ESACS: an integrated methodology for design and safety analysis of complex systems

M. Bozzano [1] & A. Villafiorita [1] & O. Åkerlund [2] & P. Bieber [3] & C. Bougnol [4] & E. Böde [5] & M. Bretschneider [6] & A. Cavallo [7] & C. Castel [3] & M. Cifaldi [8] & A. Cimatti [1] & A. Griffault [9] & C. Kehren [3] & B. Lawrence [10] & A. Lüdtke [11] & S. Metge [4] & C. Papadopoulos [10] & R. Passarello [8] & T. Peikenkamp [5] & P. Persson [12] & C. Seguin [3] & L. Trotta [7] & L. Valacca [8] & G. Zacco [1]

[1] *IRST*
via Sommarive 38, 38100 Trento, Italy

[2] *Prover Technology AB,*
Stockholm, Sweden

[3] *ONERA, Centre de Toulouse,*
2 avenue E. Belin, F-31055 Toulouse, France

[4] *AIRBUS*
1 rond point M. Bellonte, 31707 Blagnac, France

[5] *OFFIS*
26121 Oldenburg, Germany

[6] *Airbus Deutschland GmbH,*
21129 Hamburg, Germany

[7] *Alenia Aeronautica S.p.A,*
Strada Malanghero 17, 10072 Caselle (TO), Italy

[8] *Società Italiana Avionica*
Strada Antica di Collegno, 253 10146 Torino, Italy

[9] *LaBri, Université de Bordeaux*
Domaine Universitaire 351, cours de la Libération
33405 Talence Cedex 1 33405 Talence, France

[10] *Airbus UK Ltd.*
New Filton House, Golf Course Lane
Filton, Bristol BS99 7AR, UK

[11] *University of Oldenburg*
26129 Oldenburg, Germany

[12] *Saab AB*
58188 Linköping, Sweden

ABSTRACT: The continuous increase of system complexity – stimulated by the higher complexity of the functionality provided by software-based embedded controllers and by the huge improvement in the computational power of hardware – requires a corresponding increase in the capability of design and safety engineers to maintain adequate safety and reliability levels. Emerging techniques, like formal methods, have the potential of dealing with the growing complexity of such systems and are increasingly being used for the development of critical systems (e.g., aircraft systems, nuclear plants, railways systems), where at stake are not only delays in delivering products and economical losses, but also environmental hazards and public confidence. However, the use of formal methods during certain critical system development phases, e.g. safety analysis, is still at an early stage. In this paper we propose a new methodology, based on these novel techniques and supported by commercial and state-of-the-art tools, whose goal is to improve the safety analysis practices carried out during the development and certification of complex systems. The key ingredient of our methodology is the use of formal methods during both system development and safety analysis. This allows for a tighter integration of safety assessment and system development activities, fast system prototyping, automated safety assessment since the early stages of development, and tool-supported verification and validation.

1 INTRODUCTION

The dramatic improvement in the computational power of hardware brings about increasingly sophisticated software-based embedded controllers that take over complex functionality in an efficient, precise, and flexible way. These benefits allow the use of such systems in environmental conditions where delays in delivering products and/or economical losses are not the only things at stake, but also environmental hazards and public confidence.

This development, however, entails an unavoidable increase in the complexity of systems, which is expected to continue in the future. Therefore, in order to retain the benefits from more sophisticated controllers, a corresponding increase in the capability of the design and safety engineers to maintain adequate safety and reliability levels is required.

One of the most challenging issues in system development today is to take into consideration, during development, all possible failures modes of a system and to ensure safe operation of a system under all conditions. Current informal methodologies, like manual fault tree analysis (FTA) and failure mode and effect analysis (FMEA) (Vasely, 1981; Liggesmeyer & Rothfelder, 1998; Rae 2000), that rely on the ability of the safety engineer to understand and to foresee the system behavior are not ideal when dealing with highly complex systems, due to the difficulty in understanding the system under development and in anticipating all its possible behaviors.

Emerging techniques like formal methods (Wing, 1990) have the potential of dealing with such a complexity and are increasingly being used for the development of critical systems (see, for instance, (Hinchey & Bowen, 1999)). Formal methods allow a more thorough verification of the system's correctness with respect to the requirements, by using automated procedures. However, the use of formal methods for safety analysis purposes is still at an early stage. Moreover, even when formal methods are applied during system development, the information linking the design and the safety assessment phases is often carried out by means of informal specifications. The link between design and safety analysis may be seen as an "over the wall process" (Fenelon et al., 1994).

A solution to the issues mentioned above is to perform the safety assessment analysis in some automated way, directly from a formal system model originating from the design and safety engineer.

This approach is being developed and investigated within the ESACS project (Enhanced Safety Assessment for Complex Systems), 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 tools and is being trialed on a set of industrial case studies.

Outline of the paper. This paper is structured as follows. In the next section we present the ESACS approach and illustrate its use through a simple example. In section 3 we present the architecture of the ESACS platform. Finally, in section 4 we draw some conclusions and discuss related work.

2 ESACS METHODOLOGY

2.1 *Methodology*

The ESACS methodology aims to:
- Support system development and safety analyses processes.
- Provide a tight link between design and safety analysis.
- Support automated verification and validation of the design.
- Support automated safety assessment of the design.

The goals mentioned above are being achieved through two basic ingredients: firstly, the use of formal notations both for design and for safety assessment of systems and secondly, the extension of state-of-the-art tools, to provide users with a set of basic functionality that can be combined in different ways.

The use of formal methods allows for automation of analyses and for a tighter integration between system design and safety analysis (as the information exchanged between design and safety engineers is based on the same formal models, with a clear and unambiguous syntax and semantics). Tool support, achieved by implementing a set of basic functionality in state-of-the-art tools, allows for automated support of the methodology and for its flexible integration in various development and safety processes.

The ESACS methodology takes into account and supports two main scenarios. In the first scenario the process is initiated by the design engineer, who provides a formal model of the design (at a given level of abstraction) using formal modeling tools (System Model Definition). The model is then automatically enriched with failures in order to perform safety analyses (Failure Mode Definition and Failure Mode Injection). The second scenario is useful when safety assessment activities must be carried out when there is still no formal design model to start with, e.g. to evaluate a proposed system architecture. In this case, the process is initiated by the safety engineer, who builds a high-level model of the system using a library of components enriched with failure modes. Such model can then be used to perform safety assessment on the system (System Model Prototyping for Safety Assessment).

The scenarios sketched above entail for the definition of the following, tool-supported, basic functionality:
- **System Model Definition:** definition, using formal notations, of an executable specification (at a given level of abstraction) of the model of the system under development (what we call *design model*).
- **System Model Prototyping for Safety Assessment:** definition, using a library of pre-defined components, of a model suitable for safety assessment. This functionality allows performing safety analyses in the early phases of the development process.
- **Failure Mode Definition** and **Failure Mode Injection:** definition of the failure modes of the components constituting the design model. The failure modes can be automatically injected into the design model, in order to produce what we call an *extended system model* (ESM). The ESM is an executable specification of the design model in which components can fail according to the failure mode specification. Extended system models can be used by safety engineers to perform safety analyses.
- **Functional and Safety Requirements Definition:** definition of the functional requirements and of the safety requirements of the system, using a formal language (e.g. linear temporal logic and computation tree logic (Emerson, 1990)).
- **Design Assessment:** automated assessment of the design model against the functional requirements, using standard formal methods techniques (e.g., simulation, theorem proving (Boyer & Moore, 1979), and model checking (Clarke et al., 1999)).

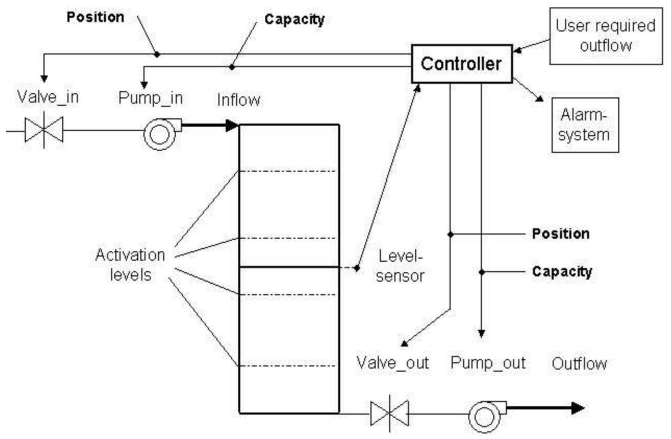

Figure 1. A simplified Tank Controller

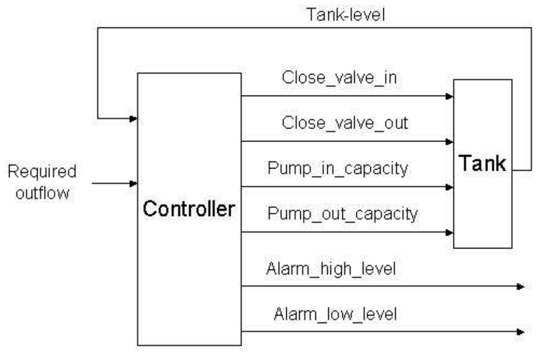

Figure 2. The model of the tank controller.

– **Safety Assessment:** automated safety assessment of the extended system model against the safety requirements, using automated techniques based on formal methods for automated fault tree analysis and automated failure mode and effect analysis.

Note that the basic functionality provided by the platform, which we have described, can be combined in different ways, in order to comply with any given development methodology one has in mind. 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).

2.2 *A simple example*

To illustrate the basic concepts of the methodology we will show its application to the design of a controller that regulates the level of a fluid in a tank.

The goal of the controller is computing the capacity of two pumps that regulate the inflow and the outflow of a liquid contained in a tank, so that the tank level remains between the two “Activation levels” in the middle of the tank. At the same time, the controller services a request, coming from the user of the system, for a minimum outflow capacity. If the tank level, obtained through a dedicated sensor, becomes either too low or too high, the controller must issue an alarm, and close the input or the output emergency valves. The user request for a minimum outflow is the only “external” input to the system. The controller monitors the tank level through a sensor providing the actual tank level. Figure 1 illustrates the components in the system.

In the following we focus on one of the two scenarios supported by the methodology, namely the scenario in which the model is built by the design engineer.

The first step of the methodology, therefore, is the definition of the design model of the system on which we will perform the automated analyses, using formal notations. Such step is performed using standard modeling tools: Figure 2 illustrates the high level decomposition of the model highlighting the inputs/outputs of the system. The model is written in SCADE[1]. The model also includes a definition of the environment with which the controller interacts: this allows the verification of the whole system (i.e., controller + sensors + actuators + tank) to ensure that it behaves as expected.

The next steps of the methodology are the definition of the functional requirements of the system and the formal assessment, to verify and validate the system against the functional requirements, in its nominal working conditions. The goal of this activity is to acquire confidence on the behavior of the system when all the components work as expected. Thus, we assume that the environment is functioning correctly and verify that the controller satisfies certain requirements. For example, we assume that the pumps and valves are functioning according to controller demand and that the level indicator is sending correct information. We investigate properties like: (1) is it possible to have an overflow, (2) is it possible to have a drain, (3) can there be a false alarm or (4) can there be an overflow without alarm, etc. The properties are formalized using standard logical formalisms and the verification of the nominal system behavior is performed automatically using theorem proving and model checking techniques. After this step we are confident that the design model behaves as expected in nominal situations.

This is, however, not sufficient to ensure that the system behaves as expected in all the possible situations. Having confirmed that all the requirements are fulfilled in the nominal case, it is then necessary to

[1] See http://www.esterel-technologies.com

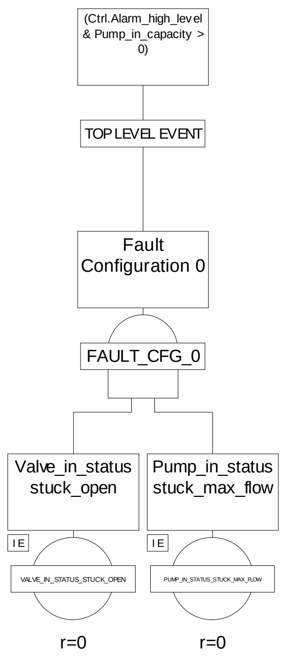
(Ctrl.Alarm_high_level & Pump_in_capacity > 0)
TOP LEVEL EVENT
Fault Configuration 0
FAULT_CFG_0
Valve_in_status stuck_open
Pump_in_status stuck_max_flow
I E
I E
VALVE_IN_STATUS_STUCK_OPEN
PUMP_IN_STATUS_STUCK_MAX_FLOW
r=0
r=0

Figure 3. A simple fault tree, generated by the ESACS platform, of the safety requirement "it is not possible to have a high level alarm while the input valve is not closed"

investigate if failures modes can make the system fail to meet the defined requirements. In order to do so, we must first identify and formalize the possible failure modes (FM) of the various components of the system. This is done in the failure mode definition phase, during which safety engineers identify and allocate all the possible failures of the components. This is performed, in the ESACS methodology, by retrieving, from a library of pre-defined failure modes, the failure modes of the different components of the system and by allocating such failures to the various components. The allocation of the failure modes to the components (Failure Mode Injection) automatically transforms the design model into a new model, that we call an *extended system model*. The extended system model enriches the possible behaviors of the system taking into account all the situations in which some of the components may fail according to the specifications provided by the safety engineers.

In the tank example, for instance, we can assume that the input pump (*Pump_in*, in Figure 1) can fail in two ways: (1) no flow can be produced or (2) the pump delivers full capacity all the time. In this case the model of the pump (whose output is modeled by the variable *Pump_in_capacity*) becomes, in the extended system model:

```
if (Pump_in_status = stuck_zero) then
  Pump_in_capacity = 0
elseif (Pump_in_status = stuck_max_flow)  then
  Pump_in_capacity = max_capacity
else
  Pump_in_capacity = nominal, (that is, the flow
                       required by the controller)
```

Notice that the occurrence of failure is revealed by the variable *Pump_in_status* that is automatically generated during failure injection. So, if no failures occur, then the pump will react as required by the controller. However, if any of the failures occurs, then the output produced by the pump is determined by the failure. The model extension algorithm also takes care of defining the behaviour of the variables encoding the failures: it is for instance possible to express the condition that no two different failures can occur at the same time on the same component or that no more that four different failures can happen for a certain simulation, etc.

Once we have an extended system model (either built using failure injection on the design model or by directly building a model with failures from the library of components), we can perform safety assessment on the model, either via failure mode and effect analysis (FMEA) or via fault tree analysis. This is done using algorithms based on model checking and theorem proving techniques for, e.g., automatically building fault trees. The algorithms supported by the ESACS methodology are similar to those described in (Liggesmeyer & Rothfelder, 1998; Coudert & Madre, 1993) and allow to analyze both monotonic and non-monotonic systems; the results produced by the analyses can be shown using commercial safety analysis tools. Figure 3, for instance, shows a simple fault tree automatically generated by ESACS algorithms and imported in FaulTree+.

3 ESACS PLATFORM

As illustrated in the previous section, an important aspect of the ESACS methodology is the support provided to the design and safety engineer by tools. This support is provided by the ESACS Platform, which has been defined and is being developed within the ESACS project.

The starting point for the development of the ESACS platform has been the tools used by the industrial partners for design and for performing safety assessment, that include (but are not limited to) Cecilia-OCAS, Statemate, SCADE, Simulink, FaultTree+. However, the task of providing a single platform that integrated all these tools and delivers to the user an interoperable environment, in which different tools and different modeling languages could be used interchangeably, was judged to be too risky for the time span and goals of the project. For instance, the problem of providing sound translators for the various input languages would have been a project in itself.

The approach taken in the ESACS project, thus, was to provide different configurations. Each configuration uses its own input language (tailored to the needs of an industrial partner) and its own set of sup-

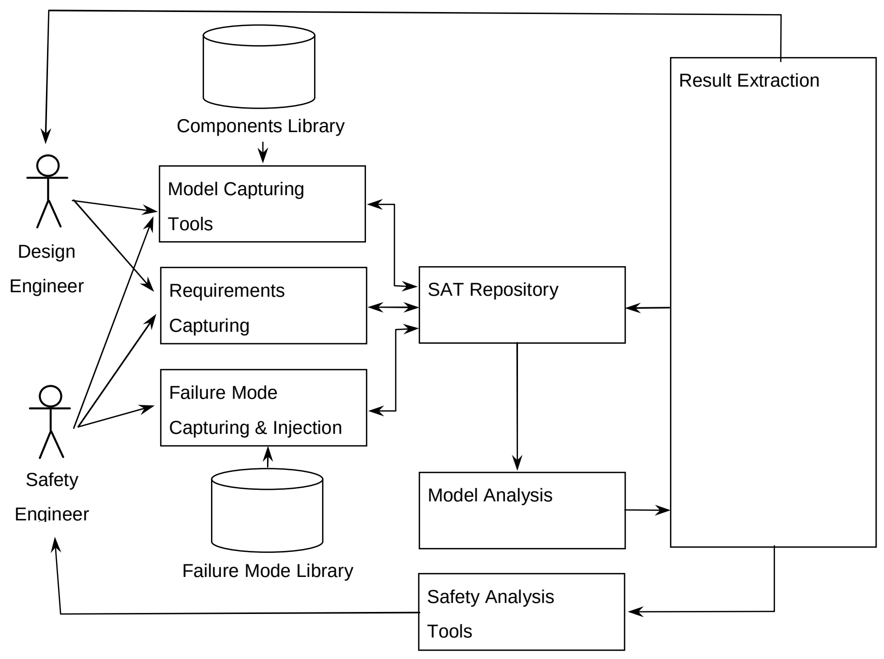

Figure 4. The ESACS Platform Reference Architecture.

port tools. All the configurations have been developed using the same underlying principles and the same architectural schema.

This has allowed the development and the delivery of the following four different configurations:

- **Altarica configuration**, based on the Cecilia-OCAS tool.
- **NuSMV configuration**, based on the NuSMV model checker.
- **SCADE configuration**, based on the on SCADE tool and on the PROVER plug-in.
- **Statemate configuration**, based on Statemate tool and on the VIS model checker.

3.1 *ESACS architecture*

The ESACS reference architecture is depicted in Figure 4. In particular, we distinguish the following blocks:

- **Model Capturing Tools**, the block used for defining the models for the analyses. It is used by the design engineer to define the design model and by the safety engineer to design models enriched with failures for safety assessment.
- **Components Library**, a library of components, extended with failures, which can be used for providing fast prototypes for safety assessment. It is used by the safety engineer to fast prototype systems on which to perform preliminary safety analysis.
- **Requirements Capturing**, the block for defining the functional and safety requirements. It is used both by design and safety engineers. The design engineer uses this block to define functional requirements against which to check the consistency of the design model. The safety engineer uses this block to define the safety requirements, against which to test the extended system model.
- **SAT Repository**, the central repository of the ESACS platform. The SAT (Safety Analysis Task) repository stores all the information about a system, like, for instance, the analyses to be performed.
- **Failure Mode Capturing & Injection**, the block for defining failure modes and injecting them into the system model; the injection yields the extended system model. This block is used by the safety engineer to define what are the degraded behaviors of the system model and to generate the extended system model, on which to perform safety analyses.
- **Model Analysis**, the block for performing all the analyses both on the design model and on the extended system model. The ESACS Platform configurations support standard formal verification analyses (e.g., simulation, model checking, and theorem proving) and safety analyses (fault tree construction and FMEA table construction).
- **Result Extraction**, the block responsible for presenting the results produced during the analyses in formats understood by commercial tools (e.g., fault trees in the FaultTree+ format).
- **Safety Analysis Tools,** a block that comprehends various commercial safety analysis tools (e.g., FaultTree+), that are used to display results in a way that is familiar to safety engineers.

Some of the components of the ESACS platform (i.e. Model Capturing tools and Safety Analysis Tools) are based on standard commercial tools: therefore, we do not illustrate them here any further.

The rest of this section, instead, focuses on the components that are particular to the ESACS Platform (namely, Components Library, Generic Failure Model Library, Failure Mode Capturing and Failure Injection, Model Analysis, and Result Extraction).

Components Library. The library of components gathers formal models of basic system components immediately suitable for safety analysis. Thus, each formal model describes both nominal and faulty behaviour of a specific system component. In the earliest phases of the system design, details of physical components are not fixed and safety engineers work with simple functional blocks and failure modes. Usually, a block offers a service (outputs) provided that some inputs and resources are available in the nominal case. Only permanent failures are considered and, after a failure, the block no longer provides an output. Some other blocks do not require inputs or resource to play their role (e.g., source of energy), or they do not provide any output (receptor). The library contains a first family of such generic blocks. It is responsibility of the safety engineer to select the appropriate failure modes to be assigned to the different components, among the available ones. During the "preliminary system safety assessment", the system components are better known and the safety engineers take into account more specific failure modes. At this stage, the library contains two other families of components. One is dedicated to components of hydraulic systems (reservoir, pump, valve, pipe, …) and deals with failure modes such as total loss, leak in a component, … The second family of components is dedicated to components of electrical systems (generator, bus, switch, receptor, …) and handles failures such as total loss, short-circuit, … The library is currently written in the AltaRica language (Arnold et al., 2000). Basically, each AltaRica model consists of two parts. An automaton describes which failure or nominal mode may be activated when a failure or a normal event occurs. Then, a set of logical assertions describes the relationship between the input/output of the components according to the current modes (see for instance (Bieber et al., 2002)). The library is implemented within the Cecilia-OCAS environment. The environment offers graphical facilities and a system model is built by drag and drop of icons of the components from the library panel to the graphical editor window. Then, the components are connected by drawing links between icons. A translator from AltaRica to Lustre has been developed [http://altarica.labri.fr/Tools/AltaLustre/] and will allow the ability to have similar libraries written in the Lustre language, usable in the SCADE environment.

Failure Mode Capturing and Failure Injection. In the design model of a system, the goal is to specify the nominal behaviour of the system fulfilling all the functionality. Suppose it is verified that this design, i.e. its nominal behaviour, fulfils all requirements. Also assume some of the inputs to and outputs from the design model represent signals coming from or going to "components", which can malfunction – i.e. there is a possibility that the nominal behaviour can be changed due to a component failure mode (FM), e.g. a valve may fail in a stuck position. In order to include FM in the analyses we extend all input and output signals, which can malfunction, with FM nodes. Having this extended system model, it is now possible to investigate if requirements are fulfilled when FMs are allowed.

FM are defined through the Failure mode capturing module that safety engineers use firstly to identify the components of the model that have to be enriched, secondly to specify the parameters of the failure, and thirdly to assign possible failures to the elements of the design. Failures are retrieved from a Generic Failure Mode Library. The Generic Failure Mode Library defines and specifies the kind of failures that can be injected into the system model: the library supports all the common failure modes, such as "stuck at", "inverted", "glitch" (the "glitch" failure allows for transient changes in the outputs delivered by a component).

Once defined, failure modes can be automatically injected into a design model via the Failure Injection module. Failure injection takes as input a design model coming from the design engineer, the failure modes defined by the safety engineers, and produces an extended system model, namely a model in which components may fail.

Only permanent failures are supported so far. The modeling of FM nodes is done such that the occurrence of a FM has priority over the nominal behaviour (see the example in Section 2).

Requirements Capturing. This module is used to define the functional requirements and the safety requirements of the extended system model. The requirements are written using standard logic formalisms (e.g. linear temporal logic and computation tree logic) using the facilities provided by the tools supporting the design of models.

SAT Management. The SAT is the central repository of all the information related to the (safety) assessment of a system. Within the SAT are stored references to design model, functional and safety requirements, references to the extended system model, and analyses task specifications, namely, the specification of the (safety) analyses that have been performed on the system.

In order to provide easy accessibility and portability, the SAT is stored in XML format (W3C, 2000). The XML tags provide the structure of the SAT and encode, through attributes, information on how the content of the tags shall be interpreted. In this way, all the different configurations share the same XML structure to encode the information and can share the same computing facilities (e.g. transformation into HTML).

The SAT is currently supported by the FSAP/NuSMV-SA configuration line, an extension of the NuSMV model checker (Cimatti et al., 2002). We conclude this section by discussing the model analysis and result extraction modules of the ESACS architecture.

3.2 *ESACS verification engine*

The model analysis and result extraction modules include the verification engines to perform verification and safety analyses on the design, and the necessary conversions algorithms to present the results of safety analyses using commercial safety analysis tools. At the moment, the following analyses are supported.

Bottom up analysis. As soon as a simulator of formal specification is available, it can be used to assist the failure mode and effect analysis. The safety engineer injects one or more failure event; the simulator computes the effects of the failure according to the propagation laws encoded in the formal texts; finally, the engineer inspects the reached states and analyses the effects. Thanks to the hierarchies of formal models, local effects can be propagated to higher views so that global effects can be identified. Let us consider an aircraft whose surfaces (spoilers, flap, ...) and some other devices are displaced thanks to hydraulic power. The aircraft model is the top level of the hierarchy. It includes surfaces and hydraulics system models. In turn, the hydraulics system model consists of models of atomic components (pumps, pipes, etc.). Failures affect these atomic components whereas the global impact is perceptible at the aircraft level. It is worth noting that a good graphical simulator makes this kind of analysis easy and intuitive for safety engineers (see a snapshot of a Cecilia OCAS simulation for instance).

Dynamic failure behaviour. Traditional FTA is a static analysis – i.e. it is done for a given system configuration – investigating the influence of failure modes on unwanted system behaviour. In our approach, since the general design includes the dynamic system behaviour, we can also investigate the influence of FM in dynamic situations. This gives us the possibility of doing new types of dynamic analyses, e.g., to see if the order of occurrence of FM is important or if an intermittent FM has the same impact as a constant FM. Given that the analysis result shows the system can malfunction, then a so-called counter-model – showing a sequence of values for all system variables – is generated. In such a case it is important to analyze the whole sequence and not only the last time-step of the sequence. Another possibility is to define the top level event, i.e. the unwanted system behaviour, with regard to the system dynamics. For example we could regard a top event to occur not until the unwanted event has existed continuously for a certain time.

Traditional fault tree generation. The ESACS Platform can compute fault trees using algorithms based on formal methods techniques. The implemented algorithms support both monotonic and non-monotonic systems; the minimization of non-monotonic systems is based on algorithms presented, e.g., in (Coudert & Madre, 1993). The underlying principle for model-checking based fault tree construction is as follows. The algorithm starts from an extended system model and a top level event and generates, using standard symbolic model checking techniques, a formula representing all the possible ways in which the top level event is not satisfied by the extended system model. From such a formula it is possible to extract all the possible combinations of failures of components. If the design model behaves correctly with respect to the top level event (i.e. if the top level event is verified by the design model), such combinations of failures are exactly the reason for the top level event not being fulfilled anymore in the extended system model (failure modes being the only change between design model and extended system model). Standard minimization techniques are then run on the combination of failures identified thus far, in order to extract from them the combinations that are minimal. The algorithm produces outputs that are suitable for integration with commercial safety analysis tools (e.g., FaultTree+).

Fault tree with ordering information. When the failure mode behaviour of a system includes both primitive and derived failures, it is also possible to perform ordering analyses on the model. This is done by selecting a minimal cut set and by verifying whether the failures related to a particular top level event only occur in a particular order. We refer the reader to (Bozzano & Villafiorita, 2003) for a detailed discussion on this topic.

4 RELATED WORK AND CONCLUSIONS

In this paper we presented the methodology that we are investigating within the ESACS project. The ESACS methodology is based on a tight integration between system design and safety analysis and on the use of formal methods for performing both design and safety assessment. The methodology is supported by state-of-the-art tools that have been extended with innovative algorithms for the safety assessment of systems.

Currently, the ESACS methodology and the ESACS platform are being tested in the following case studies:

- Air inlet control system APU (Auxiliary Power Unit) JAS39 Gripen, related to a critical subsystem of an airplane;
- Wheel Steering System, related to a critical subsystem of a family of Airbus airplanes;
- A controller of the Airbus A340 High Lift System;
- Hydraulic System A320, related to the hydraulic system of the Airbus A320;
- Secondary Power System (SPS) related to power system of the Eurofighter Typhoon.

All the case studies have been chosen to show a reasonable degree of complexity. For instance, the SPS comprises two independent channels controlled by two independent computers. 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 initiates 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. This is an example of one safety requirement of the system.

A set of formal models have been produced for the different case studies and a first series of tests have been run. The results are interesting. As common with formal methods techniques, the algorithms are sensitive to the models and to the properties provided as input and are subject to the state explosion problem. In particular, failure injection contributes to increasing the size of models, as it increases the possible behaviors.

Current work is focused on the direction of making the algorithms more efficient and in the direction of making interaction with the user easier.

5 ACKNOWLEDGMENTS

The work described in this paper has been and is being developed within the ESACS Project, an European sponsored project, G4RD-CT-2000-00361.

Several other people contributed to the work presented in this paper. We wish in particular to thank: André Arnold from LaBri-Université de Bordeaux, Jack Foisseau from ONERA, Jean Gauthier from Dassault Aviation, Jean-Pierre Heckmann from Airbus, Torgny Knutsson from SAAB, Sylvain Lajeunesse from GFI Consulting, Antoine Rauzy from IML-Université de Marseille, and Paolo Traverso from IRST.

REFERENCES

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

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

Boyer, R.S. & Moore, J.S. 1979. *A Computational Logic*. Academic Press, New York.

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

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

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

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

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

Fenelon, P., J.A. McDermid, D.J. Pumfrey, and M. Nicholson. 1994. Towards Integrated Safety Analysis and Design. In *ACM Applied Computing Review*.

Hinchey, M. G. & Bowen, J. P. 1999. *Industrial Strength Formal Methods in Practice. Springer-Verlag, London.*

Liggesmeyer, P. & Rothfelder, M. 1998. 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.

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

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

W3C 2000. Extensible Markup Language (XML) 1.0 (Second Edition), *W3C Recommendation. Available on the internet http://www.w3.org/TR/2000/REC-xml-20001006*.

Wing, J. M. 1990. *A specifier's introduction to formal methods. IEEE Computer 23(9):8-24.*