Kai Höfig
Dr. Mario Trapp
Bastian Zimmer
Prof. Dr. Peter Liggesmeyer

8

Modeling Quality Aspects: Safety

Safety is a central quality property of embedded systems. While progress in development methodologies, techniques, and tools enable the developer to manage the rapidly growing system complexity, this has long not been true for safety engineering methodologies. A promising approach to advancing the state of the art in safety engineering for software-intensive embedded systems lies in the application of model-driven development concepts to traditional safety engineering approaches. This chapter gives an overview of how safety analysis models can be integrated seamlessly into design artifacts and how model-driven development concepts can enable the modular safety assurance of platforms

K. Pohl et al. (eds.), *Model-Based Engineering of Embedded Systems: The SPES 2020 Methodology*,
DOI 10.1007/978-3-642-34614-9_8,

8.1 Introduction

Safety is typically defined as freedom from unacceptable risk (of harm). To ensure a certain level of quality, in most industrial domains the development of safety-critical systems is governed by standards. As mirrored in those standards, the development of a safety-critical system affects almost all process steps in a development lifecycle, ranging, for example, from requirements engineering through functional aspects to the technical design. For this reason, safety is not represented in a single viewpoint but as a quality aspect in the SPES modeling framework that has a crosscutting influence and is integrated into several viewpoints.

8.2 Concerns

Tight integration between safety analysis models and system development models

The growing complexity of safety-critical embedded systems is leading to increased complexity in safety analysis models. It is therefore not appropriate to develop functionality and consider safety in separate tasks. Safety aspects have to be integrated as tightly as possible into the development process and its models. Since model-based development of embedded systems deals with the increased complexity of such systems, their safety analysis has to follow this approach as well. Therefore, the goals of the quality aspect safety are:

- Provide modular, hierarchical, and model-based safety analysis models to "keep pace" with a state-of-the-art model-based development
- Ensure consistency and traceability among safety analyses in multiple views and on different layers of abstraction
- Ensure consistency between model-based safety analyses and other model-based development artifacts

The solution provided by this quality aspect is divided into two parts. The first part is component-integrated component fault trees (in short, C²FTs), as introduced in Section 8.3. Component fault trees (CFTs) provide the benefits of a modular and hierarchical safety analysis, whereas the component integration extension allows for a tight coupling between a component fault tree and a logical component. Based on C²FTs, this quality aspect contains two additional methods, one for managing consistency and protecting intellectual property over several layers of abstraction, as presented in Section 8.3.1, and one for

calculating probabilistic worst case execution times (pWCET), as presented in Section 8.3.2.

The second part of the quality aspect supports the system developer with the safety-related deployment of logical components onto technical components and is presented in Section 8.4. Deploying a logical software component onto a computer platform requires a check of whether the computer platform provides the safety requirements demanded by the software component.

8.3 Component-Integrated Component Fault Trees

The benefits of a hierarchical decomposition of complex systems and the applied principle of *separation of concerns* using (logical as well as technical) components can be perfectly transferred to the safety analysis model using the approach of component-based abstraction in fault tree analysis presented here. This decreases the complexity of safety analysis models, increases the connection between safety and development, and thereby reduces development costs.

The hierarchical decomposition of the system under development (SUD) into components and subcomponents, as well as the communication among them, follows a metamodel as specified by the logical viewpoint (see Chapter 6) but can also be applied to other model-based development languages. The central model element is the component that is embedded into a hierarchy of subcomponents and supercomponents communicating with each other using ports and connections.

Each component has an associated safety view and the propagation of failures follows *port interconnections*

The integrated metamodel for component fault trees follows this generic component model and allows the relation of a separate component fault tree for each component of the system's hierarchical decomposition. Fig. 8-1 shows the metamodel for component fault trees to be integrated with the component model: here, a component fault tree element, labeled CFT, is related to the component element of the metamodel. Faults propagate from one component to another via their interconnections (modeled using ports and connections). This is also reflected within the component fault tree metamodel: input and output failure modes are related to port elements of the component model [Domis and Trapp 2009].

The method supports the developer with automated modeling features

Based on these models, automations support the developer in handling the various component fault tree model elements such as gates, input and output failure modes, and edges between them to model the

failure behavior of the system under development using Boolean logic.

The approach was evaluated in the course of the SPES 2020 evaluation studies (see Chapter 16). Evaluation results show a good acceptance in this industrial area, since component fault trees are a model-based extension of widely accepted fault trees. Furthermore, measurements taken show that the complexity of modeling component fault trees compared to modeling classic fault trees is decreased, and the readability of component fault tree model elements is increased compared to classic fault trees. The evaluation was done by experiment in industry with more than ten people, from both the system development and safety analysis fields, and also in academia with the same amount of people and a similar background.

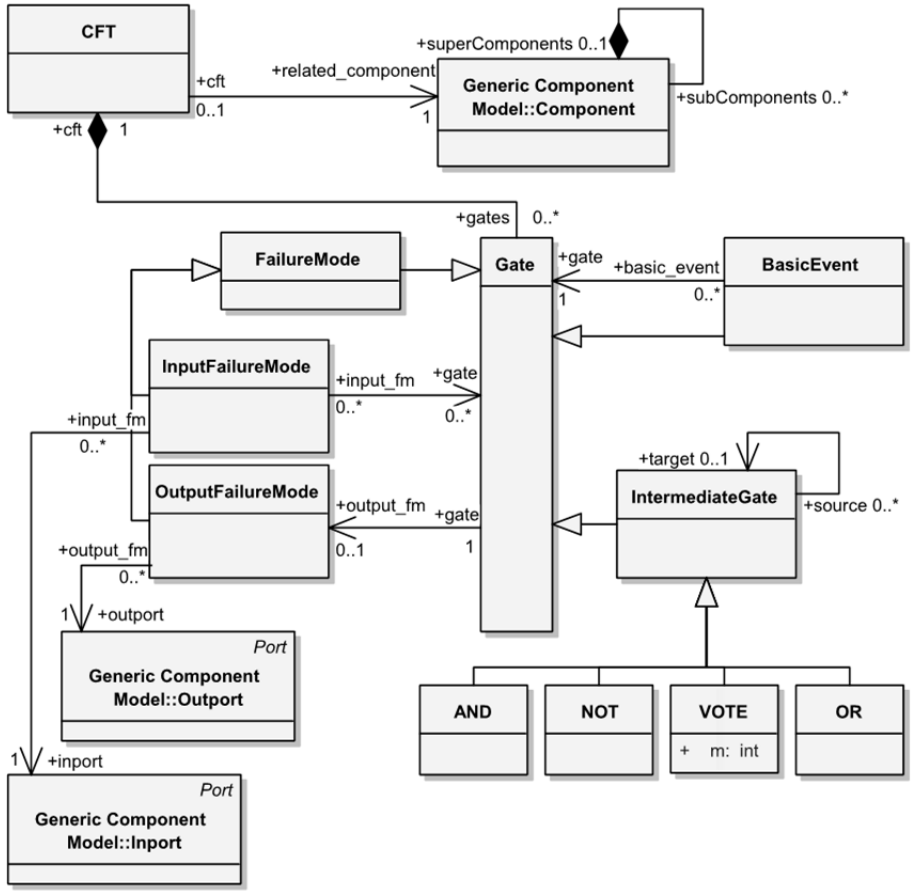

Fig. 8-1 *Metamodel for component fault trees*

8.3.1 Protecting IP and Managing Consistency across Abstraction Layers

Since the development process of modern embedded systems is often spread over many different stakeholders, e.g., different companies, this aspect is a special issue in safety analysis. During the hierarchical decomposition, components have a *specification* given by one stakeholder and are subsequently implemented by a different stakeholder using logical and technical subcomponents that *realize* the functionality as specified for the supercomponent. To analyze the failure logic of such a component, the failure logic of the supercomponent is also realized by the component fault trees of its subcomponents. To protect the intellectual property of the implementing stakeholder from the specifying stakeholder, the *white box* safety view of the subcomponents' fault trees can be transformed into a protective *black box* safety view using Boolean reduction [Domis et al. 2010]. In this way, the stakeholder of the supercomponent can still achieve the failure propagation of the specified component, but the intellectual property contained in the implementation is protected. Furthermore, the methodology allows checks regarding whether realization and specification are consistent with each other in order to ensure the consistency of the modular and hierarchical safety analysis across multiple abstraction layers.

8.3.2 Failure-Dependent Timing Analysis

This section presents a methodology that takes advantage of the tight integration of safety analysis models and system development models to combine elements of both worlds for a new execution time analysis approach.

The integration of component fault trees makes safety information accessible to other safety-related analyses such as worst case execution time analysis

Embedded real-time systems are growing in complexity and resource demand that today goes far beyond systems with simplistic closed-loop functionality. Current approaches of worst case execution time (WCET) analysis are used to verify deadlines of such systems, but these approaches calculate or measure the WCET as a single value that is used as an upper limit for a system's execution time. Overestimations are taken into account to make this upper limit a safe limit, but modern processor architectures expand these overestimations into unrealistic dimensions.

Here, therefore, probabilities of safety analysis models are combined with elements of system development models to calculate a *probabilistic* worst case execution time (pWCET). Safety analysis models are used in this approach as a source for probabilities [Adler et al. 2010]. Since safety analysis models typically reflect the occurrence of failures and

their propagation through the system under development, our approach aims at mechanisms in systems that are executed in *addition* to a failure. Such mechanisms usually belong to the area of fault tolerance and detect or process an error [Höfig et al. 2010, Höfig 2011b]. In this way, very unlikely time-intensive execution scenarios can be identified. This type of system becomes certifiable for a lower execution time if a deadline has to be guaranteed for a certain probability.

Example 8-1: Table lookup

An example of such a system is depicted in Fig. 8-2. This example is a reduced version of an example from the automotive domain as presented in [Höfig 2011a]. The right-hand side of Fig. 8-2 depicts the architecture of the system using logical components, and the left-hand side shows a part of the component fault tree model. The system measures sensor data with *Sensor A*. If this data is within a given range (plausibility test), the measured data is taken as the output of the system. If the test judges the data to be erroneous, data from a different sensor is taken to estimate the data for *sensor A* using a table lookup. Therefore, the execution of the table lookup function depends on whether *Sensor A* produced erroneous data or whether there is a failure in the *Test* component. Knowing the probability of occurrence for the failure of *Sensor A* and for the *Test* component, we can derive the probability for the execution of the time-consuming table lookup function and can calculate a probabilistic worst case execution time.

The approach has been evaluated using the tool for failure-dependent timing analysis presented in [Höfig and Domis 2011].

8.4 Efficiently Deploying Safety-Relevant Applications to Integrated Architectures

A method for finding a good deployment (a mapping between the logical and technical viewpoints) has to consider multiple aspects that influence costs and feasibility. One of these aspects is safety, and it is addressed by the approach described in this section. The SPES quality aspect safety contains a two-stepped approach for supporting a safety-related deployment. The goal of the first step is to find a promising deployment candidate using system-level information. The second step investigates the feasibility of the candidate by separately investigating the more detailed safety dependencies between each application and its host platform.

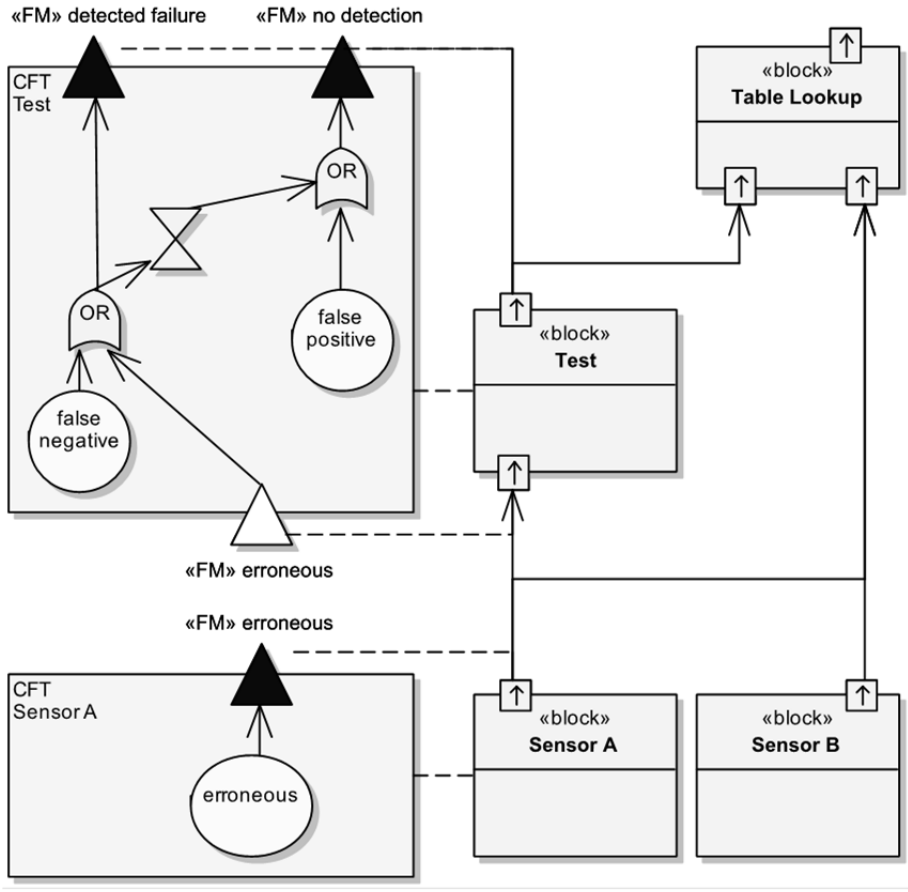

Fig. 8-2 *Sensor example system in SysML with component fault trees*

A deployment starts with a list of applications (high-level logical components), modeled as a network of communicating logical components (as specified in the logical viewpoint introduced in Chapter 6) that have to be deployed onto a set of computer platforms (as specified in the technical viewpoint introduced in Chapter 7), possibly containing several partitions. Fig. 8-3 shows a deployment calculated by a tool developed in the SPES project.

The algorithm for identifying promising candidates uses two metrics to calculate a quantified evaluation of the suitability of a deployment with regard to safety requirements [Zimmer et al. 2012].

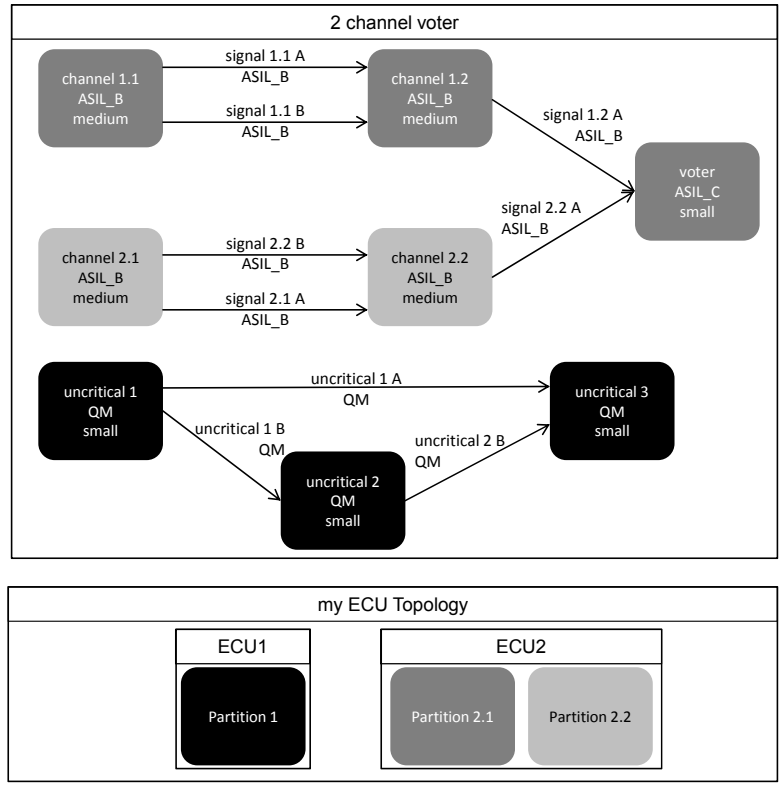

Fig. 8-3 *A network of software components deployed to a computer topology. The deployment of a component to a partition is indicated using the same shade of gray*

The cohesion metric evaluates criticality homogeneity within partitions

The first metric is called the cohesion metric and evaluates the homogeneity of application criticality levels in a partition. As already mentioned, a platform may comprise multiple partitions. Since freedom from interference is not guaranteed within one partition, every application in a partition has to be developed according to the maximum level of criticality of all applications in a partition.

Example 8-2: Cohesion metric

If, in the example shown in Fig. 8-3**,** the component *voter* and the component *channel 1.1* had both been deployed to *partition 2.1*, *channel 1.1* would have to be developed according to ASIL C. The cohesion metric reflects the costs entailed by these criticality increases.

The coupling metric evaluates volume and criticality of safety-relevant communication

The second metric is called the coupling metric and evaluates the volume of safety-relevant communication. If safety-relevant applications residing in different partitions exchange signals, undetected failures in this communication can cause a hazardous outcome. In order to prevent these inadvertent situations, safety mechanisms have to be developed and installed to detect or prevent failures. The coupling metric takes the costs caused by the volume of safety-critical communication, especially in-between platforms, into account.

After these metrics have been used to derive a deployment, the second step of the approach comes into play. The goal of this step is to assist the integrator in checking whether each application software component can run safely on its host platform, and if so, to assist in generating appropriate evidence. The method used in this second step is called VerSaI (*Ver*tical *Sa*fety *I*nterfaces) [Zimmer et al. 2011].

Demands and guarantees specify a contract-like interface

Before the *safety compatibility* between application and platform can be checked, demands and guarantees have to be specified. Demands are typically used to express all the properties an application needs the platform to have in order to be executed safely, whereas the guarantees represent the safety-related properties the platform possesses. A compatibility check is successful if a sound argument for the fulfillment of the demands with the available guarantees can be established. To enable tool-supported integration, the VerSaI approach offers a semiformal language to model these demands and guarantees.

The language comprises a finite number of elements

The language consists of a number of elements each representing a certain type of demand or guarantee exchanged by an application and a platform. This implicates the noteworthy fact that there is a finite number of language elements and, therefore, also a finite number of dependencies expressible with the language. First evaluations have shown that this is suitable because the typical service relationships between an application and a platform are finite and regular too, which is also the reason why it was possible to standardize platform interfaces in the first place.

The VerSaI language is integrated into model-based design artifacts

If the compatibility of an interface specified with the VerSaI language is checked, the demands and guarantees that have a potential relationship have to be identified first. This is done using the integration of the VerSaI language into the SPES modeling framework. A demand about the detection of a signal corruption is, for example, related to the model element representing the signal. On the other hand, a guarantee about detecting signal corruptions is related to the representation of the respective communication channel. If the detailed deployment of the signal to the com-channel is modeled, VerSaI uses this information in a

transitive manner to relate the corresponding demands and guarantees. This principle is depicted in Fig. 8-4.

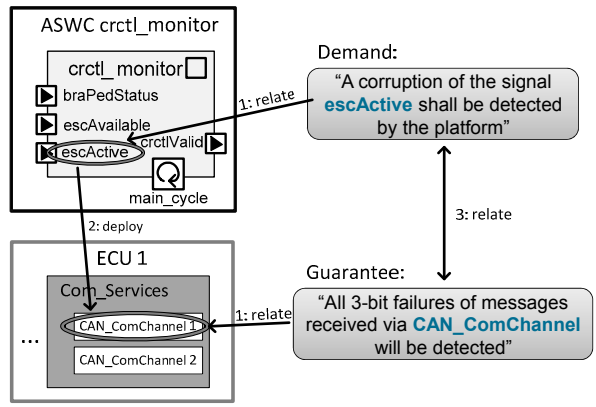

Fig. 8-4 *Relating demands and guarantees using deployment information*

The integration of applications and platforms is supported by a strategy repository

The final step of the method is checking whether each demand can be met with the guarantees identified as relevant in the previous step. In contrast to conventional interfaces, it is usually not possible to simply match the demands and guarantees respectively. In fact, an additional fragment must be generated in the safety case providing the arguments and evidences that the demands of the platform are met by the guarantees given by the platform. To this end, this step is supported by a strategy repository. The repository contains expert strategies that are selected and presented to the integrator, and that describe what guarantees are needed to fulfill the current type of demand and how to generate a piece of evidence containing a sound argument. It is important to note that, despite the formal basis of the language, each language element has a representation in natural language. This allows the human brain to read and evaluate the specification of demands and guarantees and the final argument generated after integration. This argument can be distributed for reviews and assessments.

The two-step approach presented allows the user to calculate a deployment automatically and assists the user in checking and arguing the safety of the chosen deployment.

8.5 Integration in the SPES Modeling Framework

In the SPES modeling framework, safety is represented as a quality aspect that has a crosscutting influence on and is integrated into several viewpoints. This section gives an overview of the integration of the safety quality aspect.

8.5.1 Viewpoints

Relation of the safety quality aspect to the functional, logical, and technical viewpoints

Considering the integration of component-integrated fault trees into the viewpoints of the SPES modeling framework, we have to differentiate between the capability of modularizing and hierarchizing the fault trees that comes with component fault trees, and the capability of integrating the fault trees into a component-based model. Component fault trees, on the one hand, can easily be applied to functions, logical components, and technical components, and thus, to the respective viewpoints as well. However, the component integration part works best with the logical viewpoint since the logical viewpoint metamodel represents communication between components explicitly using input and output ports, and this is best suited for component-integrated fault trees.

Since the deployment models the mapping of logical components to technical components, the methods for efficient deployment of safety-related applications belong to the logical viewpoint as well as to the technical viewpoint.

8.5.2 Abstraction Layers

Relations between the different abstraction layers of a C²FT

The relation between different abstraction layers of a component fault tree is comparable to the relation between different abstraction layers of a logical component as described in Chapter 6. A top-level component fault tree describes the failure behavior of the top-level component and can be decomposed into several lower-level component fault trees describing the failure behavior of the top-level component as an aggregation of several component fault trees.

8.6 References

[Adler et al. 2010] R. Adler, D. Domis, K. Höfig, S. Kemmann, T. Kuhn, J.-P. Schwinn, M. Trapp: Integration of component fault trees into the UML. In: Proceedings of 3rd International Workshop on Non-functional Properties in Domain Specific Languages (NFPinDSML2010). DOI: 10.1007/978-3-642-21210-9_30.

[Domis and Trapp 2009] D. Domis, M. Trapp: Component-based abstraction in fault tree analysis. In: Proc. of the International Conference on Computer Safety, Reliability and Security (SAFECOMP 2009). DOI: 10.1007/978-3-642-04468-7_24.

[Domis et al. 2010] D. Domis, K. Höfig, M. Trapp: Consistency check algorithm for component-based refinements of fault trees. In: Proceedings of International Symposium on Software Reliability Engineering , 2010.

[Höfig 2011a] K. Höfig: FDTA – A toolchain for failure-dependent timing analysis. In: Proc.11th International Workshop on Worst-Case Execution Time (WCET) Analysis, 2011.

[Höfig 2011b] K. Höfig: Timing overhead analysis for fault tolerance mechanisms. In: Proc. Zweiter Workshop zur Zukunft der Entwicklung softwareintensiver eigebetteter Systeme (ENVISION2020), LNI Vol. P-184, GI, 2011.

[Höfig and Domis 2011] K. Höfig and D. Domis: Failure-dependent timing analysis. In: Proc. 2nd International ACM Sigsoft Symposium on Architecting Critical Systems, 2011.

[Höfig et al. 2010] K. Höfig, D. Domis, M. Trapp, H. Stallbaum: Pattern-based safety engineering. Semantic enrichment of system architecture models for semi-automated safety analysis. In: Proceedings of European Safety and Reliability Conference, 2010.

[Zimmer et al. 2011] B. Zimmer, S. Bürklen, M. Knoop, J. Höfflinger, M. Trapp: Vertical safety interfaces - improving the efficiency of modular certification. In: Proceedings of the 30th International Conference of Computer Safety, Reliability, and Security, 2011.

[Zimmer et al. 2012] B. Zimmer, M. Trapp, P. Liggesmeyer, J. Höfflinger and S.Bürklen: Safety-focused deployment optimization in open integrated architectures. In: Proceedings of the 31st International Conference of Computer Safety, Reliability and Security, 2012.