SpARTA - State-Aware Fault Tree Analysis

Kai Höfig, Marc Zeller

Abstract—Conservative judgments are a common way to ease the models used for safety analysis. With the increasing complexity of safety-critical systems in domains such as automotive, healthcare, aerospace, transportation, energy or industry automation the potential for sources of hazards increases simply by the fact of increasing the number of participants required to perform a safety-critical function. An important challenge for modern safety analysis is the balancing act of keeping safety analysis models easy to comprehend and to develop and on the other hand manage the complexity of the analyzed system without using too conservative judgments for a realistic outcome of the analysis.
Therefore, we present here a methodology that allows to analyze systems with complex states using precise but comprehensible models avoiding too conservative judgments. For fault tree analysis, models exist that aim at a general analysis for a hazardous state. Complex components that provide more than simple closed loop functionality may have different operational modes that are active for different situations the system is in. In this case, the complexity of a system is not only expressed in the various number of components and their dependencies, but also in the states or modes the individual states the components are in for a specific situation of the system. We divide safety analysis models into different states or modes of a component for a state-aware fault tree analysis (SpARTA) of a system. Each component can have multiple states with a different failure behavior. The reaction of a complex system can be analyzed by identifying the active states of particular components in a certain situation.

Index Terms—safety analysis, component fault tree, compositional safety, model-based development, component-based development, embedded systems, cyberphysical systems.

1 Introduction

The precise development and analysis of safety analysis models is important during the development of safety-critical systems. Such models aim at identifying drawbacks or insufficiencies in terms of safety. The identification of such drawbacks is crucial for a safe system and a cost efficient development process. Conservative judgments are a common way to ease the models used for safety analysis. With the increasing complexity of safety-critical systems in domains such as automotive, healthcare, aerospace, transportation, energy or industry automation the potential for sources of hazards increases simply by the fact of increasing the number of participants required to perform a safety-critical function. While a sensor and an actor realized a safety function within a simple closed-loop operation in the past, there are today much more complex functions that sometimes spread over entire cyberphysical systems or entire countries, if we take energy distribution as an example. For such complexer systems the requirement to perform their safety-critical functionality remains in the same boundaries as simple closed-loop implementations. Here, conservative judgments lead to missed safety goals in safety analysis models and that in turn is a serious problem for the certification of such systems. So, an important challenge for modern safety analysis is the balancing act of keeping safety analysis models easy to comprehend and to develop and on the other hand manage the complexity of the analyzed system without using too conservative judgments for a realistic outcome of the analysis.

- *K. Höfig and M. Zeller are with Siemens AG, Corporate Technology, Munich, Germany.*
E-mail: {firstname}.{lastname}@siemens.com

Therefore, we present here a methodology that allows to analyze systems with complex states using precise but comprehensible models avoiding too conservative judgments. In many systems, the response of a system to a stimuli can vary depending on the *state* the system is currently in. For example, triggering an emergency brake of a transportation system in stop mode has *no effect* whereas the effect is *dramatically* if the system is in high-speed operation. For such a system, it makes sense to analyze the causes for a hazardous state separately for each system state, especially if causes from the one state are no courses for a hazard in another state. Doing so makes the analysis model more precise, the quantification or qualification of the hazardous state is more realistic and conservative judgments with enough safety margin can still be made since the states are separated in the analysis model.

For fault tree analysis, models exist that aim at a general analysis for a hazardous state. Modular or compositional safety analysis methodologies such as component fault trees [1] or HipHops [2] brake down the complexity for safety to smaller artefacts. In industry, development artefacts such as components or units, are often reused from existing artefacts to save time and costs. Changes are made to these artefacts to match the requirements for the new system. In software development such a reuse strategy is also known as *cone and own*. Complex components that provide more than simple closed loop functionality may have different operational modes that are active for different situations the system is in. In this case, the complexity of a system is not only expressed in the various number of components and their dependencies, but also in the states or modes the individual states the components are in for a specific situation of the system.

Therefore, we present here a methodology that is able

to divide safety analysis models into different states or modes of a component for a state-aware fault tree analysis (SpARTA) of a system. Each component can have multiple states with a different failure behavior. The reaction of a complex system can be analyzed by identifying the active states of particular components in a certain situation.

The rest of this document is structured as follows: first the related approaches are summarized in section 2 where we also briefly introduce the basic principles of component fault trees. Section 3 describes the central methodology of this document. In section 4, the SpARTA methodology is applied to our *SafeCar* demonstrator and section 5 summarizes this publication.

2 Related Work

The use of models in safety engineering processes has gained increasing attention in the last decade [3], [4], [5]. Specifically the idea is to support automatic generation of safety artifacts such as fault trees [6], [7], [8], [9], [10], [11], [12], [13], [14], [15], [16], [17], [18], [19], [20] or FMEA tables [21], [22], [23], [24], [25] from system models. Nevertheless, all these methodologies require precise models as input that have a similar complexity to the generated models. To construct the safety artefact the system models are often annotated with failure propagation models [1], [11], [15], [26], [27], [28]. These failure propagation models are commonly combinatorial in nature thus producing static fault trees. This is also driven by the industrial need to certify [29], [30], [31] their system with static fault trees. Only rarely more advanced safety evaluation models such as Dynamic Fault Trees (DFTs) [32], [33], [34], [35], Generalized Stochastic Petri Nets (GSPNs) [36], State-Event Fault Trees (SEFTs) [28], [37], [38] or Markov models [39], [40]. Beside annotating an architecture specification there is also their are also approaches to construct a safety artefact via model checking techniques [41], [42], [43], [44], [45].

In the development of a safety-critical system similar components may result in similar safety evaluation models. However, significant reuse of safety evaluation models is still rare and should be handled with great care. In fault trees a common way of reusing is to copy an entire subtree. However, Kaiser et al. [1], [28], [37], [38] realized that fault trees should be reused as subgraphs, and created Component Fault Trees and State-Event Fault Trees (SEFTs). The line of research results the ability to reuse an entire subgraph, via typing them and so called in- and out-failure ports that serve as interfaces to the rest of the fault tree. In another line of research Wolforth et al. [46], [47], [48] created a language extension for HiP-HOPS [15] that allow for pattern-based specification of reusable failure propagation models. As a result, reuse of safety models between similar components and from common patterns is possible. From the reused failure propagation models via the HiP-HOPS [15] methodology complete fault trees can be automatically constructed.

The approach presented here significantly differs from the presented approaches because we are focusing on a mode- or state-based analysis of the system. Where the current approaches analyze the system over all states, the methodology presented here allows to precisely model different states while braking complexity down to a component level.

2.1 Component Fault Trees

A component fault tree is a Boolean model associated to system development elements such as components [1]. It has the same expressive power as classic fault trees that can for example be found in [49]. As classic fault trees, also component fault trees are used to model failure behavior of safety-critical systems. This failure behavior is used to document that a system is safe and can also be used to identify drawbacks of the design of a system.

A separate *component fault tree element* is related to a component. Failures that are visible at the outport of a component are models using *Output Failure Modes* which are related to the specific outport. To model how specific failures propagate from an inport of a component to the outport, *Input Failure Modes* are used. The inner failure behavior that also influences the output failure modes is modeled using the gates *NOT*, *AND*, *OR*, and *Basic Event*.

Every component fault tree can be transformed to a classic fault tree by removing the input and output failure modes elements. Figure 1 shows on the left side a classic fault tree and on the right side a component fault tree. In both trees, the topevents or output events *TE1* and*TE2* are modeled. The component fault tree model allows, additionally to the Boolean formulae that are also modeled within the classic fault tree, to associate the specific topevents to the corresponding ports where these failures can appear. Topevent *TE1* for example appears at port *O1*. Using this methodology of components also within fault tree models, benefits during the development can be observed, for example an increased maintainability of the safety analysis model [50]. Nevertheless, distinguishing different states for each component is not feasible using classic component fault trees. In the next section, the basic principles of state-aware fault tree analysis are described to tackle this challenge.

3 State-Aware Fault Tree Analysis

In this section, we describe how the methodology of state-aware fault tree analysis (SpARTA) can be used to model the safety behavior of components under different conditions or states. Furthermore, we describe how the states of single components can be combined to a system-wide fault tree analysis maintaining a system-wide state vector over all components.

Figure 2 shows an example system with three components. The component fault tree of the component *processing* has instead of classic component fault tree elements, such as gates and basic events, two *mode* elements. These mode elements model the failure behavior of the component *processing* for two different modes. If component *processing* operates in mode *1*, a failure of type *b* will result in a failure at the outport *p5* of the component. If the component is in mode *2*, a failure of type *b* cannot occur, but a failure of type *c* will result in a failure visible at the out port *p5* of the component *processing*. So, if the system operates in a state where component *processing* operates in mode *1*,

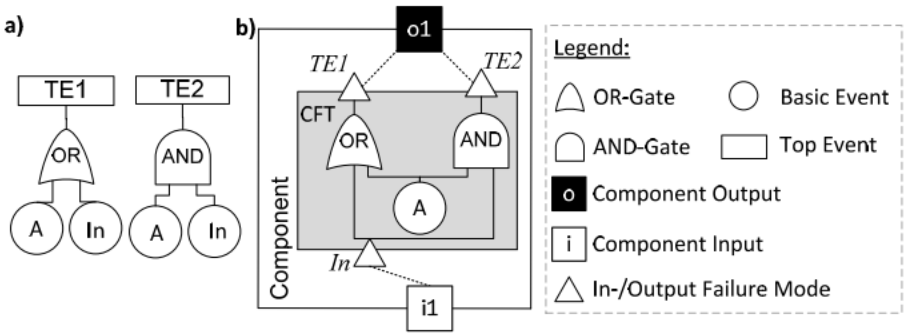

Fig. 1. Classic Fault Tree (a) and Component Fault Tree (b) [50].

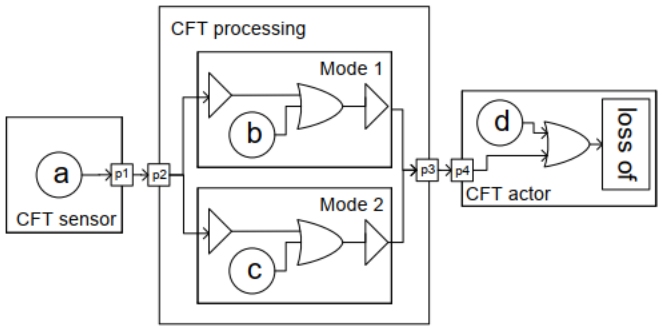

Fig. 2. Component fault tree model of three components and two different modes for the processing component.

the failure behavior of the system can be described by the fault tree depicted in figure 3. If the system operates in a state where component *processing* operates in mode 2, the failure behavior of the system can be described by the fault tree depicted in figure 4. If it is unclear, in which state the system is currently operating in, only the worst case can be assumed. As a result, both modes can be active and the failure behavior of the system can be expressed using the fault tree as depicted in figure 5.

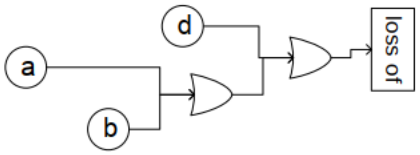

Fig. 3. Generated fault tree for the system state where mode 1 of component *processing* is active.

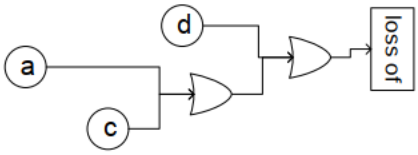

Fig. 4. Generated fault tree for the system state where mode 2 of component *processing* is active.

In the following, the methodology of state aware fault tree analysis (SpARTA) is described in detail. We show how

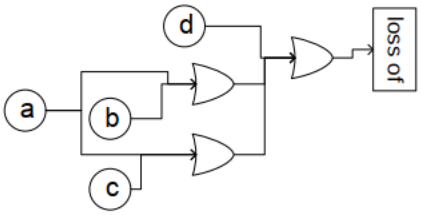

Fig. 5. Generated fault tree for the system state where both modes of component *processing* can be active.

modes can be used to generate component fault trees to represent a failure behavior for different states of a system.

Let $C = c_1, \ldots, c_n$ be the set of components of a system, $CFT = cft_1, \ldots, cft_m \cup \emptyset$ the set of component fault trees and

$$\tilde{CFT}(c) = cft \; with \; c \in C \; and \; cft \in CFT.$$

Let

$$IN(c) = in_1, \ldots, in_i, \text{ and } OUT(c) = out_1, \ldots, out_j$$

be the in- and outports of component c. Let

$$\overline{CON} = \{(out, in) \quad | \quad out \in OUT(c_1) \cup \ldots OUT(c_n), \; in \in IN(c_1) \cup \ldots \cup IN(c_n)\}$$

be the set of all possible port connections and

$$CON \subseteq \overline{CON}$$

be the set of actual port connections modeling the data flow from the outport of a component to the inport of another component.

Let

$$SPARTA(c) = x_1, \ldots, x_n$$

be the set of all modes of component c. If x is such a mode, it can be used to model the failure propagation from inports of c to outports of c using input and output failure modes. It is

$$I^x_{fm}(in) \neq \{\} \text{ and } O^x_{fm}(out) \neq \{\}$$

for an inport $in \in IN(c)$ and an outport $out \in OUT(c)$. In the example system as depicted in figure 2, the previously defined sets are as follows:

$$
\begin{aligned}
C &= sensor, processing, \\
&\quad actor \\
IN(sensor) &= \{\} \\
IN(processing) &= p_2 \\
IN(actor) &= p_4 \\
OUT(sensor) &= p_1 \\
OUT(processing) &= p_3 \\
OUT(actor) &= \{\} \\
CONN &= (p_1, p_2), (p_3, p_4) \\
SPARTA(sensor) &= \{\} \\
SPARTA(processing) &= \{1, 2\} \\
SPARTA(actor) = &= \{\}
\end{aligned}
$$

And the input and output failure modes related to the ports are

$$O_{fm}^{Mode\ 1}(p_3) = O_{fm}^{Mode\ 2}(p_3) = loss\ of,$$
$$I_{fm}^{Mode\ 1}(p_2) = I_{fm}^{Mode\ 2}(p_2) = loss\ of.$$

To use these sets for an analysis of a specific state of the system, each active mode needs to be addressed. If there is no mode available for a component, the mode is addressed using $*$. Also, if all modes of a component need to be included in the analysis, the symbol $*$ is used. For the set $C = c_1, \ldots, c_n$, the set

$$STATE(C) = \{(m_1, \ldots, m_n) \quad | \quad m_i \in SPARTA(c_i) \cup *, \\ i = 1, \ldots, n\}$$

describes all possible states of the system that are expressed using SpARTA.

For the example system, this set is as follows:

$$STATE(C) = (*, 1, *), (*, 2, *), (*, *, *)$$

Using these sets and relationships, a fault tree model can be generated from the component fault tree elements and the SpARTA modes that reflects the failure behavior of specific system states. For a state $s = m_1, \ldots, m_n \in STATE(C)$, the failure behavior for component C_i is defined by m_i and it is for every inport in and outport out of this component

$$I_{fm}(in) = I_{fm}^{m_i}(in) \text{ and } O_{fm}(out) = O_{fm}^{m_i}(out)$$

.

If $m_i = *$ and $\{m_j, \ldots, m_k\} = SPARTA(c_i)$, it is

$$I_{fm}(in) = \bigcup_{j=1}^{k} I_{fm}^{m_j}(in)$$

and

$$O_{fm}(out) = \bigcup_{j=1}^{k} O_{fm}^{m_j}(out)$$

.

If $m_i = *$ and $SPARTA(c_i) = \emptyset$, the failure behavior of component c_i is defined by its component fault tree instead.

Using this methodology, the SpARTA elements are used to generate classic fault tree elements for each mode. Each mode can now be analyzed using established component fault tree analysis methods. In the next section, this methodology is applied to a demonstrator vehicle.

4 Case Study

In this section, a case study from the automotive domain is presented to demonstrate how the SpARTA approach can be used to enable the methodology of component fault trees for a system with different modes. The system presented here is a radio-controlled demonstrator vehicle as depicted in figure 6.

Fig. 6. Demonstrator vehicle.

Figure 7 shows the architecture of a system as a SysML internal block diagram that uses two redundant ultra sonic sensors and two redundant infrared sensors to enable an emergency braking functionality in a radio controlled car. The vehicle operates in two modes. In *desk* mode, the infrared sensors at the bottom of the car detect the end of a desktop to prevent the car from falling down to the floor. In *floor* mode, the ultrasonic sensors at the front of the car prevent from driving into a wall.

The components *RadioReceiver* (R), *EmergencyBrakingControl* (EBC), *Engine* (E) and *Steering* (S) are used to control the car and perform an emergency braking function based on the inputs of the two sensors and the selected mode (floor or desk). The functionality of the emergency braking software component is basically to transmit the steering and throttle commands form the radio receiver to the steering and engine actors. If one of the pairs of sensors detects an obstacle, the throttle and steering signals are no longer forwarded to the actors. Depending on the mode, either the ultrasonic sensors (floor mode) or the infrared sensors (desk mode) are evaluated. The car is then set into emergency braking operation, where the actors are used to safely brake the car and omit forward moving signals for a certain time span.

Both pairs of sensors are a source for random faults leading to the hazard of a crash. In classic fault tree analysis, both sources are typically causes for the hazard. But since the car is exclusively either in the floor or in the desk mode, the top-event is vastly overestimated. Classic fault trees provide solutions to model this behavior. One option

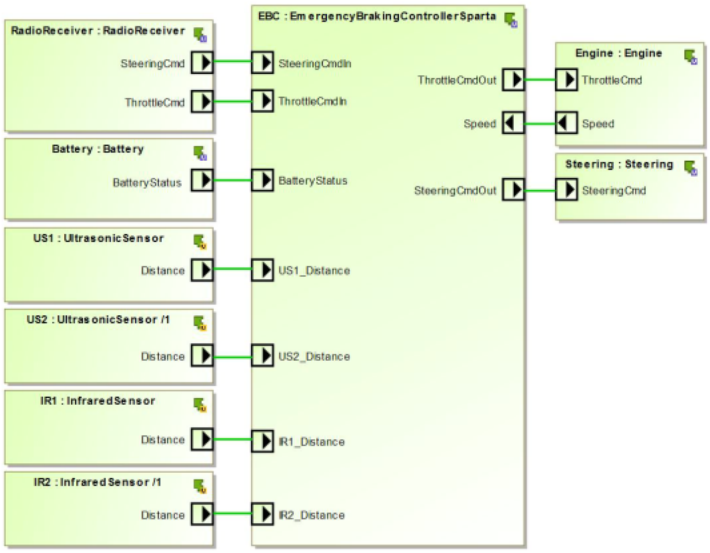

Fig. 7. Architecture system model as a SysML internal block definition diagram.

is to use XOR gates. Another option would be to dividing the fault tree analysis into two distinct top-events, one for the floor mode and one for the desk mode. But if other components in the system also have different modes, both options result in a large fault tree that can be quite complex, hard to maintain and hard to understand, especially when even more top-events are involved. Furthermore, this does not solve the problem if the system has to be analyzed only for a *certain* system state and not the maximum over all existing states.

Figure 8 shows the SpARTA component fault tree for the component *EmergencyBrakingControl* in mode *TableDrivingMode*. Two top level hazards or failure modes for the component *EmergencyBrakingControl* are modeled for the port *ThrottleCmdOut* (see also figure 7): *Brake-command-omission* and *Brake-command-commission*. The omission failure mode refers to the situation where accidentally no braking command is generated at the throttle command port of the emergency braking control component. The commission failure mode refers to the situation where accidentally a braking command is generated. Furthermore, Two top level hazards or failure modes are modeled for the port *SteeringCmdOut*: *Omission-steering_out* and *Commission-steering_out*. The omission failure mode refers to the situation where accidentally no steering command is forwarded to the steering component. The commission failure mode refers to the situation where accidentally a steering command is generated. The causes for the top level hazards to occur depend on the failure modes that are being generated by the other components in the system and therefore in this model depend on the input failure modes of other components (yellow triangles connected to inports). The SpARTA component fault tree for the component *EmergencyBrakingControl* in mode *FloorDrivingMode* is quite similar and therefore not depicted. The difference is, that the cause for the top level hazards *Brake-command-omission* and *Brake-command-commission* do not depend on the input failure modes of the inports *IR1_Distance* and *IR2_Distance* (like depicted in figure 8), but do depend on the input failure modes of the ports *US1_Distance* and *US1_Distance* instead.

According to the methodology as described in section 3, there are two SpaRTA elements modeled here: *TableDrivingMode* and *FloorDrivingMode* resulting in three different states: in a fist state, the tabel drivoing mode is active, in a second state the floor driving mode is active and in a third state, both driving modes are active. Each state can now be evaluated separately. Figure 9 shows the generated fault tree for the brake command commission top level hazard in the floor driving mode involving the ultrasonic sensors.

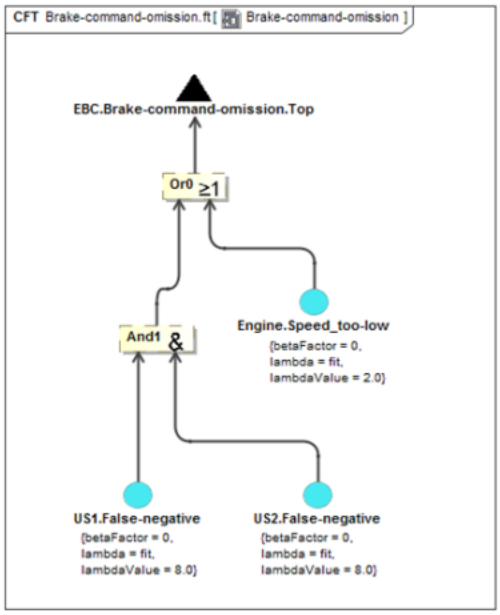

Fig. 9. Generated fault tree for the brake command commission top level hazard in the floor driving mode involving the ultrasonic sensors.

The fault tree for the brake command commission top level hazard in the table driving mode involving the infrared sensors has the same structure but involves the two causes from the infrared sensors instead and is therefore not depicted. The generated fault tree for the third SpARTA system state, where both driving modes are involved in the analysis results in the following logic:

EBC.Brake-command-omission ⇔
Engine.Speed_toolow ∨
(US1.False-negative ∧ US2.False-negative) ∨
(IR1.False-negative ∧ IR2.False-negative)

Using SpARTA model elements, the different states for table driving mode and floor driving mode can be analyzed individually. Comparing the analysis results as depicted in figure 9 for the analysis of the floor driving mode with the results of the classic analysis as described previously, it can be concluded, that using the SpARTA analysis method contains two basic events less than the classic analysis. Since these basic events, *IR1.False-negative* and *IR2.False-negative*, do not contribute to the top level hazard in floor driving mode, these basic events belong to a conservative judgment in the classic component fault tree approach.

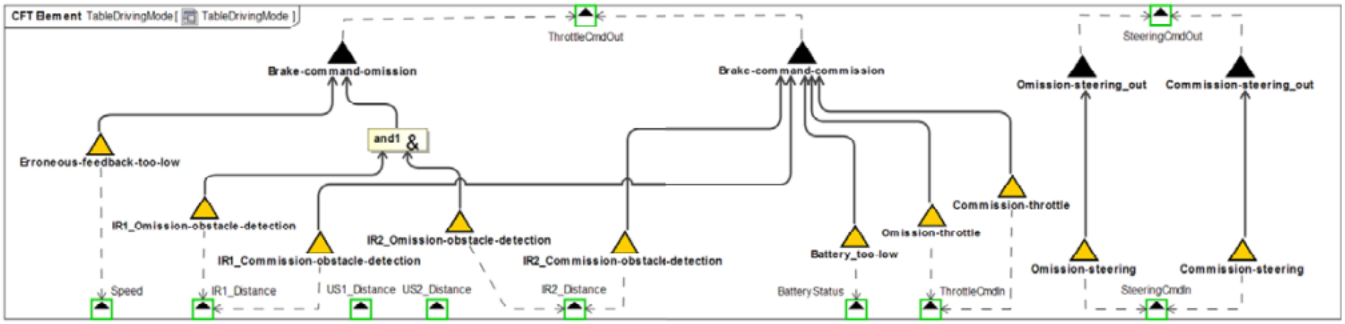

Fig. 8. SpARTA component fault tree for the component *EmergencyBrakingControl* in mode *TableDrivingMode*.

5 Conclusion

This paper describes the methodology of state-aware fault tree analysis, an extension of component fault trees to maintain comprehensible safety analysis models whilst manage the complexity of the analyzed system without using too conservative judgments for a more realistic outcome of the analysis. SpARTA allows to analyze systems with complex states using precise but comprehensible models avoiding too conservative judgments. Causes for a hazardous state can be analyzed separately for each system state. This prevents conservative judgments for independent states. As demonstrated in section 4 the analysis model becomes more precise, the quantification or qualification of the hazardous state is more realistic and conservative judgments with enough safety margin can still be made since the states are separated in the analysis model.

References

[1] Bernhard Kaiser, Peter Liggesmeyer, and Oliver Mäckel. A new component concept for fault trees. In *SCS '03: Proceedings of the 8th Australian workshop on Safety critical systems and software*, pages 37–46, Darlinghurst, Australia, 2003. Australian Computer Society, Inc.

[2] Yiannis Papadopoulos and John A. McDermid. Hierarchically Performed Hazard Origin and Propagation Studies. *Computer Safety, Reliability and Security*, 1999.

[3] A. Joshi, S. P. Miller, M. Whalen, and M. P. E. Heimdahl. A proposal for model-based safety analysis. *24th AIAA/IEEE Digital Avionics Systems Conference*, 2005.

[4] O. Lisagor, J. A. McDermid, U. K. York, and D. J. Pumfrey. Towards a Practicable Process for Automated Safety Analysis. *24th International System Safety Conference*, 2006.

[5] John McDermid and Tim Kelly. Software in Safety Critical Systems: Achievement and Prediction, 2006. University of York, UK.

[6] Rasmus Adler, Marc Förster, and Mario Trapp. Determining Configuration Probabilities of Safety-Critical Adaptive Systems. In *21st International Conference on Advanced Information Networking and Applications (AINA 2007),*, pages 548–555. IEEE Computer Society, 2007.

[7] Andrea Bondavalli, Istvan Majzik, and Ivan Mura. Automated Dependability Analysis of UML Designs. *IEEE International Symposium on Object-oriented Real-time distributed Computing*, 2, 1999.

[8] J.-L. Boulanger and Van Quang Dao. Experiences from a model-based methodology for embedded electronic software in automobile. pages 1–6, April 2008.

[9] M. Bretschneider, H. J. Holberg, E. Bode, and I. Bruckner. Model-based safety analysis of a flap control system. *Proc. 14th Annual INCOSE Symposium*, 2004.

[10] Holger Giese, Matthias Tichy, and Daniela Schilling. Compositional hazard analysis of uml component and deployment models. In Maritta Heisel, Peter Liggesmeyer, and Stefan Wittmann, editors, *Computer Safety, Reliability, and Security, 23rd International Conference, SAFECOMP 2004, Potsdam, Germany, September 21-24, 2004, Proceedings*, volume 3219 of *Lecture Notes in Computer Science*, pages 166–179. Springer, 2004.

[11] Lars Grunske. Towards an Integration of Standard Component-Based Safety Evaluation Techniques with SaveCCM. In Christine Hofmeister, Ivica Crnkovic, and Ralf Reussner, editors, *Second Int. Conf. on Quality of Software Architectures, QoSA 2006,*, volume 4214 of *LNCS*, pages 199–213. Springer, 2006.

[12] Lars Grunske and Bernhard Kaiser. Automatic generation of analyzable failure propagation models from component-level failure annotations. In *Fifth International Conference on Quality Software (QSIC 2005), 19-20 September 2005, Melbourne*, pages 117–123. IEEE Computer Society, 2005.

[13] Anjali Joshi, Steve Vestal, and Pam Binns. Automatic Generation of Static Fault Trees from AADL Models. In *DSN Workshop on Architecting Dependable Systems*, Lecture Notes in Computer Science. Springer, 2007.

[14] M. A. de Miguel, J. F. Briones, J. P. Silva, and A. Alonso. Integration of safety analysis in model-driven software development. *Software, IET*, 2(3):260–280, June 2008.

[15] Y. Papadopoulos, J. A. McDermid, R. Sasse, and G. Heiner. Analysis and synthesis of the behaviour of complex programmable electronic systems in conditions of failure. *Int. Journal of Reliability Engineering and System Safety*, 71(3):229–247, 2001.

[16] Y. Papadopoulos and M. Maruhn. Model-Based Automated Synthesis of Fault Trees from Matlab.Simulink Models. *International Conference on Dependable Systems and Networks*, 2001.

[17] Andrew Rae and Peter Lindsay. A behaviour-based method for fault tree generation. *Proceedings of the 22nd International System Safety Conference*, pages 289 – 298, 2004.

[18] G. Szabo and G. Ternai. Automatic Fault Tree Generation as a Support for Safety Studies of Railway Interlocking Systems. *IFAC Symposium on Control in Transportation Systems*, 2009.

[19] N. Mahmud, M. Walker, and Y. Papadopoulos. Compositional synthesis of temporal fault trees from state machines. In *Availability, Reliability and Security (ARES), 2011 Sixth International Conference on*, pages 429–435, Aug 2011.

[20] Martin Walker and Yiannis Papadopoulos. Qualitative temporal analysis: Towards a full implementation of the fault tree handbook. *Control Engineering Practice*, 17(10):1115 – 1125, 2009.

[21] Tadeusz Cichocki and Janusz Górski. Failure mode and effect analysis for safety-critical systems with software components. In Floor Koornneef and Meine van der Meulen, editors, *Computer Safety, Reliability and Security, 19th International Conference, SAFECOMP 2000, Rotterdam, The Netherlands, October 24-27, 2000, Proceedings*, volume 1943 of *Lecture Notes in Computer Science*, pages 382–394. Springer, 2000.

[22] Tadeusz Cichocki and Janusz Górski. Formal support for fault modelling and analysis. In Udo Voges, editor, *Computer Safety, Reliability and Security, 20th International Conference, SAFECOMP 2001, Budapest, Hungary, September 26-28, 2001, Proceedings*, volume 2187 of *Lecture Notes in Computer Science*, pages 190–199. Springer, 2001.

[23] Pierre David, Vincent Idasiak, and Frederic Kratz. Towards a Better Interaction Between Design and Dependability Analysis: FMEA Derived From UML/SysML Models. In *Safety, Reliability and Risk Analysis: Theory, Methods and Applications*, pages 2259–2266, Jan. 2008.

[24] Y. Papadopoulos, D. Parker, and C. Grante. Automating the failure modes and effects analysis of safety critical systems. In *Int. Symp. on High-Assurance Systems Engineering (HASE 2004)*, pages 310–311. IEEE Comp. Society, 2004.

[25] M. Walker, Y. Papadopoulos, D. Parker, and H. Lnn et al. Semi-automatic fmea supporting complex systems with combinations and sequences of failures. *SAE Int. J. Passeng. Cars - Mech. Syst. 2(1)*, pages 791–802, 2009.

[26] Dominik Domis and Mario Trapp. Integrating Safety Analyses and Component-Based Design. In *SAFECOMP*, pages 58–71, 2008.

[27] Jonas Elmqvist and Simin Nadjm-Tehrani. Safety-Oriented Design of Component Assemblies using Safety Interfaces. *Formal Aspects of Component Software*, 2006.

[28] Bernhard Kaiser. *State/Event Fault Trees: A Safety and Reliability Analysis Technique for Software-Controlled Systems*. PhD thesis, Technische Universität Kaiserslautern, Fachbereich Informatik, 2005.

[29] CENELEC EN 50126,128,129. CENELEC (European Committee for Electro-technical Standardisation) : Railway Applications – the specification and demonstration of Reliability, Availability, Maintainability and Safety, Railway Applications – Software for Railway Control and Protection Systems, Brussels, 2000.

[30] IEC61508. International Standard IEC 61508, 1998. International Electrotechnical Commission (IEC).

[31] ISO 26262. ISO/DIS 26262- Road vehicles Functional safety , 2009.

[32] S. Amari, G. Dill, and E. Howald. A new approach to solve dynamic fault trees. In *Reliability and Maintainability Symposium, 2003. Annual*, pages 374–379, 2003.

[33] Joanne Bechta-Dugan, Salvatore Bavuso, and Mark Boyd. Dynamic fault-tree models for fault-tolerant computer systems. *IEEE Transactions on Reliability*, 41(3):363–77, sep 1992.

[34] JOSH Dehlinger and Joanne Bechta Dugan. Analyzing dynamic fault trees derived from model-based system architectures. *Nuclear Engineering and Technology: An International Journal of the Korean Nuclear Society*, 40(5):365–374, 2008.

[35] P. Ganesh and J.B. Dugan. Automatic Synthesis of Dynamic Fault Trees from UML SystemModels. *13th International Symposium on Software Reliability Engineering (ISSRE)*, 2002.

[36] Ana-Elena Rugina, Karama Kanoun, and Mohamed Kaâniche. A System Dependability Modeling Framework Using AADL and GSPNs. In *Architecting Dependable Systems IV*, volume 4615 of *LNCS*, pages 14–38. Springer, 2007.

[37] Lars Grunske, Bernhard Kaiser, and Yiannis Papadopoulos. Model-driven safety evaluation with state-event-basedcomponent failure annotations. In *8th Int. Symp. on Component-Based Software Engineering, CBSE 2005, Proc.*, pages 33–48, 2005.

[38] Bernhard Kaiser, Catharina Gramlich, and Marc Förster. State/event fault trees–A safety analysis model for software-controlled systems. *Reliability Engineering & System Safety*, 92(11):1521 – 1537, 2007. SAFECOMP 2004, the 23rd International Conference on Computer Safety, Reliability and Security.

[39] Marco Bozzano, Alessandro Cimatti, Joost-Pieter Katoen, Viet Yen Nguyen, Thomas Noll, and Marco Roveri. Safety, dependability and performance analysis of extended aadl models. *Comput. J.*, 54(5):754–775, 2011.

[40] Marco Bozzano, Alessandro Cimatti, Joost-Pieter Katoen, Viet Yen Nguyen, Thomas Noll, and Marco Roveri. The compass approach: Correctness, modelling and performability of aerospace systems. In Bettina Buth, Gerd Rabe, and Till Seyfarth, editors, *Computer Safety, Reliability, and Security, 28th International Conference, SAFECOMP 2009, Hamburg, Germany, September 15-18, 2009. Proceedings*, volume 5775 of *Lecture Notes in Computer Science*, pages 173–186. Springer, 2009.

[41] Lars Grunske, Robert Colvin, and Kirsten Winter. Probabilistic model-checking support for FMEA. In *Fourth International Conference on the Quantitative Evaluaiton of Systems (QEST 2007)*, pages 119–128. IEEE Computer Society, 2007.

[42] Matthias Güdemann and Frank Ortmeier. A framework for qualitative and quantitative formal model-based safety analysis. In *12th IEEE High Assurance Systems Engineering Symposium, HASE 2010, San Jose, CA, USA, November 3-4, 2010*, pages 132–141. IEEE Computer Society, 2010.

[43] M. Gündemann, F. Ortmeier, and W. Reif. Using Deductive Cause-Sequence Analysis (DCCA) with SCADE. *SAFECOMP 2007, LNCS 4680*, pages 465–478, 2007.

[44] Mats Per Erik Heimdahl, Yunja Choi, and Michael W. Whalen. Deviation analysis: A new use of model checking. *Automated Software Engineering*, 12(3):321–347, 2005.

[45] Michael Lipaczewski, Simon Struck, and Frank Ortmeier. Using tool-supported model based safety analysis - progress and experiences in saml development. In *14th International IEEE Symposium on High-Assurance Systems Engineering, HASE 2012, Omaha, NE, USA, October 25-27, 2012*, pages 159–166. IEEE Computer Society, 2012.

[46] Ian Wolforth, Martin Walker, Lars Grunske, and Yiannis Papadopoulos. Generalizable safety annotations for specification of failure patterns. *Softw., Pract. Exper.*, 40(5):453–483, 2010.

[47] Ian Wolforth, Martin Walker, and Yiannis Papadopoulos. A language for failure patterns and application in safety analysis. In *IEEE Conference on Dependable Computing Systems (DEPCOSA08)*. IEEE Computer Society, 2008.

[48] Ian Wolforth, Martin Walker, Yiannis Papadopoulos, and Lars Grunske. Capture and reuse of composable failure patterns. *IJCCBS*, 1(1/2/3):128–147, 2010.

[49] William Vesely, Joanne Dugan, Joseph Fragola, Joseph Minarick, and Jan Railsback. Fault Tree Handbook with Aerospace Applications, 2002. NASA Office of Safety and Mission Assurance.

[50] Jessica Jung, Andreas Jedlitschka, Kai Hfig, Dominik Domis, and Martin Hiller. A controlled experiment on component fault trees. In Friedemann Bitsch, Jrmie Guiochet, and Mohamed Kaniche, editors, *Computer Safety, Reliability, and Security*, volume 8153 of *Lecture Notes in Computer Science*, pages 285–292. Springer Berlin Heidelberg, 2013.