Swinburne Research Bank
http://researchbank.swinburne.edu.au

Grunske, L., Kaiser, B., & Reussner, R. H. (2005). Specification and evaluation of safety properties in a component-based software engineering process.

Originally published in C. Atkinson, C. Peper, & H.-G. Gross (eds.) *Component-based software development for embedded systems: an overview of current research trends.*
Lecture notes in computer science (Vol. 3778, pp. 249–274). Berlin: Springer.

Available from: http://dx.doi.org/10.1007/11591962_13

Copyright © Springer-Verlag Berlin Heidelberg 2005.

This is the author's version of the work, posted here with the permission of the publisher for your personal use. No further distribution is permitted. You may also be able to access the published version from your library. The definitive version is available at http://www.springerlink.com/.

Specification and Evaluation of Safety Properties in a Component-based Software Engineering Process

Lars Grunske[1], Bernhard Kaiser[2], and Ralf H. Reussner[3]
grunske@itee.uq.edu.au,
bernhard.kaiser@iese.fraunhofer.de,
reussner@informatik.uni-oldenburg.de

[1] School of ITEE, The University of Queensland, St Lucia, Brisbane 4072, Australia
[2] Fraunhofer IESE, Sauerwiesen 6, 67661 Kaiserslautern, Germany
[3] Software Engineering Group, University of Oldenburg
OFFIS, Escherweg 2, 26121 Oldenburg, Germany

Abstract. Over the past years, component-based software engineering has become an established paradigm in the area of complex software intensive systems. However, many techniques for analyzing these systems for critical properties currently do not make use of the component orientation. In particular, safety analysis of component-based systems is an open field of research. In this chapter we investigate the problems arising and define a set of requirements that apply when adapting the analysis of safety properties to a component-based software engineering process. Based on these requirements some important component-oriented safety evaluation approaches are examined and compared.
Keywords. Component-Oriented Safety Evaluation (Survey), Component Specification, Component Fault Trees, FPTN, parametric Contracts

1 Introduction

Over the past years, the paradigm of component-based software engineering has been established in the construction of complex software intensive systems [1], mainly in the context of large business software projects. Models and procedures have been developed that help designing component based systems and assessing many relevant quality properties. Design by components is also a promising approach in the domain of embedded systems, where cost reduction, time-to-market and quality demands impose special constraints. In this context of embedded systems, in particular safety-critical systems, some new issues arise that are still subject of current research. Some of these issues are:

- How to specify the failure behavior of a component, when its usage and environment are unknown?
- How to evaluate the safety properties for a system built with components?
- How to adapt accepted safety assessment techniques to the special context of embedded and component-based systems?
- How to construct safety cases for a system built from components?

In this chapter we will discuss the problems in detail and give an overview of research work covering this problem domain.

The remainder of this chapter is organised as follows: In section 2 an introduction to the general safety concepts is given. Therefore, the relevant safety terms are defined. Thereafter section 3 provides an overview over the state of the art safety analysis techniques. In the main part of this chapter, section 4, we investigate the arising problems and propose some requirements to safety analysis techniques when applying them to the construction and evaluation of component based safety critical systems. Furthermore, we summarize the some important state of the art techniques for component-based analysis for safety properties. In section 5 we compare these techniques and show how each of these techniques fulfills the stated requirements. This provides support for the selection of a suitable analysis technique. Finally, section 6 contains concluding remarks and points out the directions for future work.

2 Basic Safety Concepts

To introduce into the matter of safety analysis, we first define the relevant terms and concepts used in this chapter.

Definition 1 (Component). *A component is an identifiable entity with a well defined and specified behavior. In computer science and engineering it designates a self-contained, i.e. separately deployable piece of hardware or software.*

Definition 2 (System). *A system is a set of components which act together as a whole and that is delimited by a system boundary.*

This chapter deals with purely technical systems (while safety analysis in general considers non-technical components such as user interaction as well). Due to recursive decomposition the subsystems (components) of a system can be viewed as systems on their own right, so the terms component and system are often used interchangeably.

Definition 3 (Failure). *A failure is any behaviour of a component or system, which deviates from the specified behaviour, although the environment conditions do not violate their specification.*

Based on this definition a failure is basically a derivation from the specified behavior. However, from the practical viewpoint it is useful to introduce a failure classification of finer granularity by distinguishing different ways in which the provided behavior can deviate from what the expectation. For dependable systems there is an accepted categorization which groups the failures into the following failure types or failure modes [2, 3]:

- tl timing failure of a service (expected event or service is delivered after the defined deadline has expired - reaction too late)
- te timing failure of a service (event or service is delivered before it was expected - reaction too early)
- v incorrect result of requested service (wrong data or service result - value)
- c accomplish an unexpected service (unexpected event or service - commission)
- o unavailable service (no event or service is delivered when it is expected - omission)

Definition 4 (Fault). *A fault is a state or constitution of a component that deviates from the specification and that can potentially lead to a failure.*

Definition 5 (Accident). *An accident is an undesired event that causes loss or impairment of human life or health, material, environment or other goods (similar [4]).*

To reduce the probability of an accident the preconditions under the control of the system must be distinguished from uncontrollable ones, because the system designer can only take counter-measures for the controllable ones. These controllable preconditions are called hazards and can be defined as follows:

Definition 6 (Hazard). *A hazard is a state of a system and its environment in which the occurrence of an accident only depends on factors which are not under control of the system.*

An example of a hazard is a defective car air-bag, since the accident "driver is injured" occurs only if the car hits another car. It depends on the environment, whether a hazard leads to an accident and thus the term hazard is always defined with respect to a given system environment and depends on the actual definition of the system boundary. To quantify safety it is important to consider how probable a hazard is and what the severity of the correlated accident or damage is. This is captured in the definition of risk.

Definition 7 (Risk). *Risk is the severity combined with the probability of a hazard.*

It is not practical to claim that risk be the *product* of hazard level and probability, since there are no universally accepted measures for the hazard level and the estimations of the probability are often very coarse. A practical way is to group both severity and probability in a few categories (negligible consequences . . . catastrophic, very rare . . . sure), as in [5, 6]. Both dimensions are independent from each other. A release of radioactivity in a nuclear power plant for instance can cost the lives of many people. Therefore, such kind of accident is not acceptable, even with a very low likelihood.

Definition 8 (Acceptable Risk). *Acceptable risk is the level of risk that has deliberately been defined to be supportable by the society, usually based on an agreed acceptance criterion.*

The risk acceptance depends on social factors such as applicable laws or public opinion. According to standards (e.g. [5]) the acceptable risk can be identified based on various risk acceptance principles, depending on local legislation. Some known risk acceptance principles are ALARP (the residual risk shall be As Low As Reasonably Practicable), GAMAB (globalement au moins aussi bon, French principle that assumes that there is already an acceptable system and the risk of new system shall be equivalent or lower) and MEM (Minimum Endogenous Mortality, where individual risk due to a particular technical system must not exceed $1/20^{th}$ of the minimum endogenous mortality.)

This definition of risk enables the definitions of the terms SAFETY and SAFETY REQUIREMENTS

Definition 9 (Safety). *Safety is freedom from unacceptable risks [5]*

In other words, safety is the situation where the risk is below the accepted risk level. Literally, safety is the situation where *no* hazard is present. Since this is not a practicable definition, the widely agreed definition refers to the risk level instead, incorporating the probability of a hazard.

Definition 10 (Safety Requirements). *A safety requirement is a (more or less formal) description of a hazard combined with the tolerable probability of this hazard.*

The tolerable hazard probability must be determined so that the combined risk for all hazards of the system is acceptable. This is the task of risk analysis.

In summary, the aim of safety critical systems construction is to build a system so that it fulfills all of its safety requirements. This comprises the steps

- Identification of all system level hazards
- Determination of the acceptable hazard probabilities (safety requirements)
- Taking constructive measure in order to avoid or reduce anticipated hazards
- Proof that all of these safety requirements are fulfilled (safety cases)

If the proof fails on first attempt, the last two steps have to be repeated iteratively.

3 Established Safety Analysis Techniques

There is an established set of safety analysis techniques for different purposes. Most of them have been developed at a time when safety critical tasks were exclusively performed by purely mechanical or electrical systems and do not especially consider the new apsects introduced by software control. The different techniques can be classified by different categories: they are used in different process phases, they use different formalisms, and they also differ in the kinds of qualitative and quantitative analyses that they provide. In the context of component-based system development, the techniques can also be divided into techniques that ignore the internal structure of the systems (as these are not concerned by the fact that a system is developed by components) and techniques that refer to a structural model of the systems (as these potentially need some adaptation when applied to component-based systems).

3.1 Safety Analysis Techniques on System-Level

Techniques belong to the first category e.g. because they look at the system on a coarse and abstract level, focussing on black-box properties or the effects of system-level failures to the environment. In these cases it is irrelevant whether a system is monolithic or component-based and which of the components are implemented in software or hardware. These techniques are typically applied in early process phases. In the sequel we give an overview on some techniques belonging to this category.

An example for an early safety analysis technique is Preliminary Hazard Analysis (PHA) [7], a technique that is applied during requirements analysis and early system design. Its purpose is to identify potential danger sources, to give an early assessment of severity and probability of each hazard and to suggest constructive measures to avoid or reduce risks. PHA is an inductive technique that searches for the effects of identified

hazards and the conditions in which they can arise. It is a manual and semi-formal technique that is applied on system level.

A similar technique is Functional Hazard Assessment (FHA)[6] that is increasingly used in aerospace industries. It assesses system functions without reference to the (later) technical realization. Like PHA it is used to obtain a first safety study in early process phases. Based on the potential hazards that have been identified all functions are categorised according to criticality levels. For each function and each of its failure modes the correlated effects, countermeasures and analysis or validation techniques are listed in a table. Although a FHA can be carried out on subsystem level as well, it is a manual and rather coarse technique that does not require detailed information about the component structure of the system.

Another example is Event-Tree-Analysis (ETA) [8], a graphical technique that uses a tree diagram to find and depict all potential effects that a given system level hazard has to the environment. The root of the tree is the hazard being analyzed. The branches are potential scenarios that lead to different consequences. Each branching point is associated to a condition which influences the further development of the scenario. For example if the hazard is "fire in engine", the first branching point could be "automatic extinguishing system is working properly". The TRUE branch leads to a mitigation scenario (no accident), the FALSE branch to another branching point: "fire is immediately detected by operator". Again the two branches lead to a different continuation of the story and finally each scenario leads to an accident / damage or not. The technique can yield quantitative results, if for each branching points the probability to take the TRUE or the FALSE path are known. ETA is applied manually with computer support. Since all effects considered in an ETA happen in the system environment, the internal structure of the system is not of concern.

As these techniques either regard the system as a black box or are applied in a stage where the actual implementation is yet unknown, they do not refer to components. Consequently, the aforementioned techniques can be applied to component-based systems without modification. In the following subsections we introduce some safety analysis techniques that refer to the internal structure of the system. Thus we will afterwards have to discuss in how far and with which modifications they can be applied in the context of component-based system design.

3.2 Failure Modes and Effects Analysis (FMEA)

Failure Modes and Effects Analysis (FMEA; extended variant: Failure Modes, Effects and Criticality Analysis, FMECA) is a table-based, semi-formal technique to identify possible safety or reliability issues with their effects in a systematic and roughly quantitative way. FMEA can be applied both to products (system or component level) or to a process (e.g. software development process). FMEA has been standardised in IEC 60812 [9] and is today widely applied in industry, in particular in the automotive branch. The steps to be performed are:

1. Analysis of the system structure and identification of structure elements (hierarchically arranged in a structure tree diagram)

2. Identification of the functions of each identified structure element. The functional decomposition follows the structural decomposition, i.e. functions of sub-components contribute to the functions of their respective super-components.
3. Investigation and listing of all failure possibilities of each function. Generating an FMEA table (see below) containing one row for each failure mode found
4. Estimation of (a) the failure probability of each failure mode, (b) the criticality of the failure mode and (c) the probability that the failure is not discovered early enough to prevent its consequences. For each of these three dimensions a measure out of the range 1 (most favorable case) to 10 (fatal case) is assigned. For this step the use of guiding words and predefined categories is recommended. Multiplication of the three numbers render a Risk Priority Number (RPN) between 1 and 1000. The most critical failure modes are marked with the highest RPN.
5. Redesign or improvement of the system. The ameliorations begin with the failure modes that have the highest RPN. The main goal is to reduce the occurrence frequency of failures, followed by measures to ameliorate the detection of the failures (e.g. by alarm facilities). The RPN can be used to priorize the amelioration efforts and to decide whether corrective actions are mandatory or not. After the changes a re-assesment of the system quantifies the effect of the measures. The RPN must now be significantly lower than before.

The central document of an FMCA is the table, containing the columns (Structure Element, Failure Mode, Effect on System, Possible Hazards, Risk Priority Number, Detection Means, Applicable Controls / Countermeasures). This table helps to carry out the FMEA systematically and makes it a semi-formal method.

3.3 Hazard and Operability Studies (HAZOP)

Hazard and Operability Studies (HAZOP) [10] [11] is a criticality analysis technique that has been developed in the 1970s in the context of chemical process industry and has been transferred to other industry branches, including software engineering. It focusses on unnormal deviations of process parameters from their expected values. The key element is a set of keywords that qualify the kind of deviation (e.g. no, less, more, reverse, also, other, fluctuation, early, late) for each information or material flow. The use of predefined keywords assures the completeness and consistency of the whole study. The list can be adapted or extended as appropriate. HAZOP is a session technique, conducted by a team of domain experts as early as a first material or data flow model for the system is available. The goal is to predict potential hazards that result from these deviations. The results are usually presented in a table and in the end report the system design is either accepted or changes to improve safety are requested.

3.4 Fault Tree Analysis

Fault Tree Analysis (FTA) [12] [13] [14] is a graphical safety and reliability analysis technique which has been used and accepted in different industry branches for over 40 years. It is a deductive top-down analysis technique and a combinatorial technique. FTA allows tracing back influences to a given system failure, accident or hazard. Fault Trees

(FTs) provide logical connectives (called gates) that allow decomposing the system-level hazard recursively. The AND gate indicates that all influence factors must apply together to cause the hazard and the OR gate indicates that any of the influences causes the hazard alone. The logical structure is usually depicted as an upside-down tree with the hazard to be examined (called top-event) at its root and the lowest-level influence factors (called basic events) as the leaves. In the context of FTA the term "event" is applied in its probability theory meaning: an event is not necessarily some sudden phenomenon, but can be any proposition that is true with a certain probability.

Based on a FT, several qualitative or quantitative analyses are possible. Qualitative analyses list, for instance, all combinations of failures that must occur together to cause the top-level failure. Quantitative analysis calculates the probability of the top-event from the given probabilities of the basic events. Combinatorial formulas indicate for each type of gate how to calculate the output probability of a gate from the given input probabilities. The probabilities taken into account are the probabilities that an event occurs at least once over a given mission time or they are probabilities of a failed state with respect to a given point in time. The evolution of a system over time or any dependencies between the present system behaviour and the history cannot be modelled. An important assumption to obtain correct results is the stochastic independence of the basic events, which is hard to achieve in complex networked systems where often common cause failures occur [15]. Figure 1 shows a simple FT example. The starting point of

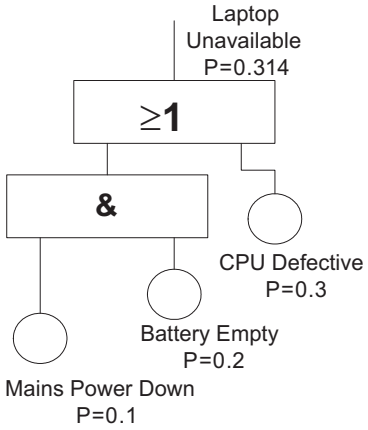

Fig. 1. Fault Tree Example

the model construction is a hazard state or a failure event that has been identified before (e.g. by means of an FMEA). In the example, the unavailability of a laptop computer is analyzed. A creative process is carried out to investigate all factors that contribute to the occurrence of this top-event. The search is performed along the system structure and examines all system functions, environment conditions (e.g. ambient temperature) and auxiliaries (such as power supply). The decomposition is stopped at a granularity level where the individual influence factors cannot or need not be refined any further.

These lowest-level factors are called basic events and are the leaves of the tree, depicted by circles. For a quantitative analysis a probability or probability distribution must be known or estimated for all basic events. This is usually achieved by probabilistic failure models, most of them empirical models.

In the example, we restrict ourselves to AND and OR gates, although many other gate types are provided by standards and FTA tools . In the figure, the graphical representation according to [13] has been chosen — in the United States different symbols are used. The AND gate is depicted by a $\&$ symbol and the OR gate by a ≤ 1 symbol (because at least one input must be true for the gate output to become true. The quantitative result shown in the figure has been obtained by application of the formulas associated with the AND and the OR gate. In case of the AND gate, the input failure probabilities F_i are multiplied with each other: $F_{output} = \prod_i F_{input_i}$, in the case of the OR gate DeMorgan's theorem indicates that all input probabilities have to be inverted (subtracted from one), then multiplied, and finally the result has to be inverted again: $F_{output} = 1 - \prod_i (1 - F_{input_i})$.

3.5 State-Based Approaches

To explain the behavior of components with respect to safety it is often not sufficient to restrict to a two states (working and failed) abstraction, as FTA does. This accounts for a different class of analysis techniques, the state-based techniques. In the context of software engineering and systems safety usually discrete state approaches are applied. The practically relevant subclasses classes are:

- Statecharts, ROOMcharts, UML State Diagrams or similar notations
- Petri Nets
- Markov Chains

The use of Statecharts or similar notations can enhance safety during system construction by providing an intuitive notation with automatic consistency checking and by partially allowing for automatic code generation. Moreover, in safety critical areas they are exploited for safety analysis as well: formal state-based models that describe the (intended) system behavior can serve as a base for model checking. Model checking is a qualitative technique that decides if a certain undesired state (hazard state) is definitely impossible to reach. If this cannot be proved, a counter example is produced, which in turn helps the analyst to formulate countermeasures how to avoid that hazard. Probabilistic variants of model checking algorithms are currently a major subject in formal methods research.

Petri Nets exist in deterministic and in probabilistic variants. They are a good means to model concurrent or collaborating systems. They also allow for different qualitative or quantitative analyses that can be useful in safety validation. However, Petri Nets are mainly applied for performance evaluation.

Markov Chains (MCs) are a probabilistic state-based modeling technique. An MC a finite state machine where the transitions occur stochastically according to defined probability distributions. MC analysis plays an important role in reliability analysis and can be used to judge the reliability or availability of safety-relevant components within

a system [16]. It is a discrete-state approach and there exists a continuous time variant and a discrete time variant. An MC is mainly a state diagram that explicitly considers working states and failed states. In contrast to the combinatorial approaches MCs allow more than two states for each component, so multiple failure modes or degrading failure (e.g. working - restricted service - completely failed) can be modelled. The states are usually depicted as circles and the state transitions as directed edges. The transition rates are annotated at the edges. A transition rate is the conditional probability, that the state will change from S_i to S_j in the next short time interval under the condition that it is in state S_i at time t.

MC analysis is performed by formulating and solving of differential equations (there are several transient and steady state analysis or simulation techniques and quite a number of tools is available). These equations can be imagined to describe the "probability flow" between different states.

4 Safety Analysis Techniques for Component-Based Systems

4.1 Problems

Since safety means freedom from unacceptable risks, the primary goal of safety analysis techniques is to identify all failures on system level that cause hazardous situations and to demonstrate that their probabilities are sufficiently low. In the context of component-based systems this involves some additional problems that do not occur in the same way in monolithic systems.

A principal question to be addressed is the compositionality of the property "safety". Is it permissible to say that a system is as safe as its components together (analogously as the combinatorial reliability models judge system reliability from component reliability)? A small part of the system, in particular a piece of software, cannot do harm to the environment and thus cannot be unsafe. We find that safety as a property is not defined on an arbitrarily low granularity level and thus fine-grained components do not possess a quality attribute "'safety"' [4]. However, the influence of component behavior, in particular software behavior on the safety of the whole system cannot be argued. In particular component failures can compromise the safety of the system. In real-time systems this applies to timing failures as well as to value failures. More exactly, safety violations result from failures that propagate to the system boundary. Thus, component based safety analysis means to conclude from component behavioral models to system safety. On a higher abstraction level, the conclusion is from various quality properties of the components (e.g. correctness, availability, reliability) to system safety. For instance, the availability of a protective device such as a car airbag or a fire detector directly influences the safety of the containing system. Consequently, the techniques applied at component level need not to necessarily be proper safety analysis techniques; analysing reliability or correctness of components can be a part of the overall safety argument and according techniques can be applied on component level. The question is which techniques to chose and how to integrate the results to a system safety case.

Another finding is that components are usually not isolated but require services from other components to provide their service correctly. Therefore, not only internal

failures, but also failures that are propagated from a foreign component can cause a component to produce failures.

The next issue concerns the development process: safety analysis techniques must integrate into the overall development process of the embedded system. In the case of component based design this means in particular that concurrent development at different places and design for later reuse in an unforeseen environment have to be considered. The different modeling techniques used within the same project should be compatible, which can be achieved e.g. by integrated tool-chains or model export and import facilities.

Another big challenge is complexity. Component-based engineering is often applied to systems that are too complex to be understood in one piece. For example, a system composed out of 10 components with only 2 state each has a state space of 1024 states; one can easily imagine the consequences for real-world systems with lots of states for each component. This problem is referred to as state-explosion. However, not only state-based approaches, but also other techniques suffer from complexity problems, e.g. by an excessive amount of causal chains that hampers the readability of the model.

This leads us to the correlated question of scope and granularity: It is impossible to consider all states and all behavioral aspects of a system. The challenge is to find the right abstraction level that makes a model expressive enough and yet analysable. We found that on the one hand, informal or even combinatorial models are sometimes not sufficient, but on the other hand, composing an integrated behavioral model and analyzing all possible sequences of actions, including failures, is far beyond feasability. Techniques on a practical granularity level and with a limited scope (i.e. expressing just the facts of interest) are necessary.

4.2 Requirements

Being aware of these particular problems we now map out some requirements that will help us to classify the safety analysis techniques that we will present in the subsequent sections.

Requirement 1 (Appropriate Component Level Models) *Each component must be annotated with an appropriate evaluation model.*

Component based safety analysis should decompose the system according to its architecture and then annotate each component with an appropriate model. The system level analysis technique must finally integrate the results from all component analyses to a sound safety case for the whole system. Different components may be implemented by different technologies and ideally it should be possible to chose for each component the most appropriate modelling technique. Due to the embedded nature of the systems, this includes techniques that are suitable for software and hardware aspects. The techniques should be able to describe the correct behavior and failure behavior by appropriate means. Further, we saw that the property safety on system level is influenced by different aspects of the component failure behavior, for instance by quality properties such as reliability, availability, timeliness and correctness on component level. Accordingly, attaching models for different quality properties to different components in order to validate each of these properties by an appropriate technique is a suitable approach.

Requirement 2 (Encapsulation and Interfaces) *The notation for the evaluation models should allow encapsulation and composition by interfaces similar to component-based design notations.*

Many current component based design notations (such as ROOM [17] or UML2.0) offer mechanisms to define components as closed capsules and ports that serve as point of information exchange between components. These ports define the externally visible interface of a component. Their semantics varies with the different modelling techniques; examples are

- incoming and outgoing messages or signals
- incoming and outgoing continuous data flows
- provided and required services

In these design fameworks, it is usually possible to refine components recursively and to integrate components to new components. Every component can be exchanged by another with the same interface. The internal implementation, i.e. everything that does not belong to the interface, is hidden from the environment. An appropriate syntax and type systems for interfaces allows to check all component interconnections automatically for consistency. If even a formal semantics is associated to the interface notation (as it is the case in interface automata, for instance) it is possible to derive the system semantics from the component semantics and the topology.

A similar construction principle is also desirable for component safety evaluation models. The interfaces of the component safety evaluation models should correspond as closely as possible to the interfaces used in the functional models from systems design phase.

Requirement 3 (Dependencies on External Components) *Safety analysis techniques must be able to express the dependencies of failures regarding provided services on failures regarding required services and on internal failures of the component.*

Due to the fact that most of the components are not self-contained and require external components to operate, the failure modes of the provided services depend on the failure modes of the provided services by other components. In consequence the failure probabilities of the provided services of a component are a function of (a) the probabilities of internal failure generation and (b) the probabilities of failures of the external environment the component interacts with.

Requirement 4 (Integration of Analysis Results) *A composition algorithm is required that constructs the evaluation model for a hierarchical component based on the architecture and the evaluation models of the used components.*

The aim of constructing a safety model for a system is not only to visualize the system for better understanding, but also to run analysis algorithms on it, for instance to calculate the probability of the system level hazard. Therefore it is necessary that the algorithms are composable, i.e. the results from component analysis can be integrated to the results (e.g. safety critical failure probability) on system level. Some compositional techniques are only analysable after the final integration and suffer from the

combinatorial explosion of the state space. Ideally the analysis algorithms allow simplifications and calculations of immediate results on component level. The advantage is that each time the component is reused only a part of the calculations has to be redone and the performance is thus acceptable. The integration of results from different modelling techniques should be automated to a high degree, as manual copy or translation between different formats is error-prone and compromises the integrity of a safety analysis.

Requirement 5 (Practicable Granularity) *The techniques applied should be on the one hand rich enough in details to express how different kind of component behaviour can influence system safety, but on the other hand coarse enough to allow affordable analysis on system level.*

Regarding granularity and scope, a compromise between expressive power and analyzing effort must be found. The approach of exhaustive modelling of all possible behavior traces to explain how a system level hazard can occur is infeasible.A plain parts count approach (system works correctly if all components work correctly) which is sometimes used in reliability analysis is not sufficient to show how components interact and how for instance a safety subsystem mitigates failures of other components. Often the two state abstraction (working versus failed) in combinatorial techniques is too coarse, but a full state based approach is not manageable due to the state-explosion-problem. A compromise could be to classify failures according to a few categories, which still allows to formulate simple causal relations between failures of different classes at different interfaces. In the case of state-based approaches it is often not feasible to examine the whole state space as determined by the functional model of the system. Instead, it is preferable to take a coarser approach by only modelling the states that are involved in a safety-relevant scenario.

Requirement 6 (Tool Support) *The safety analysis technique should be supported by appropriate and ergonomic tools.*

Some of the safety analysis techniques (FMEA, for instance) were originally designed as paper-and-pencil methods. In the present context, manual application of techniques is not practicable. First, systems that are designed by components are usually complex systems, so only computer based tools allow humans to handle systems of high complexity without making errors. Important aspects are project browsing and history tracking facilities, model design assistants and consistency checking, ergonomic user interface and structured graphical representation. Second, one main purpose of components is to design them at different places (division of labor) or at different time (reuse). Traditionally, when one team at one place created a model, intuitive knowledge and implicitly agreed assumptions helped to overcome ambiguities. In the component-based process, when working at different places or when reusing a component that has been created years before, the lack of this direct communication will likely lead to misinterpretation. By enforcing a well-defined model syntax (and ideally also semantics) and by capturing all aspects of the model in a file or database, computer based tools help creating reusable and exchangeable component analysis.

4.3 Running Example

To explain how a safety evaluation system built with components works in practice, we present a steam boiler system as a running example. The left part of figure 2 shows a schematic, similar to those process engineers use to describe the hardware of the plant. The process plant incorporates the pressure tank, 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 voter for the sensors and gives command to open both valves if a pressure higher than the allowable level is detected. 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. In a subsequent sections we will also discuss a variant of the example where it is possible to select either voter mode (three sensors) or single-sensor mode. The right part of the figure shows a structure diagram, as an

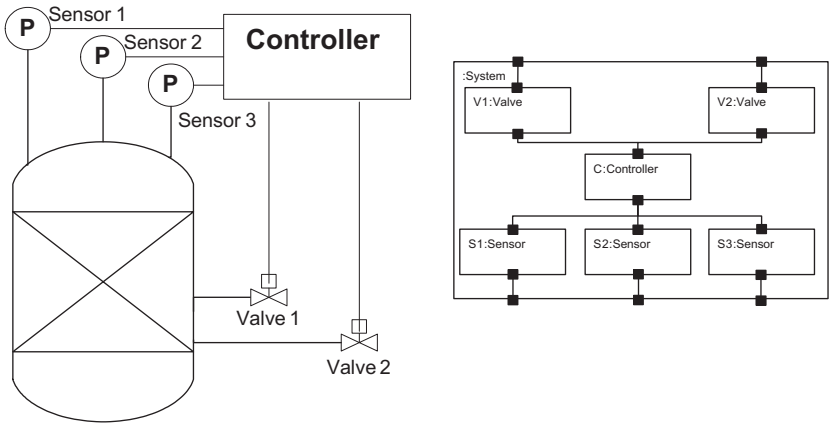

Fig. 2. Steam Boiler Schematics and Structure Diagram

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 these. During design phase, models for the behavior are attached to the components, for example state machine models that describe the reaction of components to trigger signals received via its ports [17]. During the construction phase, only the intended behavior is of relevance. Safety analysis in contrast focuses on possible deviations from the intended behavior that lead to hazardous situations.

By definition, the ports of the components are the only spots where information is exchanged and the interconnections in the structure diagram are the paths of information flow. Consequently these are also the spots where failures are propagated between components. The idea behind the component based safety analysis techniques discussed in the following subsections is to exploit the system architecture for safety analysis by attaching models for failure generation to the components.

4.4 Failure Propagation and Transformation Notation (FPTN)

The Failure Propagation and Transformation Notation (FPTN) described in [3, 18] is one of the first approaches that introduce modular concepts for the specification of the failure behaviour of components.

The basic entity of the FPTN is a FPTN-Module. This FPTN-Module contains a set of standardised sections. In the first section (the header section) for each FPTN-module an identifier (ID), a name and a criticality level (SIL) is given. The second section specifies the propagation of failures, transformation of failures, generation of internal failures and detection of failures in the component. Therefore, this section enumerates all failures in the environment that can effect the component and all failures of the component that can effect the environment. These failures are denoted as incoming and outgoing failures and are classified by the failure categorization presented above (reaction too late(tl), reaction too early(te), value failure(v), commission(c) and omission(o)). In the example which is given in figure 3 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=A:tl and for transformation C:c=A:te&&A:v and C:v=A:$tl \| B$:v). Furthermore a component can also generate failures (e.g C:o) or handle an exiting failure (e.g B:v). For this it is necessary to specify a failure cause or a failure handling mechanism and a probability. FPTN-Modules can also be nested

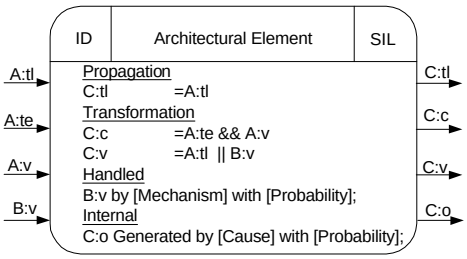

Fig. 3. Abstract FPTN-Module

hierarchically. Thus, FPTN is a hierarchical notation, which allows the decomposition of the evaluation model based upon the system architecture. If a FPTN-module contains embedded FPTN-modules the incoming failures of one module can be connected with the outgoing failures of another module. Such a connection can be semantically interpreted as a failure propagation between these two modules.

For the evaluation of an FPTN-module a fault tree is constructed for each outgoing failure based on the predicates specified inside the FPTN-module. As a result of this interpretation, a FPTN-module can be seen as a forest of fault trees, where the leaf nodes and their probabilities are extracted from the failure generation and the failure handling section inside the FPTN-module.

To show the applicability of the FPTN in figure 3 the failure behavior of the Steam Boiler System (c.p. section 4.3) is modeled. To keep the considerations simple, we assume only a few failure modes: A sensor fails with a value failure (wrong pressure indicated) if a mechanical or an electrical failure occurs. A valve can fail to open (omission) for electrical or mechanical reasons, but also as a result of a missing command (omission at the input failure cmd). The controller fails to give the open commands (omission) either if at least two of the connected sensors give wrong signals (value failure corresponding to $P1$, $P2$ or $P3$) or if there is an internal hardware defect. Based on this assumptions, for each used component a FPTN-module is created, which describes the failure behavior. These created FPTN-modules are embedded into the FPTN module "Steam Boiler System" and connected with respect to the possible failure propagation. For the evaluation of the safety properties the failure probability of both outgoing failures $Open.o$ need to be calculated. As described earlier, this can be performed by an analysis of the corresponding fault trees.

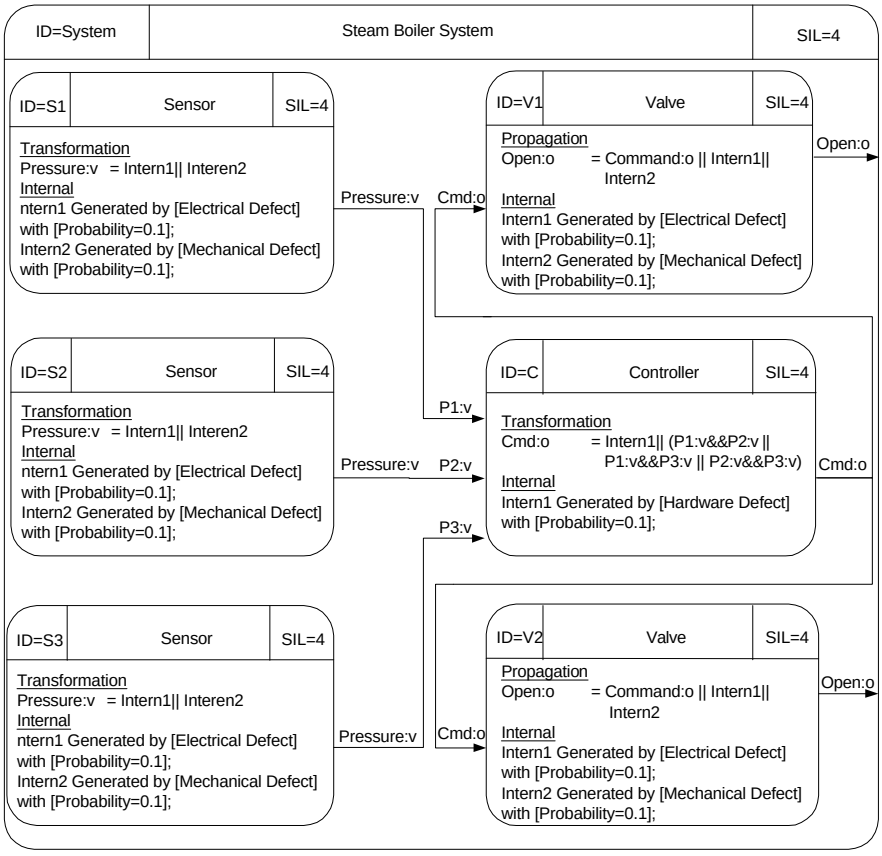

Fig. 4. Steam Boiler Example (FPTN)

4.5 CFT

Fault Tree Analysis is one of the most popular safety analysis techniques. Unfortunately they provide only a restricted decomposition mechanism: the decomposition into independent subtrees, called modules. To be compatible to the architecture model that shall serve for automatic construction of the safety case, the models for the failure behavior must be attachable to the components and account for the assignment of incoming and outgoing failures to the ports. They must take into account that the components are in general not independent from each other because the ports are access points for possible influences from other components. FTs are compositional in the sense that independent subtrees can be cut off and handled separately. Technical components however are typically influenced by other components and thus this assumption does not hold. To allow for a modularization that corresponds to the component and port concept, an extension of FTs has recently been proposed [19]. It is called Component Fault Trees (CFTs) and allows defining partial Fault Trees that reflect the actual technical components. These CFTs can be modeled and archived independently from each other. Input and output failure ports glue these parts together. While traditionally independent subtrees were regarded as compound events, CFT are treated 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 and hides all internal failure events from the environment. Components can be instantiated in different projects. Thus all necessary preconditions for an application of FTA to component based systems are fulfilled.

To model potential failures, CFTs for each component-class are generated. This is a manual task that is conveniently performed on a graphical CFT editor. Each CFT has input failure ports and output failure ports that must be associated to failure categories with respect to messages or services at the ports of the corresponding component-classes. Between input failure ports and output failure ports the failure propagation or transformation and the internal failure generation of the component-classes are modeled. For the components in the steam boiler example (c.p. section 4.3) this leads to the CFTs, which are presented in figure 5, if the same failures modes and internal faults are assumed as given in the FPTN Section (c.p. figure 4) . The CFTs given so far allow

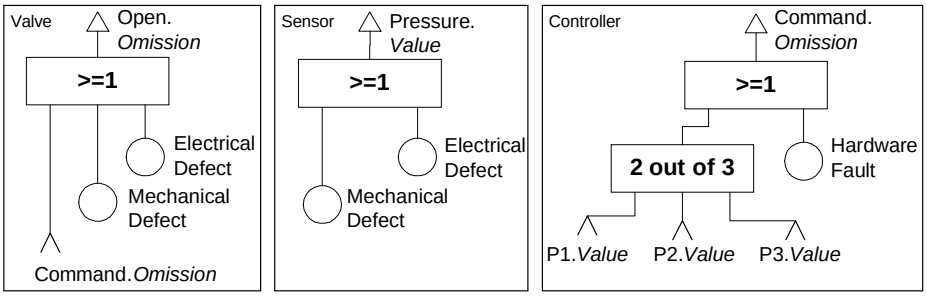

Fig. 5. Controller, Valve and Sensor CFTs

in conjunction with the structure diagram to integrate the system level CFT. However, before starting the analysis, another manual step is necessary: The user must complete the system-level Fault Tree by specifying which system hazard is to be examined. This can be performed using the graphical editor of the CFT analyzer. The resulting Fault Tree is shown in figure 6, which is a screen shot taken from our analysis tool UWG3 that will be introduced in the following section. 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 to have constant failure probability of 0.1 we calculated the hazard probability to 0,1014 using the tool UWG3.

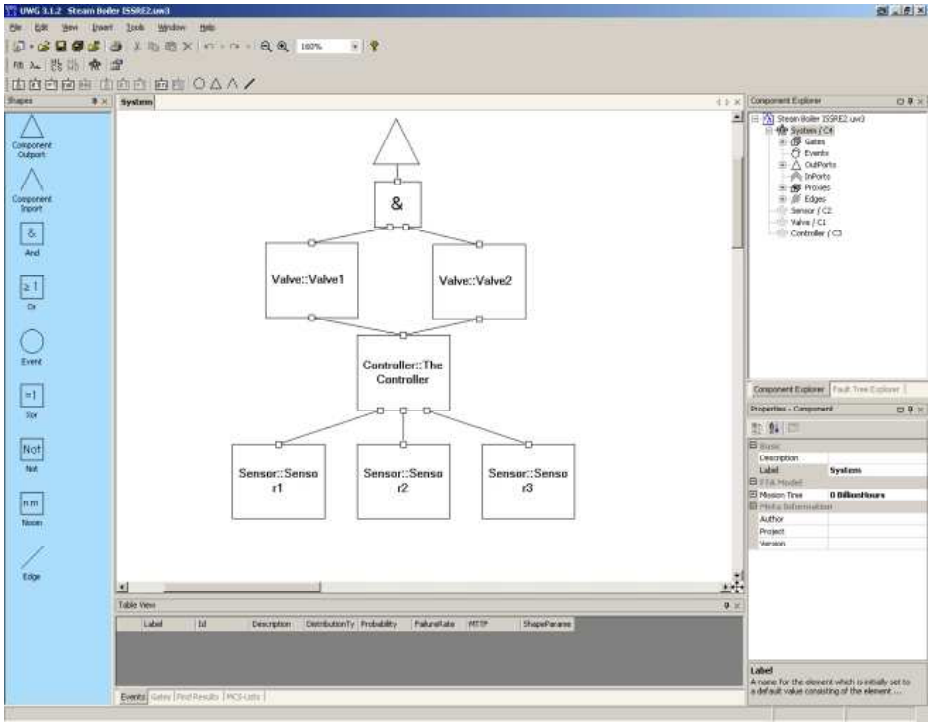

Fig. 6. System CFT

4.6 Safety Analysis with Parametric Contracts

In the following, we describe how to model safety properties in the interface of a component, using the "Quality of Service Modelling Language" (QML) [20]. This language

allows to define arbitrary quality dimensions. We define failure class definitions as quality dimensions. As the QML assumes that quality attributes are fixed values for a component and neglects the context-dependency of quality attributes, we then couple that notation with an analysis technique called "Parametric Contracts". Parametric contracts allow to model the context-dependencies of a component's safety attributes and thus the analysis of component based systems. Parametric contracts have been used for general reliability modelling before [21, 22], but are specialised for safety analysis here. The following section provides original research results.

As current interfaces ("signature-list based interfaces") specify the well-behaviour of a component service (i.e., the behaviour exposed without failures), these interfaces are unsuitable for specifying or analysing failure-propagation through a system. Therefore signature-list interfaces have to be extended by two dimensions: (a) a specification of failure classes and (b) a specification of the dependency of a service's failure behaviour on the failure behaviour of its context.

The inclusion of failure classes specification into a service signature can be done by the QML. The QML allows the specification of quality dimensions as well as the specification of Quality of Service contracts (QoS contracts, for short) specifying the actual provided or required service quality for the dimensions defined before.

In the following, for each of the failure types introduced in section 2 a "contract type" (i.e., quality dimension) is defined.

type TooEarly = contract { numberOfFailures : $\langle\langle$*decreasing*$\rangle\rangle$ *no / year; }*
type TooLate = contract { numberOfFailures : $\langle\langle$*decreasing*$\rangle\rangle$ *no / year; }*
type IncorrectValue = contract { numberOfFailures : $\langle\langle$*decreasing*$\rangle\rangle$ *no / year; }*
type Commission = contract { numberOfFailures : $\langle\langle$*decreasing*$\rangle\rangle$ *no / year; }*
type Omission = contract { numberOfFailures : $\langle\langle$*decreasing*$\rangle\rangle$ *no / year; }*

The above list assigns to each failure type the unit *numberOfFailures* which is measured by the number of occurrences per year (*no / year*). The keyword *decreasing* denotes that lower values relate to a higher "quality of service". This is important to know when matching component interfaces. In case two values are not the same, one has to know whether a higher or a lower value is acceptable.

The second extension of signature-list interfaces is modelling the context dependency of failures. Basically, for any failure of the above failure types, there are three causes:

Internal service error: a bug in the service's code causes a failure.
External call error: a call to an external service causes a failure. External calls can go to services of other domain components or to services provided by the run-time environment (operating system, middleware, etc.)
External interruption error: the run-time environment stops or interrupts the execution of the service pre-emptively and causes a failure.

In the following a failure type is denoted by $ft ::= tl|te|v|c|o$ for the failure types TooEarly, TooLate etc. The term $P_X(Y)$ is used to denote the probability that the event which is specified by the corresponding subscript X and parameter Y occurs. The subscripts is, es and ei signify internal service error, external call error and external interruption error, respectively. One can assume that on each execution trace of the software

at most one failure occurs. This assumption is justified by the fact that failure probabilities are very low. This assumption allows us to simply sum up the failure probabilities for all possible failure causes, restricting the analysis effort to linear equations. Therefore, the probability that service A fails with a failure of failure type ft is

$$P_{ft}(A) = P_{is_{ft}}(A) + P_{es_{ft}}(A) + P_{ei_{ft}}(A) \quad (1)$$

Here, $P_{is_{ft}}(A)$ is the probability that A fails (with an failure in failure type ft) because of an internal service error, $P_{es_{ft}}(A)$ because of an external service error and $P_{ei_{ft}}(A)$ an external interuption failure. If 5 failure types have been defined, then 5 equations of this style are required.

It is assumed that failures of one failure type effects only consecutive failures of the same type, e.g. if some service is provided too late, it may cause other services to be provided too late as well, but not too early or with a wrong value. This assumption holds in many practical cases. In principle however, each initial failure can result in a failure of any of the above types. To capture this, the linear equations could be extended, which in turn increases the analysis effort.

When modelling the dependencies of the component environment on the failure probability (for each failure type), the latter two terms of the above equation are important (as the first one is internal). Hence, the following considerations deal with $P_{es_{ft}}$ and $P_{ei_{ft}}$. In both cases, we need information on what happens if method A is called. In case of determining the probability of an external service error, one needs to know which external services are called (and how often are they called). In case of the external interruption error probability one needs information on the length of the execution and assume that the chance of an interruption is proportional to the execution time. Both kind of information is given by a so-called *service effect automaton* [23]. A service effect automaton (SEA) is a finite state machine, describing for each service implemented by a component, the set of possible sequences of calls to services of the context. Therefore, a service effect automaton is a control-flow abstraction. Control-statements (if, while, etc.) are neglected, unless they concern calls to the component's context. As the SEA is an automaton, it accepts a language. As the input symbols of the SEA are names of the external services called, a word of the language is a trace of service calls. By $traces(SEA)$ the set of traces of the SEA is denoted (which is the language accepted by the SEA). Figure 7 presents the SEA of the control process of the steam boiler controler. We refer to the variant of our example where the user can select between voter mode and single sensor mode. The automaton in the figure presents an abstraction of the software control-flow of the boiler control process. It first reads the value of pressure sensor 1 (`Read:P1`), then it calls the user-interface to determine whether the user selected 2-of-3 three voting mode or single sensor mode. According to this selection (let us assume a probability `[u]`for voter mode), either the other sensors are read and then the valve commands are issued or the valve commands are issued directly after the first pressure sensor reading. Hence the set of all traces is $traces(SEA_{ControlProcess}) = \{(Read : P1, Read : UI, (Read : P2, Read : P3, Cmd : Valve1)|(Cmd : Valve1), Cmd : Valve2)^n | n \in \mathbf{N}\}$.

For our purpose the SEA is extended to a Markov model, i.e., each transition is annotated with a transition probability (while the constraint holds, that for any state the

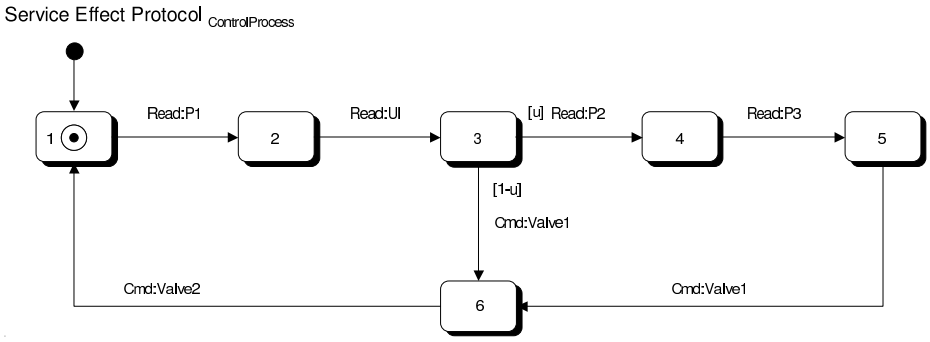

Fig. 7. Service Effect Automaton of the Steam Boiler Controler

sum of the probabilities of outgoing transitions never exceeds one.) As a result one has for each $tr \in traces(SEA)$ a function $P(tr)$ giving the probability of tr occurring in the SEA. Since execution traces of the main function in real-time systems usually are loops (repeating themselves over and over again until the device is switched off), let us first regard just the individual runs, and consider repetition later. Services that are called by the main function have one start and one end point. In our example SEA, showing the main loop, one finds two branches and thus two possible traces per run and get the probability $P(tr) = u$ for the trace $tr = Read : P1, Read : UI, Read : P2, Read : P3, Cmd : Valve1, Cmd : Valve2$ and $P(tr) = (1 - u)$ for the trace $tr = Read : P1, Read : UI, Cmd : Valve1, Cmd : Valve2$. On each trace, services are called and these services can cause a failure of one of the known failure type (still assume that failures are so improbable there is at most one failure per run.) Now one has to add up the failure probabilities of all externally called services e in each trace tr to get the failure possibility related to this trace under the condition that this trace is taken.

$$P_{es_{ft}}(tr) := \sum_{e \in tr} P_{ft}(e) \tag{2}$$

To get the total probability for one run of the main function A we refer to the definition of conditional probability. This allows to specify the probability $P_{es_{ft}}$ that a failure of type ft occurs in an arbitrary trace tr as follows:

$$P_{es_{ft}}(A) := \sum_{tr \in traces(SEA(A))} P(tr) * P_{es_{ft}}(tr) \tag{3}$$

This means, $P_{es_{ft}}(A)$ sums for all possible traces the product of the probability that tr is executed and the probability that during one execution of tr a failure of ft occurs.

Regarding the main loop that runs continuously, one finds that the probability that after n runs *no* failure has occurred is the product of the probabilities that there is no failure in the first run, no failure in the second run and so on until n. Consequently, denoting n runs of A as A^n, the probability $P_{ft}(A^n)$ is defined as follows:

$$P_{es_{ft}}(A^n) := 1 - (1 - P_{es_{ft}}(A))^n \tag{4}$$

The remaining step in order to obtain the failure probability per time unit is to estimate the number n, i.e. the number of main loop runs during one time unit. This task is feasible because the main loop is usually scheduled on a known and regular time basis, e.g. every 20 ms. If the main loop immediately starts again after completion, the number of runs per time unit can be estimated from the execution time per loop. As a result one obtains for each failure type the probability per execution time, that a failure that is caused by a call to a foreign component occurs. For practical application the method can be refined by correcting terms, e.g. the probability that the failure from the called component causes harm to the caller. These terms have to be specified by the component implementer while the function occurrence probabilities depends entirely on the usage context of the component.

The probability of an *external interruption error* $P_{ei_{ft}}$ is modelled in linear dependency on the length of the service's execution code trace. In principle, the length of the code execution trace depends on the actual path the control-flow takes through the code. The probability for a specific path taken is given by the transition probabilities of the service effect specification.

The only missing information for specifying the control flow path length (in number of instructions) is the number of instructions associated to each transition and the number of instruction associated to each state of the service effect specification. If the service effect specification is derived from existing component code, this data is available and simply needs to be attached to the service effect specification. However, without having the service implementation at hand for analysing its code, these figures might be hard to estimate in advance. Note, that this dependency of component specifications on the actual implementation makes us talking on component *implementation instances* rather than component types. Mathematically, one models the influence of external interruption errors as a linear function mapping each implemented service A to the probability that a failure of failure type ft_i occurs. Again, we refer to the definition of conditional probability.

$$P_{ei_{ft}}(A) := \sum_{tr \in traces(SEA(A))} P(tr) * L(tr) * P_{ft}(tr) \tag{5}$$

Formally, it sums over all possible traces the product of the probability that the trace is executed ($P(tr)$) and a measure for the length of the trace ($L(tr)$) and the probability $P_{ft}(tr)$ that the occurrence of an external interruption error results in an failure of failure type ft.

After these definitions, it is time to step back and to consider practical issues. First, lets summarise what our model needs as inputs:

1. The service effect automaton (SEA), a Markov model (i.e., having for all traces tr the value $P(tr)$). See [21] for a detailed discussion how to yield that data by a combination of code analysis, monitoring or simply educated guessing. However, even if the component vendor does not provide the SEA it can be generated a-posteriori out of an existing component. In addition one needs $L(tr)$, the length of a trace. But this is also given by the code of a component.
2. The failure probability $P_{ft}(e)$ for each external service and each failure type ft. This data has to be provided by the component deployer as it is part of the com-

ponent context. It can be measured (for basic operations) or predicted by using the presented model itself.
3. The probability P_{ft} that a failure of ft is caused by an external interrupt.

The second question of practical concern is how to evaluate the above formulas. The main problem is that the number of traces can be infinite, hence the sums given above cannot be simply evaluated within a loop. (Even worse, one has to show their convergence). Therefore, we refer to the Markov chain analysis for service effect automata extended to Markov models, as described in [21, 22].

5 Evaluation of Safety Analsysis Techniques

In the following we classify and evaluate the techniques presented above according to the requirements to safety analysis methods as introduced in section 4.2.

5.1 FPTN

Requirement 1: Appropriate Component Level Models: As presented in [3] the failure propagation and transformation notation provides a simple but comprehensive annotation of the failure behaviour of a component. These annotations are easy to understand and to analyse. However, failures are only differentiated according to the given five categories.
Requirement 2 + 3: Encapsulation and Interfaces and Dependencies on External Components: A FPTN-Module is encapsulated and provide with the incoming and outgoing failures a well defined interface to its environment. To specify the relation of these incoming and outgoing failures the failure transformation and propagation predicates are used. Based on these predicates the dependencies of the failure behaviour of the modelled component from its environment is defined.
Requirement 4: Integration of Analysis Results: For a hierarchical composition of FPTN-modules it is necessary to specify which failures are propagated between components. To identify this information currently no systematic procedure is specified in literature.
Requirement 5: Practical Granularity: The notation of the failure propagation and transformation notation utilizes the five relevant failure types [2] (reaction too late, reaction too early, value failure, failure of commission and failure of omission). However, the architect of a component can decide which failure types and relations between these failure types are really needed. Do to this the granularity is define by the user of the notation and thus even for a complex system the safety properties are still analysable.
Requirement 6: Tool Support Up to now there is no commercial tool that supports the specification and evaluation of FPTN-modules.

5.2 CFT

Requirement 1: Appropriate Component Level Models: Similar to the FPTN the CFTs provide a simple but comprehensive annotation of the failure behaviour of a component. The expressive power is restricted to combinatorial logic.

Requirement 2: Encapsulation and Interfaces: Each CFT is encapsulated and failure ports are used as interfaces to the capsules. These failure ports are separated into input and output failure ports. Components are reusable entities which makes the technique appropriate for component-based developoment processes.
Requirement 3: Dependencies on External Components: To describe the dependencies on external components the input failure ports are used. If they are connected with an output failure port of another component, the associated failures are propagated between these two components.
Requirement 4: Integration of Analysis Results: Due to their structure, component fault trees are hierarchically decomposable. That means the CFT of a component can contain the CFTs of the embedded components. Furthermore, the embedded CFT can be automatically connected, based on the interface specifications of the embedded components and a construction algorithm, which is presented in [24]. The quantitative analysis is usually performed by Binary Decision Diagrams (BDDs) [25] and the BDD fragments for each component can be automatically flattened to one analysable BDD.
Requirement 5: Practical Granularity: Similar to the FPTN component fault trees utilizes the five relevant failure types and the architect can decide which failure types and relations between these failure types are modelled within the CFT. Do to this the granularity is define by the user and thus even for a complex system the safety properties are still analysable.
Requirement 6: Tool Support The specification and evaluation of the CFTs is supported by a commercial tool, called UWG [19]. It that has been developed in a cooperation between the Hasso-Plattner-Institute and the companies Siemens and DaimlerChrysler for the last two years and has been used in several industrial projects where it proved its intuitive handling. It incorporates all previously mentioned features of the Component Fault Tree concept. UWG3 provides an efficient analysis algorithm that makes use of BDDs to efficiently represent even large CFTs.

5.3 Parametric Contracts

Requirement 1: Appropriate Component Level Models: As parametric contracts are specified by the service effect automata, the compositionality of the notation is given by the recursive composition of service effect automata. For that compostion, a transition marked by a call to an external method (read access or command) is replaced by the service effect automaton of that call. That construction of substituting transitions in a reversible way by service effect automata is shown in detail in [23]. However, Parametric Contracts are tailored only to a certain class of measurable quality properties.
Requirement 2: Encapsulation and Interfaces: The service effect automata are used to describe the interface of a component.
Requirement 3: Dependencies on External Components: This requirement is fulfilled, as the service automata explicitly models call to external components. Their failure probabilities are explicitly considered in the analysis. Therefore, this requirement is fulfilled.
Requirement 4: Integration of Analysis Results: As the service effect automata are again service effect automata (see above), one can apply the same analysis techniques. In fact, for given service effect autoamata, previously computed failure probabilites of

their traces traces can be used directly for the analysis of the composed service effect automaton (even without explicitly constructing the composition).
Requirement 5: Practical Granularity: Service effect automata abstract from internal computations and the influence of parameters on the failure probability of calls. This is only valid, if the parameters have no influence on the failure probabilities. This is the case e.g. if parameters are fixed or not existent (as in our example). However, the validity of these abstraction is not always given and its presence has to be validated. However, current research is concerned with more detailed usage profile models, taking parameters into account.
Requirement 6: Tool Support Tool support for the specification of parametric contracts is currently developed by the Palladio research group in Oldenburg. Currently, the analysis is not supported by dedicated programs. Commercial tools for safety analysis are currently not available.

5.4 Comparison of the three evaluation notation

Concluding the evaluation we present a table 1 a comparison of the three component-based analysis techniques for safety properties. In this table we assign a quality mark ranging from $--$ (requirements are not fulfilled) to $++$ (requirements are completely fulfilled) up to our knowledge to each analysis technique for each requirement.

Table 1. A Comparative Evaluation

Requirement	**FPTN**	**CFT**	**Param. Contracts**
Appropriate Component Level Models	$+$	$+$	$+$
Encapsulation and Interfaces	$++$	$++$	$+$
Dependencies on External Components	$++$	$++$	$++$
Integration of Analysis Results	$-$	$++$	$+$
Practicable Granularity	$+$	$+$	$+$
Tool Support	$-$	$+$	$-$

6 Conclusions

In this chapter, we have investigated the applicability of the component-based software engineering paradigm to the domain of safety critical systems. For that reason, we have discussed the relevant problems in detail and given an overview of current approaches and research covering this problem domain. As a result, we have identified a set of requirements that are needed to evaluate safety properties for a system built with components. These requirements are used to compare the state of the art specification techniques that allow for the evaluation of the probability of hazards or safety critical

failures. These specification techniques are Component Fault Trees (CFTs), Parametric Contracts and Failure Propagation and Transformation Notation Modules (FPTN Modules), which have partly been developed by the authors of this chapter and partly by other researchers. Each of these three evaluation notations has its own strengths and limitations. To increase these strengths and to reduce the limitations we try to unite the features of the three evaluation notations, which will ideally lead to a unified notation that completely fulfills all requirements that are given in this chapter.

References

1. Szyperski, C.: Component Software: Beyond Object-Oriented Programming. ACM Press, Addison-Wesley, Reading, MA, USA (1998)
2. Bondavalli, A., Simoncini, L.: Failure Classification with respect to Detection. Esprit Project Nr 3092 (PDCS: Predictably Dependable Computing Systems) (1990)
3. Fenelon, P., McDermid, J., Nicholson, M., Pumfrey., D.J.: Towards integrated safety analysis and design. ACM Computing Reviews, **2** (1994) 21–32
4. Leveson, N.G.: SAFEWARE: System Safety and Computers. Addison-Wesley Publishing Company (1995)
5. CENELEC (European Committee for Electro-technical Standardisation): CENELEC EN 50126: Railway Applications – the specification and demonstration of Reliability, Availability, Maintainability and Safety. CENELEC EN 50128: Railway Applications: Software for Railway Control and Protection Systems CENELEC, Brussels (2000)
6. SAE ARP 4754 (Society of Automotive Engineers Aerospace Recommended Practice): Certification Considerations for Highly Integrated or Complex Aircraft Systems (1996)
7. Department of Defense, United States of America: Military Standard 882C. System Safety Program Requirements (1999)
8. Deutsches Institur für Normung e.V.: DIN 25419: Ereignisablaufanalyse Ű Verfahren, graphische Symbole und Auswertung (German Standard) (1985)
9. IEC 60812 (International Electrotechnical Commission): Functional safety of electrical/electronical/programmable electronic safety/related systems, Analysis Techniques for System Reliability - Procedure for Failure Mode and Effect Analysis (FMEA) (1991)
10. IEC (International Electrotechnical Commission): Hazard and operability studies (HAZOP studies) - Application guide (2000)
11. UK Defence Standardization Organisation: Defence Standard 00-58, HAZOP Studies on Systems Containing Programmable Electronics, Part 1 and 2 (2000)
12. DIN 25424 (Deutsches Institut für Normung e.V.): Fault Tree Analysis: Part 1 (Method and graphical symbols) and Part 2 (Manual: calculation procedures for the evaluation of a fault tree) (1981/1990)
13. IEC 61025(International Electrotechnical Commission): Fault-Tree-Analysis (FTA) (1990)
14. Vesely, W.E., Goldberg, F.F., Roberts, N.H., Haasl, D.F.: Fault Tree Handbook. U. S. Nuclear Regulatory Commission (1996)
15. Mauri, G.: Integrating Safety Analysis Techniques, Supporting Identification of Common Cause Failures. PhD thesis, Department of Computer Science, University of York (2001)
16. IEC (International Electrotechnical Commission): IEC 61165: Application of Markov techniques (1995-2003)
17. Selic, B., Gullekson, G., Ward, P.: Real-Time Object Oriented Modeling. John Wiley & Sons (1994)
18. Fenelon, P., McDermid, J.A.: An integrated toolset for software safety analysis. Journal of Systems and Software **21** (1993) 279–290

19. 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 (SCS'03), Adelaide (2003)
20. Frolund, S., Koistinen, J.: Quality-of-service specification in distributed object systems. Technical Report HPL-98-159, Hewlett Packard, Software Technology Laboratory (1998)
21. Reussner, R.H., Poernomo, I.H., Schmidt, H.W.: Reasoning on software architectures with contractually specified components. In Cechich, A., Piattini, M., Vallecillo, A., eds.: Component-Based Software Quality: Methods and Techniques. Number 2693 in LNCS. Springer-Verlag, Berlin, Germany (2003) 287–325
22. Reussner, R.H., Schmidt, H.W., Poernomo, I.: Reliability prediction for component-based software architectures. Journal of Systems and Software – Special Issue of Software Architecture - Engineering Quality Attributes **66** (2003) 241–252
23. Reussner, R.H.: Automatic Component Protocol Adaptation with the CoCoNut Tool Suite. Future Generation Computer Systems **19** (2003) 627–639
24. Grunske, L.: Annotation of component specifications with modular analysis models for safety properties. In: Proceedings of the 1st International Workshop on Component Engineering Methodology (WCEM), Erfurt (2003) 737–738
25. Bryant, R.: Graph-based algorithms for boolean function manipulation. IEEE Transactions on Computers **35** (1986) 677–691