific approaches with industrial use cases. Most of these findings have been implemented in the current version of the prototypical tool already.

Findings in the FTA documentation

While the documentation of the manual FTA includes figures of the SysML model in Enterprise Architect, these screenshots do not match the current version of the model. The reason is that the manual FTA was started a year before we received the SysML model and the SysML model has been further detailed in the meantime.

In our opinion, this finding clearly motivates using CFTs instead of manually trying to keep up with the changes in the system model while performing a manual FTA.

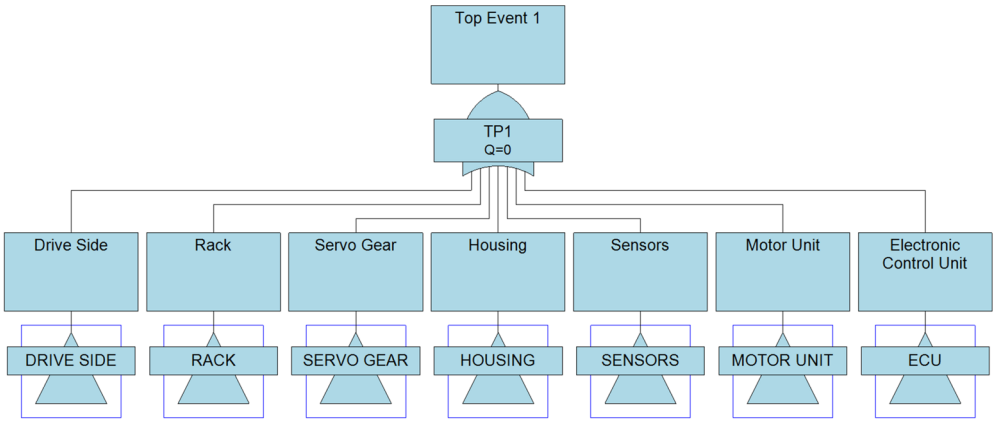
5.1.2 Comparison of manual and generated fault tree

As mentioned at the beginning of this section, we created CFTs only for one top event and stopped at a specific level of abstraction. We compared the respective part of the manually created FTA and the generated fault tree and were able to identify the following issues.

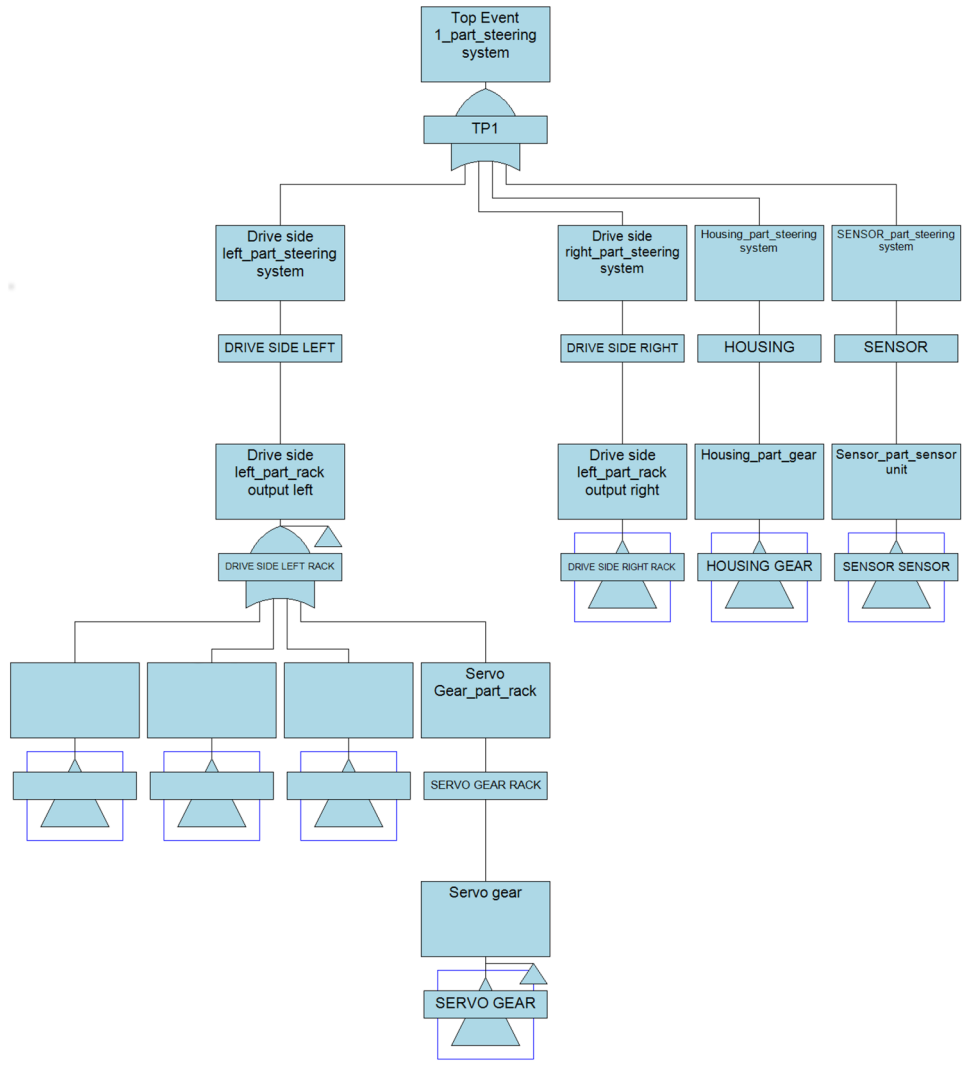
- In the manually created fault tree, the top event is resulting from one OR gate with inputs from all high-level components of the system. In the generated fault tree, the top event is resulting from an hierarchical chain of OR gates. The topmost OR gate adds the errors of the last component in the cause–effect chain and contains another OR gate for the upstream components, as detailed in Fig. 11.
- In the manually created fault tree, only one drive side is considered, as shown in Fig. 11. The FTA documentation contains a careful and reasonable argumentation why this is sufficient. In the generated fault tree, the CFT of the drive side block is instantiated twice for left and right parts automatically.
- The generated fault tree has redundant gates (in Reliability Workbench "tagged gates") that stem from the propagation of errors via separate connections to parts of same block. This is due to different variants being modeled in one system model, so there is a block for the motor unit and parts of this block for different variants of specific motors. So as soon as variant management is included in the system model, this effect should disappear.
- One interesting subtree in the FTA models the problem of an iced gear box in the electric power steering system. As shown in Fig. 12a, the manually created fault tree models this error by an AND gate with three faults connected to it: water entering the gear box, freezing temperatures, and an error in a software component that can detect a freezing gear box before it is iced completely. While the first two faults can be modeled in the CFT of the gear box, the latter error is located in another location. The respective software component is part of a software block, which is executed by the ECU that is connected to the gear box via a motor unit and a servo gear that drives a rack inside the gear box. Hence, the generated fault tree contains the complete propagation of the error in a software component, as shown in Fig. 12b.

Generally speaking, the generated fault tree has more gates than the manually created fault tree. The number of gates could be reduced in the generation process, e. g., by removing transfer gates (OR gates with only one input), e. g., for the freeze detection done in software, or by removing redundant paths. We did not implement this reduction in our generation algorithm since in our opinion; the fault tree is easier to understand and closer related to the system model if it contains more gates.

While the manually created fault tree and the generated tree look different, as detailed above, it is important to note that both trees express the same and their cut set analyses lead to identical results.

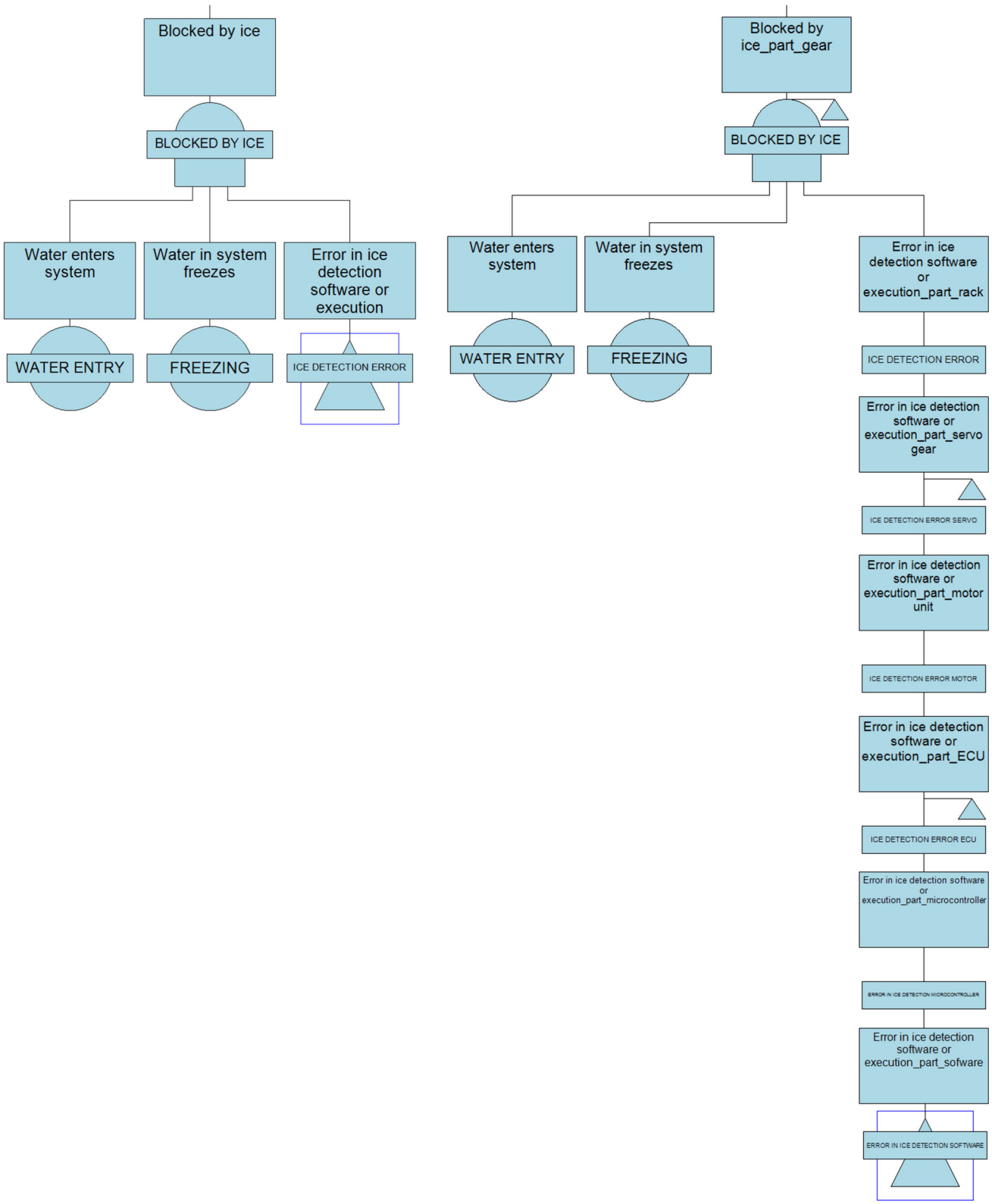


(a) Manually created



(b) Automatically generated

Fig. 11 Comparison of the first level of details of the manually created and automatically generated fault trees for the electronic power steering system. The manually created fault tree consists of one OR gate with all components of the system. The topmost OR gate of the automatically generated fault tree contains only housing, sensors, and both drive sides. The errors of the rack, the servo gear, etc. are modeled with hierarchical OR gates inside the drive sides. Please note that the names were translated, slightly obfuscated or removed for confidentiality reasons



(a) Manually created **(b)** Automatically generated

Fig. 12 Comparison of the subtrees for the error "blocked by ice." The manually created subtree models the software component error directly in the iced component, while the automatically generated fault tree contains the complete propagation path. Please note that the names were translated and slightly obfuscated for confidentiality reasons

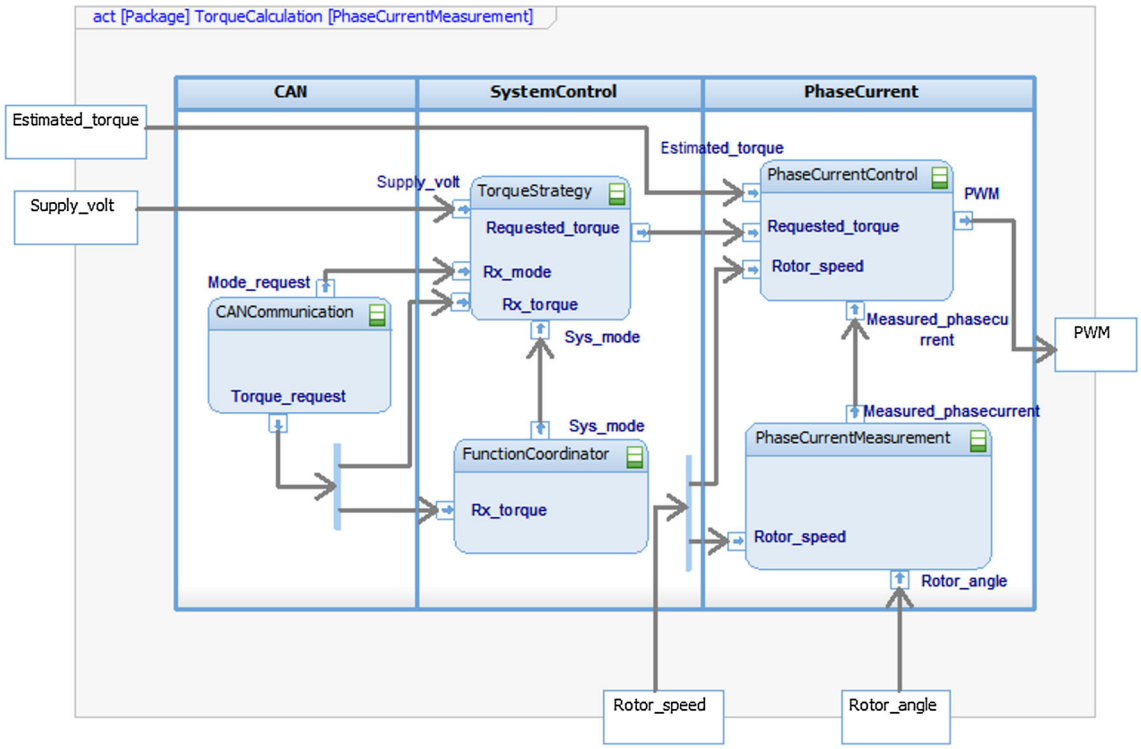


Fig. 13 Activity diagram of the BRS torque calculation, simplified for intellectual property reasons

5.2 Boost recuperation system (BRS)

The motivation for our second case study is to evaluate our approach with another system that is modeled in Rhapsody instead of Enterprise Architect and using mainly activities instead of blocks ((**D4**) and (**D6**)).

The BRS is a 48 V electrical machine integrated with the internal combustion engine of a car using a belt drive. BRS uses recuperation to recover energy for use of vehicle acceleration. The BRS consists of a 48/12 V DC/DC converter, a 48 V lead-acid battery, a DC electric motor, and a combustion engine [37].

The BRS system is modeled in SysML activity diagrams, comprising of 35 main packages, 57 swim lanes, and 123 call operation actions of hardware and software functions. Figure 13 shows a simplified BRS activity diagram, the phase-current measurement, on which we will focus in the following. Within this diagram, the *PhaseCurrentControl* provides the calculated pulse-width modulation (*PWM*) signal that is applied to the DC motor. The *PhaseCurrentControl* depends on the measured phase-current, rotor speed, and angle of the motor.

Unfortunately, the FTA for the second case study was unavailable, so we cannot compare the generated system fault tree to a manually created one. After we imported the model from Rhapsody into our prototypical tool, we extended the imported blocks with CFTs. To do so, we leveraged the pessimistic CFT generation facility before manually refining and adapting the CFTs. With the activity diagram extended by the CFTs, we are able to generate the System Fault Tree for specific output events.

Linking CFTs with SysML **`blocks`** (instead of **`parts`**) proved practical as both together become the unit of reuse, despite the additional effort when reading back the analysis results for each part, as described in Sect. 4.2.2. If a **`block`** changes and the CFT needs to be changed or refined, the change is immediately reflected in the system fault trees and the FMEA of all models that use **`parts`** of this **`block`** type (**D1**). The same is true for linking CFTs with **`operations`** (instead of **`call operation actions`**). The fault tree construction as detailed in Sect. 3.3.2 causes that changes in the CFT of **`operations`** will lead to corrected system fault trees of all activity diagrams with corresponding **`call operation actions`**.

The pessimistic assumption by Höfig et al. [16] that any hardware fault will lead to failing of all software components deployed to this hardware component does not allow to model faults that propagate only to other hardware components. It therefore showed to be too generic for our case studies. Instead, we propose **`deployment ports`** that can be used to specifically model the error propagation from hardware to software as detailed in Sect. 3.2.5.

As mentioned in the previous section, the imported models of both case studies looked different from the original model,

which hinders discussions with the business units. The underlying problem is that the original modeling tools focus on the graphical representation instead of the model itself. For example, placing blocks within other blocks is supposed to represent a containment relationship, which unfortunately is often just a visual effect in graphical modeling tools instead of being explicitly represented in the model itself.

We were able to identify several minor flaws in the imported system model while creating and analyzing the CFTs. For example, the models containing **`input ports`** unconnected to the environment or missing **`connections`** between **`blocks`**. While these issues could easily be fixed when reported to the business units, generation and analysis of the system fault tree often was impossible due to remaining loops in the models. These feedback loops stem from the control theory background of the use cases. We follow the suggestion **(K5)** of Kaiser et al. [23] and require these loops to be manually removed, but support the experts in our prototypical tool by highlighting them.

Another valuable lesson we learned is that automatic generation of a system fault tree is not only a technical challenge but also a process challenge. As we detail in our previous work [30], it is important that a process supports the safety experts when performing FTA and FMEA instead of automatically generating the resulting artifacts. Even if the generated FMEA artifacts are just used as templates for the artifacts generated during a conventional FMEA process, they might still provide good hints and ease the conventional process.

6 Discussion and remaining challenges

Our approach proved to be feasible in supporting the analysis of complex safety-critical systems, e. g., highly automated driving architectures, while several challenges still remain to be solved [5]. Apart from the challenge of compliance with relevant norms, e. g., the ISO 26262, further technical challenges remain that we want to discuss in the following.

Lots of SysML features still remain to be fully exploited for system fault tree generation, e. g., integrating not only flow ports (data-flow-oriented interaction) but also service ports **(K5)**, as this type of interaction (often asynchronous) is typical not only for software systems, but also for open networked systems and systems with user interaction **(D4)**. Also, an obvious further step is to exploit the dynamical architectural aspects covered by SysML activity diagrams, which currently get lost when generating the static system fault tree. Approaches that exploit these dynamics were already proposed, e. g., by Kaiser et al. [21] and Kabir et al. [20], but need to be evaluated for use in an industrial automotive context. To further integrate dynamic aspects into the proposed approach, enriching the fault model regarding timing aspects by techniques such as dynamic fault trees (DFT) [20], combining CFTs with Markov chains [47], or formal safety contracts are on our roadmap as well.

The FMEA support has not yet been extended for models that mainly rely on activity diagrams. Requirements often only provide an indication of the function of a block in the sense of an FMEA. Additional research on the similarities and differences between the nature of requirements and FMEA functions might help to automatically relate these.

While importing general-purpose SysML models from Rhapsody and Enterprise Architect (Sect. 4.3.1) and the automatic generation of pessimistic fault trees (Sect. 4.3.2) already allowed fast application of the proposed approach on two industrial subsystems, legacy systems with missing or insufficient models currently cannot be supported by our tooling. A potential path to circumvent this problem is complementing our approach with model mining approaches that mine system models as well as potentially even fault models from legacy code.

Further challenges include integration with model-based security analysis, which we started in [12], and analysis of the safety of the intended functionality (SOTIF).

7 Conclusion

The approach proposed in this work is, to the best of our knowledge, the first to explore the use of SysML together with component fault trees. We formulate a set of drivers from an industrial context that motivate the design decisions detailed in this work.

Our contribution with this work is twofold: firstly, we propose a scheme to construct system fault trees for an FTA and support the FMEA by generating the structure tree, function network, and failure network from SysML internal block diagrams, which is close to schemes proposed for UML and CFTs. We introduce a deployment port to explicitly model the error propagation between hardware and software. Secondly, we propose a scheme to construct system fault trees from SysML activity diagrams, which is motivated by their usage in our industrial use cases.

We introduce two industrial case studies: (i) a comparison of a manually created fault tree and a system fault generated from a SysML IBD of an electronic power steering system as well as (ii) a qualitative evaluation of the system fault tree generation from SysML activity diagrams of a boost recuperation system. We document lessons learned from applying the proposed schemes in these case studies.

Lastly, we provide an outlook on remaining challenges to further raise the expressiveness of SysML models for use in model-based safety assessment and make use of dynamic aspects, e. g., as modeled in activity diagrams.

Acknowledgements We thank our anonymous reviewers for their helpful and detailed feedback. This work was partially funded within the project SecForCARs by the German Federal Ministry for Education and Research with funding ID 16KIS0792. The responsibility for the content remains with the authors.

References

1. Adler, R., Domis, D., Höfig, K., Kemmann, S., Kuhn, T., Schwinn, J.P., Trapp, M.: Integration of component fault trees into the UML. In: Dingel, J., Solberg, A. (eds.) Models in Software Engineering, pp. 312–327. Springer, New York (2011)
2. Aizpurua, J.I., Muxika, E.: Model-based design of dependable systems: limitations and evolution of analysis and verification approaches. Int. J. Adv. Secur. **6**(1 & 2), 12–31 (2013)
3. Aizpurua, J.I., Muxika, E., Papadopoulos, Y., Chiacchio, F., Manno, G.: Application of the D3H2 methodology for the cost-effective design of dependable systems. Safety **2**(2), 9 (2016)
4. Alshboul, B., Petriu, D.: Automatic derivation of fault tree models from SysML models for safety analysis. J. Softw. Eng. Appl. **11**, 204–222 (2018). https://doi.org/10.4236/jsea.2018.115013
5. Amarnath, R., Munk, P., Thaden, E., Nordmann, A., Burton, S.: Dependability challenges in the model-driven engineering of automotive systems. In: Proceedings of the International Symposium on Software Reliability Engineering Workshops (ISSREW) (2016)
6. Bechikh, S., Datta, R., Gupta, A. (eds.): Recent Advances in Evolutionary Multi-objective Optimization. Wiley, New York (2017)
7. Biggs, G., Juknevicius, T., Armonas, A., Post, K.: Integrating safety and reliability analysis into MBSE: overview of the new proposed OMG standard. In; INCOSE International Symposium, vol. 28, no. 1, pp. 1322–1336 (2018). https://doi.org/10.1002/j.2334-5837.2018.00551.x
8. Choley, J.Y., Mhenni, F., Nguyen, N., Baklouti, A.: Topology-based safety analysis for safety critical CPS. Procedia Comput. Sci. **95**, 32–39 (2016). https://doi.org/10.1016/j.procs.2016.09.290
9. Clegg, K., Li, M., Stamp, D., Grigg, A., McDermid, J.: A SysML profile for fault trees—linking safety models to system design. In: Romanovsky, A., Troubitsyna, E., Bitsch, F. (eds.) Computer Safety, Reliability, and Security, pp. 85–93. Springer, New York (2019)
10. Deng, Y., Wang, H., Guo, B.: BDD algorithms based on modularization for fault tree analysis. Prog. Nucl. Energy **85**, 192–199 (2015)
11. Domis, D., Trapp, M.: Integrating safety analyses and component-based design. In: Proceedings of the 27th International Conference on Computer Safety, Reliability, and Security, pp. 58–71 (2008)
12. Greiner, S., Munk, P., Nordmann, A.: Compositionality of component fault trees. In: Proceedings of 6th International Symposium on Model Based Safety and Assessment (IMBSA) (2019)
13. Grunske, L.: Strukturorientierte Optimierung der Qualitätseigenschaften von softwareintensiven technischen Systemen im Architekturentwurf. Ph.D. thesis, Universität Potsdam (2004)
14. Helle, P.: Automatic SysML-based safety analysis. In: Proceedings of the 5th International Workshop on Model Based Architecting and Construction of Embedded Systems, pp. 19–24 (2012)
15. Höfig, K., Joanni, A., Zeller, M., Montrone, F., Rothfelder, M., Amarnath, R., Munk, P., Nordmann, A.: Model-based reliability and safety: reducing the complexity of safety analyses using component fault trees. In: Proceedings of the Annual Reliability & Maintainability Symposium (RAMS) (2018)
16. Höfig, K., Zeller, M., Heilmann, R.: Alfred: a methodology to enable component fault trees for layered architectures. In: Proceedings of the 41st Euromicro Conference on Software Engineering and Advanced Applications, pp. 167–176 (2015)
17. International Electrotechnical Commission (IEC): IEC 60812: Analysis techniques for system reliability—procedure for failure mode and effects analysis (FMEA) (2006)
18. International Standard Organization (ISO 26262): Road vehicles—functional safety (2018)
19. Junges, S., Guck, D., Katoen, J.P., Stoelinga, M.: Uncovering dynamic fault trees. In: Proceedings of 46th Annual International Conference on Dependable Systems and Networks, pp. 299–310 (2016)
20. Kabir, S., Papadopoulos, Y., Walker, M., Parker, D., Aizpurua, J.I., Lampe, J., Rüde, E.: A model-based extension to HiP-HOPS for dynamic fault propagation studies. In: Proceedings of the International Symposium on Model-Based Safety and Assessment, pp. 163–178 (2017)
21. Kaiser, B., Gramlich, C., Förster, M.: State/event fault trees—a safety analysis model for software-controlled systems. Reliab. Eng. Syst. Saf. **92**(11), 1521–1537 (2007)
22. Kaiser, B., Liggesmeyer, P., Mäckel, O.: A new component concept for fault trees. In: Proceedings of the 8th Australian Workshop on Safety Critical Systems and Software, pp. 37–46 (2003)
23. Kaiser, B., Schneider, D., Adler, R., Domis, D., Möhrle, F., Berres, A., Zeller, M., Höfig, K., Rothfelder, M.: Advances in component fault trees. In: Proceedings of ESREL (2018)
24. Kaleeswaran, A.P., Munk, P., Sarkic, S., Vogel, T., Nordmann, A.: A domain specific language to support HAZOP studies of SysML models. In: Proceedings of 6th International Symposium on Model Based Safety and Assessment (IMBSA) [in press] (2019)
25. Lisagor, O., Kelly, T., Niu, R.: Model-based safety assessment: review of the discipline and its challenges. In: Proceedings of the 9th International Conference on Reliability, Maintainability and Safety, pp. 625–632 (2011)
26. Mhenni, F., Choley, J.Y., Nguyen, N.: SysML extensions for safety-critical mechatronic systems design. In: Proceedings of the International Symposium on Systems Engineering, pp. 242–247 (2015)
27. Mhenni, F., Nguyen, N., Choley, J.Y.: Automatic fault tree generation from SysML system models. In: Proceedings of the International Conference on Advanced Intelligent Mechatronics (2014)
28. Möhrle, F., Zeller, M., Höfig, K., Rothfelder, M., Liggesmeyer, P.: Towards automated design space exploration for safety-critical systems using type-annotated component fault trees. In: Proceedings of the International Symposium on Model-Based Safety and Assessment, Demo Sessions (2017)
29. Mian, Z., Bottaci, L., Papadopoulos, Y., Sharvia, S., Mahmud, N.: Model transformation for multi-objective architecture optimisation of dependable systems. In: Zamojski, W., Sugier, J. (eds.) Dependability Problems of Complex Information Systems, pp. 91–110. Springer, New York (2015)
30. Munk, P., Abele, A., Thaden, E., Nordmann, A., Amarnath, R., Schweizer, M., Burton, S.: INVITED: semi-automatic safety analysis and optimization. In: Proceedings of the Design Automation Conference (DAC) (2018)
31. Nordmann, A., Munk, P.: Lessons learned from model-based safety assessment with SysML and component fault trees. In: Proceedings of the International Conference on Model Driven Engineering Languages and Systems (MODELS), pp. 134–143. (2018). 10.1145/3239372.3239373
32. Object Management Group (OMG): Systems Modeling Language Version 1.4 (2015). http://www.omg.org/spec/SysML/1.4/. Accessed 30 Oct 2019
33. Papadopoulos, Y., McDermid, J.A.: Hierarchically performed hazard origin and propagation studies. In: Felici, M., Kanoun, K. (eds.) Computer Safety, Reliability and Security, pp. 139–152. Springer, New York (1999)

34. Papadopoulos, Y., Walker, M., Parker, D., Rüde, E., Hamann, R., Uhlig, A., Grätz, U., Lien, R.: Engineering failure analysis and design optimisation with HiP-HOPS. Eng. Fail. Anal. **18**(2), 590–608 (2011)
35. Papadopoulos, Y., Walker, M., Reiser, M.O., Weber, M., Chen, D., Törngren, M., Servat, D., Abele, A., Stappert, F., Lonn, H., Berntsson, L., Johansson, R., Tagliabo, F., Torchiaro, S., Sandberg, A.: Automatic allocation of safety integrity levels. In: Proceedings of the 1st Workshop on Critical Automotive Applications: Robustness & Safety (CARS), pp. 7–10 (2010). https://doi.org/10.1145/1772643.1772646
36. Petre, M.: Why looking isn't always seeing: readership skills and graphical programming. Commun. ACM **38**(6), 33–44 (1995)
37. Robert Bosch GmbH: The BRS boost recuperation system: increased power, enhanced comfort and lower fuel consumption. https://www.bosch-presse.de/pressportal/de/media/migrated_download/de/BRS_Broschuere_RZ_en.pdf. Accessed 30 Oct 2019
38. Ruijters, E., Stoelinga, M.: Fault tree analysis: a survey of the state-of-the-art in modeling, analysis and tools. Comput. Sci. Rev. **15–16**, 29–62 (2015)
39. Sharvia, S., Kabir, S., Walker, M., Papadopoulos, Y.: Model-based dependability analysis: State-of-the-art, challenges, and future outlook. In: Mistrik, I., Soley, R., Ali, N., Grundy, J., Tekinerdogan, B. (eds.) Software Quality Assurance, pp. 251–278. Morgan Kaufmann, Burlington (2016)
40. Verband der Automobilindustrie e. V.: Quality management in the automobile industry—quality assurance in the process landscape—general, risk analyses, methods, process models: Volume 4—product- and process-FMEA (2012)
41. Völter, M.: Best practices for DSLs and model-driven development. J. Object Technol. **8**(6), 79–102 (2009)
42. Völter, M., Kolb, B., Birken, K., Tomassetti, F., Alff, P., Wiart, L., Wortmann, A., Nordmann, A.: Using language workbenches and domain-specific languages for safety-critical software development. Softw. Syst. Model **18**, 2507–2530 (2018)
43. Völter, M., Lisson, S.: Supporting diverse notations in MPS' projectional editor. In: Proceedings of GEMOC@MoDELS, pp. 7–16 (2014)
44. Walker, M., Papadopoulos, Y., Parker, D., Lönn, H., Törngren, M., Chen, D., Johannson, R., Sandberg, A.: Semi-automatic FMEA supporting complex systems with combinations and sequences of failures. Int. J. Passeng. Cars Mech. Syst. **2**, 791–802 (2009)
45. Walker, M., Reiser, M.O., Tucci-Piergiovanni, S., Papadopoulos, Y., Lönn, H., Mraidha, C., Parker, D., Chen, D., Servat, D.: Automatic optimisation of system architectures using EAST-ADL. J. Syst. Softw. **86**(10), 2467–2487 (2013)
46. Yakymets, N., Jaber, H., Lanusse, A.: Model-based system engineering for fault tree generation and analysis. In: Proceedings of the 1st International Conference on Model-Driven Engineering and Software Development (2013)
47. Zeller, M., Montrone, F.: Combination of component fault trees and markov chains to analyze complex, software-controlled systems. In: International Conference on System Reliability and Safety (ICSRS) (2018). https://doi.org/10.1109/ICSRS.2018.8688854

Peter Munk received his M.Sc. in Information Technology from the University of Stuttgart in 2013 and his Ph.D. (Dr.-Ing.) degree from the Technische Universität Berlin in 2016, where he investigated software-implemented fault-tolerance mechanisms for real-time applications on multi-core processors. Joining the Robert Bosch GmbH in 2007, Dr. Munk has more than ten years of experience in the automotive domain and the field of embedded real-time systems. He currently works as a research engineer in the Corporate Sector Research and Advance Engineering and studies model-based safety analysis for software-intensive systems. In the context of publicly funded projects, he also works on combining safety and security analysis methods and supervises a Ph.D. candidate on the topic of model-based functional insufficiencies analysis.

Arne Nordmann received his Ph.D. (Dr.-Ing.) in 2015, focused on model-driven engineering methods and domain-specific languages to help design, integrate, and analyze complex, software-intensive systems, such as collaborative robots, industrial robots, and mobile robots. Since joining Bosch Corporate Research in 2015 he applies these methods on architectures for safety-critical, software-intensive systems, such as highly-automated driving and robotics systems. Dr. Nordmann is coordinator of the euRobotics Topic Group on Software Engineering, System Integration, System Engineering in Robotics.