Automatic Generation of Analyzable Failure Propagation Models from Component-Level Failure Annotations

Lars Grunske
School of Information Technology and Electrical Engineering, University of Queensland, Brisbane, QLD 4072, grunske@itee.uq.edu.au

Bernhard Kaiser
Fraunhofer IESE, Sauerwiesen 6, 67661 Kaiserslautern, Germany, bernhard.kaiser@iese.fraunhofer.de

Abstract

Model-driven and component-based software engineering methodologies are currently key factors for the successful construction of complex software systems. To effectively apply these methodologies to mission- and safety-critical systems, component-based models should also support hazard analysis techniques and enable the automatic construction of safety cases. This paper outlines a technique, which annotates components with modular failure mode assumptions, described in the Failure Propagation Transformation Notation (FPTN) and generates an analyzable failure propagation model for the complete system. Based on this technique, a model-based safety evaluation is possible, which enables the automatic generation of safety cases based on system models. Consequently, a consistency between the safety case and the system model can be ensured, even if the system's architecture is changed.

Keywords: Safety analysis, System architectures, System analysis, Failure Propagation Transformation Notation, Component Fault Trees

1 Introduction

The control functionality of complex technical systems, such as medical systems, military aircraft and railway signalling, has slowly migrated from the hardware to the software domains. Together with the need for increased functionality, this has led to an increase in system complexity and the development of such systems has become more challenging.

It is currently accepted that a paradigm shift from code-driven to model-driven software engineering is required to tackle this problem [27]. In model-driven software engineering as recently proposed by the Object Management Group (www.omg.org) [23], the central artifacts used to develop the system are models, rather than program code as used in traditional (code driven) software engineering [10]. The used models describe the high-level functionality of the system (Platform Independent Models (PIMs)) and support code generation for various target platforms (platform specific models (PSMs)). The main objective of this paradigm shift is to increase the level of abstraction above programming languages and hardware platforms. This reduces the details required and brings the model closer to the problem domain. In this sense, model-driven software engineering is a natural continuation of the transition from assembly languages to high-level programming languages (third generation languages 3GL) which started in the 1960s.

Although there are many approaches to apply model-driven techniques to embedded and safety-critical systems [1, 14], the overall application of hazard and safety analysis techniques for these systems remain manual processes, which are performed by expert analysts [3, 15]. Consequently, to increase the suitability of model-driven software engineering processes for mission- and safety-critical systems, the involved models should also support hazard analysis techniques and enable an automatic or semi-automatic construction of safety cases.

We present in this paper a method that annotates system components with FPTN (Failure Propagation Transformation Notation) -modules [9, 8]. These FPTN-modules describe how failure modes of incoming messages together with internal faults of the components propagate to failure modes of outgoing messages. Since the message flow between components can be identified in the behavior model, we are also able to construct inter-component failure flows. To analyze these inter-component failure flows we use...
Component Fault Trees (CFTs) [18], which can be composed automatically [7] and can be used to perform the quantitative analysis of the system-level failure.

The benefits of the proposed model-driven safety evaluation method are the following:

- Reduction of errors compared to the manual construction of safety cases
- Cost reduction through automation
- Shorter time to market
- Consistency between system models and safety cases, if a new safety case is constructed when the system model evolves

The remainder of this paper is organized as follows: Section 2 introduces the used notations (FPTN and CFT). The main part of this paper is section 3, which gives an overview over the process and the transformations needed to automatically generate component fault trees from component-level failure annotations. To show the practicability of the process, the introduced techniques and transformations are applied in section 4 to the well known steam boiler example. Finally, section 5 discusses related work and section 6 contains concluding remarks and points out the directions for future work.

2 PRELIMINARIES

Before the concept of our model-driven safety evaluation process is described in detail, we first introduce the necessary terms in this section.

2.1 Failure Propagation Transformation Notation (FPTN)

The Failure Propagation and Transformation Notation (FPTN) [8, 9] is a simple, modular notation for the specification of the failure behavior of components.

The basic entity of the FPTN is a FPTN-module. This FPTN-module contains a set of standardized sections. In the first section (the header section), for each FPTN-module an identifier (ID), a name and a criticality level (SIL - Safety Integrity Level [5]) is given. The second section specifies the propagation of failures, the transformation of failures, the generation of internal failures and the detection of failures in the component. The context of FPTN-modules enumerates all failures in the environment that can affect the component and all failures of the component that can affect the environment. These failures are denoted as incoming and outgoing failures and are classified by the following failure categorization [4]:

- reaction too late(tl),
- reaction too early(te),
- value failure(v),
- commission(c),
- omission(o).

In the example, which is given in Figure 1, the incoming failures are $A:tl, A:te, A:v$ and $B:v$, and the outgoing failures are $C:tl, C:v, C:c$ and $C:o$. The propagation and transformation of failures is specified inside the module with a set of equations or predicates (e.g. for propagation: $C:tl=E:tl$ and for transformation $C:te=A:te && A:v$ and $C:v=A:tl || B:v$). Furthermore, a component can also generate a failure (e.g. $C:o$) or handle an existing failure (e.g. $B:v$). Consequently, it is necessary to specify a failure cause or a failure handling mechanism and a probability.

![Figure 1. Abstract FPTN-Module](image)

2.2 Component Fault Trees (CFT)

Fault Trees [16, 6] are one of the most popular notations for quantitative safety analysis [15]. However, as mentioned in [18], they unfortunately only provide a restricted decomposition mechanism: the decomposition into independent subtrees. To be compatible to a system model, which will serve for automatic construction of the safety case, the models for the failure behavior must be attachable to the components. They must take into account that components are in general not independent from each other. Interfaces are access points for possible influences from other components. Traditional FTs are compositional in the sense that independent subtrees can be cut off and handled separately, but as technical components are typically influenced by other components, the independence assumption is not practical. To allow for a modularization that corresponds to the component and interface (port) concept similar to the one introduced by [28], an extension of FTs has recently been proposed [18]. This extension is called Component Fault Trees.
(CFTs) and allows defining partial Fault Trees that reflect the actual technical components. These CFTs can be modelled and archived independently from each other. Input and output failure ports glue these parts together. While traditionally independent subtrees were regarded as compound events, CFTs are treated as a set of propositional formulas describing the truth-values of each output failure port as a function of the input failure ports and the internal events. CFTs can be acyclic graphs with one or more output failure ports. Each component constitutes a namespace, hides all internal failure events from the environment and can be instantiated in different projects. Thus, all necessary preconditions for an application of FTA (Fault Tree Analysis [16]) to component-based systems are fulfilled.

In figure 2 an abstract fault tree module is given, which contains two output and two input failure ports. These failure ports are graphically represented by open (inputs) or closed (outputs) triangles. Similar to the FPTN the input and output failures are typed by a failure categorization (reaction too late(tl), reaction too early(te), value failure(v), commission(c) and omission(o)). Finally, the internal faults are graphically represented as a circle, similar to ordinary fault trees (c.p. [6] [16]).

![Figure 2. Abstract CFT-Module](image)

### 3 Construction of a Safety Case for a Component-based System

To introduce the idea of model-based safety evaluation with FPTNs and CFTs, we will first introduce the basic process model and then we will describe the steps used for the construction in detail.

#### 3.1 Basic Process

The basic process contains the following steps:

1. Construct FPTN-Modules for all components in the system

2. Generate a CFT for each FPTN-Module and attach the CFT to the component type

3. Hierarchically construct and connect the input and output failure ports of the CFTs of all components with respect to the fault propagation between the components and the composition hierarchy

4. Analyze the top-level CFT1

#### 3.2 Identification of FPTN-modules

A recommendable way to identify component-level failure annotations is the Interface Focused Failure Mode and Effect Analysis (IF-FMEA) [25]. This IF-FMEA explores in a structured process the provided and required services of a component for possible failures. The causes for each possible failure of a provided service are represented as logical combinations of internal malfunctions and failures of a required service. This investigation is complemented by a forward search that finds the consequences of each possible failure of a required service. By alternating the application of these two search directions the emergence, propagation, mitigation and detection of failures in the component can be identified. These causal chains can then be represented with an FPTN-module.

Additionally, for each internal fault/failure, an appropriate probability function or measure must be specified. For hardware components there are mature models to do so, whereas for software components the determination of the failure probabilities is a complex task. There is a growing research body regarding failure probability estimation for software [3]. In industrial projects, this estimation is often based on expert knowledge. Estimation can also refer to the process model and the quality assurance techniques used [5]. Another method is to use empirical reliability growth models and testing results to forecast the probability of internal failures [22].

#### 3.3 Construction of CFTs from FPTN-modules

Generally, two steps are needed to construct a CFT from an FPTN-module. The first step is to create a CFT-frame with the same name as the FPTN-module. This CFT-frame contains all input and output failure ports that are relevant for the component. These failure ports are created based on the incoming and outgoing failures of the FPTN-module.

The second step is used to fill this frame with all internal specifications needed for the CFT. To construct this information the internal section of the FPTN-module (propagation of failures, transformation of failures, generation of

1To be able to analyze the system, the top-level CFT need to be free of cycles. Consequently, before this step the CFTs must be checked for cycles. If a CFT contains cycles, manual steps need to be performed to remove them.
internal failures and detection of failures in the component) need to be parsed. The section generation of internal failures must be investigated first and an internal fault event has to be created in the CFT-frame for each internal failure. The probability of an internal failure event in the CFT is equal to the probability of the internal failures in the FPTN-module. After this step, the section propagation of failures and the transformation of failures are parsed and based on the logical formulas, a fault tree is created between the output failure ports, the input failure ports and the internal failure events in the CFT-frame. Finally, the FPTN-section detection of failures is investigated. For each failure detection mechanism in this FPTN-part a binary AND-gate needs to be included and connected with a newly created internal failure event that specifies the failure handling mechanism.

3.4 Hierarchically Constructing and Wiring the CFTs

In this subsection we sketch an algorithm that recursively constructs CFTs for hierarchical components or the complete system based on the structure specification and the CFTs of all used component-classes. For a complete description of the algorithm we refer to [12, 13].

The algorithm generally contains three steps. In the first step, a new and empty CFT is created, that will describe the failure behavior of the considered hierarchical component or the complete system. By iterating over all subcomponents or subsystems in the system model (structure specification), the CFTs of all subcomponents are embedded into this new CFT. This is a simple step, if the component contains only flat subcomponents, because the CFTs have already been created from the FPTN-modules. If the component contains hierarchical subcomponents, the CFT of these hierarchical components must be constructed first, before they can be embedded into the new CFT. This is the reason for the recursive nature of the algorithm. In the second step the input and output failure ports of the embedded CFTs must be connected according to possible failure propagations in the system. For this, all communications or other dependencies between the components or subsystems must be investigated. Each time one component uses a service from another component or can be influenced by the other component, it is checked if the provider CFT contains an output failure port and the user CFT contains an input failure port with matching failure modes. In this case, both failure ports are connected by an edge in the newly created CFT. In the third step, the failure propagation between sub-CFTs and the environment of the enclosing component must be created. To do this, each unconnected input failure port of the embedded fault trees is identified. Then for each of these, an input failure port with the same name and failure mode is added to the new CFT and is connected with input failure ports of the embedded CFT. In a similar way, a new output failure port is added and connected in the new CFT.

3.5 Analyzing the CFTs

The result of the previous steps is a component fault tree for the complete system. This fault tree can be used to analyze the failure probability of each system level output failure port. To determine the probability of a system-level hazard, the analyst must further specify which output failures or which combinations of output failures lead to hazards. For that reason, the analyst must manually mark the output failure port that is the target for the analysis. Of course FT gates can be used to express that a combination (e.g. AND, OR) of output failures leads to the hazard; this proceeding is similar to the construction of a normal fault tree.

The quantitative analysis of the resulting fault tree (actually it is a directed acyclic graph) can be performed by any of the known algorithms [13]. Therefore, the hierarchy has to be resolved (flattened) and events that affect the top-event by several paths (repeated events) have to be split. However, it is more efficient to apply a modified BDD algorithm that exploits the component hierarchy and translates each component only once into a Binary Decision Diagram (BDD), no matter how often it is instantiated, and performs an insertion operation on the BDD fragments afterwards. This algorithm has been described in [19].

4 Example

To explain how a safety evaluation and the construction of component fault trees works in practice, we use a steam boiler system (depicted in figure 3) as a small case study. The left part of this figure shows a schematic diagram, which is usual for engineers to describe the hardware of a system. The diagram incorporates the steam boiler, a triple-redundant pressure sensor and a double-redundant safety valve. Further, the system contains a software controller that implements a two-out-of-three voting for the sensors and gives command to open both valves if the pressure is higher than the allowable level. The voter pattern assures that if at least two out of the three sensors indicate the right value, the controller takes the correct decision. Furthermore, each of the valves is sufficient as a pressure relief; so if one fails, the system is still safe.

The right part of the figure shows a structure diagram, as an embedded systems engineer would use it to describe the system. The structure diagram describes the static architecture of a system, consisting of components and interconnections between them. During the design phase, models for the behavior are attached to these components; for example state machines that describe the reaction of components to trigger signals received via its ports [28]. During
the construction phase, only the intended behavior is of relevance, safety analysis rather focuses on possible deviations from the intended behavior that lead to hazardous situations. As the interfaces (ports) and interconnections of the components are the spots where information is exchanged they can also be used to identify the paths where failures propagate between components.

Based on the structural model of the system an expert team can produce FPTN-modules for each component of the system by performing the already mentioned IF-FMEA (Interface-Focused FMEA). The result of this IF-FMEA could be the three FPTN-modules that are depicted in figure 4.

These three FPTN-modules are used to create a CFT for each component, strictly following the sketched algorithm in section 3.3. These CFTs are presented in figure 5.

In the last step these component-level CFTs must be instantiated and connected (wired) in the system-level CFT. The resulting component fault tree is shown in figure 6. The lower part of the structure has been generated automatically, the top-event and the AND gate have been added manually by the user. The AND gate attached to the failure output ports V1open.omission and V2open.omission specifies that if both valves fail to open when expected, the hazard to be examined is present. Assuming all events have a constant failure probability of 0.1, we calculated the hazard probability of 0.10049 using the tool UWG3, which was developed within the ESSaReL-project [7].

5 Related Work

The model-based safety-evaluation process which is presented in this paper is related (1) to other fault tree generation algorithms and (2) to quality evaluation approaches for other non-functional properties in the field of model-driven software engineering.

Current fault tree generation algorithms are focused on specific system artifacts and modelling languages. As an example [20] generates fault trees based on finite state machines and model checking. Papadopoulos et al. describe in [25, 24] the HiPHOPS methodology, which enables systematic fault tree generation based on Matlab-Simulink models. Grunske [12] and Giese et al. [11] have independently developed an approach to generate Fault Trees from UML 2.0 component and deployment diagrams and ROOM specifications [28]. However, in contrast to these approaches, our approach aims to identify a general methodology that can be applied to different system artifacts and modelling languages.

Model-driven approaches are currently also used to evaluate other non-functional properties like security [17, 21], reliability [26], and performance [2]. Similar to the model-driven safety evaluation method presented in this paper, all these approaches annotate the original system and component model with assumptions to reason about the fulfillment of the non-functional requirements. As an example model-driven performance evaluation techniques use annotations such as Queuing Networks, Petri Nets, Process Algebras, and Markov Processes [2].
6 Conclusion and Outlook

In this paper, we have proposed a technique for the automatic generation of component fault trees from component-level failure annotations. The basic procedure of this technique is to first annotate system components with FPTN-modules, an informal specification formalism to describe modular failure behaviors. Then, the FPTN modules are translated into analyzable CFT-specifications. Based on the data and information flow in arbitrary system models a CFT for the complete system can be constructed automatically. This system-level CFT allows for the evaluation of hazards or system-level failure probabilities. Our approach supports the current MDA-vision, if the used system specifications and analysis models (e.g., system architectures and component fault trees) are UML and MOF 2.0 compliant. Both can easily be archived by typing the system model- and CFT-elements with MOF meta-classes (e.g., structured classifier, structured class, connection and port).

The benefits of this procedure are twofold. On one hand it is the simple specification formalism of the local failure behavior with the failure propagation and transformation notation, together with their familiarity to system safety engineers. On the other hand, it is the effective tool support, which is provided for evaluating component fault trees. Consequently, this paper combines the strength of both notations: failure propagation and transformation notation and component fault trees.

The described method is subject to current research and we have identified some potential for new research directions. A fundamental improvement is the research for better models to estimate the probabilities of internal failures, in particular for software components. Up to now, predictions depend strongly on expert knowledge and premature models.

Additionally, this method would benefit from a component library that contains FPTN-module/CFTs for recurring component classes and readily connected models for recurring patterns in safety critical systems. With respect to the analysis performance we tend to exploit the component structure to reduce the analysis workload, especially if components are reused very often. This can be achieved for example by caching presimplified BDD versions of the Boolean structure of CFTs.

Finally, an appropriate algorithm is needed to remove cycles in the initial FPTN specification, in order to enable the evaluation of the resulting CFTs.

References


