# Formal Support for Quantitative Analysis of Residual Risks in Safety-Critical Systems

Jonas Elmqvist and Simin Nadjm-Tehrani
Department of Computer and Information Science
Linköping University
Linköping, Sweden
{jonel, simin}@ida.liu.se

## Abstract

*With the increasing complexity in software and electronics in safety-critical systems new challenges to lower the costs and decrease time-to-market, while preserving high assurance have emerged. During the safety assessment process, the goal is to minimize the risk and particular, the impact of probable faults on system level safety. Every potential fault must be identified and analysed in order to determine which faults that are most important to focus on.*

*In this paper, we extend our earlier work on formal qualitative analysis with a quantitative analysis of fault tolerance. Our analysis is based on design models of the system under construction. It further builds on formal models of faults that have been extended for estimated occurence probability allowing to analyse the system-level failure probability. This is done with the help of the probabilistic model checker PRISM. The extension provides an improvement in the costly process of certification in which* all *forseen faults have to be evaluated with respect to their impact on safety and reliability. We demonstrate our approach using an application from the avionic industry: an Altitude Meter System.*

## 1 Introduction

A current trend in many safety-critical domains such as the automotive and aerospace industries is an increasing complexity in electronics and software. This is due to the introduction of more advanced features in cars and aircrafts and replacing mechanical functions with software and electronics. In order to deal with the complexity and achieve efficient upgrades of systems, the Integrated Modular Avionics (IMA) concept [28] in avionics and AUTOSAR [2] in the automotive industry are proposed. These are intended to ease the deployment of replaceable units efficiently. One big challenge in these industries is now to lower the costs and decrease time-to-market while preserving high assurance. A major cost for deployment of electronic units and software is the mounting costs of safety assessments as systems get more complex and updates more frequent.

The main goal of the safety assessment process is to minimize the probability of *hazards* [22]. A hazard is a state of the system that may lead to an accident, e.g. a system failure due to a *fault* inside or in the environment of the system. Ideally, every hazard should be removed which means that the occurrence of every potential fault in the system (or in its environment) must be removed or the effect of the fault must be mitigated by the system. This is impossible in practice for complex digital systems as in the avionic and automotive industries since these systems operate in harsh environments and can be affected by internal or external natural faults in hardware [3]. Instead, safety engineers strive for minimizing the *risk*, which is characterized by the severity and the likelihood of occurrence of the given hazard. Hence, one prominent issue in the safety assessment process is to know which faults are tolerated by the system and to know the severity and likelihood of risks posed by the residual faults.

There exists a wide variety of qualitative and quantitative methods for analysing safety, for example Failure Mode and Effects Analysis (FMEA) and Fault-Tree Analysis (FTA) [16]. However, many of these techniques become intractable to use for systems with embedded software. First of all, deriving the failure propagation inside the system becomes tedious or nearly impossible by hand, and the resulting fault trees are extremely large. One way of dealing with the increased complexity in safety assessment is to integrate the two separate activities of functional design and safety assurance through introduction of formal models [7, 14]. With this approach, safety assessment is based on the system design model and formal fault models, and formal analysis. This model-based approach enables verification tools, such as model checkers, to automatically check, at design time, if the system design tolerates the modelled faults.

Our earlier work has combined the component-based approach with the development of safety-critical systems while integrating design and safety analysis using the concept of *safety interfaces* [9, 8]. Briefly, the safety interface can be seen as a formal description of the effect on the behaviour of a component when single or multiple (known) faults are present. This approach enables a compositional safety analysis where the safety interfaces of components are used for overall system safety analysis. By reusing earlier verification results, this approach is particularly effective during component upgrades and reuse. However, not all faults can be taken care of in this manner. Some faults are shown to be *not*-tolerated by a system design, and some are simply incompletely analysed since the computation complexity barrier leads to non-terminated proofs.

In this paper, we extend our work with a method for quantitative analysis of residual risks by using probabilistic model checking. This process begins where the qualitative analysis stops, i.e. quantifying the probability of a system level failure based on the non-tolerated faults found in the initial qualitative analysis. By defining the probability of each potential fault, we are able to derive the system failure probability using the probabilistic model checker PRISM. This approach enables safety engineers to analyse the residual faults and quantifying their severity in order to identify which faults are most important to focus on.

The contribution of the paper can be summarised as follows. We present a method that supports analysis of fault tolerance in a system with embedded electronic components. Given a description of the system in the form of a formal model for each component, and the derived safety interfaces (as presented in earlier work), our method narrows down the range of faults to be considered for quantitative evaluation. These are the faults that can be excluded from the focus of the safety assurance process through qualitative analysis. We then propose to focus on faults for which (a) the qualitative process did not terminate or (b) the proof of fault tolerance ended with a negative answer. For these faults we propose an extension of the concept of safety interface so that each considered fault can be modelled with a probability of occurrence. Finally, we illustrate the use of probabilistic model checking for quantifying the probability of a system level failure (much in the spirit of FTA or FMEA, but using formal analysis tools).

The paper is structured as follows: Section 2 describes the motivation behind this work and gives an overview of the related work in this area. Section 3 gives an overview of our previous work including some basic definitions needed, and a brief introduction to probabilistic reasoning and PRISM. In section 4, we present our approach for a quantified analysis of residual risks. In Section 6 we illustrate our approach on a case study from the aerospace industry. Section 7 concludes the paper and presents future work.

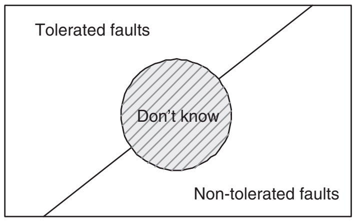


**Figure 1. Three sets of faults**

## 2 Motivation and Related Work

The safety assessment process is a complex and time consuming process encompassing the whole system life cycle. As mentioned in the introduction, the goal of safety assurance is to minimize the likelihood of hazards. Our previous work on qualitative safety analysis enables the safety engineer to ascertain, compositionally, which faults in a given set of potential faults the system is resilient to [8].

Ideally, the output of the analysis is two sets of faults, one set of tolerated, and a set of non-tolerated faults in the system. However, in some cases, due to combinatorial explosion in models, formal techniques for generating safety interfaces do not terminate (discussed in [9]). Thus, in these cases we get a third set of faults, marked as "don't know" (see Figure 1).

The safety assessment process of course does not stop here with identifying these three sets of faults. We can put aside the faults we know the system can tolerate and the next step is to focus on mitigating actions to avoid or minimize the effect of the residual faults. Perhaps, we must re-design our system in order to tolerate the fault, or try to minimize the probability of the fault becoming active. However, an in-depth analysis of all faults and trying to take care of every residual fault is both time consuming and costly. This is even more significant if every revision of the system would require going through the whole process again (which is the cause of current costly process). The question then boils down to deciding which are the non-tolerated faults that should be focused on.

In this work, we focus on analysing the two sets of "non-tolerated" faults and "don't know" faults in order to quantify the residual risks. The composed model of the system is exposed to a formal model of an injected (non-tolerated) fault, whereby the safety engineer can derive the probability of the system safety being affected by occurance of that fault.

## 2.1 Related work

Probabilistic model checking has been used for quantitative reliability evaluation in a number of case studies. For example, in [25], PRISM is used for analysing the reliability of von Neumann's NAND multiplexing technique. In [20], reliability of a simplified embedded system is evaluated. The work we present in this paper is also closely related to the approach proposed by Grunske et. al. [11]. They propose a method for probabilistic model-checking support for FMEA. In their approach, probabilistic FMEA tables are generated with the help of PRISM based on behavioural models that have been augmented with information about component failures. However, these approaches require explicit modelling of failure behaviour and propagation *inside* a component. Once the analysis is completed the fault models are excluded, and the functional models can be used for code generation and integration. In our work, propagation inside the components already exists in the formal functional model and the faulty behaviour is modelled as "external" modules interacting with the system. Also, by using the result from our qualitative analysis, as input to the quantitative analysis, we already have narrowed down the number of non-tolerated faults that need to be analysed.

Probabilistic model checking has also been used for performance evaluation of dependable system architectures [5, 31, 32, 23]. These approaches address reliability modelling and enable preliminary evaluations of the system dependability during the early phases of the development process. Based on the evaluation, architectural decisions can be made in order to achieve high reliability. Even though not a primary goal in our work, their approach of using evaluation results as input to an architectural decision process could be applied in our setting.

There are a number of approaches addressing tool support for qualitative reliability analysis using techniques for automatic generation of FMEA tables or fault trees [12, 26, 27, 30]. However, these approaches do not base their fault modelling on formal design models. Instead, they require safety engineers to specify impact of component failures on the system which requires in-depth knowledge of the system. In some sense, this is exactly what our design models facilitate and automate. The work by Joshi et. al [18, 17] on model-based safety-analysis is similiar to our approach on qualitative analysis and has no probabilistic element.

# 3 Background

This section presents some basic definitions and gives an overview of our previous work regarding compositional qualitative safety analysis.

## 3.1 Basic Definitions

In our framework for qualitative analysis, we have chosen to represent the systems using a synchronous formalism based on the notion of *reactive modules* [1]. The definitions below are restated from earlier work [9, 8] in order to clarify the general modelling approach.

**Definition 1 (Module)** *A synchronous module $M$ is a triple $(V, Q_{init}, \delta)$ where*

- $V = (V_i, V_o, V_p)$ *is a set of typed variables, partitioned into sets of input variables $V_i$, output variables $V_o$ and private variables $V_p$. The controlled variables are $V_{ctrl} = V_o \cup V_p$ and the observable variables are $V_{obs} = V_i \cup V_o$;*
- *A* state $q$ *is an interpretation of the variables in $V$. The set of controlled states over $V_{ctrl}$ is denoted $Q_{ctrl}$ and the set of input states over $V_i$ as $Q_i$. The set of states for $M$ is $Q_M = Q_{ctrl} \times Q_i$;*
- $Q_{init} \subseteq Q_{ctrl}$ *is the set of initial control states;*
- $\delta \subseteq Q_{ctrl} \times Q_i \times Q_{ctrl}$ *is the* transition relation.

The state space for the module is determined by the values of its variables. For a state $q$ over variables $V$ and a subset $V' \subseteq V$, $q[V']$ denotes the projection of $q$ onto the set of variables $V'$. The successor $q'$ of a state $q$ is obtained at each round by updating the controlled variables of the module according to the transition relation $\delta$.

The execution of a module produces a state sequence $\bar{q} = q_0 \dots q_n$. A trace $\bar{\sigma}$ is the corresponding sequence of observations on $\bar{q}$, with $\bar{\sigma} = q_0[V_{obs}] \dots q_n[V_{obs}]$.

A property $\varphi$ on a set of variables $V$ is defined as a set of traces over $V$. This work focuses on *safety properties* [24, 13] as opposed to liveness properties.

**Definition 2 (Safety property)** *A safety property $\varphi$ is a set of traces over a set of variables $V$ such that for all traces $\sigma$, $\sigma \in \varphi$ iff every finite prefix $\sigma'$ of $\sigma$, is in $\varphi$.*

Generally, safety properties are used to model critical requirements that a system needs to fulfill (or *satisfy*).

**Definition 3** *A module $M$ satisfies a property $\varphi$, denoted $M \models \varphi$ iff every trace of $M$ (projected on the variables of $\varphi$) belongs to (the set of traces) $\varphi$.*

Composing simple modules in order to model complex modules is a necessity. Here, we define the parallel operator:

**Definition 4 (Parallel composition)** *Let $M = (V^M, Q^M_{init}, \delta^M)$ and $N = (V^N, Q^N_{init}, \delta^N)$ be two modules with $V^M_{ctrl} \cap V^N_{ctrl} = \emptyset$. The parallel composition of $M$ and $N$, denoted by $M \parallel N$, is defined as*

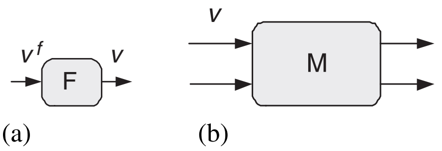


**Figure 2. a) Input fault mode. b) Module.**

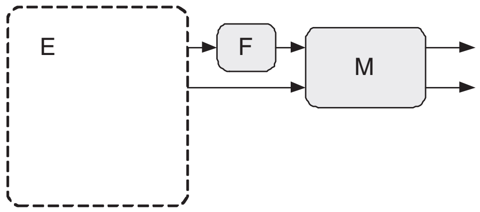


**Figure 3. A fault mode affecting the input to M.**

- $V_p = V_p^M \cup V_p^N$
- $V_o = V_o^M \cup V_o^N$
- $V_i = (V_i^M \cup V_i^N) \setminus V_o$
- $Q_{init} = Q_{init}^M \times Q_{init}^N$
- $\delta \subseteq Q_{ctrl} \times Q_i \times Q_{ctrl}$ *where* $(q, i, q') \in \delta$ *iff* $(q[V_{ctrl}^M], (i \cup q)[V_i^M], q'[V_{ctrl}^M]) \in \delta^M$ *and* $(q[V_{ctrl}^N], (i \cup q)[V_i^N], q'[V_{ctrl}^N]) \in \delta^N$.

In traditional safety analysis, faults can be classified into the following high-level categories: *omission faults*, *value faults*, *commission faults* and *timing faults* [6, 9, 3]. In this work, we do not focus on timing faults and our work does not include the process of *identifying* fault modes, which is itself a different research topic.

We model faults in the environment as delivery of faulty input to the component and call each such faulty input a *fault mode* for the component. The faulty behaviour is explicitly modelled in a *new* module that is placed in between the environment and the affected module. The input fault of one component thereby corresponds to the output fault of a component connecting to it.

To be able to apply formal analysis of the behaviour of a component in presence of faults in its environment, we need to define a formal model of the faults and a fault composition operator.

**Definition 5 (Input Fault Mode)** *An input fault mode $F$ of a module $M$ is a module with one input variable $v^F \notin V^M$ and one output variable $v \in V_i^M$, both of the same type $D$ (see Figure 2).*

**Definition 6 (Fault Composition)** *Let $E$ be an environment (modeled as a module) with an output $v^E$ and $F$ be a fault mode with input $v^F$ and output $v \in V_o^E$. We denote $E \circ F = E[v^E/v^F] \parallel F$ where $E[v^E/v^F]$ is the module $E$ with the substitution $v^F$ for $v^E$.*

Consider a module $M$ (see Figure 2 b), an environment $E$ and a fault mode $F$ (see Figure 2 a)) that affects the variable $v$ from $E$ to $M$. We model this formally as a composition of $F$ and $E$, which has the same variables as $E$ and can then be composed with $M$. In the resulting faulty environment $E \circ F$, the original output $v$ of $E$ becomes the input $v^F$ of $F$, which produces the faulty output $v$ as input to $M$ (see Figure 3).

Our technique for capturing fault modes is general and can model arbitrary faults that affect environment outputs in unrestricted and arbitrary ways.

Given a module, we wish to characterize its fault tolerance in an environment that represents the remainder of the system together with any external constraints. Whereas a module represents an implementation, we wish to define an interface that provides all information about the component that the system integrator needs. Traditionally, these interfaces do not contain information about safety or fault tolerance of the component. We have defined a safety interface that captures the resilience of the component in presence of faults in the environment with respect to a given safety property $\varphi$.

The safety interface makes explicit which single and double faults the component can tolerate, and the corresponding environments capture the assumptions that $M$ requires for resilience to these faults.

**Definition 7 (Safety Interface)** *Given a module $M$, a system-level safety property $\varphi$, and a set of fault modes* **F** *for $M$, a safety interface $SI^\varphi$ for $M$ is a tuple $\langle E^\varphi, \mathsf{single}, \mathsf{double}\rangle$ where*

- *$E^\varphi$ is an environment in which $M \parallel E^\varphi \models \varphi$.*
- $\mathsf{single} = \{\langle F_1^s, A_1^s\rangle, \ldots, \langle F_n^s, A_n^s\rangle\}$ *where* $F_j^s \in \mathbf{F}$ *and $A_j^s$ is a module composable with $M$, such that* $M \parallel (A_j^s \circ F_j^s) \models \varphi$
- $\mathsf{double} = \{\langle F_1^d, A_1^d\rangle, \ldots, \langle F_n^d, A_n^d\rangle\}$ *with* $F_k^d = \{\langle F_k^1, F_k^2\rangle \mid F_k^1, F_k^2 \in \mathbf{F},\ F_k^1 \neq F_k^2\}$ *such that* $M \parallel (A_k^d \circ (F_k^1 \parallel F_k^2)) \models \varphi$

Note that $E^\varphi$ is an environment that the component needs in order to satisfy the safety property $\varphi$ in absence of any faults. $A^s$ and $A^d$ respectively are further constraints on the environment for the component to be resilient in presence of single (respectively double) faults. Intuitively, the safety interface tells the system integrator what the component developer knows about the component and its impact on the safety property $\varphi$. If the component developer knows nothing about the resilience of the component, then the single/double part of the safety interface is empty.

**Definition 8 (Component)** *Let $\varphi$ be a system-level safety property, $M$ a module and $SI^{\varphi}$ a safety interface for $M$. A component $C$ with a safety interface for property $\varphi$ is the tuple $\langle M, SI^{\varphi} \rangle$.*

### 3.2 Qualitative Analysis using Safety Interfaces

In earlier work we have demonstrated how qualitative compositional reasoning about safety can be performed with help of our modelling approach [9, 8]. The formalism we have presented above allows us to use any modelling and analysis tool that shares this underlying semantics. In all our practical work so far concerning qualitative reasoning, we have used the synchronous tools SCADE [34] and Esterel Studio [33] for modelling and analysis.

We have derived assume-guarantee reasoning rules in presence of fault modes [9] which enable us to perform compositional reasoning about the fault tolerance in the system.

The input to this analysis is the set of potential faults in the system and its environment. The assume-guarantee reasoning determines whether the whole assembly satisfies the specific safety properties in presence of these faults. For qualitative reasoning, we focus on only single and double faults with no loss of generality, since higher number of simultaneous faults are typically shown to be unlikely.

Assume we want to check whether the system consisting of a set of modules $M_1, ..., M_n$ can tolerate single fault $F_i$ affecting component $C_m$, i.e.:

$$(M_1 \parallel \ldots \parallel M_{m-1} \parallel M_{m+1} \parallel \ldots M_n) \circ F_i \parallel M_m \models \varphi \quad (1)$$

The assume-guarantee rules enable us to decompose this formula into a number of premises to check ($\mathcal{O}(n^2)$), were each individual check is less complex than the composed formula.

### 3.3 Probabilistic Model Checking and PRISM

Our quantitative approach uses the PRISM tool [19], a probabilistic model checker that supports three types of probabilistic models: discrete-time Markov chains (DTMCs), continuous-time Markov chains (CTMCs) and Markov decision processes (MDPs), and the two logics Probabilistic Computation Tree Logic (PCTL) and Continuous Stochastic Logic (CSL) [4, 21]. Here we give a brief overview, for a detailed overview of probabilistic model checking, see for example[21]. For more information about the PRISM tool and the range of case studies to which it has been applied, see [19] or the tool website [29].

The PRISM language is based on the reactive module formalism [1], i.e. the same formalism which our modules are based on. A system in PRISM consists of a set of parallel modules and each module has a set of local variables. The state space of a probabilistic model described in the PRISM language is the set of all possible evaluations of the modules variables. The behaviour of each module is described by commands, which include a guard and one or more updates. The guard is a predicate over the variables in the module and all global variables in the system. The update describes the actual state transitions and each update is also associated with a probability. The transition is enabled if the guards are true, and one of the updates can be chosen with the defined probability.

For example, a typical transition is expressed as:

```
[] <guard> -> <rate_1> : <update_1> +
           ... + <rate_n>: <update_n>;
```

In the command above, if the guard is true, then one of the update actions is chosen with the probability of the rate associated with it. The square brackets `[]` marks the start of the transition and allows synchronization between multiple transitions by placing a common action inside the brackets. For example, marking two transitions in two different modules with `[tick]` forces the modules to make synchronous transitions.

The main differences between the different models (DTMCs, CTMCs, and MDPs) are how time is modelled, how transition probabilities are modelled and the behaviour upon composition. DTMC models time as discrete time steps and the transition probabilities are also discrete. This makes them suitable for modelling and analysing simple probabilistic behaviour. MDPs extend DTMCs by adding the possibility of combining nondeterminism and probability. CTMCs provides modelling of continuous time and the ability to define properties in continous time. We model our systems using CTMCs since it allows translating reliability requirements to probabilities of failure in the model more naturally (see Section 4.3).

### 3.4 Property specification

In order to evaluate the specified probabilistic models we first need to specify relevant properties. PRISM provides two types of temporal logics for this; PCTL (for DTMCs and MDPs) and CSL (for CTMCs), probabilistic extensions of the classical temporal logic CTL.

The main operators used to define properties are

- `P` which refers to the probability of an event occurring.

- S which is used to reason about the steady-state behaviour of a model.

The P-operator is used to check the probability of a specific path property. A path property is evaluated to either true or false for a single path in a model defined using the traditional operators:

- X: "next
- U: "until"
- F : "eventually" or "future"
- G : "always" or "globally

The P-operator can be used in two ways: either to check *if* the probability that a path property holds meets a specified bound, or for actually computing a bound. For example, if *prop* is a path property and *bound* is a probability bound, this is written:

$$\mathtt{P} >_{bound} [prop] \quad (2)$$

Probability bounds can the be defined using $>, <, \leq, \geq$. The answer to the query is then yes/no. One can also use the P-operator to compute the probability of a specified behaviour in the model. This is formulated as follows:

$$\mathtt{P} =_{?} [prop] \quad (3)$$

With this formulation, the model checker will calculate the actual probability that *prop* holds. This is how we incorporate probabilistic reasoning using PRISM in our methodology.

Time bounds can also be added to the operators U, F, and G. For example, a bounded until property can be written $prop_1$ U $prop_1$ where time can be used to define upper limit ($>= t$), lower limit ($<= t$), or time intervals $[t_1, t_2]$ and $prop_1$ and $prop_2$ are two state properties.

# 4 Quantitative Analysis

## 4.1 Probabilities and Safety Interfaces

The safety interface serves as a specification of the fault resilience of a component. Until now, we have only been able to reason qualitatively. We now need to assign probabilities into our model in order to be able to reason quantitatively.

Our typical control system is deterministic and in absence of faults can be modelled by PRISM modules in which transitions have a probability of 1, while we want our fault modes to be probabilistic. This means that we have to add a probability attribute in order to model the likelihood of each fault mode. In this case, we will add the attribute *failure rate* (in seconds) for each fault mode in the safety interface. For example, if we (by some means) know that a specific sensor will at most fail once every year, the failure rate of the sensor, denoted $\lambda_s$, will be $1/(365*24*60*60)$.

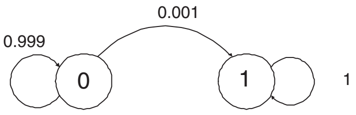


**Figure 4. A probabilistic fault mode**

## 4.2 Modelling Probabilistic Fault Modes

Compared to the non-probabilistic fault modes modelled earlier, these fault modes have a probability associated with them. Each fault mode has two states, one initial state where the fault is inactive, and a second state where the fault is active. In figure 4, an automata corresponding to the fault mode $F$ is depicted. Here, the initial state is 0, and the "active fault" state is 1. The transition between the normal state and the "active fault" state is labelled with a probability $P(F_i)$ (corresponding to the rate) for fault mode $F_i$. Figure 4 shows that the probability of the fault mode becoming active is 0.001.

## 4.3 Probabilistic Analysis using PRISM

Our qualitative analysis only answered the question *whether* a system would tolerate a specific fault. Now, when analysing the system quantitatively, we would like to know the *probability* of tolerating the faults. Since we have added failure rates to failure modes and now have a probabilistic system, we now have to formulate our assurance goals also probabilistically.

For example in avionics, safety is typically related to the number of system level failures per flight hour. As an example, for a military aircraft a safety requirement might be formulated in terms of absence of reliability breaches for some subsystems, e.g. at most one Level A failure in $10^6$ flight hours, or at most one Level B failure in $10^4$ flight hours.

In order to specify the relevant property in temporal logics (as briefly introduced in Section 3.4), we'll use the P-operator to find the actual probabilities, and we use the the bounded "until"-operator (U) to specify a path property. Thus, we should write:

$$\mathtt{P} =_{?} [\, \mathtt{true}\; U^{\leq T} \neg\varphi \,] \quad (4)$$

The above CSL formula denotes the question: "what is the probability that the safety property $\varphi$ ceases to be valid by time T from the initial state". This means, by setting T to $10^6 * 60 * 60$ [s], we will get the answer how probable it is that a system level failure (non-valid safety property) will occur during $10^6$ flight hours.

## 5 Computing probability of failure

We have now shown how a system model $M$, a fault mode $F$, and a safety property ($\varphi$) is represented in PRISM. To compute the residual risk resulting from presence of a non-tolerated fault, we need to compute what is the probability that the system jeopardizes property $\varphi$ in presence of the fault $F$. To do this we use the built-in model checker in PRISM.

The tool chain supporting our framework is presented in Fig 5. As mentioned, the generation of safety interfaces is automatically done using a front-end to Scade [8]. The output of this tool is safety interfaces for every component which are then used in the compositional qualitative safety analysis (as described in earlier work [8, 9]). That step is supported by the built in Design Verifier in Scade. The output of this process is three sets of faults, tolerated, non-tolerated and “don’t know”.

The contribution in this paper is the follow up quantitative analysis of the residual faults, thus enabling the computation of the residual risks.

## 6 Case study: Altitude Meter

To illustrate the use of probabilistic safety interfaces we have applied the approach on a digital Altitude Meter subsystem.

The Altitude Meter subsystem calculates the altitude of an unmanned aerial vehicle (UAV) above a fixed level. Input to the Altitude Meter system is the atmospheric pressure supplied from two static ports outside the aircraft. The pressure is then transformed into a corresponding altitude. This value is then used by the UAV for planning and controlling the flight. This means that the Altitude system is a safety-critical function of the UAV, since an incorrect value from it can have severe consequences. In order to achieve high reliability and fault tolerance of the altitude system, redundancy is added to parts of the subsystem.

Generally, during nominal behaviour of the aircraft, the altitude system calculates accurate altitude. However, during strong climbs and descents, the system might lag behind the aircraft’s actual altitude. Hence, some compensation is necessary for this reason, which is done by using other sensor values, such as air speed and vertical acceleration.

### 6.1 Architectural view

To reduce the complexity and to illustrate a component-based approach to this case study suitable for future IMA solutions, the functionality of the Altimeter is divided into four different types of components (see Figure 6):

| *Fault* | *Type* | *Affected Component* |
|---|---|---|
| $F_1$ | Value | ADC1 |
| $F_2$ | Omission | Alt. Func. 1 |
| $F_3$ | Value | RS-485 |

**Table 1. Fault modes**

- **ADC** The Air Data Computers (ADCs) are advanced transducers that convert the input data from the sensors (pressure) to an altitude. This is the “pure” altitude value, without any correction or filtering. The system consists of three ADCs, and all of them send their status as output to the System Computer (SC).
- **Altitude function** The altitude function’s goal is to filter and correct the altitude in order to get as accurate a value as possible. This is done taking the air speed and the aircraft’s acceleration into account. The system consists of two versions of the Altitude function, both run on the System Computer.
- **Voter** the role of the Voter is to compare the outputs from the Altitude Function subsystems and decide which of these values to use as output from the system.

The two ADCs (ADC1 and ADC2) are connected directly to the System Computer. Inside the SC, the values from the ADC1 and ADC2 are filtered and corrected. The altitude function filters the altitude and compensates for the UAV air speed and vertical acceleration. Output from these are sent to the Voter. The ADCs also emit a 2 bit signal, indicating “ok”, “degraded” or “total outage” which the voter may use for fault detection.

To cope with any malfunction of the SC, the altitude from the ADC3 is directly connected to the Voter with an RS-485 bus.

### 6.2 Safety Requirements and Fault modes

A part of the preliminary safety assessment process is to identify possible faults in the system. In this case study, we have considered and modelled three types of fault modes. A value fault mode $F_1$ of the signal from sensor $S1$ (input to $ADC1$), an omission fault mode $F_2$ of the altitude signal from ADC1 to Altitude Function 1 modelling a communication failure, and a value fault mode $F_3$ on the RS-485 bus (see Table 1).

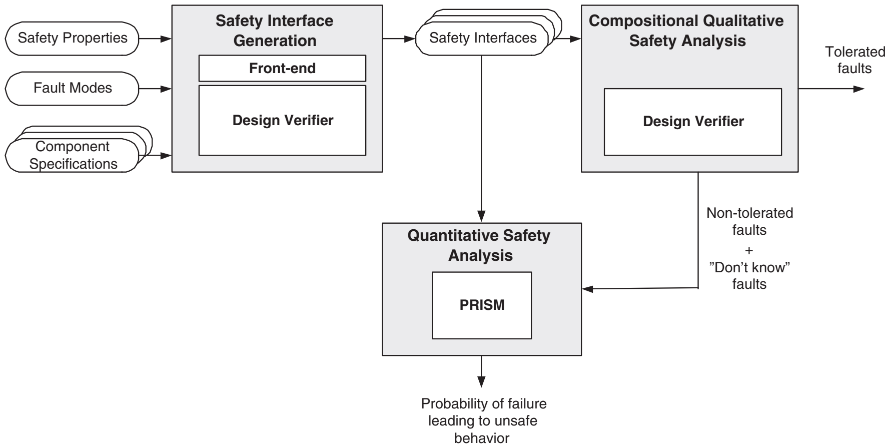


**Figure 5. The tool chain.**

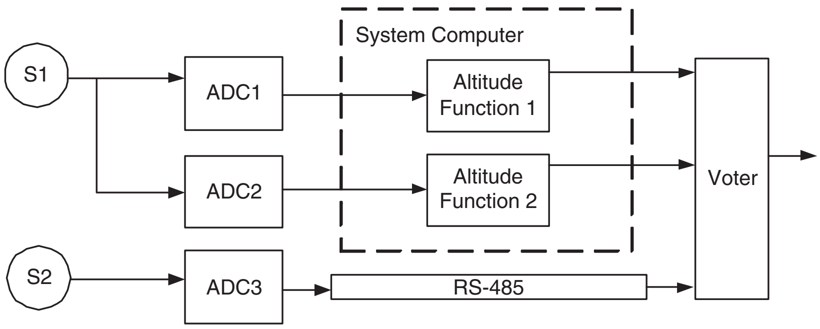


**Figure 6. The Altitude meter architecture.**

| *Fault* | *Non-Tolerated* | *Tolerated* |
|---|---|---|
| $F_1$ | • | |
| $F_2$ | | • |
| $F_3$ | | • |
| $\langle F_1, F_2 \rangle$ | • | |
| $\langle F_2, F_3 \rangle$ | • | |
| $\langle F_1, F_3 \rangle$ | • | |

**Table 2. Result of the qualitative analysis**

### 6.3 Compositional Qualitative Safety Analysis

Initially, the safety interfaces were generated using the EAG-algorithm [9] that is implemented as part of a front-end to the modelling tool Scade with its built in verification tool Design Verifier [10].

Using the safety interfaces, a qualitative safety analysis was performed. The result was that both $F_2$ and $F_3$ were tolerated by the system while $F_1$ was not tolerated (the safety property did not hold in presence of F1). Neither were the double faults $\langle F_1, F_2 \rangle$, $\langle F_1, F_3 \rangle$, $\langle F_2, F_3 \rangle$ tolerated (as seen in Table 2).

### 6.4 Quantitative Analysis of Fault Tolerance

The next step in the analysis is to focus on the residual fault $F_1$. To find out how this fault may affect the safety property in a quantitative setting we need to follow the work flow in the bottom part of Figure 5.

The Scade models of the Altitude meter were then translated into CTMC. In order to model synchronous events, where two modules make transitions in parallel, we use the synchronization feature in the PRISM language. By labelling commands with a *shared action* `tick`, we force the modules to make transitions simultaneously.

```
[tick] <guard> -> <rate> : <update>;
```

In the implementation, the `tick` action is triggered with the specific execution rate of the system (in our case 15 Hz).

Fault modes are modelled with probabilistic transitions in order be invoked as specified. The controller design, however, is modelled as deterministic modules as before, but in the PRISM language. This means that all transitions have the probability 1, as long as the guards are enabled.

The RS-485 bus is naively modelled as a simple buffer but with the possibility to send the status (correct or incorrect) of the signal.

Table 3 shows the fault modes and the assigned failure rates. This paper does not address how these probabilities are derived. If the components in question are hardware, the probabilities are normally derived by experimental testing and fault injection.

| *Fault* | *Failure rate* | $\lambda$ |
|---|---|---|
| $F_1$ | Once every month | 1/(30*24*60*60) |
| $F_2$ | Twice every year | 2/(365*24*60*60) |
| $F_3$ | Once every year | 1/(365*24*60*60) |

**Table 3. Fault modes with probabilitites**

### 6.5 Experimental results

Having modelled the fault modes (with probabilistic behaviour), and the system design described in the PRISM language, we can now compose the whole system. The qualitative analysis typically considers single or double faults, since the probability of more than two simultaneous faults is assumed to be very low. If that approach is considered as optimistic, and we wish to allow more than two faults at a time, this can be done by simply composing any faults that may be considered likely to happen simultaneously. Here we show the composition of all three faults in the Altitude meter system.

$$ADC1 \circ F_1 \parallel ADC2 \parallel ADC3 \parallel ALT1 \circ F_2 \parallel ALT2 \parallel Voter \parallel RS485 \circ F_3$$

where ALT = Altitude Function. Then, we formulate the probabilistic safety property that PRISM can compute.

$$\mathtt{P} =_? \ [\, \mathtt{true} \ U^{\leq T} \ \neg\varphi \,] \tag{5}$$

where $\varphi$ is defined as "the voter should always emit a valid altitude". Valid in this sense, is defined as being within a specific interval around the correct altitude, which means that the system tolerates small deviations without a system failure. In our specific case study, we tolerate a deviation of $\pm 10$ meters. We also set the time bound T as $10^6$ flight hours.

The result of the model checking confirms that the probability of a system level failure (i.e. that the voter emits an non-valid altitude) within $10^6$ flight hours is 0.9923.

Further, to quantify the likelihood of the individual single (double) faults in the system, we can check the probability of failure for each (non-tolerated) fault (pair) one by one. In our case we need $F_1$, $\langle F_1, F_2 \rangle$, $\langle F_1, F_3 \rangle$ and $\langle F_2, F_3 \rangle$ to be present (see Table 4).

This enables the safety engineer to identify which of the potential faults in the system needs further action, i.e. architectural changes or other remedial/mitigation actions.

The analysis shows that

| *Faults "enabled"* | *System failure probability* |
|---|---|
| *All* | 0.9923 |
| $F_1$ | 0.8778 |
| $\langle F_1, F_2 \rangle$ | 0.8991 |
| $\langle F_2, F_3 \rangle$ | 0.8827 |
| $\langle F_1, F_3 \rangle$ | 0.8903 |

**Table 4. Probabilistic results**

- $F_1$ leads to a safety violation with a 0.8778 probability,
- $\langle F_1, F_2 \rangle$ leads to a safety violation with a 0.8991 probability,
- $\langle F_2, F_3 \rangle$ leads to a safety violation with a 0.8827 probability, and
- $\langle F_1, F_3 \rangle$ leads to a safety violation with a 0.8903 probability.

## 7 Conclusions

In this paper, we extended our earlier work on qualitative safety analysis with a quantitative analysis. By introducing probabilities into the fault modes we are able to analyse the system-level failure probability. The analyis focuses on (potential) non-tolerated faults.

First, we transform our fault modes into stochastic modules by adding probabilities to the transitions in each residual fault mode. Then, we translate our modules into the PRISM language and rewrite our safety properties into probabilistic properties. This enables the PRISM tool to be used for computation of the failure probability of the composed system.

By introducing the quantitative analysis on the same design models, we are able to reason about the faults that a formal verification engine in a qualitative setting does not manage to analyse. This extends the application of formal techniques in system safety analysis and provides valuable support to the safety engineer in constructing the safety case.

### 7.1 Future Works

This paper has presented a proof of concept for the quantitative analysis of the residual risks. So far, the translation of the system model from the qualitative analysis to the underlying PRISM model for the purpose of quantitative analysis was performed manually. Due to the common underlying semantics, this translation can obviously be performed automatically and is a straight forward extension to this work.

A more demanding extension, is the support for quantitative analysis during upgrades. As in earlier qualitative analysis we would like to be able to reuse as much of the earlier analyses as possible when one module of the system is replaced with a new slightly different version.

Our earlier work supports compositional qualitative analysis of safety-critical systems. However, our quantitative analysis is not compositional. This is due to the fact that generation of safety interfaces probabilitically requires the ability to generate counter examples from a probabilistic model checking exercise, which is an open research issue [15]. Future work in this area would be to develop a compositional quantitative method.

## 8 Acknowledgments

This work was supported by the Swedish National Aerospace research program NFFP4, and project SAVE financed by the Swedish Strategic Research Foundation (SSF). Thanks are due to our industrial collaborators in particular Sam Nicander from SAAB Aerosystems, Kristina Forsberg and Stellan Nordenbro from SAAB Avitronics for valuable inputs on the case study. The second author was partially supported by University of Luxembourg.

## References

[1] R. Alur and T. A. Henzinger. Reactive modules. In *Proceedings of the 11th Symposium on Logic in Computer Science (LICS '96)*, pages 207–218. IEEE Computer Society, 1996.

[2] AUTOSAR. http://www.autosar.org. URL, October 2006.

[3] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing*, 01(1):11–33, 2004.

[4] A. Aziz, K. Sanwal, V. Singhal, and R. K. Brayton. Verifying continuous time markov chains. In *CAV '96: Proceedings of the 8th International Conference on Computer Aided Verification*, pages 269–276, London, UK, 1996. Springer-Verlag.

[5] C. Baier, B. R. Haverkort, H. Hermanns, and J.-P. Katoen. Automated performance and dependability evaluation using model checking. In *Performance Evaluation of Complex Systems: Techniques and Tools, Performance 2002, Tutorial Lectures*, pages 261–289, London, UK, 2002. Springer-Verlag.

[6] A. Bondavalli and L. Simoncini. Failures classification with respect to detection. In *2nd. IEEE Workshop on Future Trends in Distributed Computing Systems*, pages 47–53, Cairo, Egypt, September 30 - October 2 1990. also Esprit PDCS (Predictably Dependable Computing Systems) report 1st Year Deliverables, 1990.

[7] M. Bozzano, A. Villafiorita, O. kerlund, P. Bieber, C. Bougnol, E. Bde, M. Bretschneider, A. Cavallo, C. Castel, M. Cifaldi, A. Cimatti, A. Griffault, C. Kehren,

B. Lawrence, A. Ldtke, S. Metge, C. Papadopoulos, R. Passarello, T. Peikenkamp, P. Persson, C. Seguin, L. Trotta, L. Valacca, and G. Zacco. ESACS: an integrated methodology for design and safety analysis of complex systems. In *ESREL 2003*, pages 237–245. Balkema, June 2003.

[8] J. Elmqvist and S. Nadjm-Tehrani. Safety-oriented design of component assemblies using safety interfaces. In *Third International Workshop on Formal Aspects of Component Software (FACS'06)*, pages 1–15, Prague, Czech Republic, September 2006. ENTCS.

[9] J. Elmqvist, S. Nadjm-Tehrani, and M. Minea. Safety interfaces for component-based systems. In R. Winther, B. A. Gran, and G. Dahll, editors, *SAFECOMP*, volume 3688 of *Lecture Notes in Computer Science*, pages 246–260. Springer Verlag, 2005.

[10] Esterel Technologies. *Design Verifier User Manual*, 2004.

[11] L. Grunske, R. Colvin, and K. Winter. Probabilistic model-checking support for FMEA. *Quantitative Evaluation of Systems*, pages 119–128, Sept. 2007.

[12] L. Grunske, P. A. Lindsay, N. Yatapanage, and K. Winter. An automated failure mode and effect analysis based on high-level design specification with behavior trees. In J. Romijn, G. Smith, and J. van de Pol, editors, *IFM*, volume 3771 of *Lecture Notes in Computer Science*, pages 129–149. Springer, 2005.

[13] N. Halbwachs, F. Lagnier, and P. Raymond. Synchronous observers and the verification of reactive systems. In *Third Int. Conf. on Algebraic Methodology and Software Technology*, pages 83–96. Springer Verlag, 1993.

[14] J. Hammarberg and S. Nadjm-Tehrani. Formal verification of fault tolerance in safety-critical reconfigurable modules. In *International Journal of Software Tools for Technology Transfer (STTT)*, volume 7, pages 195–291. Springer Verlag, June 2005.

[15] T. Han and J. P. Katoen. Counterexamples in probabilistic model checking. In O. Grumberg and M. Huth, editors, *Proceedings of the 13th International Conference on Tools and Algorithms for Construction and Analysis of Systems*, volume 4424 of *Lecture Notes in Computer Science*, pages 72–86. Springer Verlag, July 2007.

[16] E. Henley and H. Kumamoto. *Reliability Engineering and Risk Assessment*. Prentice Hall, 1981.

[17] A. Joshi and M. P. Heimdahl. Model-Based Safety Analysis of Simulink Models Using SCADE Design Verifier. In *SAFECOMP*, volume 3688 of *LNCS*, pages 122–135. Springer-Verlag, Sept 2005.

[18] A. Joshi and M. P. E. Heimdahl. Behavioral fault modeling for model-based safety analysis. In *HASE '07: Proceedings of the 10th IEEE High Assurance Systems Engineering Symposium*, pages 199–208, Washington, DC, USA, 2007. IEEE Computer Society.

[19] M. Kwiatkowska, G. Norman, and D. Parker. PRISM: Probabilistic symbolic model checker. In *Computer Performance Evaluation / TOOLS*, pages 200–204. Springer Verlag, 2002.

[20] M. Kwiatkowska, G. Norman, and D. Parker. Controller dependability analysis by probabilistic model checking. *Control Engineering Practice*, 15(11):1427–1434, 2006.

[21] M. Kwiatkowska, G. Norman, and D. Parker. Stochastic model checking. In *Formal Methods for the Design of Computer, Communication and Software Systems: Performance Evaluation*, volume 4486 of *Lecture Notes in Computer Science*, pages 220–270. Springer Verlag, june 2007.

[22] N. Leveson. *Safeware*. Addison-Wesley, 1995.

[23] I. Majzik, A. Pataricza, and A. Bondavalli. Stochastic dependability analysis of system architecture based on UML models. In *Architecting Dependable Systems*, volume 2677 of *Lecture notes in computer science*, pages 219–244. Springer Verlag, 2003.

[24] Z. Manna and A. Pnueli. *The temporal logic of reactive and concurrent systems: Specification*. Springer-Verlag, 1992.

[25] G. Norman, D. Parker, M. Kwiatkowska, and S. Shukla. Evaluating the reliability of NAND multiplexing with PRISM. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 24(10):1629–1637, 2005.

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

[27] Y. Papadopoulos, D. Parker, and C. Grante. Automating the failure modes and effects analysis of safety critical systems. In *Proc. of the 8th IEEE International Symposium on High Assurance Systems Engineering (HASE'04)*, March 2004.

[28] P. J. Prisaznuk. Integrated modular avionics. In *National Aerospace and Electronics Conf*, volume 1, pages 39–45, May 1993.

[29] PRISM Model Checker webpages. http://www.prismmodelchecker.org. URL, March 2008.

[30] D. Raheja. Software system failure mode and effects analysis (SSFMEA) – a tool for reliability growth. In *International Symposium on Reliability and Maintainability (ISRM'90)*, pages 271–277, Tokyo, Japan, 1990.

[31] R. H. Reussner, H. W. Schmidt, and I. H. Poernomo. Reliability prediction for component-based software architectures. *Journal of System and Software*, 66(3):241–252, 2003.

[32] G. Rodrigues, D. Rosenblum, and S. Uchitel. Using scenarios to predict the reliability of concurrent component-based software systems. In *Fundamental Approaches to Software Engineering*, volume 3442 of *Lecture Notes in Computer Science*, pages 111–126. Springer Verlag, 2005.

[33] E. Technologies. *Esterel Studio 5.0 User Manual*, 2004.

[34] E. Technologies. *Scade Suite 4.3 User Manual*, 2006.