# Improving Safety Assessment of Complex Systems: An industrial case study *

Marco Bozzano
ITC-irst
bozzano@irst.itc.it

Antonella Cavallo
Alenia Aeronatica
acavallo@aeronautica.alenia.it

Massimo Cifaldi
Società Italiana Avionica
cifaldi@sia-av.it

Laura Valacca
Società Italiana Avionica
valacca@sia-av.it

Adolfo Villafiorita
ITC-irst
adolfo@irst.itc.it

**Abstract**. The complexity of embedded controllers is steadily increasing. This trend, stimulated by the continuous improvement of the computational power of hardware, demands for a corresponding increase in the capability of design and safety engineers to maintain adequate safety levels. The use of formal methods during system design has proved to be effective in several practical applications. However, the development of certain classes of applications, like, for instance, avionics systems, also requires the behaviour of a system to be analysed under certain degraded situations (e.g., when some components are not working as expected). The integration of system design activities with safety assessment and the use of formal methods, although not new, are still at an early stage. These goals are addressed by the ESACS project, a European-Union-sponsored project grouping several industrial companies from the aeronautic field. The ESACS project is developing a methodology and a platform – the ESACS platform – that helps safety engineers automating certain phases of their work. This paper reports on the application of the ESACS methodology and on the use of the ESACS platform to a case study, namely, the Secondary Power System of the Eurofighter Typhoon aircraft.

**Keywords:** *Formal Verification and Safety Assessment of Complex Systems, Automated Fault Tree Computation, ESACS*

## 1. Introduction

In the *development cycle* of a complex system, it is possible to identify a certain number of steps each involving different processes and tasks that the system

* This work has been and is being developed within ESACS, an European-sponsored project, Framework Programme 5 - Growth Contract no. G4RD-CT-2000-00361

development team has to carry out. In the classic *waterfall model*, the principal phases are: *requirements analysis* and *specification*, *design*, *implementation*, *testing*, *analyses* and *maintenance*. In the last decades, many variations of this model were proposed. Some of these are based on virtual prototyping and simulation, incremental development, reusable software and automated synthesis. As specification errors and misconceptions found in later phases of the system development cycle are extremely expensive to fix, it is evident that meticulous comprehension of the system and of its behaviour should be carried out as early as possible in the development cycle. Dedicated languages are therefore used in the requirements capturing phase to build a *model* of the system, and model checking techniques are used to analyse it in detail. The availability of a model is important for all participants in the system development; for example, if an unambiguous and executable model is available early on, customers and subcontractors can become aware with it, and can approve or improve the functionality and behaviour of the system before investing heavily in the implementation stages. Precise and detailed models are also in the best interest of the designers, analysers and testers of the system.

If the system under development is a *safety critical system*, in parallel to the standard development process described above, it is necessary to carry out a set of activities - *safety assessment activities* - whose goal is assessing the robustness of the system in degraded situations, that is, when some of the components are not working as expected. The phases, activities, and outputs of the safety assessment process are coded by various standards (e.g., ARP4754). The first step is defining the safety requirements of the system, that is, the minimum safety levels that the system must achieve. As an example, a safety requirement may be something like: "*no single failure shall yield to a loss of a given output*". The next step is assessing the safety of the architecture, by determining what are the combinations of failures of components that may cause a safety requirement to be violated. During this activity, safety engineers produce, e.g., fault trees, that are compact representations of the combination of failures leading to the violation of a given safety requirement [VGRH81]. System certification typically requires the probability of such combination of failures to be below a given threshold. The traditional safety verification process, that relies on the ability of the safety engineer to understand and to foresee the system's behaviour, is very difficult to carry out and error prone when dealing with highly complex systems. Moreover, even when formal methods are used during system development, the information passed to the safety engineer is still transmitted by means of informal specifications and the communication between system design and safety assessment activities can be seen as an "over the wall process" [FMPN94].

A solution to these issues is to perform the safety assessment analysis in some automated way, directly from the formal system model coming from the design engineer. This approach is being developed and investigated in ESACS (*Enhanced Safety Assessment for Complex Systems*), a European-Union-sponsored project in the area of safety analysis, involving several research institutions and leading avionics and aerospace industrial companies. The methodology developed within the ESACS project is supported by state-of-the-art and commercial tools for system modelling

and safety analysis. Furthermore, the effectiveness of the ESACS methodology is being tested against a set of real-world industrial case studies. In this paper we report on the application of the ESACS methodology to one of such industrial case studies, namely the Secondary Power System (SPS, for short) of the Eurofighter Typhoon and we report on the obtained results.

*Outline of the paper* The paper is structured as follows: in the next section we present the ESACS methodology and platform; in Section 3 we present the SPS case study and discuss our experience. In Section 4 we report on related work and in Section 5 we draw some conclusions and discuss future work.

## 2. The ESACS Methodology & the ESACS Platform

The main goal of the ESACS project is the definition of a methodology, that is compliant with the design and safety assessment processes of the industrial partners involved in the project. The methodology must also be supported by tools, that can be gently integrated with the other tools already in use by the industrial partners.
In order to achieve the above-mentioned goals, within ESACS we defined a methodology based on a set of key steps, that can be adapted by the various industrial partners according to their needs, and we set up a platform, called the *ESACS platform*, which is shipped in different configurations. The configurations of the ESACS platform are based on different tools, sharing the same architecture and providing the same basic functionalities. The use of different tools, although has lead to configurations that are not interoperable, has considerably eased the issue of integrating the platform within the development processes of the industrial partners.

The following two subsections describe in more details the methodology and the platform.

### 2.1 The ESACS Methodology

The main characteristic of the ESACS methodology is the capability of integrating the system design and the system safety assessment processes by providing an environment in which formal notations are the common and shared language to be used both during system design and safety assessment.

The methodology, sketched in Figure 1, is based on the following steps.

**Model Capturing** The starting point of the ESACS methodology is a *formal model*, that is, a model written in some formal language. The formal model can be either written by the design engineer or by the safety engineer. This alternative gives rise to two different scenarios.
In the first scenario, that is, when the formal model is written by the design engineer, the model, that we call system model (SM), includes only the nominal behaviour of the system. The SM is used by the design engineer to verify the functional requirements and it is then passed to the safety engineer, to assess its safety. In order

to validate the system with respect to the safety requirements, the safety engineers will enrich the behaviour of the SM by automatically *injecting* failure modes on the SM, according to what described in more details below.

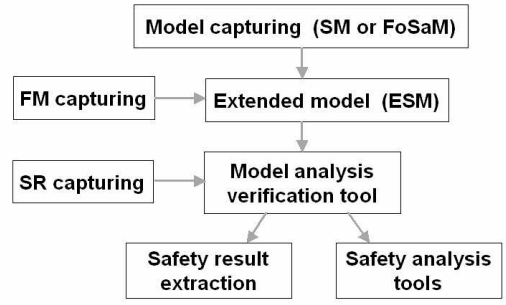


**Figure 1 – ESACS methodology steps**

In the second scenario, the formal model is built directly by the safety engineer and we call it FoSaM (Formal Safety Model). This model represents a formal view of the system highlighting its safety characteristics. To write a FoSAM, the safety engineer can browse a library of system components (that include both nominal and faulty behaviours) and a library of architectural safety patterns (containing typical structures of components to build a safety architecture, like, for instance, primary/backup of N-version systems). This second scenario is followed during the early phases of the system life cycle, when there are still no design models available, but only some system specification. In this second scenario, the main goal is assessing the system architecture.

**Failure Mode Capturing, Model Extension** The second step of the methodology includes the failure modes (FMs) capturing, the model extension, and the safety requirements capturing phases. When the SM is written by the design engineer, in order to use it for safety analyses, the safety engineer must first extend it by *injecting* with failure modes, that is, with a specification of how the various components of the system can fail. This step yields to a model, that we call *extended system model* (ESM), in which all the components of the SM can fail according to the specified failure modes. The typologies of failure modes to inject into a SM can be stored and retrieved from a library of generic failure modes, the so-called Generic Failure Modes Library (GFML) and then automatically injected into the formal system model through an extension facility.

**Safety Requirements Capturing** As long as a SM/ESM or a FoSaM is available, it is possible to verify its behaviour with respect to the desired functional (nominal behaviour) and safety requirements (degraded behaviour). During the safety requirements capturing phase, therefore, design and safety engineers define functional and safety requirements, that will be used at a later stage to assess the behaviour of the system. In particular the design engineer and/or the safety engineer will verify the

system either by writing directly the system requirements using some formal notation (e.g., temporal logic [Eme90]) or by loading the basic safety requirements of a safety critical system from a so-called Generic Safety Requirement Library (GSRL).

**Model Analysis** This is the phase in which the behaviour of a system is assessed against functional and safety requirements. The model analysis phase is performed by running formal verification tools (e.g., model checkers) on the given system properties. In particular, model analysis includes two possible verification tasks. In case of a system property, the model checking engine can test validity of the property, and generate a counterexample in case the system property is not verified (e.g., assuming the property is required to hold for every possible path of the system, the model checking engine generates a counterexample showing one particular path along which the property is falsified). In case of a safety requirement, the model checking engine generates all possible minimal combinations of components failures, called Minimal Cut Sets (MCS), that violate the safety requirements. Minimal cut sets can be arranged in the so-called Fault Tree representation [VGRH81]. Fault trees provide a convenient representation of the combination of events resulting in the violation of a given *top level event*, and are usually represented in a graphical way, as a parallel or sequential combination of AND/OR logical gates.

**Result Extraction**. During this phase the results produced by the model analysis phase are processed to be presented in human-readable format. In particular, the result extraction phase is responsible for conveniently displaying all the outputs automatically generated by the model checking engine, e.g., simulation traces and minimal cut sets, and to present results of safety analyses in formats that are compatible with traditional fault tree analysis tools used by safety engineers.

### 2.2 The ESACS Platform

The ESACS platform supports and automates the application of the methodology described in the previous subsection.

The ESACS platform is shipped in four possible configurations, namely the Altarica configuration [AGPR00], based on the Cecilia-OCAS tool, the FSAP/NuSMV-SA configuration (http://sra.itc.it/tools/FSAP), based on the NuSMV2 model checker [CCG+02], the SCADE configuration, based on the SCADE tool (http://www.esterel-technologies.com) and on the PROVER plug-in [SS00], and the Statemate configuration, based on the Statemate tool (http://www.ilogix.com) and on the VIS model checker [BHS+96].

All the configurations of the ESACS platform share the same architectural principles and functional requirements. The delivery in four different configurations has guaranteed a more flexible integration of the platform within the industrial partners' processes, by allowing, for instance, choice on the formal notation to use for writing SM/FoSAM. The general architecture of the ESACS platform is shown in Figure 2.

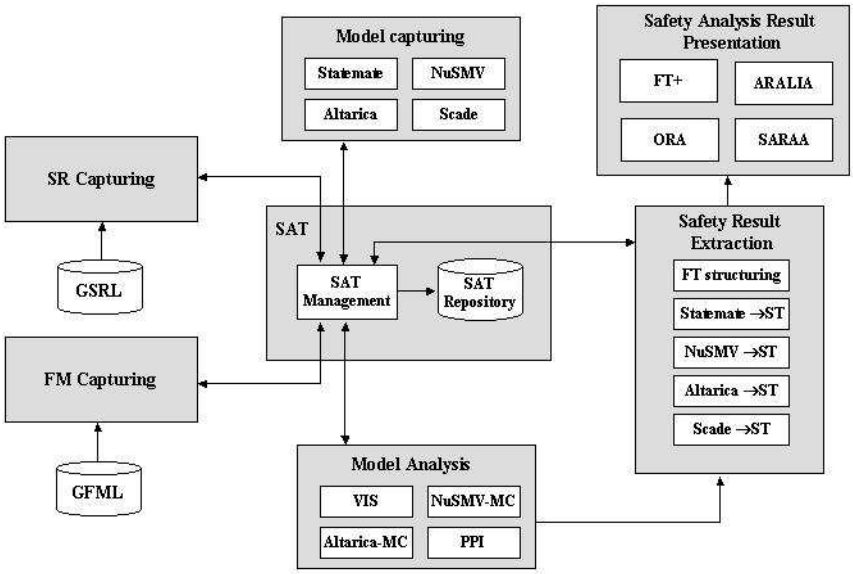


**Figure 2 – ESACS Platform Architecture**

The core of the ESACS platform is the so-called Safety Analysis Task (SAT). This block provides the core of the interaction with the system and allows users to access the libraries and to store and manage all the information relevant to the assessment of a system (SAT Repository). In particular, the SAT repository can store the following information: system models (i.e., SM, ESM or FoSaM), failure modes, system requirements (i.e., system properties and safety requirements), and the specifications and results of the analyses to perform.

The other blocks of the ESACS platform include facilities for system design, for automated system verification, and for automated safety assessment. The architecture is composed of both commercial off-the-shelf tools (i.e., for model capturing, for the verification of system properties, and for the presentation of safety analysis results) and components (both libraries and algorithms) specifically developed for the ESACS project (e.g., for system requirements capturing, failure mode capturing and system model extension, model analysis – for the generation of MCS starting from a safety requirement, and safety result extraction).

The commercial tools provided for the different ESACS platform configurations are the following. For the *model capturing* block, which is used by the design engineer to define the system formal model, the following different modelling tools are used: Altarica, NuSMV, Statemate and Scade. The *model analysis* block, used to verify the SM/ESM or FoSaM with respect a specific system property, is based on one of the following model checking engines: VIS, Prover Plug-In, NuSMV-SA (an

extension of NuSMV2, with safety analysis algorithms), and Altarica. Finally the *safety analysis result presentation* block is used to display the output of the automated fault tree generation in the traditional safety analysis tools, namely Isograph FT+, Aralia, ORA and SARAA.

The components specifically defined for ESACS include facilities for *failure mode capturing* and *model extension*, *system requirement capturing*, *model analysis* (in particular automated fault tree generation) and *result extraction*. The *failure mode capturing* block allows the user retrieve the specification of the failure modes from a specifically developed library of failure modes (GFML) and to instantiate them to a specific system model. When all the failure modes have been retrieved and instantiated, the *model extension* facility allows for the automatic extension of the SM into a ESM. The *safety requirements capturing* block allows to write functional and safety requirements; these can either be extracted from a library of generic properties (GSRL) or directly written using some standard logic formalism (e.g., temporal logic). Finally the *model analysis* and the *result extraction* blocks implement the most important facility of the ESACS approach, e.g., the computation algorithm for the automated generation of fault trees, based on formal methods techniques, and the necessary conversion algorithms to present the result of safety analysis using standard commercial tools (e.g., for analysing fault trees).

As a final remark, we stress that the basic functions provided by the ESACS platform may be combined in different ways, in order to comply with various development and safety assessment processes. For instance, 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).

## 3. A Case Study: the Secondary Power System

One of the case studies investigated in the ESACS project is the Secondary Power System (SPS hereafter) of the Eurofighter Typhoon aircraft. The case study has been chosen for the following reasons: it is of industrial interest, it is a heterogeneous system comprising various types of components like electromechanical components (e.g., control valves, relays), mechanical components (e.g., shafts, gearboxes, freewheels), electronic transducers (e.g., speed sensors, pressure sensors) and electronic controllers (SPS computers), and it has been judged of the right (medium/high) complexity to be analysed within the project. The case study has been conducted in collaboration among Alenia Aeronautica, Società Italiana Avionica, and ITC-IRST. The aims were twofold: on the one hand we wanted to investigate the behaviour of the SPS, on the other we wished to use the case study as a way to test two configurations of the ESACS platform.

The Secondary Power System drives the hydraulic and the electrical utilities of both the left and right hand side of the aircraft and therefore it can be considered as a "critical" system from the safety point of view. To satisfy the basic safety requirement, i.e. "*no single failures shall cause the total loss of the SPS utilities*", the architecture of the system includes two basic redundancies: there are two independent and perfectly symmetric lines, whose purpose is to drive the left and the right hand side utilities, respectively; for each side, the mechanical drive of the relevant utilities (*normal mode*) is redounded by a pneumatic drive (*cross-bleed mode*) in case of failure of one of the components in the mechanical line.

Figure 3 shows a schematic view of the SPS. The SPS normal operation consists in transmitting the mechanical power from the engines to the relevant hydraulic and electrical generators. In case of an engine failure, the SPS computers automatically initiate a *cross-bleed* procedure consisting in driving the hydraulic and electrical generators by means of an air turbine motor, using bled air from the opposite engine. Correct functioning of this procedure is an example of one safety requirement of the SPS system.

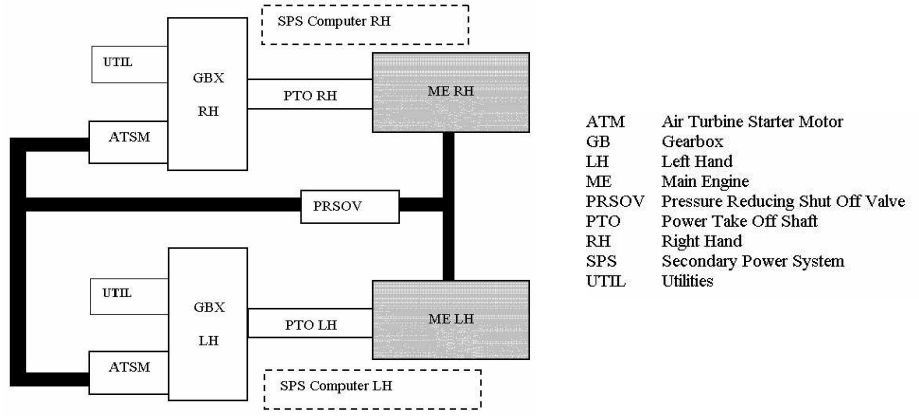


**Figure 3 – SPS schematic view**

In order to investigate the behaviour of the SPS, a set of formal models, described at different level of details, has been set up, using two configurations of the ESACS platform, namely the Statemate-based configuration and FSAP/NuSMV-SA. This hierarchy of models can be summarised as follows (in increasing order of complexity):

1. the simplest model, which – in the standard development process – is also representative of the first specification that the safety engineer receives from the design engineer - is a sort of block diagram. In our case, this simple model includes both the left and right hand side of the SPS and a very simplified model of the SPS computer. The variables used are all Boolean and the components are blocks which may be either working or not working; Figure 4 highlights the various components and the data flow among them;

2. in the second model, the behaviour of the components of the SPS is more realistic, even if the SPS computer is still very simplified. Using a *discretization* approach, variables representing physical quantities have been encoded by means of integer variables ranged between 0 and 20. Moreover, by exploiting the functional symmetry of the system, we removed some of the components, and we limited reasoning on just one side of the system;
3. the third model is as the previous one but with both sides of the system included;
4. the fourth model is derived from the previous one by enriching the SPS computer model; in the Statemate-based configuration we used real-valued variables, whereas in FSAP/NuSMV-SA we used discretized integer variables with ranges closer to the real values provided by the system;
5. finally, we realised two very detailed models in which both the nominal and the faulty behaviour of each component is modelled in detail. Variables are encoded like in the previous model. Graphs obtained by simulating these detailed models are in accordance with the graphs obtained from the telemetry on the real system.

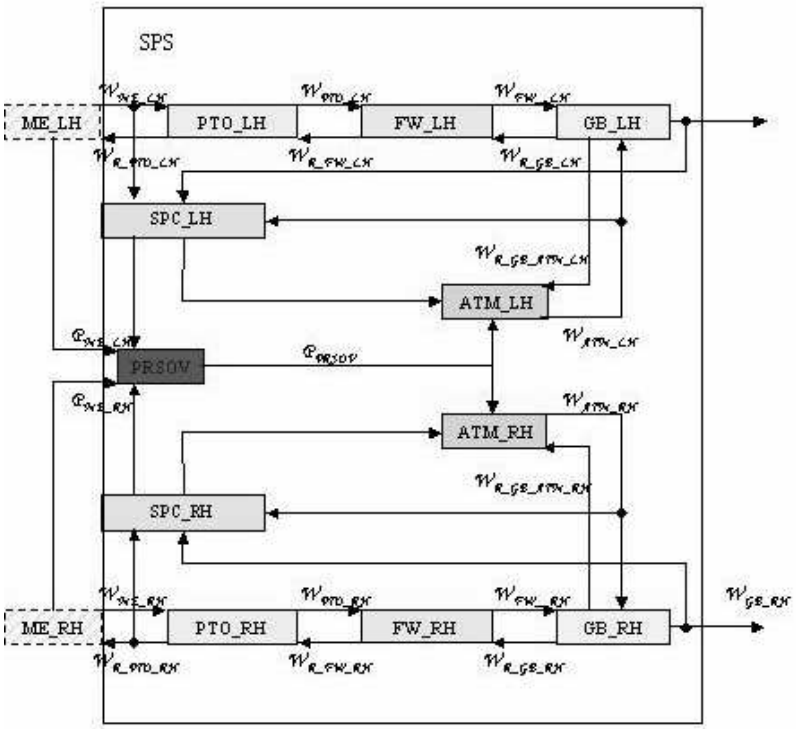


**Figure 4 – SPS block diagram model**

## 4. Results

The industrial partners involved in the SPS case study (Alenia Aeronautica and Società Italiana Avionica) performed a cycle of application, as end-users of the ESACS methodology and tool-set, in which the various SPS models were tested, in order to highlight *pros* and *cons* of the ESACS approach and to devise possible methodology and tool platform improvements. The main criteria of the evaluation were the following:

1. effectiveness of the methodology to improve the integration of the "design" and "safety" activities on the complex system;
2. effectiveness of the tool-set in the implementation of the different steps defined by the methodology (i.e., failure mode definition and injection in the system model, system property definition, system verification with respect to the functional and safety requirements, fault tree generation).

In the following we briefly summarise the results of our evaluation.

Representational Issues: one interesting aspect of the case study concerned the modelling of the various components of the SPS. In particular, one of the most challenging modelling issue has been the modelling of hydraulic and mechanical components. For such systems, in fact, when reasoning about degraded situations, the standard input/output modelling with functional blocks may be particularly difficult. (For instance, a leakage in a pipe may cause loss of pressure in the whole pipe. As a second example, in certain situations, e.g. *grippage*, mechanical forces may need to be propagated in "reverse" - e.g., by affecting functional blocks that are further "up" in the functional chain). To address these issues, particular care had to be taken in modelling such kind of aspects. More in general, we think that the use of hybrid systems modelling tools may be extremely effective for such kind of models.

Integration of the "design" and "safety" activities: we experienced that the ESACS approach effectively improves and encourages the interaction between design and safety engineers as they, for instance, can "speak" the same unambiguous language, sharing the same formal system model. Moreover, the safety evaluation of the proposed system architecture, thanks to the possibility to simulate and verify the system model, can be performed in the very early phases of system design. However, an important issue concerns the possible "semantic gap" between the model provided and the actual design/system. In fact, while existing modelling and simulation tools (e,g., the ones provided by the Statemate-based configuration) support very rich input languages, often, in order to be able to formally verify properties it is necessary to scale the model down, e.g., by abstracting away certain characteristics of the design model. This leads to models whose accuracy with respect to the real models has to be agreed upon by specialists. On the other hand, future improvements of current model checking tools can help to ameliorate this problem. We will come back to this point in the next section.

Failure mode definition and injection: within the ESACS approach, as already discussed, there are two possibilities to include the failure modes of system components in the system model: the definition of user-defined failure modes as a particular class of variables included in the system model, and/or the injection of typical failure modes (such as *stuck-at*, *ramp-down*, *delay*, ...) to take into account all the possible faulty behaviours of system components. The facility for failure mode injection and system model extension experimented during the different test cycles of the ESACS platform works well but it is based on a library of generic failure modes specifically created for the ESACS purpose. As a consequence, the library needs to be enriched in the next future to include the failure modes typical of the main types of complex system components (e.g., electronic, electric, mechanical, pneumatic components and so on).

System property definition: the ESACS approach allows the definition of different types of verification tasks on system models, like, for instance, reachability of a given state (e.g., gearbox failure) or the fulfilment of a given condition (e.g., output from one utility a certain percentage under its nominal value). In any case, in order to write the system property, it is necessary to use some particular formalism (e.g., LTL or CTL temporal logics), that are often difficult to understand, especially by people who are not expert in formal verification. As a consequence, a future improvement of the ESACS platform concerns the inclusion of different classes of system properties to instantiate the main formalisms to be used for performing the different verification tasks.

System property verification: performing model checking of functional requirements on the system model often leads to the "*state explosion*" problem. In order to mitigate this problem, the definition of a set of different models has helped, as it has been possible to define "specialised" models to be used for certain properties.
From an industrial point of view, the possibility of using simulation and exhaustive techniques to "drive" the system into a state has been proven particularly useful, for example, to show that a safety critical state cannot be reachable when failure modes are disabled.

Fault tree generation: automatic generation of fault trees has been possible with satisfactory results only for the first two types of formal model, whereas with the more complex models we encountered difficulties due to the "*state explosion*" problem. Nonetheless, the generated fault trees can still be very informative for the safety engineers (see, for instance, the example of generated fault tree in Figure 5). The possibility for the safety engineer to have more complex system models, which, at least, can be used to perform simulations, remains also a valuable aspect.

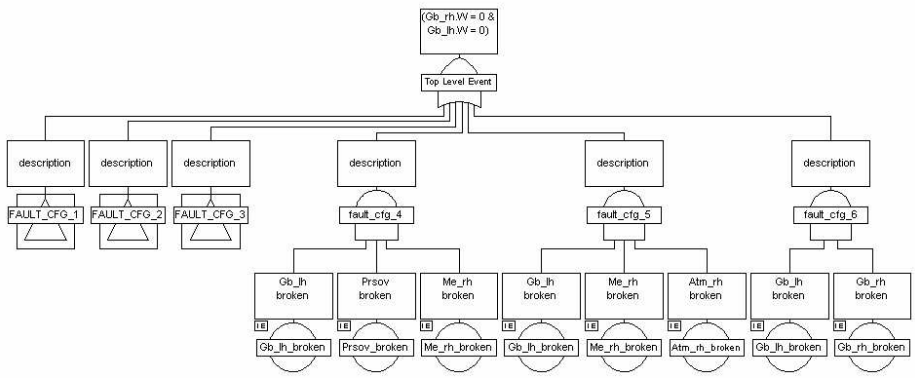


**Figure 5 – An example of generated Fault Tree**

## 5. Related Work

**ESACS Project and Platform** The work described in this paper has been developed within the ESACS project. For more details about the project we refer the reader to the URL http://www.esacs.org/. The different configurations of the platform have been tested on various case studies, among which we would like to mention:

1. Air inlet control system APU (Auxiliary Power Unit) JAS39 Gripen, related to a critical subsystem of an airplane. The work has been carried out using the SCADE based configuration.
2. Wheel Steering System, related to a critical subsystem of a family of Airbus airplanes.
3. A controller of the Airbus A340 High Lift System.
4. Hydraulic System A320, related to the hydraulic system of the Airbus A320. The work carried out using the Altarica and (partly, FSAP-NuSMV-SA), is described in [BCS02].

**FSAP/NuSMV-SA Configuration** Concerning the NuSMV-based configuration, the safety analysis capabilities provided by this platform include traditional fault tree generation [VGRH81] together with formal verification capabilities typical of model checking [CGP00, CCG+02]. The algorithms for cut set and prime implicant computation mentioned in Section 2.1 are based on classical procedures for minimization of boolean functions, specifically on the implicit-search procedure described in [CM92, CM93], which is based on Binary Decision Diagrams (BDDs) [Bry92]. This choice was quite natural, given that the NuSMV model checker [CCG+02] makes a pervasive use of BDD data structures. The ordering analysis procedure mentioned in Section 2.1 also makes use of these algorithms [BV03].

**Fault Tree Computation** The ESACS Platform can compute fault trees using algorithms based on formal methods techniques. Related work includes, e.g., [LR98, Rae00]. The implemented algorithms support both *monotonic* and *non-monotonic* systems. We also mention [MDCS98, SDC99], which describe DIFTree, 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.

**Probabilistic Safety Assessment** A large amount of work has been done in the area of probabilistic safety assessment (PSA) and in particular on dynamic reliability [Siu94]. 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 [MZDL98]. 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 [Ald87, Pap94 ], the dynamic event tree methodology [CIMP92], and direct simulation via Monte Carlo analysis [SD92, MZDL98].

## 6. Conclusions and Future Work

In this paper we have presented the ESACS safety analysis platform and methodology. The ESACS platform 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, the platform provides a set of basic functions which can be combined in arbitrary ways to realize different process development methodologies. The functionality includes 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 system maintenance easier. The major benefits provided by the use of the ESACS platform and methodology are a tight integration between the design and the safety teams, mechanisation of (some of) the activities related both to the verification and to the safety analysis of systems in a uniform environment, and support for the realization of different development methodologies (e.g., incremental development approach, based on iterative releases of a given system model at different levels of detail).

Concerning the works on dynamic reliability cited in Section 5, 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 [Pap94]) or simulation (e.g., Monte Carlo simulation [MZDL98], the TRETA package of [CIMP92]). Current work is focusing on some improvements and extensions in order to make the methodology competitive with existing approaches. In particular, we need to extend our framework in order 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., [DS94]). We also want to overcome the current limitation to permanent failures.

As far as FSAP/NuSMV-SA is concerned, the 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 [Hen96, HHW97]. This would allow both to model more complex variable dynamics, and also a more realistic modelling of time (which, currently, is modelled by an abstract transition step). Furthermore, this would ameliorate the problem of state explosion, which is partly due to the current use of discretized integer variables. Another direction of research that we are investigating is the use of SAT-based model-checking verification techniques [BCCZ99], which have been shown to be extremely efficient for model debugging and bug hunting [ABC+02, ACKS02]. In the near future, we plan to use these techniques both for interactive fault tree generation and for formal specification debugging.

## 7. Acknowledgements

Several other people contributed to the work presented in this paper. We wish in particular to thank: Ove Akerlund (Prover), Pierre Bieber (ONERA), Christian Bougnol (AIRBUS), E. Boede (OFFIS), Matthias Bretschneider (AIRBUS-D), Charles Castel (ONERA), Alain Griffault (LaBri, Universit´e de Bordeaux), C. Kehren (ONERA), Benita Lawrence (AIRBUS-UK), Andreas Luedtke (University of Oldenburg), Silvayn Metge (AIRBUS-F), Chris Papadopoulos (AIRBUS-UK), Renata Passarello (SIA), Thomas Peikenkamp (OFFIS), Per Persson (Saab), Christel Seguin (ONERA), and Luigi Trotta (Alenia Aeronautica).

Finally, FSAP/NuSMV-SA would have not been possible without the help of Paolo Traverso, Alessandro Cimatti, and Gabriele Zacco.

## 8. References

[ABC+02] Audemard, G. & Bertoli, P. & Cimatti, A. & Kornilowicz, A. & Sebastiani R. A SAT Based Approach for Solving Formulas over Boolean and Linear

Mathematical Propositions. In *A. Voronkov (Ed.), Proc. Conference on Automated Deduction (CADE-18)*, volume 2392 of LNAI, pages 195-210, Springer-Verlag, 2002.

[ACKS02] Audemard, G. & Cimatti, A. & Kornilowicz, A. & Sebastiani R. Model Checking for Timed Systems. In *D. Peled, M.Y. Vardi (Eds.), Proc. Conference on Formal Techniques for Networked and Distributed Systems (FORTE 2002)*, volume 2529 of LNCS, pages 243-259, Springer-Verlag, 2002.

[AGPR00] Arnold, A. & Griffault, A. & Point, G. & Rauzy, A. The AltaRica formalism for describing concurrent systems. *Fundamenta Informaticae*, 40:109-124, 2000.

[Ald87] Aldemir, Y. Computer-assisted Markov Failure Modeling of Process Control Systems. *IEEE Transactions on Reliability*, R-36:133-144, 1987.

[BCCZ99] Biere, A. & Cimatti, A & Clarke, E.M. & Zhu, Y. Symbolic Model Checking without BDDs. In *R. Cleaveland (Ed.) 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.

[BCS02] Bieber, P. & Castel, C. & Seguin, C. Combination of Fault Tree Analysis and Model Checking for Safety Assessment of Complex System. In *Proc. 4th European Dependable Computing Conference*, volume 2485 of LNCS, page 19-31, Springer-Verlag, 2002.

[BHS+96] Brayton R.K. & Hachtel G.D. & Sangiovanni-Vincentelli A.L. & Somenzi F. & Aziz A. & Cheng S.-T. & Edwards S.A. & Khatri S.P. & Kukimoto Y. & Pardo A. & Qadeer A. & Ranjan R.K. & Sarwary S. & Shiple T.R. & Swamy G. & Villa T. VIS: A System for Verification and Synthesis**.** *In R. Alur and T.A. Henzinger (Eds.), Proc.8th International Conference on Computer Aided Verification (CAV'96)*, Volume 1102 of LNCS, pages 428-432, Springer-Verlag, 1996.

[Bry92] Bryant, R.E. Symbolic Boolean Manipulation with Ordered Binary Decision Diagrams. *ACM Computing Surveys*, 24(3):293-318, 1992.

[BV03] Bozzano, M. & Villafiorita, A. Integrating Fault Tree Analysis with Event Ordering Information. In *Proc. European Safety and Reliability Conference (ESREL 2003)*, Maastricht, The Netherlands, 2003.

[CCG+02] Cimatti A. & Clarke, E.M. & Giunchiglia, E. & Giunchiglia, F. & Pistore, M. & Roveri, M. & Sebastiani, R. & Tacchella, A. NuSMV2: An OpenSource Tool for Symbolic Model Checking. In *Proc. International Conference on Computer-Aided Verification (CAV 2002)*, Copenhagen, Denmark, 2002.

[CGP00] Clarke, E. & Grumberg, O. & Peled, D. *Model Checking*. MIT Press, 1999.

[CIMP92] Cojazzi, G. & Izquierdo, J.M. & Meléndez, E. & Perea, M.S. 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.

[CM92] Coudert, O. & Madre, J. 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.

[CM93] Coudert, O. & Madre, J. Fault Tree Analysis: $10^{20}$ Prime Implicants and Beyond. In *Proc. Annual Reliability and Maintainability Symposium*, 1993.

[DS94] Devooght, J. & Smidts, C. Probabilistic Dynamics; The Mathematical and Computing Problems Ahead. In *T. Aldemir, N.O. Siu, A. Mosleh, P.C. Cacciabue and B.G. Göktepe (Eds.), Reliability and Safety Assessment of Dynamic Process Systems*, NATO ASI Series F, 120:85-100, Springer-Verlag, 1994.

[Eme90] Emerson, E. Temporal and Modal Logic. In *J. van Leeuwen (Ed.), Handbook of Theoretical Computer Science*, Volume B, pp. 995-1072. Elsevier Science, 1990.

[FMPN94] Fenelon, P. & McDermid, J.A. & Pumfrey D.J. & Nicholson. M. Towards Integrated Safety Analysis and Design. *ACM Applied Computing Review, 2(1):21-32*, ACM Press, 1994.

[Hen96] Henzinger, T.A. 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.

[HHW97] Henzinger, T.A. & Ho, P.-H. & Wong-Toi, H. Hytech: : A Model Checker for Hybrid Systems. *Software Tools for Technology Transfer*, 1:110-122, 1997.

[LR98] Liggesmeyer, P. & Rothfelder, M. Improving System Reliability with Automatic Fault Tree Generation. In *Proc. 28th International Symposium on Fault Tolerant Computing (FTCS'98)*, Munich, Germany, pp. 90-99. IEEE Computer Society Press, 1998.

[MDCS98] Manian, R. & Dugan, J.B., & Coppit, D. & Sullivan, K.J. 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.

[MZDL98] Marseguerra, M., & Zio, E. & Devooght, J. & Labeau, P.E. A concept paper on dynamic reliability via Monte Carlo simulation. *Mathematics and Computers in Simulation*, 47:371-382, 1998.

[Pap94] Papazoglou, I.A. Markovian Reliability Analysis of Dynamic Systems. In *T. Aldemir, N.O. Siu, A. Mosleh, P.C. Cacciabue and B.G. Göktepe (Eds.), Reliability*

*and Safety Assessment of Dynamic Process Systems*, NATO ASI Series F, 120:24-43, Springer-Verlag, 1994.

[Rae00] Rae, A. 2000. Automatic Fault Tree Generation – Missile Defence System Case Study. *Technical Report 00-36*, Software Verification Research Centre, University of Queensland, 2000.

[SD92] Smidts, C. & Devooght, J. Probabilistic Reactor Dynamics II. A Monte-Carlo Study of a Fast Reactor Transient. *Nuclear Science and Engineering*, 111(3):241-256, 1992.

[SDC99] Sullivan, K.J., & Dugan, J.B., & Coppit, D. 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.

[Siu94] Siu, N.O. Risk Assessment for Dynamic Systems: An Overview. *Reliability Engineering ans System Safety*, 43:43-74, 1994.

[SS00] Sheeran M. & and Stalmarck G. A tutorial on Stalmarck's proof procedure for propositional logic, *Formal Methods in System Design*, vol. 16(1):23–58, 2000.

[VGRH81] Vesely, W. & Goldberg, F. & Roberts, N. & Haasl D. 1981. Fault Tree Handbook, *Technical Report NUREG-0492*, Systems and Reliability Research Office of Nuclear Regulatory Research U.S. Nuclear Regulatory Commission.