Formal assessment techniques for embedded safety critical system

Christel Seguin, Pierre Bieber, Charles Castel, Christophe Kehren
ONERA - Centre de Toulouse,
2 avenue E. Belin
France
{firstname.name@onera.fr}

Keywords: formal models, safety assessment, embedded systems, AltaRica, robotic systems

Abstract

Recently, ONERA was involved in the ISAACS European project. The aim of this project was to investigate new safety assessment techniques based on the use of formal design languages and associated tools. ONERA studied more specifically the applicability of the AltaRica language and the Cecilia OCAS environment to perform the safety assessment of some Airbus aircraft systems. In this paper, we first recall the methodology developed for such traditional embedded safety critical system. Then we discuss its applicability to robotics systems.

Introduction

During the last three years, ONERA was involved in the ISAAC (Improvement of Safety Activities on Aeronautical Complex Systems) European project (ref. 1, http://www.isaac-fp6.org/). This project aimed at developing safety assessment techniques based on the use of formal specification languages and associated tools. So called formal models are traditionally used to specify the expected normal behaviours of software based system. ISAACS partners investigated first how to generalize such models to deal with faulty behaviours of various kinds of systems. Then they proposed new tools or new uses of existing tools to check whether the generalized formal models met qualitative safety requirements. These tools provide not only interactive simulation capabilities but also take advantage of formal language features to support advanced capabilities such as model-checking or fault tree generation. The approach was validated on some existing aircraft systems.

In this paper we first present how AIRBUS and ONERA tackled modelling and safety assessment of aircraft systems using the AltaRica language and a subset of the associated tools. Moreover, the concepts are illustrated by a case study inspired by the hydraulic system of the Airbus A320.
Safety assessment based on formal models raised two main issues. The first issue is to get formal system models meaningful for safety analysis whereas models more often used are fault trees. ISAACS partners are interested in failure propagations in complex dynamic systems. So they consider formal notations for reactive systems, used to support system design such as Statechart (models are automata), Scade (models are equations between synchronous data flows) or dedicated to safety such as AltaRica (models mixing automata and equation concepts). To cope with failure propagations, system models can either be produced by system designers and then extended with failure modes specified by safety engineers, or can directly be produced by safety specialists using libraries. It is worth noting that formal models of failure propagations should have the correct granularity level to ease model exploitation. On one hand, advanced simulation capabilities have good performances when the analyzed model does not go into detailed arithmetic computations. On the other hand, a correct granularity is reached when the scenarios, leading to a failure condition, extracted by the tools are similar to what safety analysts would have envisioned if they had to design a fault tree. In order to get the appropriate granularity at first shot, we chose to define libraries of AltaRica components that focus on failure mode propagation and abstract details of nominal behaviours.
The second issue is related to the choice of the adequate techniques for assessing qualitative safety requirement of complex dynamic systems. Interactive simulation facilities enable to perform a preliminary bottom up analysis since failures can be injected and their effects computed not only locally but at system or even aircraft level. This will be detailed later on. Top down analyses are guided by qualitative requirements such as “no single failure leads to the system loss”. We propose to use model-checkers to assess such kind of requirements. They perform “exhaustive” simulation to check whether a requirement is always met. Moreover, they can distinguish subtle temporal situations such as a transient loss of a function (during a recovery phase for instance) from a permanent one.

The paper has the following structure. First section describes one traditional safety critical aircraft system: a hydraulic system inspired by A320 system. We focus on its safety requirements and architecture. Section 2 introduces the AltaRica language through examples. We explain the modeling philosophy used to build the hydraulic formal model at a satisfying granularity level for safety assessment. Section 3 deals with the benefit of advanced simulation capabilities to assess qualitative safety requirements on dynamic models. We show how the models were analyzed using interactive simulation facilities of Cecilia OCAS and SMV (Symbolic Model Verifier) model-checker. The last section discusses the applicability of such formal assessment techniques for robotic systems.

Case-study Presentation

The role of the hydraulic system is to supply hydraulic power to devices which ensure aircraft control in flight like the flaps, slats, or spoilers as well as devices which are used on ground like the braking system. As the loss of devices powered by this system could lead to the loss of aircraft control, the main safety requirement of this system is:

> A total loss of hydraulic power is considered to be catastrophic. The probability of occurrence of this failure condition should be smaller than 10^{-9} per flight hour and no single failure should lead to this failure condition.

The hydraulic system is mainly composed of three independent sub-systems which generate and transmit the hydraulic power to the consumers. Three kinds of pumps were used in the model of an A320-like hydraulic system. The first one is the Electric Motor Pump (EMP) which is powered by the electric system, the second one is the Engine Driven Pump (EDP) that is powered by one of the two aircraft engines and the last one is the RAT pump that is powered by the Ram Air Turbine. The hydraulic system also contains other types of components such as tanks, valves and gauges.

To meet its main safety requirement, the system is constituted of three channels: Green, Blue and Yellow. The Blue channel is made of one electric pump EMPb, one RAT pump and two distribution lines: prioritary (Pdistb) and non-prioritary (NPdistb). When priority valve PVb is closed consumers connected to Npdistb do not receive hydraulic power. The Green system is made of one pump driven by engine 1 EDPg and two distribution lines Pdistg and NPdistg. The Yellow system is made of one pump driven by engine 2 EDPy, one electric pump EMPy and two distribution lines Pdisty and NPdisty. Moreover a reversible Power Transfer Unit (PTU) transmits pressure between green and yellow channels as soon as the differential pressure between both channels exceeds a given threshold.

These components are controlled by crew actions and reconfiguration logics. The RAT is automatically activated in flight when both engines are lost. The EMPb is automatically activated when the aircraft is in flight or on ground when one engine is running. EMPy is activated by the pilot on ground. We assumed that EDPy, EDPg were activated whenever the corresponding engine was started.

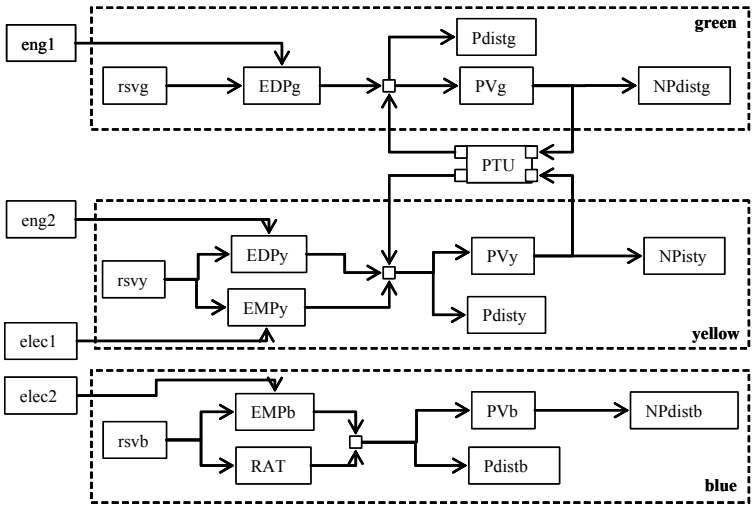

Figure 1 − Hydraulic System Architecture

System Modelling in AltaRica

The AltaRica Language: AltaRica (ref. 2) is a formal language developed at LaBRI (Laboratoire Bordelais de Recherche en Informatique) for modelling both functional and dysfunctional behaviours of systems. Thanks to the language well defined semantics and syntax, safety assessments of AltaRica models can be analysed by numerous reliability or validation tools. Moreover, its capacity to realise compositional and hierarchical models is a great advantage when complex systems must be modelled. The development of AltaRica models is supported by Cecilia OCAS workshop (ref. 3) of Dassault Aviation that provides graphical edition and simulation facilities and integrates fault tree generators. We will now briefly describe this language.
Each system component is modelled by a "node". A node is a mode automaton (ref. 4) defined by three well identified parts. First part is the declaration of the different kinds of node parameters: state, flow and event. States are internal variables which memorize current functioning modes (failure modes or normal ones). Flows are node inputs or outputs. Possible types of states and flows are integer interval, enumeration and boolean. Events are phenomena, which trigger transitions from an internal state to another. They can model pilot actions or the occurrence of failures, or reactions to input conditions (the key word "no_event" is used in this case). This particular event plays a significant role when modelling the impact of cascading failures in the system.

The second part describes the automaton transitions. A transition is a tuple `g |- evt -> e` where `g` is the guard of the transition, `evt` is an event name and `e` is the effect of the transition. The guard is a boolean formula over state or flow variables. It defines the configuration in which the transition is fireable if the event `evt` occurs. The effect `e` is a list of assignations of value to state variables. So the transition part describes how functioning or failure states can evolve.

The third part is a set of assertions. Assertions are atomic equalities or more structured equations using `if-then-else` or `case` construction. They establish relations between the states and the flows of the component and so, describe how component outputs are determined by component inputs and current functioning mode.

These concepts are illustrated by the following example. The component `block` has one `input`, one `output` flow ranging over the domain `{no, low, ok, max}`, one boolean internal state `ok` and one `failure` event. The transition means "if the system is `ok` and if `failure` occurs then the system is no more `ok`". The assertion means "if the system is `ok` then the `output` is equal to the `input` else the `output` value is `no`". We used here the `case` structure but we could use similarly an `if then else` structure.

```
node block
 state
  ok : boolean;
 flow
  input : {no, low, ok, max} : in;
  output :{no, low, ok, max} : out;
 event
  failure;
 trans
  ok |- failure -> ok:= false;
 assert
  output = case {ok : input,
  else : no};
edon
```

In a system model, instances of such nodes are interconnected by assertions which plug input-output flows. Hierarchy of nodes can be used to build complex components and structure the system model.

Case-Studies Modelling: The main step prior to model a system is to collect information on it (e.g. architecture, failure modes). We particularly paid attention to Airbus Functional Hazard Assessment document performed on aircraft functions that describes the failure conditions, effects and severity levels (i.e. catastrophic, hazardous, major or minor) and to the System Safety Assessment which demonstrates that safety objectives are met. In this section we

describe how to model a system using these documents as inputs and use the example of the hydraulic pipe as an illustration.

Failure Modes: Failure modes that could cause the loss of energy supply were modelled. We considered that all components could fail to generate, transmit or deliver energy. We also supposed that leaks could occur in pipes. Finally, blocked positions for valves and PTU were also considered. Table 1, hereunder, sums up the failure modes considered in our component libraries.

Components	Failure modes
Pipe	Leakage
Reservoir	Failed, leakage
Pump	Failed, overheat
Valve	Failed, stucked
Consumer	Failed, leakage
PTU	Failed, stucked

Table 1 − Failure Modes of the hydraulic components

Failure Propagation: We have to know what kinds of information are exchanged between the components and how failures will be propagated through pipes. This information is really specific to each system. If we consider a hydraulic circuit pipe we cannot model a leakage only by considering the absence or the presence of fluid in the pipe. Indeed, the real consequence of a leakage is a sudden pressure decrease for all the components located downwards the faulty component and, at last, a lack of fluid in the circuit. As a result, a pipe must transmit the couple fluid/pressure in order to take into account and to correctly propagate the leakage information throughout the model. Moreover, as all the components (i.e. downwards but also upwards) have to be informed of such a failure, the fluid/pressure signal has to be bidirectional.

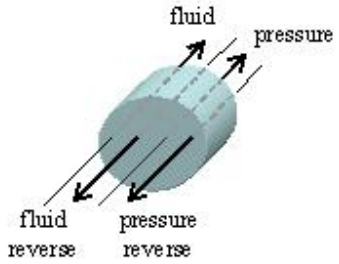

Figure 3 − Pipe section

In the component models, failure propagation will be modelled by assertions that constrain the values of flow variables. The following example shows in details the failure propagation in the pipe model.

Example:

```
node pipe
 flow
  output_pressure : {max,ok,low,no} : out;
  output_fluid : {yes,low,no} : out;
  output_pressure_reverse_info : {max,ok,low,no} : in;
  output_fluid_reverse_info : {yes,low,no} : in;
  input_pressure : {max,ok,low,no} : in;
  input_fluid : {yes,low,no} : in;
  input_pressure_reverse_info : {max,ok,low,no} : out;
  input_fluid_reverse_info : {yes,low,no} : out;
 state
  state_ : {ok,leakage};
 event
  leak;
 trans
  (state_ = ok) |- leak -> state_ := leakage;
```

```
assert
 output_fluid = (case {
 (state_ = ok) : input_fluid,
 (state_ = leakage) and ((input_fluid = yes) or (input_fluid = low)) : low,
 else no}),
 input_fluid_reverse_info = (case {
 (state_ = ok) : output_fluid_reverse_info,
 (state_ = leakage) and ((input_fluid = yes) or (input_fluid = low)) : low,
 else no}),
 output_pressure = (case {
 (state_ = ok) and not((input_fluid = no)) : input_pressure,
 else no}),
 input_pressure_reverse_info = (case {
 (state_ = ok) and not((input_fluid = no)) : output_pressure_reverse_info,
 else max});
init
 state_ := ok;
edon
```

When a pipe is not leaking, output pressures and output fluid levels are equal to input ones. When a leak occurs, the fluid level decreases from `yes` to `low` until the reservoir is empty (the input fluid level is `no`). Moreover, while the pipe is not empty (fluid different from `no`), the leak increases the upwards pressure and decreases the downwards one.

In Cecilia OCAS workshop, each node is associated to an icon and belongs to a library. Once the component library created, the system is easily and quickly modelled. Components are dragged and dropped from the library to the system architecture sheet and then linked graphically. The whole hydraulic system model is made of about 15 component classes.

Safety Assessment Techniques

Formal Safety Requirements: As stated in the case-study presentation section, the main safety requirement for the hydraulic system is: "*Total loss of hydraulic power is classified catastrophic*". We also considered two related requirements: "*Loss of two hydraulic channels is classified major*", "*Loss of one hydraulic channel is classified minor*". We associate with this set of safety requirements three qualitative requirements of the form "*if up to N individual failures occur then the loss of N+1 power channels of hydraulic system shall not occur*" with N = 0,1,2.

To model these qualitative requirements we first have to model the loss of N+1 power channels. Let N = 2, so we consider the total loss of hydraulic power. A first approach consists in using propositional formula 3_Hyd_Loss that would be true whenever the value of flow output_pressure of the distribution lines of the three hydraulic channels is equal to no**.** But this formula fails to adequately describe the failure condition. It could hold in evolutions of the system during a small period of time and then it would no longer hold as the hydraulic power is recovered due to appropriate activation of a backup such as the RAT for instance. The correct description of the failure condition should model the fact that hydraulic power is definitively lost. Hence we use Linear Temporal Logic (ref. 5) operators to model a failure condition. The following temporal formula models the permanent loss of hydraulic power:

```
Permanent_3_Hyd_Loss: F G 3_Hyd_Loss
```

where F is the eventually (or Finally) operator, G is the always (or Globally) operator. Formula Permanent_3_Hyd_Loss can be read "*eventually Hydraulic power is totally lost in all future time steps*". So the general form of qualitative requirements we check is:

```
No_N+1_S_Loss: G upto_N_failures -> ~ F G N+1_S_Loss
```

with N = 0,1,2 and `upto_N_failures` is a property that holds in all states of a system such that up to N individual failures have occurred.

Graphical Interactive Simulation: A Safety Engineer can check the effect of failure occurrences on the system architecture using Cecilia OCAS graphical interactive simulator. The system architecture is depicted by a set of interconnected boxes that represent nodes of the AltaRica model. Icons are associated with a node state. For

instance, a green box is displayed if a distribution line delivers power and a red box is displayed otherwise. These icons help to rapidly assess the component current state.

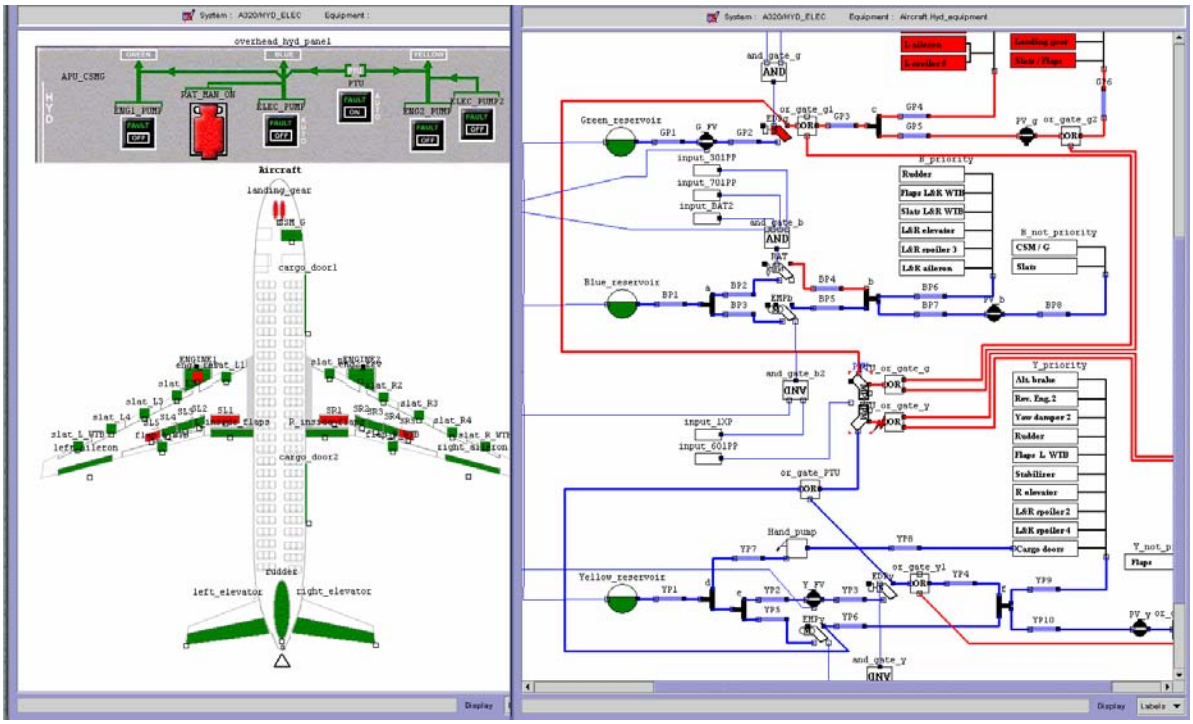

Figure 4 − Cecilia OCAS Graphical Simulator

To observe more complex situation such as the loss of several channels, special nodes called "observers" are added into the model. An observer internal state only depends on the value of other components outputs.
First, the simulator computes the initial state. Then, when the safety engineer selects a node the simulator proposes the set of events that can be performed at this step. This is the set of events with a guard that is true in the current state. The safety engineer chooses an event and the resulting state is computed by the simulator. As failures are events in the AltaRica model, the safety engineer can inject several failure events into the model in order to observe whether a failure condition is reached (such as loss of one or several power channels).

Figure 4 shows the graphical user interface of Cecilia OCAS. The Hydraulic system is displayed in the right window. All basic icons represent a component (tank, pump, distribution line …) of this system. The left window displays a set of observers that show whether aircraft devices powered by the hydraulic system are available or not. At the top of this window, we designed a control panel similar to the aircraft panel with button components that are used to activate or inactivate components in the hydraulic system.

Model-checking: A model-checker as Cadence Labs SMV (ref. 6) performs symbolically an exhaustive simulation of a finite-state model. The model-checker can test whether the qualitative requirements stated as temporal logic formulae are valid in any state of the model. Whenever a formula is not valid, the model-checker produces a counter-example that gives a sequence of states that lead to a violation of the safety requirement.

We developed tools to translate a model written in Altarica into a finite-state SMV model. Thus, we were able to check that both system models enforced their qualitative safety requirements. All requirements were verified in less than ten seconds although the truth value of some formulae depended in each state on as much as 100 boolean variables.

Applicability of the approach to robotic systems

Applicability scope of the approach with respect to traditional safety assessment process: The kinds of models and analysis presented before are devoted to a pivot step of the safety assessment process. Former analysis aim at identifying and classifying the failure conditions according to their criticality. The pivot step is used to demonstrate that a system architecture enable to meet safety requirements. Following steps deal with the verification of the hypothesis used to build the pivot models: are the considered failure modes the good ones? For quantitative part of the analysis, how representative are the considered failure rates? The verification process depends on the kind of system components. In the simplest case, it may consist in checking the compliance of the hypothesis with available data base. When considering software based component, it requires testing the software more or less heavily according to its criticality. Formal assessment techniques like the one presented before were primarily defined to perform such software verification. Nevertheless their application is today more or less successful according to the software complexity. The interested reader may consult for instance ref. 7 and ref 8 for further details.

The case of robotic systems: In the following, we focus on the applicability of the pivot step previously detailed to robotic systems. According to our understanding, such systems consist in a physical devices (sensors, actuators, …) monitored and controlled by embedded software that are structured in a more or less sophisticated control architecture.
We insisted in section 2 on building system models at the right granularity level with respect to the analysis purpose. The selected granularity level identifies the main functions provided by basic component and highlights how the quality of the function outputs depend on the quality of the function inputs, on the current (faulty or nominal) function modes and on protections inserted in the component. Such generic principles can be applied to model and assess simplest robot control architectures against safety requirements.
In most sophisticated architectures where plans are computed on board, the applicability is not straightforward but seems still possible. In such architecture, a part of the policy used to control the robot may be implicitly defined by a plan generation module in order to cope with a numerous number of procedures. In this context, it is useless to enter into the details of the planning algorithms since one is interested only in finding dependencies between components that propagate failure. Nevertheless, these dependencies can be numerous depending on the combination of use of basic components generated by the planner. We already met a similar case when studying a highly reconfigurable aircraft electrical system (ref. 9). In such a case, one can left open the control of the devices piloted by plan. The analysis enables to find out bad plans (with our without combination of failures) that lead to critical situations. The proposal is to derive new safety requirements to avoid the generation of such plans. Then, the analysis can be conducted under these new requirements.

Conclusion and Future Work

Our experiment about traditional embedded safety critical systems shows that safety system modelling and analysis are possible and fruitful using a formal approach provided that models have the right level of detail. We have to observe that our models should not confine to failure propagation related with the functional analysis. They should also include failure propagations that could be related to system-level risks as specification errors, assumptions, synergistic considerations through-out the life-cycle, energy effects, …Previous works such as references 10 and 11 present modelling approaches that, as ours, abstract nominal physical details and focus on failure propagation. However, the author main goal was to generate fault trees, so their models focus on system architecture and do not enable temporal analysis of highly dynamic reconfiguration mechanisms. Following our approach, we could state formally interesting qualitative and temporal safety requirements of aircraft systems and perform assessment analysis with interactive simulation and model-checking tools without performance problems.

Nevertheless, as discussed in section 4, it is worth noting that this fruitful approach assists only one specific step of the safety assessment process. Moreover, if our paper gives a flavour of the use of model-checking techniques for a specific purpose, it does not pretend to give a comprehensive view of the available formal techniques and their uses. For instance, model-checking techniques are also used to generate plan in the robotic field (e.g. ref 12) or to validate the model for some model based planners. Future works at ONERA intends to build such a more comprehensive approach of the safety analysis for autonomous systems and more specifically unmanned vehicle like. Such a step is mandatory if one wants to use more widely drone submitted to stringent regulations.

Acknowledgement

The experiment described in this paper has been developed within the ISAAC Project, a European sponsored project, FP6-2002-Aero-1-501848.
Thank you to CNES, LAAS and ONERA colleagues of AGATA project for fruitful discussion.

References

1. O. Akerlund, P. Bieber, E. Boede, M. Bozzano, M. Bretschneider, C. Castel, A. Cavallo, M. Cifaldi, J. Gauthier, O. Lisagor, A. Lüdtke, S. Metge, C. Papadopoulos, T. Peikenkamp, L. Sagaspe, C. Seguin, H. Trivedi and L. Valacca, ISAAC, a framework for integrated safety analysis of functional, geometrical and human aspects, proceedings Embedded Real Time Software 2006, Toulouse (France), Janvier 2006.

2. A. Arnold, A. Griffault, G. Point, A. Rauzy. The AltaRica formalism for describing concurrent systems. Fundamenta Informaticae n°40, p109-124, 2000.

3. Manuel utilisateur OCAS V3.2, Dassault Aviation, 2005.

4. A. Rauzy. Mode automata and their compilation into fault trees. Reliability Engineering and System Safety, 2002.

5. Z. Manna, A. Pnuelli. The temporal logic of reactive and concurrent systems. Springer − Verlag, New-York, 1992, ISBN 0-387-97664-7.

6. K.L. MacMILLAN. Symbolic Model Checking. Kluwer Academic Publishers, 1993, ISBN 0-7923-9380-5.

7. Patrick Cousot, Radhia Cousot, Jérôme Feret, Antoine Miné, David Monniaux, Laurent Mauborgne, Xavier Rival.
The ASTRÉE Analyzer. ESOP 2005: The European Symposium on Programming, Edinburgh, Scotland, April 2—10, 2005. Lecture Notes in Computer Science 3444, © Springer, Berlin, pp. 21—30.

8. Dirk Beyer, Thomas A. Henzinger, Ranjit Jhala, and Rupak Majumdar. The Software Model Checker Blast: Applications to Software Engineering. *Int. Journal on Software Tools for Technology Transfer*, Invited to special issue of selected papers from FASE 2005.

9. Ch. Kehren, *et al*, Advanced Simulation Capabilities for Multi-Systems with AltaRica, in Proceedings of the 22nd International System Safety Conference (ISSC), 2004, System Safety Society.

10. Y. Papadopoulos, M. Maruhn. Model-based automated synthesis of fault trees from Matlab-Simulink models. DSN2001, Int. Conf. on Dependable Systems and Networks (former FTCS), Gothenburg, Sweden, pages 77-82, ISBN 0-7695-1101-5, July 2001.

11. Fenelon, P., McDermid, J.A., Nicholson, M. & Pumfrey, D.J. Towards Integrated Safety Analysis and Design, ACM Computing Reviews, Vol. 2, No. 1, p. 21-32, 1994.

12. A. Cimatti, M. Pistore, M. Roveri, P. Traverso. Weak, Strong, and Strong Cyclic Planning via Symbolic Model Checking , *Artificial Intelligence* , 147 (1-2):35-84, 2003. Elsevier Science publisher