2010 Fourth IEEE International Conference on Secure Software Integration and Reliability Improvement Companion

Analysing System Failure Behaviours With PRISM

Xiaocheng Ge, Richard F. Paige and John A. McDermid
Department of Computer Science, University of York, UK.
{xchge, paige, jam}@cs.york.ac.uk

Abstract—**The verification of safety-critical systems using formal techniques is not something new[15]. Traditionally, safety-critical systems are verified using hazard analysis techniques, e.g., fault tree analysis. As safety-critical systems have become larger and more complex, several analysis techniques with compositional capabilities were developed. However, these techniques were not able to analyse stochastic systems. In this paper, we present a model-based compositional safety analysis technique (i.e., failure propagation analysis) and explore the feasibility of integrating this safety analysis technique with techniques of probabilistic model checking, more precisely the PRISM model checker. By doing so, we make it possible to rigorously verify a model while system failure behaviours are quantitatively analysed.**

Index Terms—**Component-based safety assessment, PRISM, Model verification, Probabilistic analysis**

I. Introduction

The failure of safety-critical systems may result in a tragic loss of human life or property. Traditionally, safety-critical systems are verified using *hazard analysis* techniques. Typical techniques are Fault tree analysis (FTA) [2], Hazards and operability analysis (HAZOP) [10], and Failure modes and effects analysis (FMEA) [1]. Those techniques have been successfully applied to real-world safety-critical systems for many decades.

Safety-critical systems are becoming more and more complex, which makes the development and analysis of these complex systems more difficult. Component-based development has emerged as a promising solution for developing complex systems, via an approach of composing smaller, independently developed components into larger assemblies. However traditional hazard analysis techniques rely on a monolithic decomposition of the system with regards to the hierarchy of failure effects rather than on the system's architectural model; thus, they are inadequate to demonstrate effectiveness at an architectural level, where compositional capability is required. In order to improve the effectiveness of model-based safety analysis, new compositional techniques were proposed such as Failure Propagation Transformation Notation (FPTN) [8], Hierarchically Performed Hazard Origin and Propagation Studies (HiP-HOPS) [20], Component Fault Trees (CFT) [12], and Fault Propagation and Transformation Calculus (FPTC) [21].

A major limitation of all of these hazard analysis techniques is that the analysis heavily relies on the experience and skills of the analysts: a failure unknown to the analysts would not be covered, thus hazards related to the failure are not foreseen. The negative effects of the limitation are increasing as safety-critical systems also increase in complexity. As a result, there is a trend to use methods that are more automatic and exhaustive than hazard analysis, for example model checking [5], [11]. The verification of safety-critical systems using formal techniques is not new [15]. The application of model checking to safety-critical system verification can base on various formal models for example Petri nets [16], finite state machines [5], Statecharts [4], SCADE [7], and Altarica [4], [3].

The work presented in this paper is based on our previous work of FPTC [21], [19]. The major goal of our recent work is to explore the feasibility of integrate model checking techniques, more precisely probabilistic model checking, with compositional safety analysis technique because there is little research in this area so far. The contribution of this work is to show how a probabilistic model checking tool, i.e., PRISM, can support failure propagation analysis (via the FPTC approach of [21]), thus making it feasible to use hazard analysis techniques for stochastic systems.

The paper is structured as follows: the first section will briefly review applications of model checking to safety-critical systems and introduce the probabilistic model checker, PRISM[1]. Then we will demonstrate the use of PRISM for safety analysis; the demonstration is via example.

II. Background

An overall objective of our research is to explore and integrate model checking techniques with a compositional safety analysis technique: FPTC. In this section, we first review several compositional analysis techniques; then we describe the application of model checking techniques to the safety domain; and at the end, we introduce the probabilistic model checker – PRISM – used in our research.

A. Compositional safety analysis techniques

Approaches to compositional safety analysis are all based on the idea of carrying out safety analysis on an architectural model of a system: the architectural model, besides capturing components and connectors, encapsulates failure behaviour, and expresses how incoming *input failures* of individual components propagate or transform to their outgoing output failures. Such approaches attempt to break down system-level analysis and assessment into more manageable tasks, applied to individual components, and a system model is then produced by composing models of individual components. In general, the composition of individual component models is

[1]http://www.prismmodelchecker.org/

carried out by connecting output failures of a component to input failures of other components.

Failure Propagation and Transformation Notation (FPTN) [8] was the first such approach to be developed. FPTN aims to provide a simple notation to capture both system architecture and the way in which failures within that architecture interact.

The Hierarchically Performed Hazard Origin and Propagation Studies (HiP-HOPS) [20] method consists of three main stages: functional failure analysis, component failure analysis and fault tree synthesis. At the beginning of the HiP-HOPS process, the system is decomposed into subsystems and components that are abstractions of system functionality. Then safety engineers use a modification of classical FMEA – Interface-Focused FMEA (IF-FMEA) – to generate a model of the local failure behaviour of components. Once the local failure behaviour of all components are determined, fault trees are constructed by exploiting the structure of the hierachical model and information about failure behaviour of components that is contained in IF-FMEAs. HiP-HOPS also introduced the idea that a partial fault tree can be used to reflect the failure logic of an actual component, and input and output ports of different components can be glued together. This idea was influential in developing other failure analysis techniques, such as FPTC.

Similarly, [12] proposed a technique, called Component Failure Tree (CFT), which aims to help capture the failure logic of a component as a function of input failures and internal events. Components can be modelled independently and each component constitutes a namespace and hides all internal events from its environment so that CFTs can be archived and reused in different projects.

Compared with FPTN, both HiP-HOPS and CFT are tool-supported and automated safety assessment techniques. This is important because it is needed for successful technology transfer to industrial practise, since manually performed safety analysis techniques are error-prone or even infeasible for complex component-based systems.

A limitation of all of these methods is their inability to handle cyclic data- and control-flow structures in the system architecture. This is a major drawback of these techniques as many real-world control systems contain closed feedback loops. To overcome this limitation, two more recent approaches have been proposed: AltaRica [3] and Fault Propagation and Transformation Calculus (FPTC) [21].

The AltaRica language was a product of the ESACS (Enhanced Safety Assessment for Complex System) European project, which aimed to invested new safety techniques based on the use of formal design languages and associated tools. One of the advantages of using AltaRica is its supporting tools. A model expressed in the AltaRica language can be validated by graphical interactive simulation and symbolic model checking. It has been successfully applied to many industrial case studies.

FPTC is a compositional failure analysis technique, which solves the problem of cyclic dependencies in a failure propagation model by using fixed-point evaluation techniques. The calculus is supported by a tool implemented in a number of domain-specific languages (in this sense, the approach is similar to AltaRica) [19]. FPTC has been applied in several industrial case studies, including analysis of failure modes in engine controllers, and for FPGA-based systems.

¿From a high-level viewpoint, these compositional techniques have a similar conceptual foundation that components and failure mode flows in failure behaviour model correspond to the components and their connections in the system design. Differences can be identified in the representation (graphical or textual), tool and methodological support.

B. Model Checking in safety domain and PRISM

Model checking [6] is a collection of techniques for automatically analysing *reactive systems* [9], [17]. The inputs to a model checker are a (usually finite state) description of the system to be analysed and a number of properties, often expressed as formulae of temporal logic; these properties are expected to hold of the system. The model checker can confirm that the properties hold, or may report that they are violated, or may be unable to produce a response in a timely manner (due to restrictions on available computational resources).

The verification of safety-critical systems using formal techniques is not new [15]. The application of model checking to safety-critical system verification has been based on various formal models, e.g., Petri nets [16], finite state machines [5], Statecharts [4], SCADE [7], and Altarica [4], [3].

Formal verification techniques, and in particular model checking, offer a powerful and rigorous approach for establishing the truth of curtain properties of a complex system. We choose to use PRISM [13], [14] because it is a sound and powerful probabilistic model checker which is well-suited to the purpose of analysing stochastic systems. In the case of probabilistic model checking, models are typical variants of Markov chain, in the sense that they encode the probability of making a transition between states instead of simply expressing the existence of a transition. Probabilistic model checking is an automatic procedure for establishing if a desired property holds in a probabilistic system model. PRISM requires two inputs:

- a description of the system to be analysed, typically given in some high-level modelling language;
- a formal specification of quantitative properties of the system that are to be analysed, usually expressed in variants of temporal logic.

The first input is a probabilistic variant of a state-transition model: each state represents a possible configuration of the system modelled: and each transition represents a possible evolution of the system from one configuration to another over time. Transitions are labelled with quantitative information regarding the probability and/or timing of the transition's occurrence.

A PRISM model comprises a set of *modules* which represent different components of the system being modelled. The state of each module is defined by a set of finite-ranging variables. The behaviour of a module, i.e., the changes to its state that can occur, is specified by a set of *guarded commands*. These commands take the form:

$$[syn]\ guard -> probability\ :\ update;$$

where syn is an (optional) synchronisation label, $guard$ is a predicate over the variables of the model, $probability$ is a (non-negative) real-valued expression and $update$ is update state of local variables.

The value of using PRISM is that (i) the PRISM model checker can be used to help to verify the system model; (ii) the tool can calculate the probability that a certain event occurs so that this functionality is used to predict the likelihood of the failure event; and (iii) the language of PRISM has a modular structure, thus making it easier to see how to apply the approach to larger, more complex problems. For our particular setting, the modular structure of the PRISM language makes it easier to integrate with compositional safety analysis techniques, which are inherently decompositional.

C. Framework for model-based safety analysis

We propose to combine the use of compositional safety analysis and model checking to support safety analysis.

Several compositional safety analysis techniques were briefly reviewed in the previous section. All these techniques are based on a de facto standard analysis process. In practice, the development of a software architecture is an iterative process from an intermediate model to a complete and rigorous solution. To provide effective feedback to the modelling process, the safety assessment should be performed paralleled with modelling process. An iteration of the modelling process consists of four basic stages. The first phase is functionality-based architectural design, which is the process of designing a domain model based on the functional requirements only. The second phase is architectural transformation. Software architecture is constructed for the purpose of functionality. It represents the processing units of the system and the interactions between them. Safety assessment is based on a model which represents the failure logic (flow) of the system. The failure logic model is closely related to that of the software architecture, but they may not be same at all times. During the second phase, the architectural model is transformed to the failure logic model so that safety analysis can start. The third phase is architecture assessment. During this stage, the software architecture is evaluated with respect to safety concerns. Finally, architecture refinement is the stage in which system developers consider the improvements of the architecture by introducing protective mechanisms in the architecture, or by converting a safety requirement to a functional requirement and then re-modelling.

A software architecture may use different notations for different views to support different purposes. An architecture be described in terms of a module view, component-and-connector (C&C) view, and allocation view. Our focus is the failures and their propagation in the system. They are run-time properties of the system. So C&C and allocation views

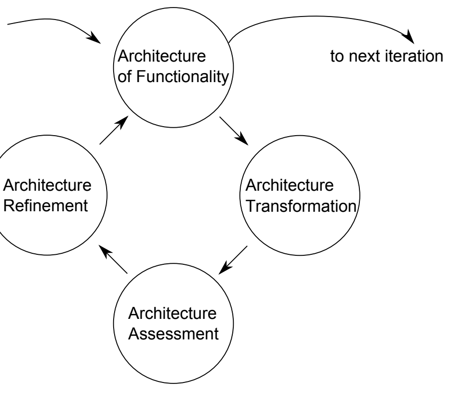

Fig. 1. Iterative Development Process of Safety-critical Systems

are essential for gaining a clear picture of the failure behaviour of a system.

III. Failure Behaviour Modelling

In our study, a system is an entity that interacts with other entities, i.e., other systems – including hardware/software systems, humans, and the physical world with its natural phenomena. From an architectural point of view, a system is composed of a set of components bound together in order to interact; each component can be another system. This recursion stops when a component is considered to be atomic: any further internal architecture cannot be discerned, or is not of interest and can be ignored. To avoid confusion, the term target system (abbreviated as **system**) in this paper refers to the top-level, whole system, instead any subsystems or modules are called **components**. In general, a component is an abstraction of a principal processing unit. It consists of interfaces which are ports to communicate with others, and a *"body"* embedded a logical function which may vary for different purposed of modelling. The two catagories of ports, input ports and output ports, are always distinguished for purposes of the design and analysis. To deliver function, components are connected via a **connector** which is a model of interaction or communication mechanism between components.

A service delivered by a system is one of its functions. Correct service is delivered when the service implements its system specification. A service failure is an event that occurs when the delivered service deviated from correct service.

The failure of a component is an event during run-time. *Mode* is a general term to describe the manner by which a behaviour of a component is observed. A component is designed and expected to behave normally. The default mode of a component is its non-failure model, which is usually tagged as *normal*.

Components are connected (interact) with others in the system. Inputs of a component can be the stimuli of the abnormal behaviours of the component. When a component receives (normal or abnormal) inputs, it can introduce failures (e.g., because of an exception or crash), or may propagate failures (e.g., data that is erroneous when it arrives at the component remains erroneous when it leaves), or transforms a failure into a different kind of failure (e.g., data that arrives late may propagate as a value failure). Furthermore, a component may correct or mask failures that it receives. At each time, only one mode can be propagated or transformed by the component. But statistically, a component can have many failure modes, which describes the way the failure occurs. Thus when a component receives inputs of a particular failure mode, it will generate one of the following responses.

$$output = \begin{cases} normal \\ same\ failure \\ different\ failure \end{cases}$$

A failure means that at least one (or more) external state of the component deviates from the correct (specified) state. The deviation from correct state may assume different forms which are called **failure modes**. Failure modes of a component can be identified and classified using a set of keywords such as those from SHARD [18].

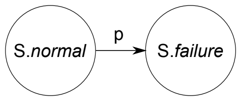

Fig. 2. Transition of Failure

A failure can be seen as a transition from correct service to incorrect service. In Figure 2, p represents the probability that a component transforms from $normal$ state to a $failure$ state. In addition, a component can fail to many failure state. Therefore the transition from $normal$ state can be arbitrary too.

One key concern in failure modelling is its expressive power with regards to describing failure behaviour of a component. The behaviour of a component is always recorded by its external states, i.e, the states of its outputs. However, because failure stimuli may arise in the external environment, or inside a software or hardware component in the system, the information of the state of component's inputs are also needed to describe the failure behaviours. In addition, the failure behaviour can be modelled by making the occurrence of failure event a deterministic choice or non-deterministic choice. The decision regarding the choice between determinism and non-determinism depends on the degree of the knowledge about the failure behaviour of a component. The non-determinism is not a feature of software system but can be one of specification at a higher level of abstraction due to incomplete knowledge about system behaviour. Above all, the failure behaviour of a component is expressed in the following format:

$$input\ state -> \ output\ state,\ probability$$

In the expression, the *probability* is conditional, i.e., the probability of an event that the output of a component in a certain state given the condition of a particular input.

IV. Failure Behaviour Model in PRISM With Example

The previous section discussed how a failure behaviour is modelled. We will demonstrate how to model a component in PRISM in this section. Our example is a processor. Typical failure behaviours of a processor can be: crash failure (i.e., permanent omission failure), transient timing failure, transient value failure. The model of the processor is illustrated in Figure 3.

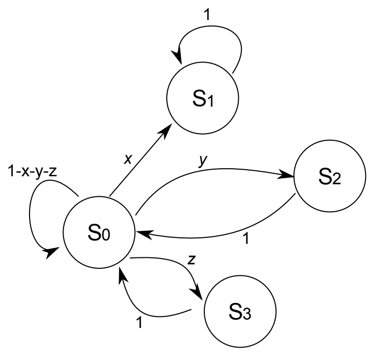

Fig. 3. Model of a Processor

In the model, s_0 are the initial state of the processor (we assume it is a normal state), and s_1, s_2, s_3 donates the state of the processor in failure mode of *crash failure*, *timing failure* and *value failure*. We did not claim the probability values for the transitions from the initial state to various $failure$ state in the model (i.e., x, y, and z in the Figure 3) and we will demonstrate the reason later in this paper.

The behaviours of the processor can be expressed as followings:

$$normal -> \ crash,\ x$$

$$normal -> \ timing,\ y$$

$$timing -> \ normal,\ 1$$

$$normal -> \ value,\ z$$

$$value -> \ normal,\ 1$$

The processor can be defined as a *module* in the PRISM model. The following is the PRISM model.

```
probabilistic

module processor
  s : [0..2] init 0;
  in : [0..2] init 0;
  out : [0..4] init 0;

  [] in=0 -> (in'=1);

  [] s=0 & in=1 ->
      (1-x-y-z):(out'=1) & (s'=1)
      + x:(out'=2) & (s'=1)
      + y:(out'=3) & (s'=1)
          + z:(out'=4) & (s'=1);
  [] s=1 & out=3 -> (in'=0) & (s'=0);
  [] s=1 & out=4 -> (in'=0) & (s'=0);
endmodule
```

In this PRISM model, we used three state variables: s, in, and out — s represents the state of a component, which can be either 0 which indicates initial state of a component, or 1 that the component is processing data, or 2 that the output is ready; in represents the modes of input where 0 is normal and 1 is abnormal; and out is the state of output where there are four modes, $normal$, $omission$, $timing\ failure$, and $value\ failure$. Both input and output have an initial mode which is normal.

Actually, the state machine model of the processor (Figure 3) can be transformed into different PRISM models with different structure. We choose such a structure with separate concept of in/output because it is a compositional structure. The advantage of the PRISM model with this in/output structure will become obvious when the system model is constructed.

V. Compositional Failure Modelling

In a compositional failure analysis approach, the system is constructed by connecting models of individual components. The tokens of failure mode flow from the output of a component to the input of another component. PRISM itself has a modular structure. Each module in a PRISM can be the model of a individual component. The problem of compositional system analysis becomes to the problem of how each module can be easily connected. In our PRISM model, the state of input and output has already been separated so that it is easy to build a system failure behaviour model by assigning the mode of the output of a component to the input of another component. In addition, the state of a component and the synchronisation mechanism of PRISM are also important features when building the system model.

Given the example of Triple modular redundancy (TMR), we will show how system is analysed in a compositional fashion in PRISM. Figure 4 shows the architecture of TMR[2].

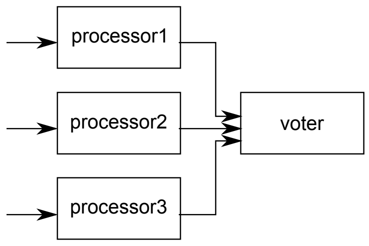

Fig. 4. Architecture of TMR

The system is a discrete-time Markov model. How to model the state machine of the system is not our focus. We listed our PRISM model in following:

[2]Due to the simplicity of the example, the failure model is as same as its architecture.

```
probabilistic

const double x;
const double y;
const double z;

module processor1

  s1 : [0..2] init 0;
  in1 : [0..2] init 0;
  out1 : [0..4] init 0;

  [a] in1=0 -> (in1'=1) & (s1'=1);

  [] s1=1 & in1=1 ->
      (1-x-y-z):(out1'=1) & (s1'=2)
      + x:(out1'=2) & (s1'=2)
      + y:(out1'=3) & (s1'=2)
      + z:(out1'=4) & (s1'=2);

endmodule

module processor2

  s2 : [0..2] init 0;
  in2 : [0..2] init 0;
  out2 : [0..4] init 0;

  [a] in2=0 -> (in2'=1) & (s2'=1);

  [] s2=1 & in2=1 ->
      (1-x-y-z):(out2'=1) & (s2'=2)
      + x:(out2'=2) & (s2'=2)
      + y:(out2'=3) & (s2'=2)
      + z:(out2'=4) & (s2'=2);

endmodule

module processor3

  s3 : [0..2] init 0;
  in3 : [0..2] init 0;
  out3 : [0..4] init 0;

  [a] in3=0 -> (in3'=1) & (s3'=1);

  [] s3=1 & in3=1 ->
      (1-x-y-z):(out3'=1) & (s3'=2)
      + x:(out3'=2) & (s3'=2)
      + y:(out3'=3) & (s3'=2)
      + z:(out3'=4) & (s3'=2);

endmodule

module voter

  v : [0..2] init 0;
  inv1 : [0..4] init 0;
  inv2 : [0..4] init 0;
  inv3 : [0..4] init 0;
  outv : [0..2] init 0;

  [] v=0 & s1=2 & s2=2 & s3=2 ->
      (inv1'=out1) & (inv2'=out2)
      & (inv3'=out3) & (v'=1);

  [] v=1 & ((inv1=1 & inv2=1) |
        (inv1=1 & inv3=1) |
        (inv2=1 & inv3=1))
        -> (outv'=1) & (v'=2);
    [] v=1 &
        ((inv1!=1 & inv2!=1 & inv3!=1)
        -> (outv'=2) & (v'=2);

endmodule
```

In this modular model, we used a module to model each component and connected each module by our proposed in/output mechanism. The PRISM model of TMR example illustrates that constructing a system model is easier if every component is modelled in a modular structure (discussed in previous section). Considering the execution sequence of the components, the only thing to do when connecting components is to ensure that the component get the inputs when the outputs of previous component are ready.

VI. System Assessment

The safety assessment is based on the failure model transformed from the architecture. In the failure model, components are treated that they may have various failure behaviours and links only propagate the failure from an output of a component to the input of another component.

As a probabilistic model checker, PRISM firstly can help us verify our probabilistic system model once it is built, which is so important if the system model is too complex to be checked manually.

In addition, sensitivity analysis is some times desired in system analysis. PRISM provides an ability to run a serial of experiments with variables which makes the sensitivity analysis easier.

In the example, we can examine the effect of the crash rate of processors by adding an experiment in PRISM. Figure 5 is the analysis result which can be used as a component selection criterion. The curve in Figure 5 shows the relationship of crash failure rate of processors and the probability of TMR working normally, given the rate of timing and value failures of processors are both 0.1.

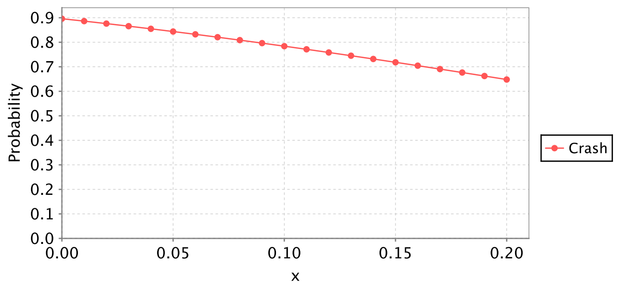

Fig. 5. Result of Probabilistic Analysis

The example demonstrates the use of PRISM in our failure behaviour analysis. In the application of our failure analysis technique with PRISM, some lessons are learned. In next section, we briefly evaluate our work.

VII. Evaluation

The key to the effectiveness of safety analysis should lie in at least the following five aspects:

1) an underlying formal model where imprecision and ambiguity can be avoided;
2) compositional capabilities which determines safety properties of a system by composition of its components;
3) a systems modelling approach where hardware, software and environment components can be modelled in a unified or integrated model;
4) sufficient expressive power to capture failures in a straightforward manner;
5) automation support to enable safety analysis to repeated with minimal effort.

Consider those criteria of compositional safety analysis techniques, our technique with the help of PRISM is based on a mathematical model and can be systematically verified by the model checker. By introducing probability, the expressive power of the technique has been enhanced. Failures of hardware and software components are easily modelled in the technique. From another angle, our work demonstrates that probabilistic model checker can be used to improve the failure propagation analysis.

There is not yet sufficient project experience to judge the practicability of our approach. However, early experience suggests that it is relatively easy to produce the required failure model and assessment, and that some re-use of the model is possible. The technique, therefore, can give economic benefits, as same time as improve the effectiveness of safety assessment. However, there are many areas for future improvement, for instance, the implementation of the tool which makes the model checking techniques more integrated in safety analysis process.

VIII. Conclusions

We have presented a model-based technique for analysing the failure propagation model of a system. The proposed technique enables the assessment of failure propagation from the analysis of components to the system. It integrates the system development (specifically, the development of component-based system) and safety analysis, and in the process of assessment, works on a hierarchical architecture of the system so that it can ensure the system model is consistent.

We are now extending our technique to enable analysis of not only probabilistic failure models, but also *timed* probabilistic failure models.

References

[1] IEC 60812. Functional safety of electrocal/electronical/programmable electronic safety/related systems, analysis techniques for system reliability - procedure for failure mode and effect analysis (FMEA). Technical report, International Electrotechnical Commission (IEC), 1991.

[2] IEC 61025. Fault-tree analysis (FTA). Technical report, International Electrotechnical Commission (IEC), 1990.

[3] Pierre Bieber, Christian Bougnol, Charles Castel, Jean pierre Heckmann, Christophe Kehren, Sylvain Metge, Christel Seguin, P. Bieber, C. Bougnol, C. Castel, J. p. Heckmann, C. Kehren, and C. Seguin. Safety assessment with altarica - lessons learnt based on two aircraft system studies. In *In 18th IFIP World Computer Congress, Topical Day on New Methods for Avionics Certification*, page 26, 2004.

[4] M. Bozzano, A. Cavallo, M. Cifaldi, L. Valacca, and A. Villafiorita. Improving safety assessment of complex systems: An industrial case study. In *Proceedings of International Formal Methods European Symposium*, pages 208–222, 2003.

[5] M. Bozzano and A. Villafiorita. Improving system reliability via model checking: The FSAP/NuSMV-SA safety analysis platform. In *Proceedings of International Confenerence of Computer Safety, Reliability and Security*, 2003.

[6] E.M Clarke. *Model Checking*. MIT Press, 1999.

[7] J. Deneux and O. Akerlund. A common framework for design and safety analysis using formal methods. In *Proceedings of International Conference of Probabilistic Safety Assurance and Management (PSAM) and European Safety and Reliability Conference*, 2004.

[8] Peter Fenelon and John A. McDermid. An integrated toolset for software safety analysis. *The Journal of Systems and Software*, 21(3):279–290, June 1993.

[9] David Harel and Amir Pnueli. On the development of reactive systems. *Logics and Models of Concurrent Systems*, F13:477–498, 1985.

[10] IEC. Hazard and operability studies (HAZOP studies) - application guide. Technical report, International Electrotechnical Commission (IEC), 2000.

[11] J. Jacky. Formal safety analysis of the control program for a radiation theraphy machine. In *proceedings of 13th International Conference of Use of Computers in Radiation Theraphy*, 2000.

[12] Bernhard Kaiser, Peter Liggesmeyer, and Oliver Mäckel. A new component concept for fault trees. In *In the proceedings of the 8th Australian Workshop on Safety Critical Systems and Software (SCS'03)*, 2003.

[13] Marta Z. Kwiatkowska, Gethin Norman, and David Parker. PRISM: Probabilistic symbolic model checker. In *in the proceedings of Computer Performance Evaluation, 12th International Conference of Modelling Techniques and Tools, TOOLS 2002*, volume 2324 of *LNCS*, pages 200–204, London, UK, April 2002. Springer.

[14] Marta Z. Kwiatkowska, Gethin Norman, and David Parker. Stochastic model checking. In *In proceedings of 7th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, SFM 2007*, volume 4486 of *LNCS*, Bertinoro, Italy, May 2007. Springer.

[15] Nancy G. Levenson. *Safeware: System Safety and Computers*. Addison-Wesley, 1995.

[16] Nancy G. Levenson and Janice L. stolzy. Safety analysis using petri nets. *IEEE Transactions of Software Engineering*, SE-13:386–397, 1987.

[17] Zohar Manna and Amir Pnueli. *The Temporal Logic of Reactive and Concurrent Systems — Specification*. Springer-Verlag, New York, 1992.

[18] J. McDermid, M. Nicholson, D. Pumfrey, and P. Fenelon. Experience with the application of HAZOP to computer-based systems. In *Compass '95: 10th Annual Conference on Computer Assurance*, pages 37–48, Gaithersburg, Maryland, 1995. National Institute of Standards and Technology.

[19] Richard F. Paige, Louis M. Rose, Xiaocheng Ge, Dimitrios S. Kolovos, and Phillip J. Brooke. Automated safety analysis for domain-specific languages. In *In proceedings of Workshop on Non-Functional System Properties in Domain Specific Modeling Languages, co-located with 11th International Conference of Model Driven Engineering Languages and Systems, MoDELS 2008*, volume 5421 of *LNCS*, Toulouse, France, October 2008. Springer.

[20] Yiannis Papadopoulos, John A. McDermid, Ralph Sasse, and Guenter Heiner. Analysis and synthesis of the behaviour of complex programmable electronic systems in conditions of failure. *Reliability Engineering and System Safety*, 71:229–247, 2001.

[21] Malcolm Wallace. Modular architectural representation and analysis of fault propagation and transformation. *Electronic Notes in Theoretical Computer Science*, 141(3):53–71, 2005.