Software and Systems Modeling (2020) 19:1465–1481
https://doi.org/10.1007/s10270-020-00792-8

THEME SECTION PAPER

Hazard-driven realization views for Component Fault Trees

David Santiago Velasco Moncada[1]

Received: 14 June 2019 / Revised: 11 March 2020 / Accepted: 13 March 2020 / Published online: 24 March 2020
© The Author(s) 2020

Abstract
Traditionally, the preferred means of documentation used by safety engineers have been sheets- and text-based solutions. However, in the last decades, the introduction of model-driven engineering in conjunction with Component-Based Design has been influencing the way safety engineers perform their tasks; especially in the area of fault analysis, model-driven approaches have been developed aimed at coupling fault trees with architecture models. Doing this fosters communication between engineers, may reduce design effort, and makes artifacts easier to maintain and reuse. In this paper, we want to move forward in this direction and take another step in the modeling of Component Fault Trees in combination with the modeling of the architecture design. We propose a hazard-centric approach for the definition of multiple realization views for fault analysis using Component Fault Trees. The approach is composed of a modeling method and a tool solution. We illustrate our approach with a real-life example from the automotive industry.

Keywords Model-driven engineering · Component-based · Hazard-centric · Component Fault Trees · Realization view

1 Introduction

The development of safety-relevant functions in the automotive domain is faced with increasing complexity as the result of more intelligent features, more automation, and more functions realized by software. Component-based development has proven to be effective in handling the complexity of such systems by fostering modularity and thereby reuse.

Because of the increasing interaction between the architecture and the safety life cycles, safety engineers have had a hard time maintaining design and analysis artifacts. Traditionally, this has been the case for fault trees [1], as they are not strongly related to architecture design models. In order to tackle this issue, several approaches have been proposed in recent years. With the aim of modularizing fault trees, Component Fault Trees (CFTs) [2] were introduced. In [3], [4], and [5], Component Integrated Fault Trees (C^2FT) extended this modularization concept and presented the means to associate modular UML constructs like SysML blocks with Component Fault Trees. In 2018, the paper "Advances in Component Fault Trees" [6] summarized the most important contributions with respect to CFTs since their definition. Moreover, the authors (including the original contributors of CFT and C^2FT) agreed, for the sake of simplicity, on keeping the original name of the technique (namely CFT), which we will use throughout this article.

The previous studies on CFTs did not investigate how models could be structured in order to facilitate information exchange, maintenance, and review processes.

The most important idea in this regard was to use as reference the architecture models (specifically a logical network) to drive the composition of associated CFTs. With the aim of achieving a better separation of concerns and coping with the complexity of the analysis and review processes, we introduce in this work the idea of using hazards to drive the definition of several realization views for CFTs (one view per hazard). Giving the compositionality of CFTs, this separation of concerns might also occur along the hierarchical composition (i.e., at sub-CFTs) and will be influenced according to the reuse possibilities of those sub-CFTs.

This paper is structured as follows: In Chapter 2, we introduce the relevant model-based design and fault tree concepts, followed in Chapter 3 by a description of the problem. In Chapter 4, we present a solution concept, followed by an

Communicated by Federico Ciccozzi, Antonio Cicchetti and Andreas Wortmann.

✉ David Santiago Velasco Moncada
santiago.velasco@iese.fraunhofer.de
https://blog.iese.fraunhofer.de/author/santiago-velasco/

[1] Embedded systems Quality, Fraunhofer IESE, Kaiserslautern, Germany

example use case in Chapter 5. Finally, Chapters 6 and 7 present related work and our conclusions, respectively.

2 Preliminaries

2.1 Foundations of model-driven engineering

Model-driven engineering is typically associated with automatic generation or transformation of higher-level models into low-level artifacts (e.g., source code generation) [7]. In the context of this work, we focus primarily on the modeling aspect and less on the generation or transformation aspect. Therefore, we will concentrate on the idea that models conform to meta-models, from which modeling languages can be defined [8].

In the world of model-based design, the Unified Modeling Language (UML) is one of the best-known and most commonly used languages within the field of software engineering. Due to its great versatility, UML is considered a General Purpose Language (GPL) that can be used in different fields. However, due to its high level of abstraction, it is not usually practical in real-world engineering projects. For this reason, Domain Specific Languages (DSLs) are used quite often [9]. A DSL is a modeling language tailored to a particular subject area or a certain group of stakeholders. The idea behind a DSL is to create models that help to raise the performance of the stakeholder's tasks. As part of GPLs and DSLs, different viewpoints are often defined to emphasize the modeling of one aspect or another. In the IEEE1471/ISO 42010 standard [10, 11], a distinction is made between the terms *View* and *Viewpoint*; see Fig. 1. Viewpoints relate to the languages (i.e., models, notations, product types) used to define views. Viewpoints are generally defined with the concerns of a specific stakeholder in mind. Viewpoints restrict, in a certain manner, the information displayed through a view, which can be seen as a projection of the system under development.

UML, for example, uses different diagram types to focus on one aspect or another of the system. For instance, class diagrams are used to depict logical structures and component diagrams focus on the composition and sequence diagrams on the behavior. Generally, systems are too complex to show everything in one diagram (i.e., the view). For this reason, it is common to see a functional or logical structuring of the views, leading to a large collection of partial views that, taken together, sum up the system definition.

2.2 Foundations of Fault Tree Analysis

Fault Tree Analysis (FTA) is a technique recommended for use in the development of safety-critical systems [12, 13] to identify faults within a system [1]. To use this technique, the first step is to define a top event (black triangle in Fig. 2), which represents an undesired event (typically a hazard). The system is then analyzed and the combinations of faults (i.e., basic events, blue circles in Fig. 2) that might lead to that event are linked through Boolean logic. Usually, only OR and AND gates are used, but negative logic can also be used by means of NOT or XOR gates. Figure 2 shows the FTA of a simplified airbag system.

Generally, two problems are associated with airbag systems: failure to deploy and unwanted deployment. The former is generally considered less critical because only that part of the injury that could have been prevented by proper deployment, presumably a severe injury, is compensable. The latter case, i.e., undesired activation, is investigated in Fig. 2. This is a dangerous situation because the force generated during deployment can cause an accident, serious injury, or even death (if children are affected).

Component Fault Trees (CFTs) [2] represent the evolution of FTA with respect to the handling of complexity in large systems. By applying component-based principles [14, 15], CFTs integrate modularization and instantiation concepts into the modeling of fault trees. Figure 3 presents the added modeling constructs.

CFTs encapsulate the failure behavior of a component. They depict incoming failures as Input Failure Modes and outgoing failures as Output Failure Modes. This defines the interface of the CFT, which can be exposed by means of CFT Instances when integrated into the CFT of a larger system. Figure 4 presents the CFT of the airbag system.

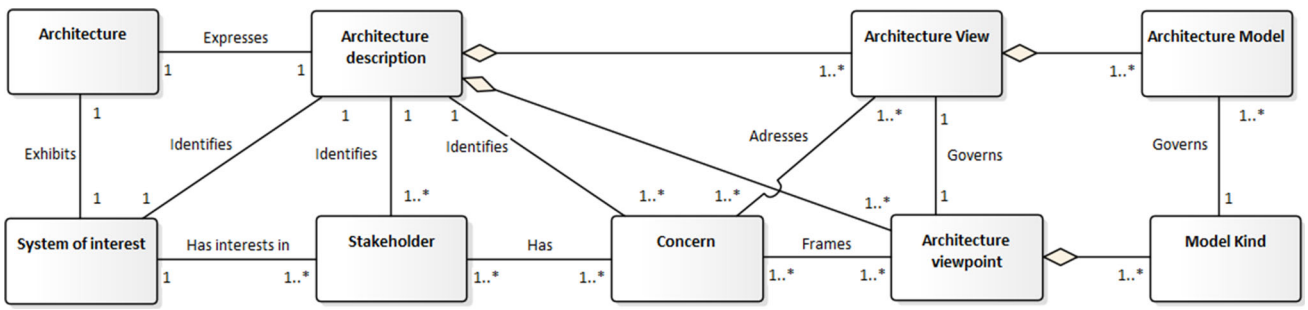


Fig. 1 IEEE1471/ISO 42010 standard meta-model

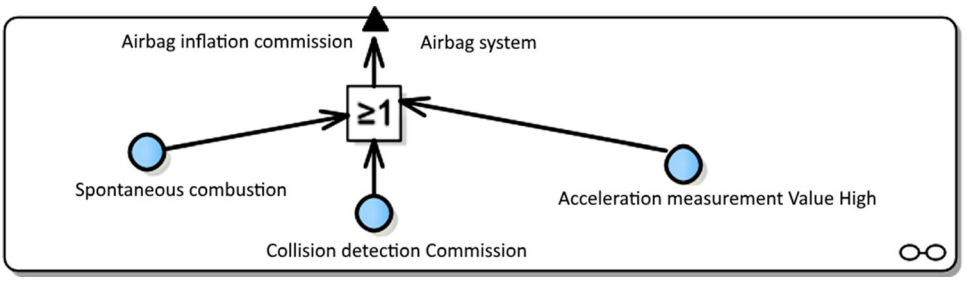


Fig. 2 FTA of a simplified airbag system

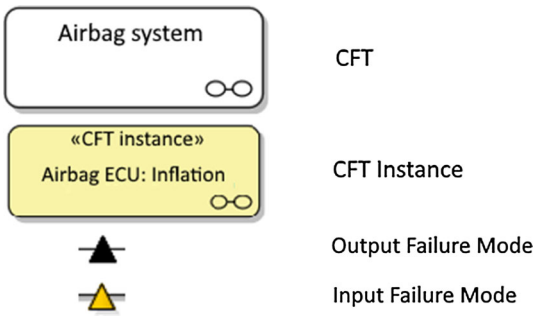


Fig. 3 CFT modularization constructs

At the top, Fig. 4a presents the logical network of the system. Figure 4b depicts the CFT of the system integrating the CFTs of the components through CFT instances and depicts the failure propagation through the connectors. This way of documenting the system faults makes it easier to understand the source of consequences (compared to FTA) and facilitates the planning of where to integrate safety measures in order to prevent or correct the identified issues. Figure 4c presents the CFTs of the components, depicting their specific failure behavior. The dotted lines establish the relationship between failure modes and logical interfaces.

One of the main goals of CFTs [3] is to bring designers and safety engineers closer, given their natural dependency. The goal of a safety engineer is to guarantee that faults identified through fault trees are removed or mitigated. In this sense, FTA influences the design, since the identified faults lead to the definition of safety requirements for which new functions/components will be integrated into the architecture. In order to enhance the interaction between these stakeholders, CFTs propose the use of a component-centric modularization approach where the composition of functions or components (e.g., Logical/Hardware), as provided by the system's architecture design, is used to structure the Fault Tree Analysis. Therefore, the techniques explicitly associate CFT modules with architecture modules as well as their interfaces, see Fig. 4. Each interface from the architecture is analyzed and failures modes are identified and linked.

One further advantage of CFTs with respect to FTA is that the construction effort can be distributed over several modelers. In this sense, it would be possible to carry out the analysis almost in parallel, since one person will only deal with a certain part of the system at a time. CFTs are also better than FTA in terms of the maintainability of the model. Typically, changes performed in the design models need to be considered in the Fault Tree Analysis. However, it is also usual that these changes occur locally to one or the other components at a time. CFTs facilitate the maintenance in that only for those components which have been modified the associated CFT needs to be reevaluated.

3 Problem statement

The complexity of the Fault Tree Analysis process is typically as high as the level of details provided by the architecture. Generally, this is reflected in the input architecture network. In early states of the development process, only high-level networks exist. In such a scenario, the failure descriptions are typically done at a high level of abstraction to get a big picture of the most critical system interactions, without having to invest too much effort into the analysis. In contrast, if the architecture network has already been refined, then typically more details of the system are known and the Fault Tree Analysis will be carried out more precisely.

FTA is known as a deductive analysis technique; for this reason, the input or starting point of the analysis is the hazard or undesired event represented in the form of top events. From this point on, the analyst checks the system for possible faults that could lead, either alone or in combination with other faults, to the undesired event. In CFTs, it is common to perform the analysis in a systematic way, by investigating each of the data flows. This occurs backward, starting with the data flow from the actuators and going back to the sensors. Each entity along the interaction chain is checked. See Fig. 5.

Depending on the modularization applied in the construction of the architecture network, three typical situations occur:

- Some entities are not included in the analysis because they are not safety relevant.
- Some entities only play a role for one specific hazard and are therefore not relevant in the context of other hazards.

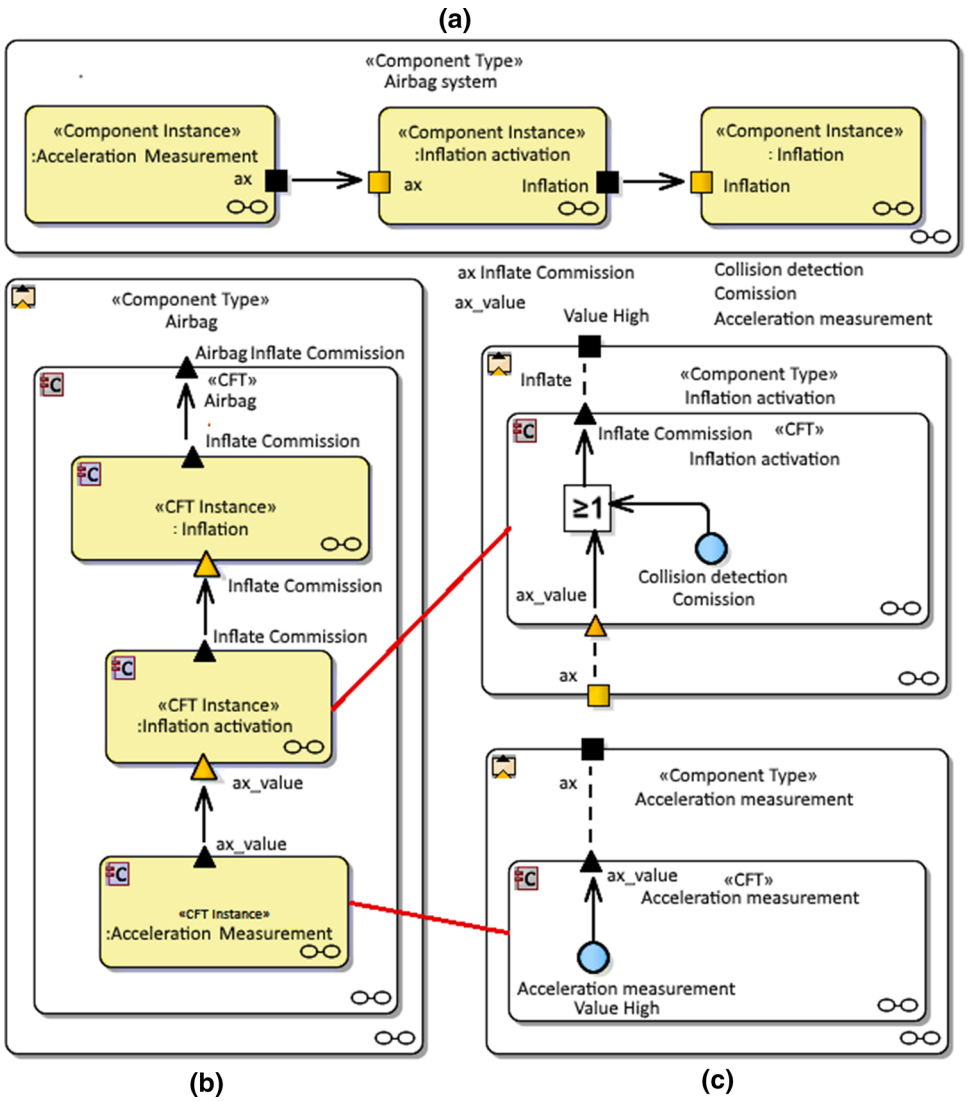


Fig. 4 CFT example of the airbag system

- Some entities (functions/components) are part of the interaction chain in the context of several hazards; i.e., that an entity is analyzed from two or more different perspectives.

In CFTs, all hazards are modeled in the same diagram. This typically leads to complex fault propagation networks in which all entities related to all hazards are shown. This situation is shown in Fig. 6, in which a real-life example is presented.

Due to a Non-Disclosure Agreement, further details cannot be displayed. However, it can be seen from the figure that CFTs do not scale well when considering many hazards at the same time, as the model has become very complex, even though only five hazards were examined. Furthermore, it is not always possible to identify which parts of the system are relevant to one hazard or another as shown in the model section. Without the yellow and blue borders, it cannot be identified that the component highlighted by the red boundary is only relevant for the context of the second hazard. Furthermore, it cannot be seen that for the component above, the incoming interfaces enclosed in the green boundary are not relevant for the context of the second hazard.

Note that this situation becomes even worse if the entities are refined into other entities (e.g., subfunctions/subcomponents), because parts of the system are not visible directly, but are rather hidden behind some of these entity specifications.

Another important aspect that we have seen in our many years of experience is that not all details are known in the early stages of architecture design. This often leads to the definition of failure modes that makes no distinction for specific situations, so that these can easily be misinterpreted and their criticality might be disregarded. This is illustrated in Fig. 7b. Figure 7a shows the logical network that represents the part of a vehicle where the sensor functionality provides the distance to the vehicle in front to the ACC controller as well as to the automatic parking controller. In the Fault Tree Analysis (Fig. 7b), a collision with a vehicle in

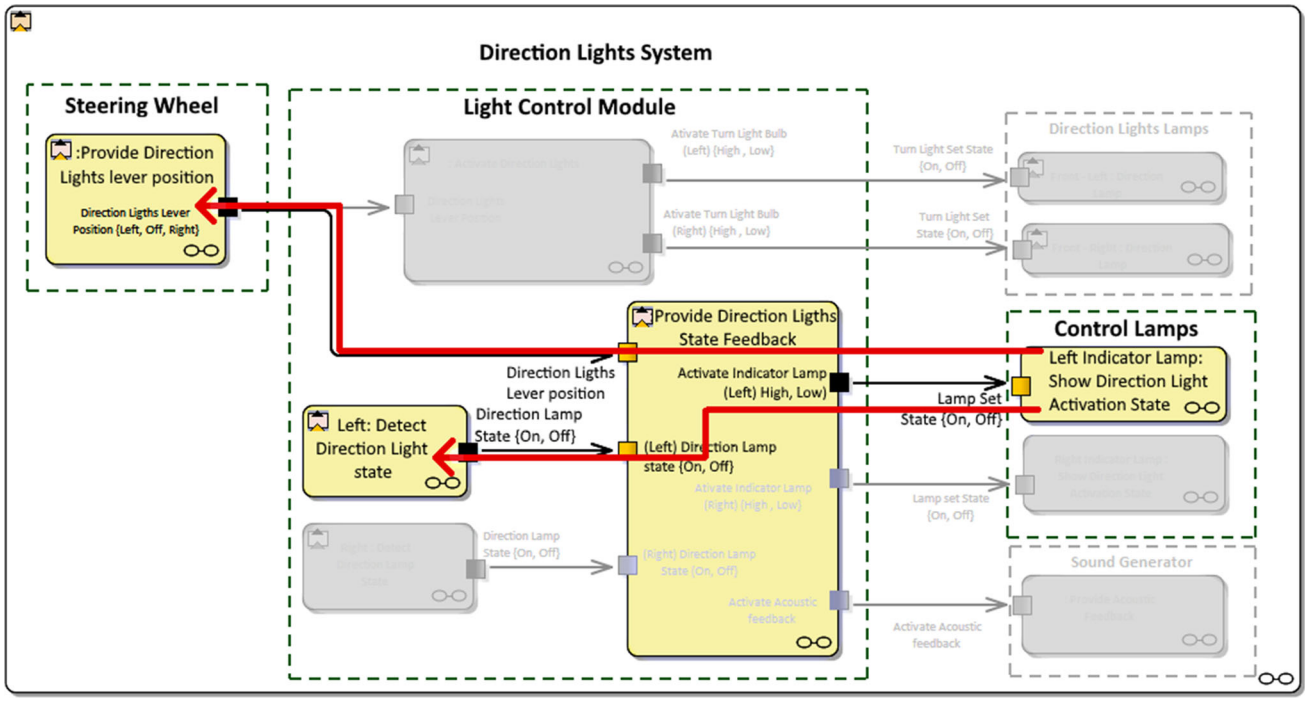


Fig. 5 Data flow backward analysis. The figure is presented with all details in Fig. 12

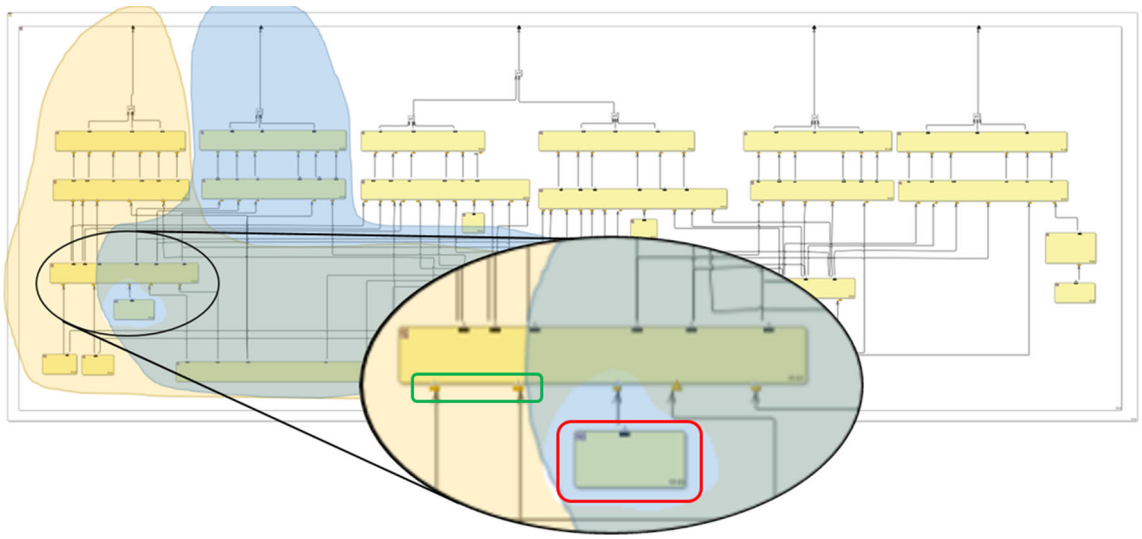


Fig. 6 Overview of CFT for a system with multiple hazards

front is investigated. The undesired event can always occur if the control systems do not behave correctly or if the sensors do not provide the correct information. The problem in this scenario is that the failure modes are too general and do not distinguish the criticality of the underlying situations. Typically, the ACC can only be activated when the vehicle's speed is greater than 30 km/h, and parking assistance can only be activated when the vehicle is stationary. The activation conditions are clearly very different. Moreover, in the event of an accident, injuries caused by the ACC are more likely to be more critical than those caused by parking control functionality. This would generally lead to a different ASIL (Automotive Safety Integrity Level) classification of these functions. Another aspect to consider about this example is the abstraction level used. Most failure modes from Fig. 7 can be considered as not sufficiently specified. For instance, it is assumed that the same sensor is used for both functionalities (ACC and parking), which is very unlikely as the operating conditions are very different. Due to such situations, a finer

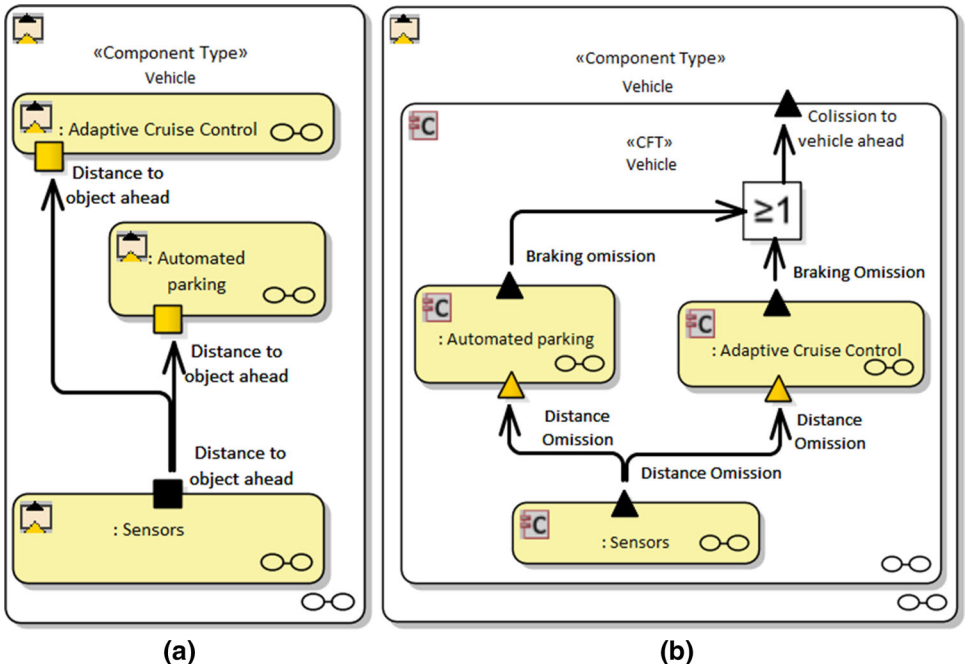


Fig. 7 Early Fault Tree Analysis issue

distinction of hazards and failure modes is useful, as we will see along this article.

One last aspect to be considered is the model presentation and communication. When more details are integrated in a CFT, the model grows in size and complexity as shown in Fig. 6. Such large models are impractical because they can hardly be displayed on a standard monitor. If enlarged, the modeler can get the details, but loses the overview, since most of the information will be hidden and can only be displayed by scrolling, which ultimately decreases the navigability and understandability of the model. This makes the modeling and reviewing processes harder and more error-prone.

4 Solution concept

Our concept is hazard-centric; i.e., the hazards identified from the Hazard and Risk Assessment (HARA) are the drivers for the definition of the realization views of CFTs. During the hazard analysis, hazards are identified for system-level functions, which are composed of simpler, possibly atomic functions. These functions are generally realized by components that, together, represent the structure of the system. As explained in Chapter 2, ideally a CFT should be defined for each of these components. While the Fault Tree Analysis could be performed in a generic way, to be able to reuse the Component Fault Tree in other contexts, it is quite often the case that achieving a properly reusable description is difficult and at reuse time, the models appear incomplete and are hard to understand. In addition, as shown in Chapter 3, context information might get lost and the analysis then becomes incomplete. In our experience, analysts deal with this situation by describing the hazards and failure modes in a more concrete way (e.g., omission due to H1), see Fig. 8.

Figure 8 presents such a model where the failure modes were modeled more precisely in comparison with the model presented in Fig. 7. In this way, the modeler can be more specific, which facilitates the review process. However, doing so inevitably leads to CFT modules with a large number of failure mode interfaces and connections, making the overall model more complex, as pointed out in Chapter 3 (see Fig. 6).

In order to keep the modeling and review processes simple for safety engineers, we propose in this work the definition of multiple realization views for CFTs based on the identified hazards. While the CFT of a component contains the definition of the failure behavior of that component for the different hazards, realization views (by means of diagrams) make visible only that part of that behavior specification that is relevant for one hazard at a time, as shown in Fig. 9.

Our solution is composed of three aspects:

1. A modeling aspect
2. A methodological aspect
3. A tooling aspect.

4.1 The modeling aspect

The meta-model for our modeling approach is depicted in Fig. 10. This figure presents the concept-relevant part of the CFT Domain Specific Language as an extension of UML constructs.

Figure 11a shows the representation of the "CFT Realization View" entity, while Fig. 11b shows the model composition for the vehicle system CFT, as presented in

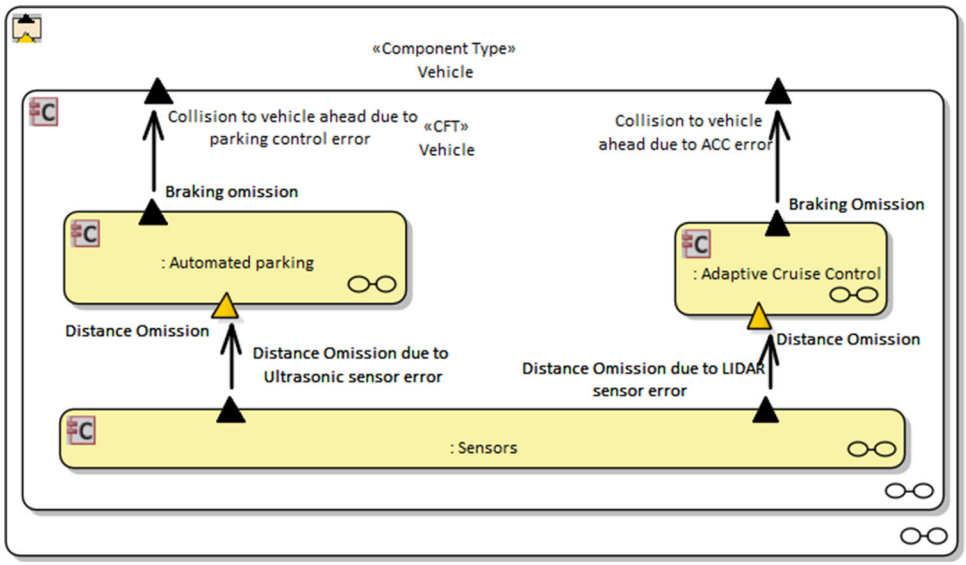


Fig. 8 Refined Fault Tree Analysis

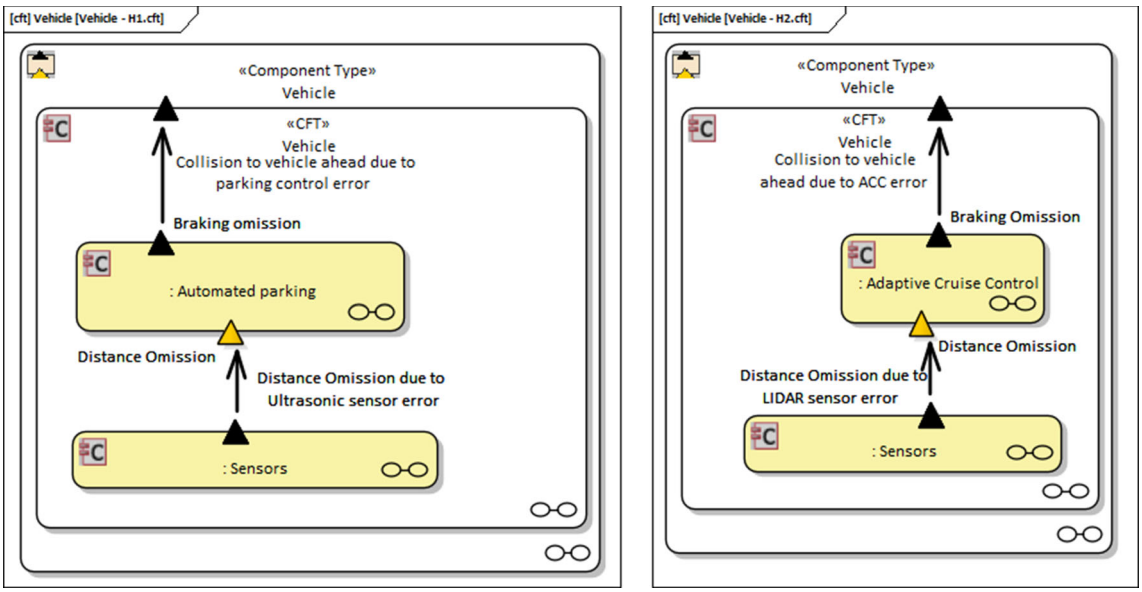


Fig. 9 Two realization views of the same CFT

Fig. 9. Realization views are highlighted through the red boundary.

CFTs supports the composition of other sub-CFTs similar to "Components" in UML [16]. This is necessary because CFTs are defined for components of logical networks, as shown in Chapter 2. At the modeling level, composition is supported with the help of "CFT Instances" and "CFT Realization Views."

As with Component-Based Design, CFTs encapsulate their realization in an internal view (white-box view) and expose their interfaces through an external view (black-box view). In our modeling approach, these views are supported such that a CFT has at least one such internal view, (i.e., a default CFT Realization View) and exposes the external view through "CFT Instances" when instantiated in a realization view of another CFT. This is already partially presented in Fig. 9. Figure 12 now depicts the realization views of the "Sensors" entity to clarify the concept.

As can be seen in Figs. 9, 11, and 12, the CFT of the "Sensors" has been instantiated twice, once in each of the realization views of the CFT of the "Vehicle." It should be noted that each of these instances shows a different interface specification although both have the same type, namely the Sensors CFT. In this way, a "CFT Instance" behaves similarly as a UML Property [16], with the difference that it shall only expose the black-box view corresponding to the assigned preferred realization view; see the "preferred RV" attribute in Fig. 10.

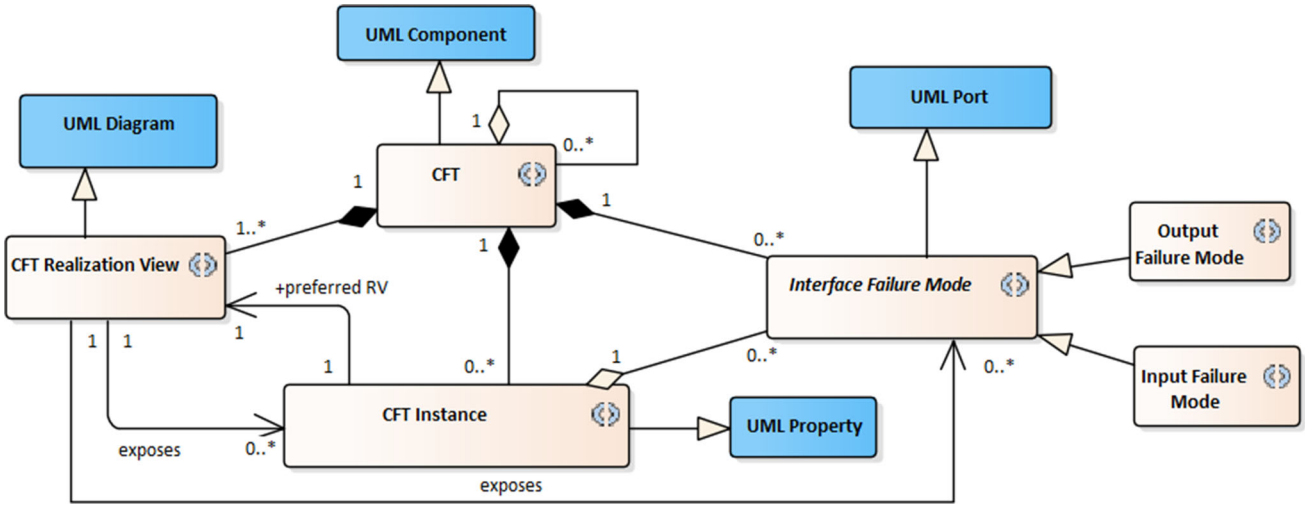
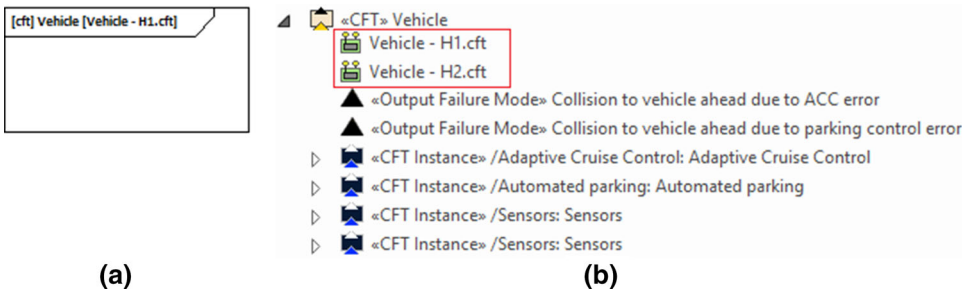


Fig. 10 CFT realization views meta-model extension (UML constructs in blue)

Fig. 11 Realization view modeling construct example



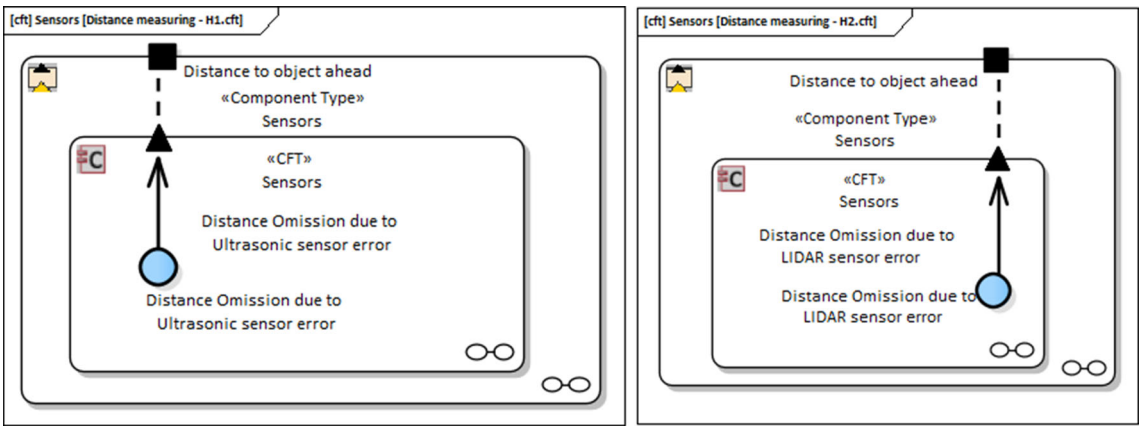


Fig. 12 Realization views of the sensors entity

We have added the "CFT Realization Views" as a modeling object to facilitate the understanding of the concept and because in practice, these entities should exist as part of the data model. These views are supported in our approach with diagrams (see UML diagram element in Fig. 10), although they could be supported by other presentation means. The use of model elements is required because they need to be referenceable model artifacts, as already shown by the "preferred RV" attribute. Moreover, these artifacts will be associated with other important artifacts of the safety life cycle (e.g., the safety case) due to traceability requirements.

4.2 The methodological aspect

The proposed modeling approach provides more modeling and expressiveness power than pure CFTs. However, a modeler might be confused as to how to use the approach effectively. For this reason, we propose as part of the solution

the definition of the realization views from the point of view of the safety engineer, taking as a basis the hazards/undesired events being analyzed.

As pointed out in Chapter 3, fault analysis with CFTs is typically performed for the entire system, starting from the actuators and then following the data flow backward. Therefore, the modeling process should start by defining a realization view (i.e., diagram) of the CFT associated with the topmost component type, which shall be related to one undesired event at a time. More undesired events might be integrated if there is high reuse potential for existing modeling artifacts and if the complexity of the diagram remains low, i.e., following the traditional modeling approach for CFTs. However, we recommend creating one realization view per undesired event to facilitate the review process, since this will be performed systematically, one hazard at a time.

While moving backward in the data flow, the user will encounter architecture entities to be analyzed. For these, a dedicated CFT realization view in the context of the current top event should also be defined unless it is possible to reuse an existing one or parts of it (e.g., existing failure modes, basic events, and failure logic).

By following this procedure for the complete functional chain up to the sensors or inputs of the system, the modeler will complete the CFT realization view of the topmost component, in which all CFT instances have a dedicated realization view exclusively for the current context under analysis.

It should be noted that this approach primarily targets a top-down development process, because in order to depict the failure behavior of the subsystems and components, good knowledge about their implementation is required. In this work, we do not investigate how distributed development could be supported, i.e., how existing models from suppliers could be integrated at the Original Equipment Manufacturers (OEM) level.

In summary, this methodology reflects the focus we have placed on the safety engineer's tasks, since the goal is to emphasize the modeling in one single undesired event at a time. This approach is intended to keep the model simple and understandable, thereby facilitating the review process.

4.3 The tooling aspect

Thanks to the integration of realization views into the modeling approach, the Fault Tree Analysis can be divided into manageable pieces to handle the complexity of the system. However, this has a negative impact on the maintenance effort, given the fact that more views have to be considered. Because of this and in order to facilitate the modeling process, we have conceived several automation features, which we have implemented as part of the safeTbox modeling tool [17].[1] The following aspects have been considered:

- Definition and assignment of realization views, for classifiers and instances.
- Cloning (deep and shallow) of existing realization views.
- Definition of the preferred realization view and navigation to multiple realization views.

Special attention has been given to the consistency (with respect to the functional/logical interfaces) between component and CFT realization views. As noted in Chapter 2, CFTs are defined for each of the components of the architecture. In addition, as shown in Figs. 4 and 12, the components and their interfaces are shown as part of the realization views of the CFTs. This is done as proposed in [4]. In safety engineering, it is very important to demonstrate that the analysis is consistent and complete. This is typically achieved when all entities in the architecture are considered during the fault analysis or when a convincing argument is provided why this is not the case. Looking back at our solution, one can see that the modeling approach is quite simple, since it is built on the basis of entities (functions/components) and their interfaces. For the first type of constructs, one would typically expect to see in a CFT realization view a CFT instance for each component instance as defined in the component realization view. However, we deliberately decided not to implement any mechanism enforcing composition consistency (e.g., for component instances). The reason for this decision was that during the analysis, entire entities are quite frequently left out, since they might not play any role in the system-level function under analysis. For the interfaces, we did, however, implement a synchronization mechanism that, based on the association between the component and CFT, guarantees that the interfaces remain consistent. This mechanism supports the user by showing the impact of changes, e.g., when showing/hiding interfaces. This functionality is required because the user might not be aware of which CFTs will be affected by performing changes on the architecture. Moreover, this is intended to help guarantee the completeness argument, since all inputs and outputs of a component have been considered. Figure 13 shows a dialog supporting the impact of changes when showing or hiding interfaces.

Besides modeling support for realization views, safeTbox also offers the possibility for CFTs to perform qualitative (e.g., Minimal Cut Sets) and quantitative analysis (Unavailability, Cut Sets Importance, Common Cause Failures). It also supports the definition of type systems for ports and failure modes, as proposed in [20]. CFTs in safeTbox make part of the model-based safety engineering approach inte-

[1] safeTbox is an Enterprise Architect [18] extension supporting an integrated modeling framework for safety engineering.

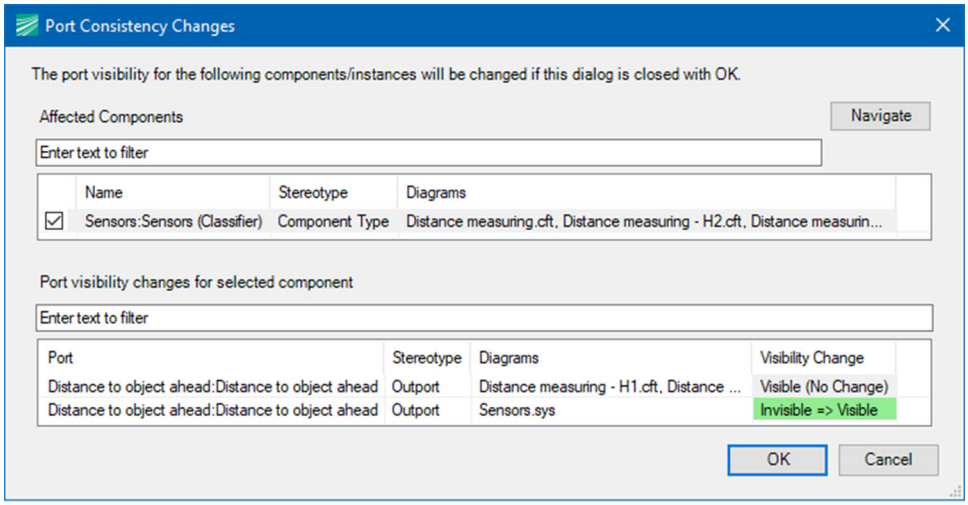


Fig. 13 safeTbox—port consistency supporting dialog

grating modeling support for a hazard and risk assessment and support for the construction of modular safety concepts and safety cases.

5 Case study

The evaluation of the concept was carried out in the context of the automotive domain as part of the development of several vehicle subsystems (e.g., external lights system, brake system, battery management system, power train, etc.). For the sake of intellectual property protection, the models shown in this document have been simplified, abstracted, or obfuscated.

5.1 Architecture design

In order to show the application of the approach, the "direction lights" functionality is selected. Direction lights serve to announce to other road users the intention of changing the direction of travel. In total (and to ensure visibility from all relevant perspectives), three flashing lights are installed on each side of the vehicle: one in the front, one in the back, and one on the side. The driver can announce the direction of travel by making use of the direction signal lever on the left side of the steering wheel (raised position→ turn right, lowered position→ turn left, middle position→ off). As long as the switch remains in the selected position, the lights of the selected side will flash periodically. At the same time, the driver is informed of the direction lights' activation state through acoustic and visual feedback (control lamps). Figure 14 shows a high-level architecture of the direction lights functionality.

To keep the model simple, only the front (left and right) direction lights have been modeled. Moreover, stereotypes have been omitted. Starting from the left:

- The steering wheel communicates the current position (i.e., Left, Off, Right) of the direction lights lever to the Light Control Module (LCM); see function "Provide Direction Lights Lever Position."
- The LCM receives this information and intermittently activates the direction lights accordingly; see function "Activate Direction Lights."
- In order to provide appropriate acoustic and visual feedback to the driver, the LCM integrates the functions for detecting the current activation state of the direction lamps; see functions "Detect Direction Lamps State" and "Provide Direction Lights State Feedback." This last function also receives the information of the Direction Lights Lever Position, which is used to check the plausibility of the direction lights' activation state.
- Finally, the indicator lamps and the sound generator provide the visual and acoustic feedback, respectively.

The behavior of the function "Provide Direction Lights State Feedback" is specified as follows:

Plausibility check passed when:

- Lever position is left AND left direction light is active
- Lever position is right AND right direction light is active
- Lever position is off AND all lights are inactive
- → System reaction: Activate/deactivate accordingly.

Plausibility check fails when:

- Lever position is left AND right direction light is active OR both lights are active
- Lever position is right AND left direction light is active OR both lights are active
- Lever position is off AND any direction light is active
- → System reaction: provide visual (by activating both indicator lamps) and acoustic feedback with a frequency twice as fast as usual.

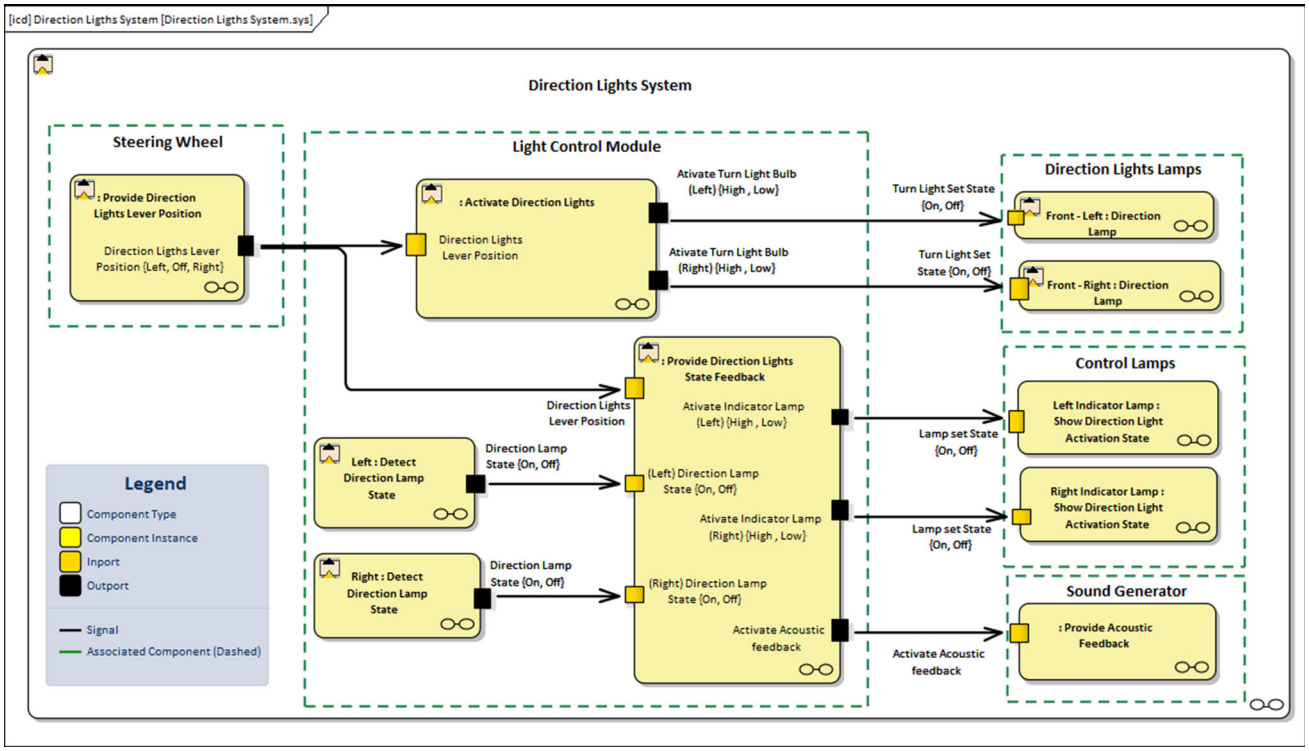


Fig. 14 Function network for the Direction Lights System

Plausibility check fails when:

- Lever position is left or right AND both lamps are inactive
- → System reaction: provide visual (only the respective indicator lamp) and acoustic feedback with a frequency twice as fast as usual.

5.2 Safety analysis

As mentioned earlier, the starting point for performing fault analysis with fault trees is to identify the undesired events. In order to do that, a Hazard and Risk Analysis (HARA) is typically performed. However, this will not be shown here, as it is not relevant for the current approach. The resulting hazards identified from the HARA are:

1. Inverted activation of the direction indicators without feedback
2. Missing activation of the direction indicators without feedback
3. Undesired activation of the direction indicators without feedback

For the sake of simplicity, in this document we will only show the application of the methodology in the first two cases. Figure 15 shows the realization view in which the inverted scenario has been modeled for the CFT of the Direction Light System.

As explained in Chapter 3, we analyze the data flow in the system backward. In this case, as can be seen in the model, the direction lights, the indicators lamps, and the sound generator have not been included for the sake of simplicity. While turn lights and indicators might be turned on as the result of a fault, e.g., due to a short circuit to the battery, it is highly unlikely that they will work intermittently, reflecting an actual activation from LCM. The sound generator has also been excluded since it is activated consistently with the indicator lamps. For this reason, the analysis starts at the LCM boundaries.

Without having the intention to deepen the fault tree modeling aspects, this model reflects the situation in which the driver has selected the direction with the lever (e.g., left), but due to errors in the system, the opposite direction (i.e., right) is activated, while visual feedback continues to be received that is consistent with the lever position (i.e., left). This situation is safety-critical, as the driver has no chance to identify the inverted activation, and other drivers will most likely interpret his intentions wrongly, which may lead to an accident.

Figure 16 shows the realization view in which the omission scenario has been modeled for the CFT of the Direction Lights System. In this scenario, the driver would have selected one side with the lever (e.g., left), but the system would have failed to activate the corresponding direction lamp. At the same time, it would have wrongly activated

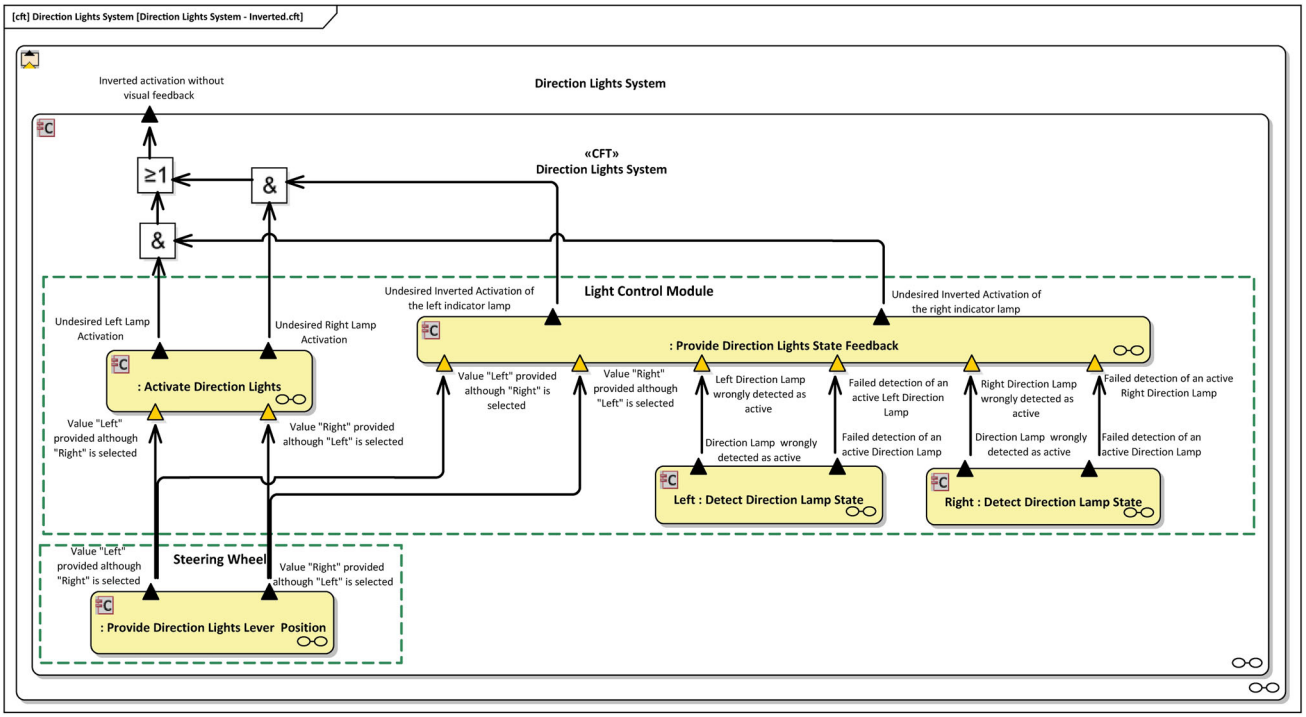


Fig. 15 "Direction Lights system—Inverted" realization view

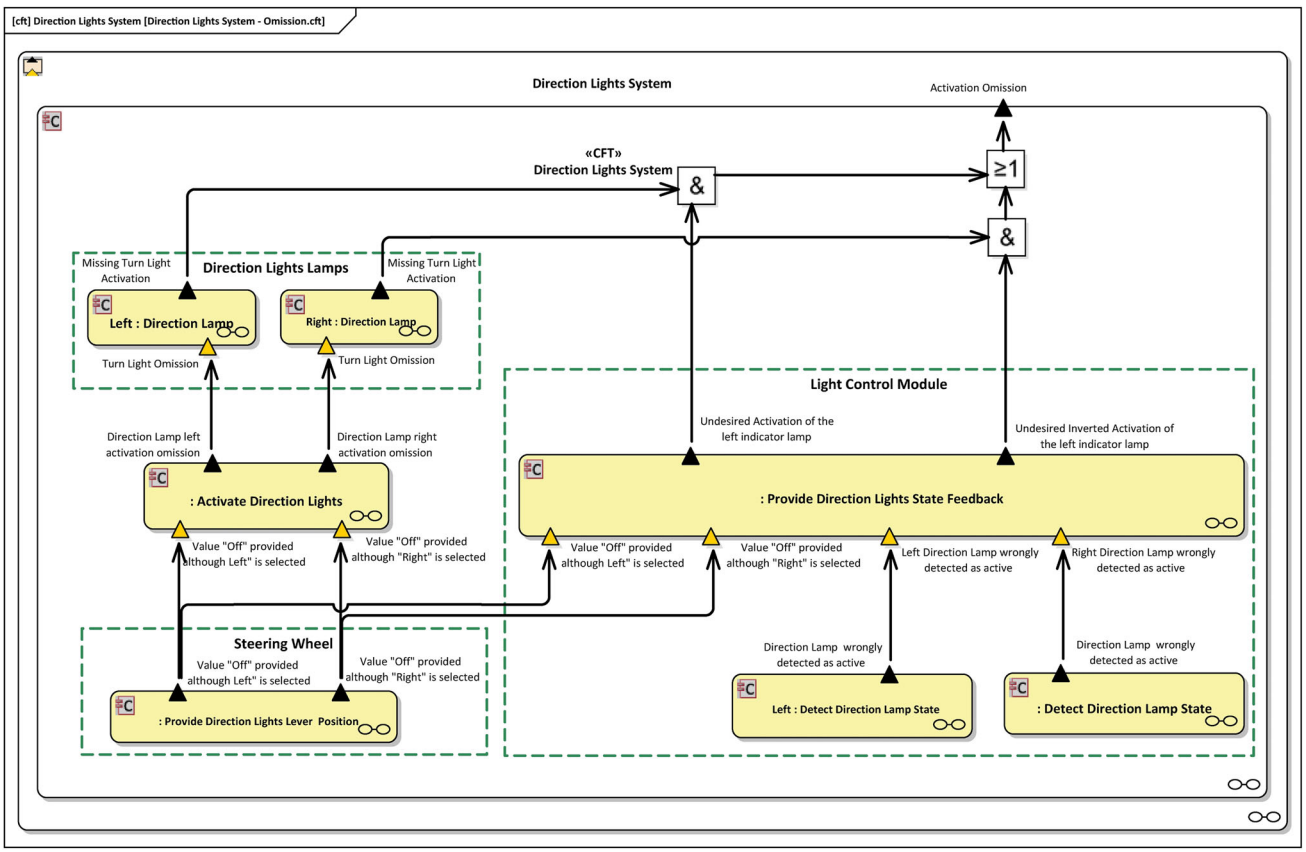


Fig. 16 "Direction Lights System—Omission" realization view

the indicator lamp (i.e., left side). This situation is again safety-critical for similar reasons as explained in the inverted scenario. In particular, in this model the turn signals should be included, since the failure of the lamp can be considered as a root cause and a system behavior has been defined in relation to it.

When comparing the two models, it is easy to find that:

- Both views have many functions in common. This is natural since the analysis is driven by the architecture network and we follow the CFT principles. Nevertheless, not all functions are relevant in both analysis scenarios.
- Only few failure modes have been reused (e.g., "Direction lamp wrongly detected as active"). This happened because the analysis was performed at a low level of detail, as it was desired to distinguish more specifically the fault combinations leading to the defined top events. This becomes clearer when we compare the two realization views of the function "Provide Direction Lights State Feedback"; see Fig. 17.
- Most modeled entities (including logical gates, failure modes, CFT instances, and connectors) are different. Although this is hard to see from the diagrams, it is possible to identify this aspect if the concept was properly understood. Recalling Chapter 4—The Modeling Aspect, Component Instances have the "preferred RV" attribute, which defines for a CFT instance the realization view associated with it and reflects the interface specification as defined in that view. From this, we can conclude that the realization views of the Direction Lights System must have a completely different set of CFT instances, since each of them shows a different interface specification.

During the creation of the second and further realization views, one would expect the modeler to manually add all modeling entities. In order to save some time in such situations, we have implemented the cloning of realization views as pointed out in Chapter 4—The Tooling Aspect. For the presented modeling scenario, the deep cloning mechanism is of particular interest. This mechanism creates a new realization view on the basis of a selected one. It creates new modeling entities, whose layout is exactly as in the source realization view. During this process, thanks to the "preferred RV" attribute, the newly created CFT instances point to the same realization views as before. As can be expected, the algorithm does not work recursively along the composition. The reason for this is to avoid making the cloning process too complex for the modeler.

From the method point of view, the modeler has three options for creating the second or further realization views:

- Create an empty realization view. That is, the modeler has to start from scratch, adding new modeling objects or showing (i.e., reusing) existing ones. In this case, the layout of the model also needs to be done manually.
- Create a shallow clone. In this alternative, the diagram is cloned; i.e., the new diagram shows exactly the same modeling objects as before.
- Create a deep clone. As shown above, new modeling objects are created, but references to existing realization views are kept for CFT instances.

In any of these cases, the modeler is still confronted with the question of whether to keep the CFT instances as they are or associate them with a new realization view. The former is a better option if the failure modes have been modeled in a relatively abstract way and can be reused in different contexts. The second option should be preferred if the desired level of detail of the analysis is much lower, as was the case in the provided example. If the second option is preferred, then it makes sense that the realization view to be associated should already be modeled.

Figure 18 shows the merged realization view for the Direction Lights System, as it would be obtained by following the typical CFT analysis. As can be seen, even with two top events it is already difficult to identify which functions or failure modes are relevant for which top event. In order to get that information, it is necessary to navigate the realization view of the modeled CFT instances. This, however, has the negative effect that the context is lost, and the modeler is required to keep the information in mind while switching continuously between the realization views in order to get the right picture in their mind. The model becomes harder to understand the more top events are considered, similarly as shown in Fig. 6.

Figure 19 presents the merged realization view of the "Provide Direction Lights State Feedback" function. In comparison with the separated realization views, one can see that the model has become less understandable, even though great effort was invested in avoiding connector crossings. Although the logic itself is not complex, the model appears to be so due to the large number of connectors. In this case, the engineer not only needs to take care of modeling the failure behavior properly, which clearly becomes more difficult and error-prone, but also has to ensure the proper layout of the modeling elements in order to be able to communicate the model to the reviewer later on.

6 Related work

From the methodology point of view, several efforts have been made with respect to integrating safety and the architecture design, for example:

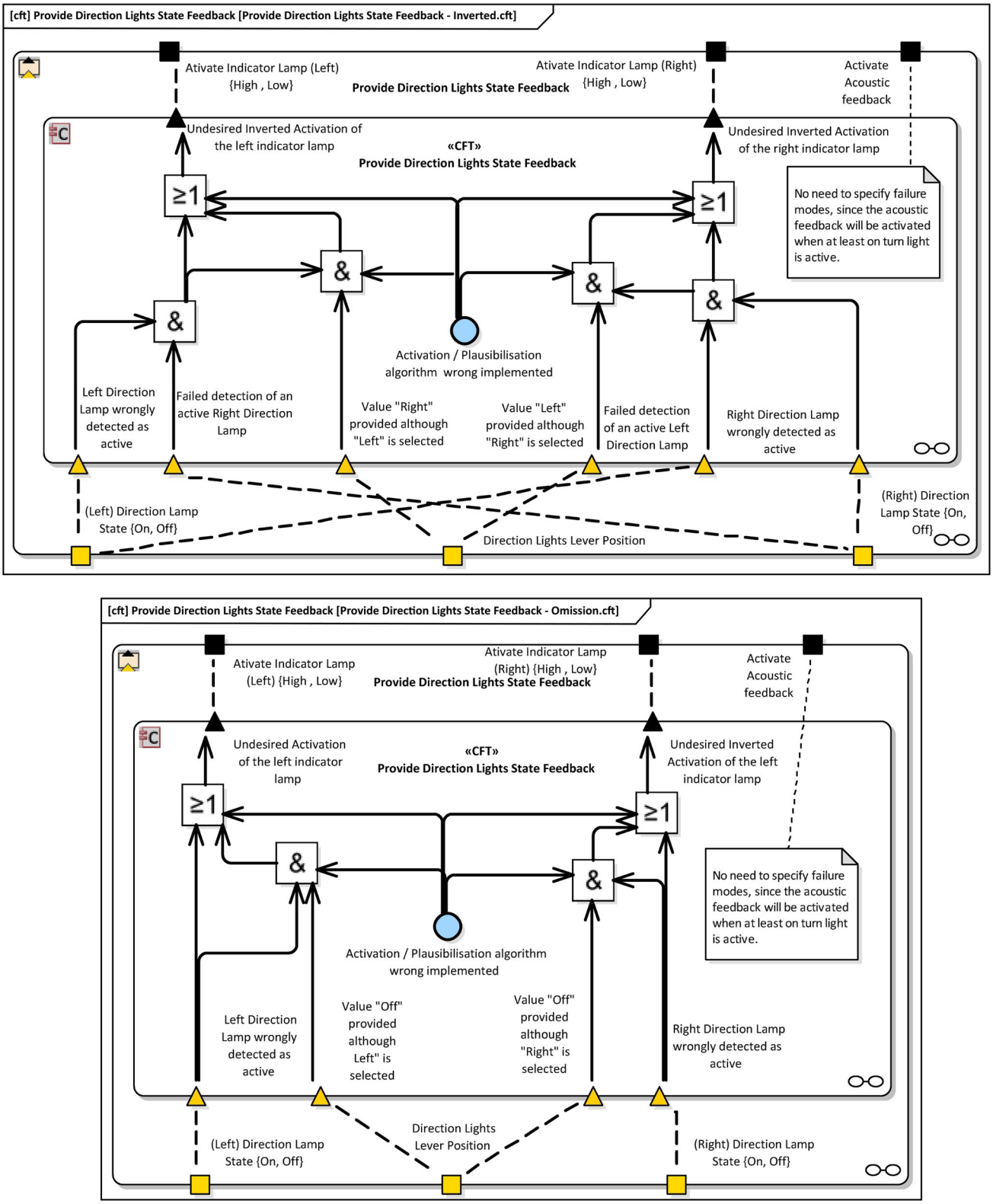


Fig. 17 Realization views "Provide Lights State Feedback—Inverted" (top) and "Provide Lights State Feedback—Omission" (bottom)

- SafeML [21] defines a profile for modeling safety information in conjunction with SysML. The main goal was to enhance the communication between stakeholders, since this is considered an important source of safety-related

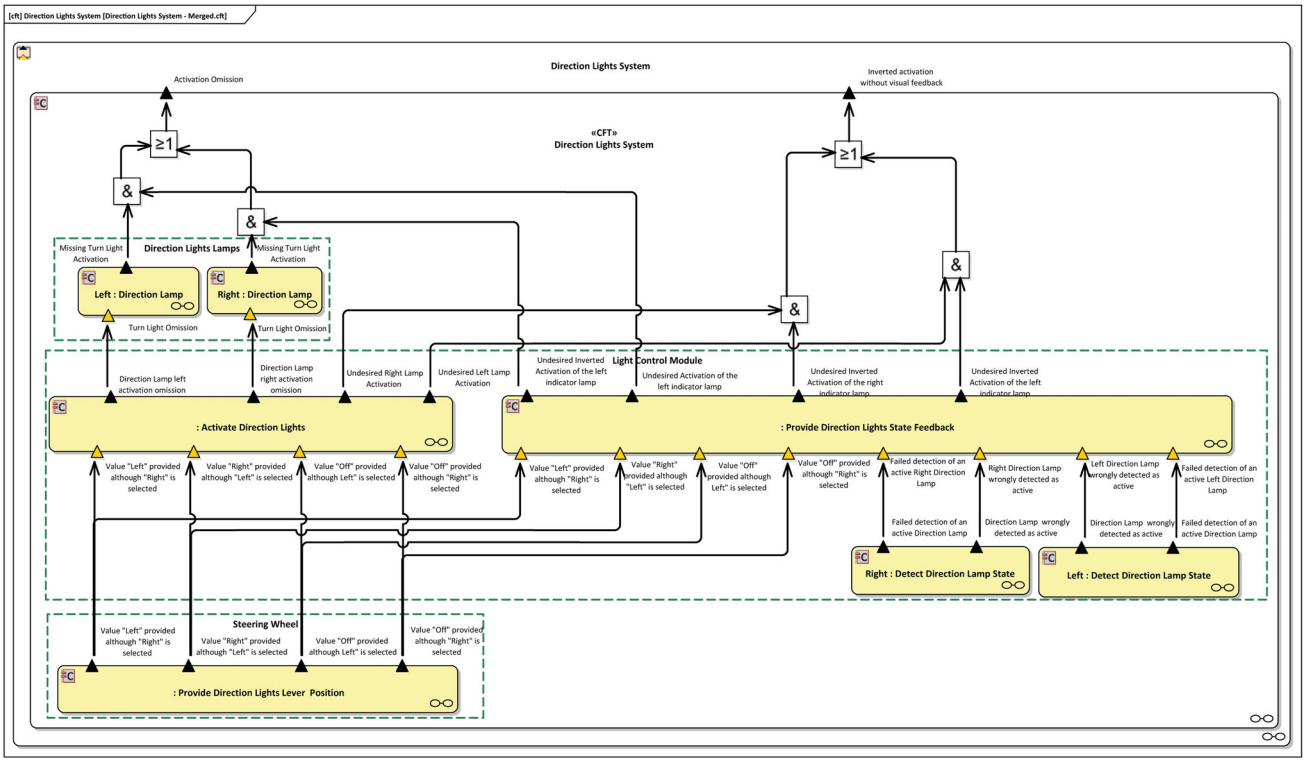


Fig. 18 "Direction Lights System—Merged" realization view

issues. The work presents the integration of concepts and techniques as well as tool support.

- SPESXT [22]. In the SPESXT research project, the definition of a safety perspective was investigated. This perspective is built on top of and across the SPES modeling framework, in which different viewpoints (requirements, functional, logical, and technical) have been defined for the documentation of the system architecture. Safety artifacts are well interconnected in their perspective and mapped to the architecture constructs to achieve traceability. One of the main aspects investigated in this project was modularization. Concerning safety, it aimed at enabling the construction of heterogeneous fault analysis models integrating fault trees, Markov chains, and FMEAs.
- The Eclipse Safety Framework [23] presents a meta-model and tool implementation to allow enrichment of the system design (functional and physical architectures) with safety information using annotations. One of the main goals of this open source project is to drive the discussion of such integration with the ultimate goal of standardizing the meta-model.

While these efforts are more focused on the conceptual perspective of the integration, most of them do not have such a strong focus on the modeling aspect, but merely on bringing the concepts together. At the same time, support for the methodology and tool aspects is relatively weak.

In the area of tools, several vendors such as Ansys-Medini analyze [24], Enco-Sox [25], or Vector-Preevision [26] offer model-based solutions that support most of the safety life cycle. These tools cover many different aspects, including modeling, traceability, analysis, reporting, process management, etc. However, to date, none of these tools or any other commercial tool offers proper support for Component Fault Trees.

7 Conclusion and future work

Our solution extends the CFT concepts further by integrating mechanisms for defining and mapping multiple realization views of architecture entities with realization views of CFTs. Moreover, a method for concrete use during system safety analysis in a scenario with multiple hazards is provided, as well as a tool implementation. Our solution presents the following advantages:

- First, the solution increases the modeling power of pure CFTs by integrating new concepts for the use of existing model constructs (e.g., realization views in the form of diagrams), which allow structuring models in a more manageable way. Moreover, it allows keeping artifacts consistent with each other. This facilitates the integration of changes performed in the architecture into the CFT real-

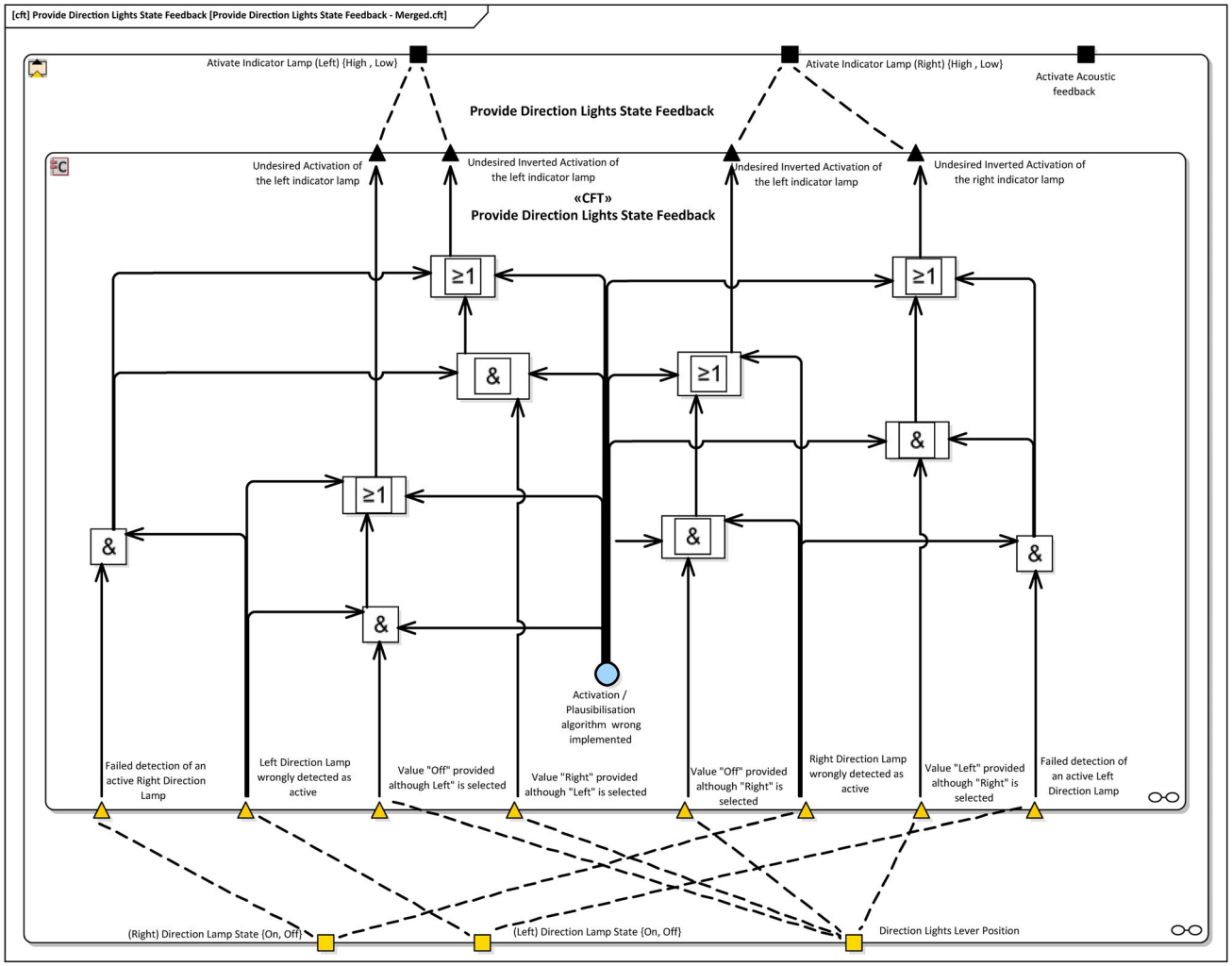


Fig. 19 "Provide Lights State Feedback—Merged" realization view

ization views, which ultimately leads to modeling effort being saved.

- Second, the provided method proposes to the safety engineer to divide the Fault Tree Analysis into understandable blocks of information by distinguishing the views on the basis of the undesired events to be analyzed. The modeling and the review process were the focus of the concept development, since the model will be kept simple by focusing on only one specific concern at a time.
- Third, a tool solution operationalizes the concept and provides usability and automation functionalities to enable appropriate usage of the proposed approach.

The proposed solution is superior to the original CFT modeling approach, except for one aspect: When modeling all hazards in a single view, the safety engineer has the advantage of getting the big picture of the interaction of all components for all relevant safety-critical scenarios. However, we consider this a minor issue, since such an integrated view can be generated out of the existing independent realization views. A functionality that builds such an integrated view automatically is planned to be implemented in the tool solution. One of the main difficulties in this regard is getting good layout results.

Other aspects are being considered as part of our future work, e.g., support for product line engineering and management of product families, as well as visualization features to facilitate the review process. With respect to this last aspect, an initial prototype targeting the switching context issue presented at the end of Chapter 5—Safety analysis, introduces the idea of using an information zoom mechanism that allows depicting information hidden behind the black-box views [19].

Acknowledgements Open Access funding provided by Projekt DEAL.; Please verify relation to: Fraunhofer Institute for Experimental Software Engineering (IESE) (1050).

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by/4.0/.

References

1. Ramamoorthy, C., Ho, G., Han, Y.: Fault Tree Analysis of Computer Systems, pp. 13–17. ACM, New York (1977)
2. 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, October 01, Canberra, Australia (2003)
3. Domis, D., Trapp M.: Integrating safety analyses and component-based design. In: International Conference on Computer Safety, Reliability and Security (SafeComp), pp. 58–71 (2008)
4. Domis, D., Trapp, M.: Component-Based Abstraction in Fault Tree Analysis, pp. 44–55. Springer, Hamburg (2009)
5. Adler, R. et al.: Integration of component fault trees into the UML. In: Proceedings in Models in Software Engineering, Workshops and Symposia at MODELS 2010-Reports and Revised Selected Papers, pp. 312–327 Springer, Berlin (2011)
6. Kaiser et al. B.: Advances in component fault trees. In: Safety and Reliability-Safe Societies in a Changing World: Proceedings of ESREL 2018, June 17–21, 2018, Trondheim, Norway, Taylor & Francis (CRC Press)
7. Stahl, T., Volter, M., Czarnecki, K.: Model-Driven Software Development: Technology, Engineering, Management. Wiley, London (2006)
8. Romero, J.R., Jaen, J.I. Vallecillo, A.: Realizing correspondences in multi-viewpoint specifications. In: 2009 IEEE International Enterprise Distributed Object Computing Conference, Auckland, pp. 163–172 (2009). doi: 10.1109/EDOC.2009.23
9. Kowalski, M., Wilkosz, K.: A domain specific language in dependability analysis. In: 2009 Fourth International Conference on Dependability of Computer Systems, Brunow, pp. 324–331 (2009). https://doi.org/10.1109/depcos-relcomex.2009.14
10. IEEE Architecture Working Group: IEEE Std 1471-2000, Recommended practice for architectural description of software-intensive systems. IEEE Technical Reports (2000)
11. ISO/IEC/IEEE: Systems and software engineering-architecture description" ISO/IEC/IEEE 42010:2011(E) (Revision of ISO/IEC 42010:2007 and IEEE Std 1471-2000), pp. 1–46 (2011)
12. International Organization for Standardization: IEC 61508: Functional Safety of Electrical/Electronic/Programmable Electronic Safety-Related Systems. The International Electrotechnical Commission, Geneva, Switzerland (1998)
13. International Organization for Standardization. ISO/DIS 26262-Road Vehicles-Functional Safety, Geneva, Switzerland: Technical Committee 22 (ISO/TC 22) (2011)
14. Crnkovic, I., Malavolta, I., Muccini, H., Sharaf, M.: On the use of component-based principles and practices for architecting cyber-physical systems. In: 2016 19th International ACM SIGSOFT Symposium on Component-Based Software Engineering (CBSE), Venice, 2016, pp. 23–32. doi: 10.1109/CBSE.2016.9
15. Atkinson, C., Bostan, P., Brenner, D., Falcone, G., Gutheil, M., Hummel, O., Juhasz, M., Stoll, D.: Modeling components and component-based systems in KobrA. In: Rausch, A., Reussner, R., Mirandola, R., Plášil, F. (eds.) The Common Component Modeling Example. LNCS, vol. 5153, pp. 54–84. Springer, Heidelberg (2008)
16. Object Management Group (OMG): Unified Modeling Language, OMG Document Number formal/2017-12-05 (2017). https://www.omg.org/spec/UML/2.5.1/PDF
17. https://www.safetbox.de/
18. https://www.sparxsystems.de
19. Velasco Moncada, D.S., Reich, J., Tchangou, M.: Interactive information zoom on component fault trees. In: Schaefer, I., Karagiannis, D., Vogelsang, A., Méndez, D., Seidl, C. (eds.) Modellierung 2018, pp. 311–314. Gesellschaft für Informatik e.V, Bonn (2018)
20. Möhrle, F. et al.: A formal approach for automating compositional safety analysis using flow type annotations in component fault trees. In: Proceeding of the 27th European Safety and Reliability Conference (ESREL): Safety and Reliability—Theory and Applications. Taylor & Francis (CRC Press), Portorož, Slovenia (2017)
21. Biggs, G., Sakamoto, T., Kotoku, T.: A profile and tool for modelling safety information with design information in SysML. Softw. Syst. Model. (2014). https://doi.org/10.1007/s10270-014-0400-x
22. Pohl, K., Manfred, B., Daembkes, H., Hönninger, H.: Advanced Model-Based Engineering of Embedded Systems: Extensions of the SPES 2020 Methodology. Springer, Berlin (2016)
23. https://www.eclipse.org/esf/
24. https://www.ansys.com/products/systems/ansys-medini-analyze
25. https://www.enco-software.com/
26. http://www.vector.com/preevision

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

David Santiago Velasco Moncada obtained his M.Sc. degree in computer science with a major in software engineering from the University of Kaiserslautern in 2011. Since his graduation, he has been working in the Embedded Systems Quality Assurance Department of Fraunhofer IESE. He has participated in several research projects with the focus on integrated model-based solutions for safety engineers. In his current position as team leader, he coordinates the development of the safeTbox modeling and analysis tool at the institute.