Improving System Reliability via Model Checking: the FSAP/NuSMV-SA Safety Analysis Platform *

Marco Bozzano and Adolfo Villafiorita

ITC-IRST, Via Sommarive 18,
38050 Trento, Italy
ph.: +39 0461 314481, fax: +39 0461 314 591
{bozzano,adolfo}@irst.itc.it
http://sra.itc.it/people/{bozzano,adolfo}

Abstract. Safety critical systems are becoming more complex, both in the type of functionality they provide and in the way they are demanded to interact with their environment. Such growing complexity requires an adequate increase in the capability of safety engineers to assess system safety, including analyzing the bahaviour of a system in degraded situations. Formal verification techniques, like symbolic model checking, have the potential of dealing with such a complexity and are more often being used during system design. In this paper we present the FSAP/NuSMV-SA platform, based on the NuSMV2 model checker, that implements known and novel techniques to help safety engineers perform safety analysis. The main functionalities of FSAP/NuSMV-SA include: failure mode definition based on a library of failure modes, fault injection, automatic fault tree construction for monotonic and non-monotonic systems, failure ordering analysis. The goal is to provide an environment that can be used both by design engineers to formally verify a system and by safety engineers to automate certain phases of safety assessment. The platform is being developed within the ESACS project (Enhanced Safety Analysis for Complex Systems), an European-Union-sponsored project in the avionics sector, whose goal is to define a methodology to improve the safety analysis practice for complex systems development.

1 Introduction

Controllers for safety critical systems are typically required to operate effectively not only in nominal conditions – i.e., when all the (sub)components of the system work as expected – but also in degraded situations – that is, when some of the (sub)components of the system are not working properly. This requirement is common in various safety critical sectors like, e.g., aeronautics, in which degraded operational conditions are stated as a set of *safety requirements*, available in

* This work has been and is being developed within ESACS, an European- sponsored project, contract no. G4RD-CT-2000-00361.

the System Requirements Specification. Therefore, the standard development process is paired by a new set of activities (safety analysis), whose goal is to identify all possible hazards, together with their relevant causes, and to certify that the system behaves as expected in all the operational conditions.

Safety critical systems are becoming more complex, both in the type of functionality they provide and in the way they are demanded to interact with their environment. Such growing complexity requires an adequate increase in the capability of safety engineers to assess system safety. Current informal methodologies, like manual fault tree analysis (FTA) and failure mode and effect analysis (FMEA) [34], that rely on the ability of the safety engineer to understand and to foresee the system behaviour, are not ideal when dealing with highly complex systems. Emerging techniques like formal methods [35] are increasingly being used for the development of critical systems (see, e.g., [12, 9, 8, 21]). Formal methods allow a more thorough verification of the system's correctness with respect to the requirements, by using *automated* and hopefully *exhaustive* verification procedures. In particular, model checking [13] is increasingly being used for several real-world safety-critical industrial applications. However, the use of formal methods for safety analysis purposes is still at an early stage. Moreover, even when formal methods are applied, the information linking the design and the safety assessment phases is often carried out informally. The link between design and safety analysis may be seen as an "over the wall process" [18].

In this paper we present the FSAP/NuSMV-SA platform, which is being developed at ITC-IRST. FSAP/NuSMV-SA is based on two main components: FSAP (Formal Safety Analysis Platform), that provides a graphical front-end to the user, and NuSMV-SA, based on the NuSMV2 [10] model checker, that provides an engine to perform safety assessment. The main functionality of FSAP/NuSMV-SA include: support for model construction (e.g., failure mode definition based on a library of predefined failure modes), automatic fault injection, support for safety requirements definition (in the form of temporal logic formulas), automatic fault tree construction for both monotonic and non monotonic systems, user-guided or random simulation, counterexample trace generation, and failure ordering analysis. FSAP/NuSMV-SA provides an environment that can be used both by design engineers to formally verify a system and by safety engineers to automate certain phases of safety assessment.

The major benefits provided by the FSAP/NuSMV-SA platform are a tight integration between the design and the safety analysis teams, and a (partial) automation of the activities related to both verification and safety assessment. The basic functions provided by the platform can be combined in different ways, in order to comply with any given development methodology one has in mind. It is possible to support an incremental approach, based on iterative releases of a given system model at different levels of detail (e.g., model refinement, addition of further failure modes and/or safety requirements). Furthermore, it is possible to have iterations in the execution of the different phases (design and safety assessment), e.g., it is possible to let the model refinement process be driven by the safety assessment phase outcome (e.g., disclosure of system flaws requires fixing

the physical system and/or correcting the formal model). Therefore, in order to support the flow of information which is likely to be required between design and safety engineers, the FSAP/NuSMV-SA platform implements the concept of *repository* for safety analysis task results. The repository contains information about which safety analysis tasks (e.g., verification of temporal properties, fault tree generations) have been performed for which model, keeping trace of which properties do hold for a particular model and which do not, and marking tasks as being up-to-date or not. The repository thus provides traceability capabilities and makes reuse and evolution of safety cases easier.

The FSAP/NuSMV-SA platform has been and is being developed within the ESACS project [6] (Enhanced Safety Assessment for Complex Systems, see `http://www.esacs.org`), an European-Union-sponsored project in the area of safety analysis, involving several research institutions and leading companies in the fields of avionics and aerospace. The methodology developed within the ESACS project is supported by state-of-the-art and commercial tools for system modeling and traditional safety analysis. The tools, collectively referred with the name of ESACS platform, have been extended to support the methodology and to automate certain phases of safety analysis. Both the methodology and the ESACS platform are being trialed on a set of industrial case studies. The ESACS platform comes in different configurations, tailored to the needs of the industrial partners participating in the project.

The rest of the paper is structured as follows. In Section 2 we give an overview of the safety analysis process, we discuss its connection with model checking and the safety analysis capabilities integrated into FSAP/NuSMV-SA. In Section 3 we give an overview of the FSAP/NuSMV-SA platform. In Section 4 we discuss some related work, and, finally, in Section 5 we draw some conclusions.

2 Safety Analysis via Model Checking

Model checking [13] is a well-established method for formally verifying temporal properties of finite-state concurrent systems. It has been applied for the formal verification of a number of real-world safety-critical industrial systems [22, 23, 10]. In particular, the engine of FSAP/NuSMV-SA is an extension of the model checking tool NuSMV2 [10], a BDD-based symbolic model-checker developed at ITC-IRST, originated from a re-engineering and re-implementation of SMV [28]. NuSMV2 is a well-structured, open, flexible and well-documented platform for model checking, and it has been designed to be robust and close to industrial standards [11]. Typically, system specifications are written as temporal logic formulas, and efficient symbolic algorithms (based on data structures like BDDs [7]) are used to traverse the model and check if the specification holds or not.

Being an extension of NuSMV2, FSAP/NuSMV-SA provides all the functionality of NuSMV2. Below, however, we will focus on the safety assessment capabilities of FSAP/NuSMV-SA. We do so by providing a typical scenario of usage of the platform on a toy example, namely a two-bit adder. The presented scenario derives from the ESACS methodology (see [6], for more details). The

```
MODULE bit(input)
VAR
  out : {0,1};
ASSIGN
  out := input;

MODULE adder(bit1,bit2)
VAR
  out : {0,1};
ASSIGN
  out := (bit1 + bit2) mod 2;
```

```
MODULE main
VAR
  random1 : {0,1};
  random2 : {0,1};
  bit1    : bit(random1);
  bit2    : bit(random2);
  adder   : adder(bit1.out,bit2.out);
```

Fig. 1. A NuSMV model for a two-bit adder

example is deliberately simple for illustration purposes and should not be regarded as modeling a realistic system. Following the ESACS methodology, the use of the FSAP/NuSMV-SA platform is based on the following phases:

System Model Definition In this phase a formal model of the system under development, called *system model*, is provided.

Requirements Definition It is the phase in which the desired properties of the system model are specified. The properties may refer to the behaviour of the system both in nominal and in degraded situations.

Failure Mode Definition In this phase, the failure modes of the components of the design model are identified.

Fault Injection and Model Extension In this phase the failure modes defined in the previous phase are injected into the system model. As a result, a new model, called *extended system model*, is generated. The extended system model enriches the behaviour of the system model by taking into account all the set of degraded behaviours identified during the previous phase.

Formal Verification and Safety Assessment It is the phase in which the system model and/or the extended system model are checked against a set of requirements. Verification strategies may include, e.g., guided or random simulation, formal verification of properties, generation of fault trees.

Note that the responsibility of the different phases described above may belong to different disciplines, e.g., design engineers or safety engineers. As discussed in the introduction, there are no strict constraints on the way in which the different functions can be invoked, and the overall process in which FSAP/NuSMV-SA is used can be shaped up so as to comply with different development methodologies.

2.1 System Model and Failure Mode Definition

System model definition provides an executable specification (at a given level of abstraction) of the model of the system under development. As an example, consider the simple example, written in the syntax of NuSMV2 [10], in Figure 1.

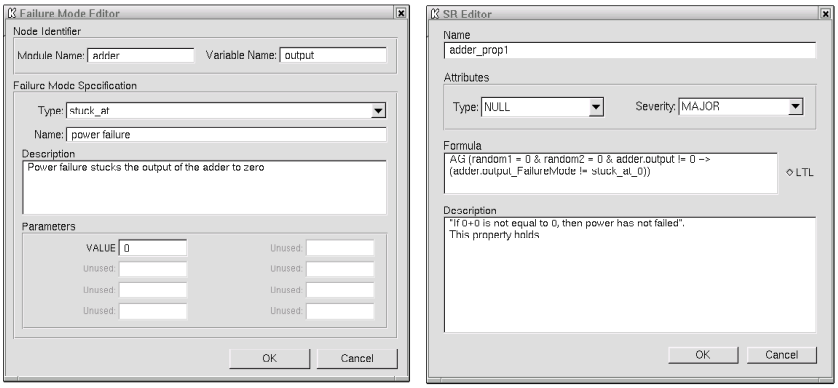

Fig. 2. Inputing of failure modes and safety requirements in FSAP

It is composed of three modules: the `bit` module, which simply copies the input bit to the output, the `adder` module, which computes the sum (module two) of two given input bits, and the `main` module, which defines the overall system as composed of an adder which takes as input two bits that may vary in a random way. In order to study the behaviour of the adder circuit in presence of degraded situations, failure mode definitions can be added to the previous specification.

In FSAP/NuSMV-SA, failure modes are defined using a graphical user interface, in which the safety engineer specifies which nodes of the system model can fail, in what ways, and according to what parameters. Figure 2 (left-hand side) shows an example of the interface currently provided by FSAP/NuSMV-SA for defining failure modes. Failure modes are retrieved from a library, called Generic Failure Mode Library (GFML, for short). The library contains the specifications of the behaviours induced by the failures and the specification of the parameters (whose values must be set by the user) that characterize the failures. The standard GFML, that is the library distributed with the platform, provides specification of failures like, e.g., *stuck-at*, *random output*, *glitch*, *inverted*. The library can be extended to include user-defined failure modes. In the adder case, examples of failure modes may include, e.g., the adder output being stuck at a given value (zero or one), and an input bit corruption (*inverted* failure mode).

2.2 Fault Injection and Model Extension

The failure modes defined at the previous step can be *automatically* injected by FSAP/NuSMV-SA into a system model. The result is the so-called *extended system model*, that is a model in which some of the nodes can fail according to the specification of the failure modes. As an example, consider the *inverted* failure mode for the output of the `bit` module in Figure 1. Injection of this failure mode causes the system model to be extended with a new piece of NuSMV code (instantiated from the GFML), that is automatically inserted into the extended

```
VAR    out_nominal     : {0,1};
       out_FailureMode : {no_failure, inverted};
ASSIGN out_nominal := input;
DEFINE out_inverted := ! out_nominal;
DEFINE out := case
         out_FailureMode = no_failure : out_nominal;
         out_FailureMode = inverted   : out_inverted;
       esac;
ASSIGN next(out_FailureMode) := case
         out_FailureMode = no_failure : {no_failure, inverted};
         out_FailureMode = inverted   : inverted;
       esac;
```

Fig. 3. Injecting a fault in the `bit` module

system model. The new piece of code (see Figure 3) replaces the old definition of the `out` variable by taking into account a possible corruption of the input bit.

2.3 Requirements Definition

System model definition, failure mode definition and model extension are just a part of the verification and safety assessment process. Formal verification is carried out by defining properties in the form of temporal specifications. For instance, the following properties may be specified for the adder example:

```
AG (random1 = 0 & random2 = 0 → adder.out = 0)
AG (random1 = 0 & random2 = 0 & adder.out != 0) →
     (bit1.out_FailureMode = inverted | bit2.out_FailureMode = inverted)
```

The first one states that the output of the adder must be zero whenever both input bits are zero (this is clearly not the case in degraded situations), whereas the second one states that whenever the sum of the zero input bits yields one it is the case that at least one of the two input bits is corrupted. Requirements defined in this way can subsequently be exhaustively verified via the underlying model checking verification engine provided by NuSMV. Properties in FSAP/NuSMV-SA are defined via a graphical user interface, in which users can enter information such as type and severity of the safety requirement. Figure 2 (right-hand side), for instance, shows how safety requirements are specified by the user. The graphical user interface does not provide, at the moment, facilities for simplifying inputing of formulas, such as patterns or visual representations.

2.4 Formal Verification and Safety Assessment

During this phase the model under development is tested against safety requirements. Using the facilities provided by the NuSMV2 engine, it is possible to perform guided or random simulation, and several kinds of formal verification analyses. In particular, below we will focus on fault tree construction and failure ordering analysis, which are more specific to the safety analysis process.

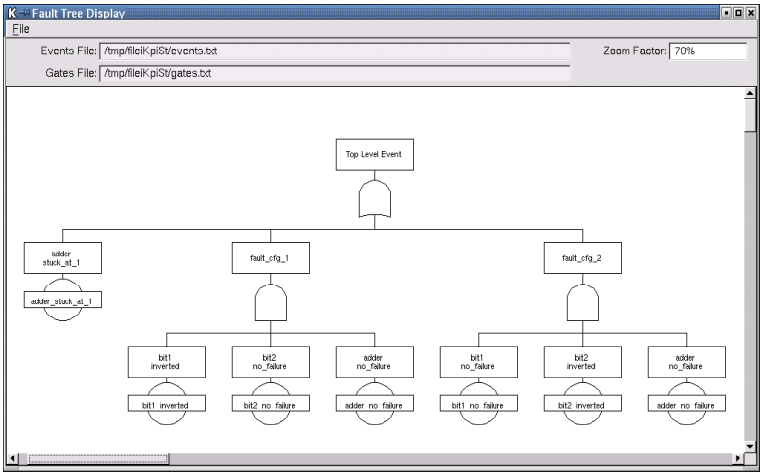

Fig. 4. A fault tree generated for the adder model

Fault Tree Construction Fault Tree Analysis (FTA) [34, 24, 30] is a safety assessment strategy which is complementary with respect to exhaustive property verification. It is a deductive, top-down method to analyze system design and robustness. It usually involves specifying a *top level event* (TLE hereafter) to be analyzed (e.g., a *failure state*), and identifying all possible sets of basic events (e.g., basic *faults*) which may cause that TLE to occur. FTA allows one to identify possible system reliability or safety problems and find out root causes of equipment failures. *Fault trees* provide a convenient symbolic representation of the combination of events resulting in the occurrence of the top event. They are usually represented in a graphical way, as a parallel or sequential combination of AND/OR gates. The FSAP/NuSMV-SA platform can be used to automatically generate fault trees starting from a given model and TLE. Model checking techniques are used to extract *automatically* all collections of basic events (called *minimal cut sets*) which can trigger the TLE. The generated cut sets are minimal in the sense that only failure events that are strictly necessary for the top level event to occur are retained.

Each cut set produced by FSAP/NuSMV-SA represents a situation in which the top level event has been violated owing to the occurrence of some failures. Under the hypothesis that the system model does not violate the top level event, such failures are the cause of the violation of the top level event. Notice that the violation may be due not to a static analysis of the system but, rather, to complex interactions caused by the various failing and non-failing components of the system. Since the fault tree representation provides a static representation, NuSMV-SA associates, to each cut set, a counter-example that shows a trace, step by step, of how the top level event is violated by the failures represented in the cut set. Figure 4 shows an example of fault tree computed for the adder. It

has been generated for the top level event

```
random1 = 0 & random2 = 0 & adder.out != 0
```

and it comprises three cut sets (the first one of them is a single failure, whereas the remaining two include three basic events). The fault tree states that the top level event may occur if and only if either the output of the adder is stuck at one, or one of the input bits (and *only* one) is corrupted (with the adder working properly). We note that minimality of the generated cut sets implies that, e.g., the case in which both input bits and the adder are failed is not considered (though causing the top level event as well).

Finally, we note that the fault tree in Figure 4 shows an example of *non-monotonic* fault tree analysis, i.e., basic events requiring system components *not* to fail can be part of the results of the analysis. The traditional *monotonic* analysis (i.e., where only failure events are considered) is also supported by FSAP/NuSMV-SA. The choice between the different kinds of analyses is left to the user, which may label a system model as being monotonic or non-monotonic.

Failure Ordering Analysis A further functionality of the FSAP/NuSMV-SA platform is the so-called *event ordering analysis*. For further information on the material of this section, we refer the reader to [5], which describes the algorithm for ordering analysis, its implementation and applications in detail.

In traditional FTA, cut sets are simply flat collections (i.e, conjunctions) of events which can trigger a given TLE. However, there might be timing constraints enforcing a particular event to happen before or after another one, in order for the TLE to be triggered (i.e., the TLE would not show if the order of the two events were swapped). Ordering constraints can be due, e.g., to a causality relation or a functional dependency between events, or caused by more complex interactions involving the dynamics of a system. Whatever the reason, event ordering analysis can provide useful information which can be used by the design and safety engineers to fully understand the ultimate causes of a given system malfunction, so that adequate countermeasures can be taken.

The ordering analysis phase can be tightly integrated with fault tree analysis, as described below. Given a system model, the verification process consists of the following phases. First of all, a top level event to analyze is chosen (clearly, the analysis can be repeated for different top level events). Then, fault tree analysis is run in order to compute the *minimal cut sets* relative to the top level event. For each cut set, the event ordering analysis module of the platform generates a so-called *ordering information model* and performs ordering analysis on it. The outcome of the ordering analysis is a *precedence graph* showing the order among events (if any) which must be fulfilled in order for the top level event to occur.

3 FSAP/NuSMV-SA Platform Overview

This section briefly describes the architecture of FSAP/NuSMV-SA. The platform is based on two main components: FSAP (Formal Safety Analysis Platform)

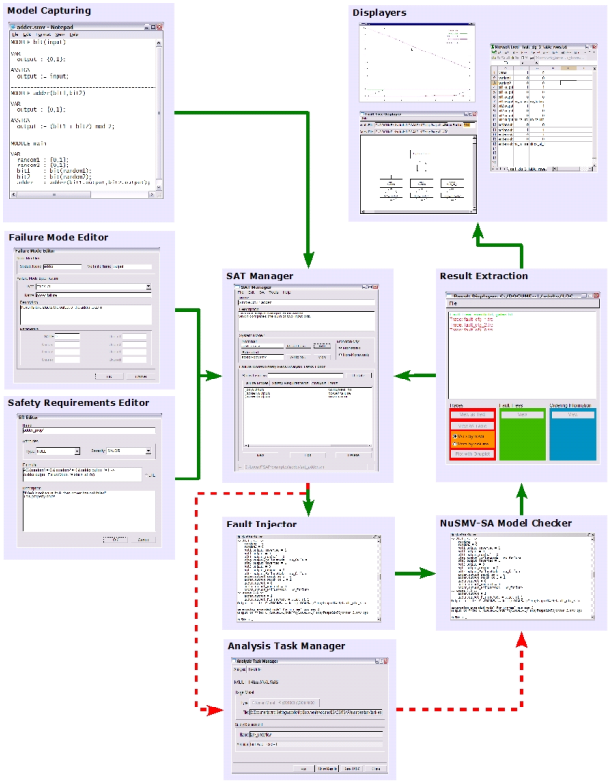

Fig. 5. The FSAP/NuSMV-SA components

provides a graphical user interface and a manager for a repository that can be used by safety engineers and design engineers to share information related to the system under development and the analysis performed; NuSMV-SA, based on the NuSMV2 model checker, provides the core algorithms for formal analysis.

FSAP is implemented in C++ as a cross-platform environment. The graphical user interface is based on the FLTK (see `http://www.fltk.org`) cross platform toolkit. The data produced by the platform are stored in XML format, and the parser is based on the expat library (see `http://www.expat.org`). As a result, FSAP/NuSMV-SA currently runs on Windows and Linux platforms (as for NuSMV, running NuSMV-SA on Windows currently requires the Cygwin environment to be installed). Figure 5 shows the components of FSAP/NuSMV-SA and the data flow (solid lines). We distinguish the following blocks:

SAT Manager The SAT manager is the central module of the platform. It is used to store all the information relevant to verification and safety assessment. It contains references to the system model, failure modes, location

of the extended system model, safety requirements, and analyses to be run. From the SAT, it is possible to call all the other components of the platform.

Model Capturing System models are written using the NuSMV input language, that is text based. FSAP/NuSMV-SA provides users with the possibility of using their preferred text editor for editing the system model.

Failure Mode Editor & Fault Injector These are the modules for, respectively defining failure modes and generating an extended system model.

Analysis Task Handler This is the module to define analysis tasks. Analysis tasks are a convenient way to store the specification of the analyses to be run, they are saved in the SAT and can be retrieved across different sessions.

NuSMV-SA This is the core, based on the NuSMV2 model checker.

Result Extraction and Displayers All the results produced by the platform can be viewed using the result extraction and displayers. In particular, it is possible to view counterexamples in textual, structured (XML), graphical, or tabular fashion. Fault trees generated by the platform can be viewed using commercial tools (e.g. FaultTree+ v9.0 and v10.0) or using a displayer we especially developed within the project and can be exported in XML format.

4 Related Work

The FSAP/NuSMV-SA platform has been and is being developed within the ESACS project (Enhanced Safety Analysis for Complex Systems), an European-Union-sponsored project involving various research centers and industries from the avionics sector. For a more detailed description of the ESACS methodology and the project goals, we refer the reader to [6], which also discussed more realistic examples to which the methodology has been applied.

The safety analysis capabilities provided by the platform include traditional fault tree generation [34, 24, 30] together with formal verification capabilities typical of model checking [28, 13, 22, 23, 10]. The algorithms for cut set and prime implicant computation described in Section 2.4 are based on classical procedures for *minimization* of *boolean functions*, specifically on the implicit-search procedure described in [15, 16], which is based on Binary Decision Diagrams (BDDs) [7]. This choice was quite natural, given that the NuSMV model checker makes a pervasive use of BDD data structures. The ordering analysis procedure described in Section 2.4 also makes use of these algorithms (we refer the reader to [5] for a discussion of the related literature). Explicit-search and SAT-based techniques for computation of prime implicants are described, e.g., in [26].

We also mention [25, 33], which describe DIFTree (Dynamic Innovative Fault Tree), a methodology supporting (however, still at the manual level) fault tree construction and allowing for different kinds of analyses of sub-trees (e.g., Markovian or Monte Carlo simulation for dynamic ones, and BDD-based evaluation for static ones). The notation for non-logical (dynamic) gates of fault trees and the support for sample probabilistic distributions could be nice features to be integrated in our framework.

A large amount of work has been done in the area of probabilistic safety assessment (PSA) and in particular on *dynamic reliability* [31]. Dynamic reliability is concerned with extending the classical event or fault tree approaches to PSA by taking into consideration the mutual interactions between the hardware components of a plant and the physical evolution of its process variables [27]. Examples of scenarios taken into consideration are, e.g., human intervention, expert judgment, the role of control/protection systems, the so-called failures *on demand* (i.e., failure of a component to intervene), and also the ordering of events during accident propagation. Different approaches to dynamic reliability include, e.g., state transitions or Markov models [1, 29], the dynamic event tree methodology [14], and direct simulation via Monte Carlo analysis [32, 27].

Concerning ordering analysis (see Section 2.4), the work which is probably closer to ours is [14], which describes dynamic event trees as a convenient means to represent the timing and order of intervention of a plant sub-systems and their eventual failures. With respect to the classification the authors propose, our approach can support *simultaneous* failures, whereas, at the moment, we are working under the hypothesis of *persistent* failures (i.e., no repair is possible).

5 Conclusions

In this paper we have presented the FSAP/NuSMV-SA safety analysis platform. The verification engine of the platform is based on the NuSMV2 model checker [10]. FSAP/NuSMV-SA can be used as a tool to assist the safety analysis process from the early phases of system design to the formal verification and safety assessment phases. The goal is to provide an environment that can be used both by design engineers to formally verify a system and by safety engineers to automate certain phases of safety assessments. To achieve these goals, FSAP/NuSMV-SA provides a set of basic functions which can be combined in arbitrary ways to realize different process development methodologies.

The functionalities provided by FSAP/NuSMV-SA integrates traditional analysis methodologies like fault tree generation, together with exhaustive property verification capabilities typical of model checking, plus model construction facilities (e.g., automatic failure injection based on a library of predefined failure modes) and traceability capabilities, which improve exchange of information and make reuse and evolution of safety cases easier. The FSAP/NuSMV-SA platform supports automatic fault tree generation, both in the case of *monotonic* systems (computation of *minimal cut sets*) and in the case of *non-monotonic* ones (computation of *prime implicants*). Furthermore, the results provided by fault tree generation can be conveniently integrated by the so-called *ordering analysis* phase, which allows one to extract ordering constraints holding between basic events in a given cut set, thus providing a deeper insight into the ultimate causes of system malfunction. As discussed in [5], timing constraints can arise very naturally in industrial systems. For a more extensive discussion about the use of model checking for safety analysis, the tool usage experience, and for a more realistic example of application of the methodology, we refer the reader to [4].

Concerning the works on dynamic reliability cited in Section 4, the most notable difference between our approach and the works mentioned there is that we present *automatic* techniques, based on model checking, for both fault tree generation and ordering analysis, whereas traditional works on dynamic reliability rely on manual analysis (e.g., Markovian analysis [29]) or simulation (e.g., Monte Carlo simulation [27], the TRETA package of [14]). Automation is clearly a point in our favour. Furthermore, we support automatic verification of arbitrary CTL properties (in particular, both safety and liveness properties).

Current work is focusing on some improvements and extensions in order to make the methodology competitive with existing approaches and usable in realistic scenarios. First of all, there are some improvements at the modeling level. The NuSMV models used so far are discrete, finite-state transition models. In order to allow for more realistic models, we are considering an extension of NuSMV with hybrid dynamics, along the lines of [19, 20]. This would allow both to model more complex variable dynamics, and also a more realistic modeling of time (which, currently, is modeled by an abstract transition step). Furthermore, we need to extend our framework to deal with *probabilistic* assessment. Although not illustrated in this paper, associating probabilistic estimates to basic events and evaluating the resulting fault trees is straightforward. However, more work needs to be done in order to support more complex probabilistic dynamics (see, e.g., [17]). We also want to overcome the current limitation to permanent failures.

Concerning the FSAP/NuSMV-SA platform, we are currently working on further improving user interaction (e.g., by working on pattern-based inputing of top level events) and experimenting on SAT based techniques [3, 2] for fault tree construction. The FSAP/NuSMV-SA platform is available for evaluation purposes from `http://sra.itc.it/tools/FSAP` (the download is currently password protected; the password can be obtained by sending an e-mail to the authors).

Acknowledgments. The work presented in this paper would have not been possible without the help of Paolo Traverso, Alessandro Cimatti, and Gabriele Zacco.

We would also like to thank the people working in the ESACS project and, in particular: Ove Åkerlund (Prover), Pierre Bieber (ONERA), Christian Bougnol (AIRBUS), E. Böde (OFFIS), Matthias Bretschneider (AIRBUS-D), Antonella Cavallo (Alenia Aeronautica), Charles Castel (ONERA), Massimo Cifaldi (SIA), Alain Griffault (LaBri, Université de Bordeaux), C. Kehren (ONERA), Benita Lawrence (AIRBUS-UK), Andreas Lüdtke (University of Oldenburg), Silvayn Metge (AIRBUS-F), Chris Papadopoulos (AIRBUS-UK), Renata Passarello (SIA), Thomas Peikenkamp (OFFIS), Per Persson (Saab), Christel Seguin (ONERA), Luigi Trotta (Alenia Aeronautica), and Laura Valacca (SIA).

References

1. T. Aldemir. Computer-assisted Markov Failure Modeling of Process Control Systems. *IEEE Transactios on Reliability*, R-36:133–144, 1987.
2. G. Audemard, P. Bertoli, A. Cimatti, A. Korniłowicz, and R. Sebastiani. A SAT Based Approach for Solving Formulas over Boolean and Linear Mathematical

Propositions. In Andrei Voronkov, editor, *CADE-18: Conference on Automated Deduction*, number 2392 in LNAI, pages 195–210. Springer, 2002.
3. A. Biere, A. Cimatti, E.M. Clarke, and Y. Zhu. Symbolic Model Checking without BDDs. In R. Cleaveland, editor, *Proc. 5th International Conference on Tools and Algorithms for Construction and Analysis of Systems (TACAS'99)*, volume 1579 of *LNCS*, pages 193–207. Springer-Verlag, 1999.
4. M. Bozzano, A. Cavallo, M. Cifaldi, L. Valacca, and A. Villafiorita. Improving Safety Assessment of Complex Systems: An industrial case study. In *Proc. Formal Methods Europe (FME'03)*, 2003.
5. M. Bozzano and A. Villafiorita. Integrating Fault Tree Analysis with Event Ordering Information. In *Proc. European Safety and Reliability Conference (ESREL'03)*, 2003.
6. M. Bozzano et al. ESACS: An Integrated Methodology for Design and Safety Analysis of Complex Systems. In *Proc. European Safety and Reliability Conference (ESREL'03)*, 2003.
7. R.E. Bryant. Symbolic Boolean Manipulation with Ordered Binary Decision Diagrams. *ACM Computing Surveys*, 24(3):293–318, 1992.
8. A. Chiappini, A. Cimatti, C. Porzia, G. Rotondo, R. Sebastiani, P. Traverso, and A. Villafiorita. Formal Specification and Development of a Safety-Critical Train Management System. In M. Felici, K. Kanoun, and A. Pasquini, editors, *18th Conference on Computer Safety, Reliability and Security (SAFECOMP'99)*, volume 1698 of *LNCS*, pages 410–419. Springer-Verlag, 1999.
9. A. Cimatti. Industrial Applications of Model Checking. In F. Cassez, C. Jard, B. Rozoy, and M.D. Ryan, editors, *Modeling and Verification of Parallel Processes (MOVEP'00)*, volume 2067, pages 153–168. Springer-Verlag, 2001.
10. A. Cimatti, E.M. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella. NuSMV2: An OpenSource Tool for Symbolic Model Checking. In E. Brinksma and K.G. Larsen, editors, *Proc. 14th International Conference on Computer Aided Verification (CAV'02)*, LNCS, pages 359–364. Springer-Verlag, 2002.
11. A. Cimatti, E.M. Clarke, F. Giunchiglia, and M. Roveri. NuSMV: a new symbolic model checker. *International Journal on Software Tools for Technology Transfer*, 2(4):410–425, 2000.
12. A. Cimatti, P.L. Pieraccini, R. Sebastiani, P. Traverso, and A. Villafiorita. Formal Specification and Validation of a Vital Communication Protocol. In J.M. Wing, J. Woodcock, and J. Davies, editors, *World Congress on Formal Methods, (FM'99), Volume II*, volume 1709 of *LNCS*, pages 1584–1604. Springer, 1999.
13. E.M. Clarke, O. Grumberg, and D.A. Peled. *Model Checking*. MIT Press, 2000.
14. G. Cojazzi, J. M. Izquierdo, E. Meléndez, and M. S. Perea. The Reliability and Safety Assessment of Protection Systems by the Use of Dynamic Event Trees. The DYLAM-TRETA Package. In *Proc. XVIII Annual Meeting Spanish Nucl. Soc.*, 1992.
15. O. Coudert and J.C. Madre. Implicit and Incremental Computation of Primes and Essential Primes of Boolean Functions. In *Proc. 29th Design Automation Conference (DAC'98)*, pages 36–39. IEEE Computer Society Press, 1992.
16. O. Coudert and J.C. Madre. Fault Tree Analysis: 10^{20} Prime Implicants and Beyond. In *Proc. Annual Reliability and Maintainability Symposium*, 1993.
17. J. Devooght and C. Smidts. Probabilistic Dynamics; The Mathematical and Computing Problems Ahead. In T. Aldemir, N. O. Siu, A. Mosleh, P. C. Cacciabue, and B. G. Göktepe, editors, *Reliability and Safety Assessment of Dynamic Process Systems*, volume 120 of *NATO ASI Series F*, pages 85–100. Springer-Verlag, 1994.

18. P. Fenelon, J.A. McDermid, M. Nicholson, and D.J. Pumfrey. Towards Integrated Integrated Safety Analysis and Design. *Applied Computing Review*, 2(1):21 – 32, 1994.
19. T. A. Henzinger. The Theory of Hybrid Automata. In *Proc. 11th Annual International Symposium on Logic in Computer Science (LICS'96)*, pages 278–292. IEEE Computer Society Press, 1996.
20. T. A. Henzinger. HyTech: A Model Checker for Hybrid Systems. *Software Tools for Technology Transfer*, 1:110–122, 1997.
21. M.G. Hinchey and J.P. Bowen, editors. *Industrial Strength Formal Methods in Practice*. Formal Approaches to Computing and Information Technology. Springer-Verlag, 1999.
22. G.J. Holzmann. The Model Checker SPIN. *IEEE Transactions on Software Engineering*, 23(5):279–295, 1997.
23. K.G. Larsen, P. Pettersson, and W. Yi. UPPAAL in a Nutshell. *International Journal on Software Tools for Technology Transfer*, 1(1-2):134–152, 1997.
24. P. Liggesmeyer and M. Rothfelder. Improving System Reliability with Automatic Fault Tree Generation. In *Proc. 28th International Symposium on Fault-Tolerant Computing (FTCS'98)*, pages 90–99, Munich, Germany, 1998. IEEE Computer Society Press.
25. R. Manian, J.B. Dugan, D. Coppit, and K.J. Sullivan. Combining Various Solution Techniques for Dynamic Fault Tree Analysis of Computer Systems. In *Proc. 3rd International High-Assurance Systems Engineering Symposium (HASE'98)*, pages 21–28. IEEE Computer Society Press, 1998.
26. V.M. Manquinho, A.L. Oliveira, and J.P. Marques-Silva. Models and Algorithms for Computing Minimum-Size Prime Implicants. In *Proc. International Workshop on Boolean Problems (IWBP'98)*, 1998.
27. M. Marseguerra, E. Zio, J. Devooght, and P. E. Labeau. A concept paper on dynamic reliability via Monte Carlo simulation. *Mathematics and Computers in Simulation*, 47:371–382, 1998.
28. K.L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publ., 1993.
29. I. A. Papazoglou. Markovian Reliability Analysis of Dynamic Systems. In T. Aldemir, N. O. Siu, A. Mosleh, P. C. Cacciabue, and B. G. Göktepe, editors, *Reliability and Safety Assessment of Dynamic Process Systems*, volume 120 of *NATO ASI Series F*, pages 24–43. Springer-Verlag, 1994.
30. A. Rae. Automatic Fault Tree Generation - Missile Defence System Case Study. Technical Report 00-36, Software Verification Research Centre, University of Queensland, 2000.
31. N. O. Siu. Risk Assessment for Dynamic Systems: An Overview. *Reliability Engineering ans System Safety*, 43:43–74, 1994.
32. C. Smidts and J. Devooght. Probabilistic Reactor Dynamics II. A Monte-Carlo Study of a Fast Reactor Transient. *Nuclear Science and Engineering*, 111(3):241–256, 1992.
33. K.J. Sullivan, J.B. Dugan, and D. Coppit. The Galileo Fault Tree Analysis Tool. In *Proc. 29th Annual International Symposium on Fault-Tolerant Computing (FTCS'99)*, pages 232–235. IEEE Computer Society Press, 1999.
34. W.E. Vesely, F.F. Goldberg, N.H. Roberts, and D.F. Haasl. Fault Tree Handbook. Technical Report NUREG-0492, Systems and Reliability Research Office of Nuclear Regulatory Research U.S. Nuclear Regulatory Commission, 1981.
35. J.M. Wing. A Specifier's Introduction to Formal Methods. *IEEE Computer*, 23(9):8–24, 1990.