Combination of Fault Tree Analysis and Model Checking for Safety Assessment of Complex System

Pierre Bieber, Charles Castel, Christel Seguin

ONERA-CERT,
2 av. E. Belin,
31055 Toulouse Cedex - France
{bieber, castel, seguin}@cert.fr

Abstract. Safety assessment of complex systems traditionally requires the combination of various results derived from various models. The Altarica language was designed to formally specify the behaviour of systems when faults occurs. A unique Altarica model can be assessed by means of complementary tools such as fault tree generator and model-checker. This paper reports how the Altarica language was used to model a system in the style of the hydraulic system of the Airbus A320 aircraft family. It presents how fault tree generation and model-checking can be used separately then combined to assess safety requirements.

Introduction

Safety assessment of aircraft systems according to standard as ARP 4754 [12] requires the combination of various results derived from various models. For instance, on one hand safety engineers build fault tree in order to compute minimal cut sets and occurrence probabilities of failure conditions. On the other hand, system designers will use specific languages such as SCADE [15], SABER [3], Simulink [9] to model the system under study in order to perform simulations. Such an amount of modeling languages and models is not easily shared by the different teams involved in the system design and assessment. Moreover, this heterogeneity does not facilitate the result integration.

The Altarica [1] language was designed by University of Bordeaux and a set of industrial partners to overcome such difficulties. Altarica models formally specify the behavior of systems when faults occurs. These models can be assessed by means of complementary tools such as fault tree generator and model-checker.

One issue is now to integrate such a practice into existing safety processes. When can such models be used fruitfully ? At early design stages, in order to assess the first drafts of a system architecture? In latest phases, in order to assess a detailed design? What are the benefits of the available assessment techniques?

We carried out some experiments with the Altarica language and tools in order to assess the validity of some possible answers. We study more specifically the hydraulic system of the Airbus A320 at early stage of design. In this paper, we

report how we use the Altarica language to model this system and we present how fault tree generation and model-checking can be used separately then combined to assess safety requirements.

1 Case-study : AIRBUS A320 Hydraulic System

1.1 System description

The role of the hydraulic system is to supply hydraulic power with an adequate safety level both to devices which ensure aircraft control in flight (servocontrols, flaps, ...) and to devices which are used on ground (landing gear, braking systems, ...). In this paper we assume that the hydraulic system is only made of pumps and distribution lines which generate and transmit the hydraulic power to the devices. The actual hydraulic system contains other types of components that we do not consider here such as tanks, valves, gauges, ... There are three kinds of pumps : Electric Motor Pump (EMP) that are powered by the electric system, Engine Driven Pumps (EDP) that are powered by the two aircraft engines and one RAT pump that is powered by the Ram Air Turbine. So some pumps (namely the EDPs) cannot be used on ground when aircraft engines are shut down and the RAT use requires that the aircraft speed is high enough.

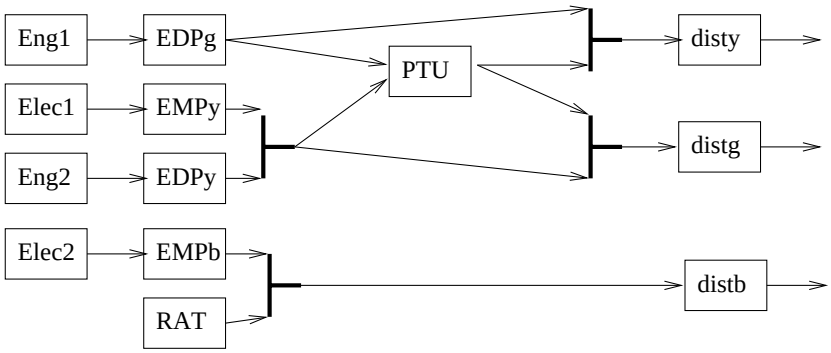

Fig. 1. Hydraulic system structure

To meet safety requirements, this system contains a physical sub-system that is composed of three hydraulic channels : Green, Blue and Yellow. The Blue channel is made of one electric pump EMPb, one RAT pump and a distribution line distb. The Green system is made of one pump driven by engine 1 EDPg and a distribution line distribg. The Yellow system is made of one pump driven by engine 2 EDPy, one electric pump EMPy and a distribution line distriby. Moreover a power transfer unit PTU opens a transmission from green hydraulic power to yellow distribution line and vice versa as soon as the differential pressure between both system is higher than a given threshold.

The physical sub-system is controlled by crew actions and reconfiguration logics that activate the various pumps. The RAT is automatically activated in flight when the speed of the aircraft is greater than 100 knts and 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. In the following, we suppose that all other pumps (EDPy, EDPg), the PTU and distribution lines are always activated

1.2 Safety requirements

AIRBUS follows ARP 4761 [13] recommendations used to define safety assessment process related with airworthiness certification. One step of this process is the definition of a safety requirement document that contains requirements derived from the Aircraft level FHA (Functional Hazard Analysis). For the Hydraulic system we find for instance :

- TotalLoss : "*Total loss of hydraulic power is classified catastrophic*",
- Loss2 : "*Loss of two hydraulic systems is classified major*",
- Loss1 : "*Loss of one hydraulic system is classified minor*",

Quantitative requirement are associated with each of theses requirements, they are the form "*the probability of occurrence of failure condition FC shall be less than* 10^{-N} *per flight hour*" with N=9 (resp. 7 and 5)) if FC is of severity catastrophic (resp. hazardous and major). We also associate qualitative requirements of the form "*if up to N individual failures occur then failure condition FC shall not occur*", with N=2 (resp. 1 and 0) if FC is of severity catastrophic (resp. hazardous or major, minor).

2 Altarica model

2.1 Altarica language

An Altarica model of a system consists of hierarchies of components called nodes. A node gathers flows, states, events, transitions and assertions.

```
node block
 flow
  O : bool : out ;
  I, A : bool : in ;
 state
  S : bool ;
 event
  failure ;
 trans
  S |- failure -> S:= false;
 assert
```

```
  O = (I and A and S);
 extern initial_state = S = true ;
edon
```

- Flows are the visible parameters of the component, whereas states are internal variables. Let us consider a basic component called "block". A block stands for a basic energy provider. It receives two boolean inputs, `I` that is true whenever the component receives energy, and `A` that is true whenever the component is activated. It has a boolean output `O` that is true whenever it produces energy. It has the internal state `S`, which is true whenever the block is safe. Initially, we assume that `S` is true.
- Assertions are boolean formulae that state the constraints linking flows and internal states. For instance, the output`O` is true whenever the input energy is available (true), the activation order is true and the component state is safe. Assertions are system invariants: they must always be true.
- Transitions describe how the internal states may evolve. They are characterised by a guard (a boolean constraint over the component flows and state), an event name and a command part (value assignation of some state variables). For the block example, an unique transition is considered. It can be taken only if the component is safe (`S` is true) and the event failure occurs. After this transition, the component is no more safe : `S` is set to false. This transition models the only failure mode we consider in this paper, i.e. loss of a component.

The whole system node (main node) is built by connecting the basic nodes. Global assertions allow the definitions of the flow connections. For instance, a global assertion states that input flows of a node are output flows of another node. Connections may also be related to events shared by a set of nodes (synchronisation of events). This feature is not used in our example, the interested reader may find more details in [1].

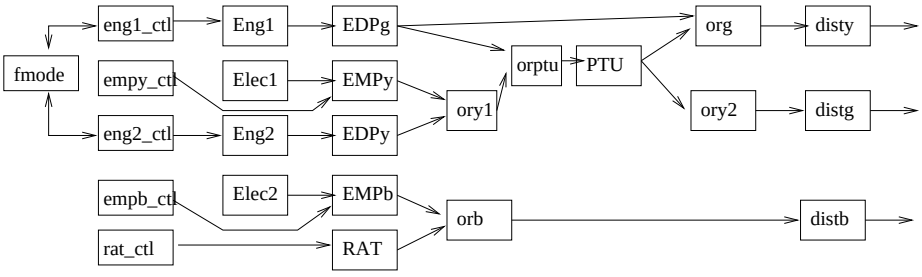

Fig. 2. A320 Hydraulic Model

2.2 Physical sub-system model

We model the various Pumps (EDPy, Edpg, EMPy, EMPb, RAT) with "block" nodes, they receive energy from the engines or electric system as input and produce hydraulic power as output. As we do not model aircraft speed we consider that the RAT is always energised. Distribution lines (disty, distg and distb) are also modelled with "block" nodes, they receive hydraulic power as input and produce hydraulic power as output. We suppose that they are always activated. To connect the outputs of two pumps on the input of a distribution line we use an "or" node that is described in the following Altarica code.

```
node or2
 flow
  O : bool : out ;
  I1, I2 : bool : in ;
assert
  O = (I1 or I2);
edon
```

The PTU component is modelled with a combination of an "or" node that merges the hydraulic power produced by yellow and green pumps and delivers it to a "block" node that is always active. So the PTU is able to deliver hydraulic power whenever it has not failed and it receives hydraulic power on I1 (green hydraulic power) or I2 (yellow hydraulic power).

In our study we did not model in great detail the behaviour of dependent systems as "*electric generation and distribution system*" or "*engine system*". We suppose that each engine acts as a basic block that is always energised. Engines are activated by pilot actions. We suppose that each electric bar (Elec1 and Elec2) acts as a basic block that is always active and is energised if at least one engine is active.

2.3 Control sub-system model

The control sub-system model contains models of pilot actions and reconfiguration logic. To control hydraulic system components the pilot or the computers need information about aircraft environment such as the current flight mode. Node `flight_mode` contains one state variable `s_on_ground` that is true when the aircraft is on ground and that is false when the aircraft is in flight. Events `take_off` and `land` change the value of `s_on_ground` variable, `take_off` guard contains the property `eng1_A and eng2_A` to model the fact that both engines should be activated before take-off. Finally, the value of the `s_on_ground` internal variable can be observed thanks to the flow `on_ground` as stated in the assertion part of the node.

```
node flight_mode
 flow
  eng1_A,eng2_A : bool : in ;
```

```
 on_ground : bool : out ;
 state
  s_on_ground : bool ;
 event
  take_off, land ;
 trans
  s_on_ground and eng1_A and eng2_A |- take_off -> s_on_ground:=false;
  not s_on_ground  |- land -> s_on_ground:=true
 assert
  (on_ground = s_on_ground);
 extern initial_state = s_on_ground = true ;
edon
```

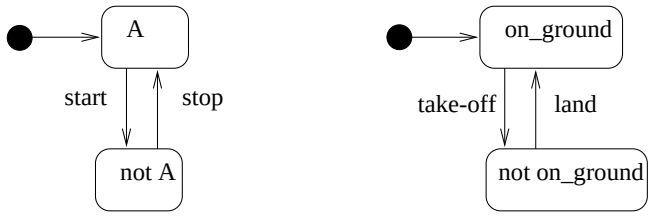

Fig. 3. Flight-mode and engine control models

An engine is controlled by `eng_cntrl` node that contains state variable `s_A` that are set to true or false according to pilot actions start and stop. We consider that an engine can only be deactivated on ground if the other engine is activated (i.e. flow `other_A` is true). As previously, flow `A` allows the observation of the corresponding internal state.

```
node eng_cntrl
 flow
  A : bool : out ;
  on_ground, other_A : bool : in ;
 state
  s_A : bool ;
 event
  start, stop ;
 trans
  not s_A |- start -> s_A :=true;
  on_ground and s_A and other_A |- stop -> s_A := false;
 assert
  (A = s_A);
 extern initial_state = s_A = true ;
edon
```

The EMPy is controlled by `empy_cntrl` node that contains variable s_A. EMPy can be deactivated in flight by pilot action `stop`, it can activated by pilot `p_start` action and it has to be activated on ground by `c_start` action.

```
node empy_cntrl
 flow
  A : bool : out ;
  on_ground : bool : in ;
 state
  s_A : bool ;
 event
  p_start,c_start, stop ;
 trans
  on_ground and not s_A |- c_start -> s_A :=true;
  not s_A |- p_start -> s_A :=true;
  (not on_ground) and s_A |- stop -> s_A := false;
 assert
  (A = s_A);
 extern initial_state = s_A = true ;
edon
```

There exists two other nodes `empb_cntrl` and `rat_cntrl` that control respectively EMPb and RATb.

2.4 Safety requirement model

To model safety requirements described in the previous chapter we should first model failure conditions such as "Total loss of hydraulic power". This means that the three distribution lines (green, yellow and blue) output is equal to false. A first approach consists in using formula :

$TotalLoss : \neg distb.O \wedge \neg distg.O \wedge \neg disty.O$.

But this formula fails to describe adequately the failure condition because it could hold in evolutions of the system during a small period 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 the hydraulic system is lost permanently or during a period exceeding some allowed amount of time. Hence we use Linear Temporal Logic (see [16]) operators to model a failure condition. For instance, we could use the two following temporal formulae :

$PermanentLoss : \Diamond\Box Totalloss$

$2TimeStepsLoss : \Diamond(TotalLoss \wedge \circ TotalLoss)$

where $\Diamond$ is the Future operator, $\Box$ is the always operator and $\circ$ is the next operator.

The semantics ot temporal formulae such as $\Diamond p$, $\Box p$, $\circ p$ can be understood by studying the truth value of p on a sequence of states. Sequence 1 of Figure 4 satisfies formula $\circ p$ in the initial state as p is true in the next state. Sequence

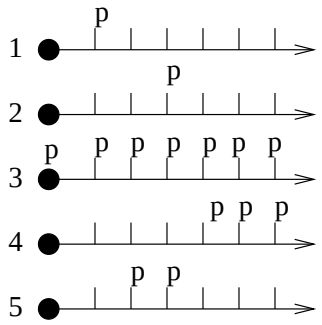

Fig. 4. Linear Temporal Logic models

2 of Figure 4 satisfies formula $\Diamond p$ in the initial state as there is a state such as p is true in this state. Sequence 3 of Figure 4 satisfies formula $\Box p$ in the initial state as p is true in any state. Sequence 4 of Figure 4 satisfies formula $\Diamond\Box p$ in the initial state as there exist a state such p is true in any subsequent state. Sequence 5 of Figure 4 satisfies formula $\Diamond(p \wedge \circ p$ in the initial state as there exist a state such p is true and p is also true in the subsequent state.

Formula $PermanentLoss$ models the permanent loss of the hydraulic system it can be read "*eventually the hydraulic power is lost in all future time steps*". Whereas the second formula models the inability to recover hydraulic power in one time step, it could be read "*eventually the hydraulic power is lost during two consecutive time steps*".

So the general form of safety requirements we consider is
$NoPermanentLoss : \Box N\_failures \Rightarrow \neg\Diamond\Box fc$
or is
$No2TimeStepsLoss : \Box N\_failures \Rightarrow \neg\Diamond(fc \wedge \circ fc)$
where fc denotes a non-temporal failure condition formula as TotalLoss and $N\_failures$ is a property that holds in all states of a system such that up to N component failures have occurred. According to the rules of Linear Temporal Logic, these formulae are equivalent to :
$No2TimeStepsLoss : \Box N\_failures \Rightarrow \Box(fc \Rightarrow \circ\neg fc)$
$NoPermanentLoss : \Box N\_failures \Rightarrow \Box\Diamond\neg fc$

3 Safety Assessment

We propose to apply a three steps approach to perform safety assessments based on Altarica models:

1. Generate a fault-tree of the failure condition of interest. The generation is based on the physical sub-system model and the failure condition considered is non-temporal. Analyse the fault-tree in order to extract a set of unexpected failure and activation combinations.

2. Define and check the activation interface. This is a set of properties that eliminate the unexpected combinations found at the previous stage. Use a model-checker to verify that activation properties are enforced by the control sub-system model.
3. Compose the various results in order to prove that safety requirements are met by the global system. This step is performed thanks to a set of inference rules.

In the following sections we show how to use this approach to check safety requirements of A320 hydraulic systems.

3.1 Fault-tree generation and analysis

Altarica fault-tree generator (see [14]) takes an Altarica model as input and an unexpected event, then it generates a fault-tree describing the failure situations that lead to the unexpected event. We use this tool to generate a fault-tree for the non-temporal failure condition (i.e. failure condition without temporal operators) using the Altarica model of the physical sub-system.

Once the fault-tree is generated, we use ARALIA (see [4]) fault-tree analyser to compute the set of prime implicants of the non-temporal failure condition. This gives us a set of combinations of the form :

$\{c1.failure, c2.failure, ..., \neg c'1.A, \neg c'2.A, ...\}$

where $c1.failure$ means that component c1 has failed and $\neg c'1.A$ means that $c'1$ is not activated. Each combination implies that the failure conditions occurs. A combination related to failure condition FC is unexpected if it contains less than N failure with N = 2 (resp. 1, 0) if FC is of severity Catastrophic (resp. Hazardous or Major, Minor).

We compute the prime implicants for formula $\neg distb.O \wedge \neg distg.O \wedge \neg disty.O$ that represent the instantaneous total loss of hydraulic system. There are 30 unexpected combinations (i.e. combinations with less than 3 failures) in this set. One combination contains no failure at all : $\{\neg Eng1.A, \neg Eng2.A, \neg RATb.A\}$ that means that when both engines are stopped and the RAT is not activated the three hydraulic systems cannot produce hydraulic power. Six combinations contain only one failure :

- $\{\neg Eng1.A, \neg Eng2.A, RATb.failure\}$,
  $\{\neg Eng1.A, \neg Eng2.A, distb.failure\}$,
  $\{\neg Eng1.A, Eng2.failure, \neg RATb.A\}$,
  $\{Eng1.failure, \neg Eng2.A, \neg RATb.A\}$
- $\{EDPg.failure, \neg EMPb.A, \neg EMPy.A, \neg Eng2.A, \neg RATb.A\}$,
  $\{EDPy.failure, \neg EMPb.A, \neg EMPy.A, \neg Eng1.A, \neg RATb.A\}$

The four first combinations represent situations where both engines are not working (because they are stopped or have failed) and the RAT is not working or the blue distribution line has failed. The two last combinations represent situations where one engine driven pump has failed and the other engine is stopped

and electric pumps and RAT are not activated. The 23 remaining combinations contain two failures.

An alternative way to define unexpected combinations can be followed if failure occurrence probabilities are known for each individual components. In this case, we would set the probability of event $c.A$ to 0 and use ARALIA to compute the probability of each combination. Then we would consider as unexpected combination any prime implicant with a probability greater than the probability objective associated with the failure condition.

3.2 Interface Definition and Model-checking

All the unexpected combinations found at the previous stage contain members of the form $\neg c.A$ meaning that component c has not been activated. Our basic assumptions is that it is the goal of the control sub-system to activate the physical component in a adequate order to avoid the occurrence of unexpected combinations. We propose to define an activation interface, this is a set of properties on $c.A$ variables that eliminate the unexpected combinations. The control sub-system should guarantee these properties.

Finding a good set of activation properties can be difficult. One option is to consider that all components are always active, this trivially eliminates combinations including a member of the form $\neg c.A$. This option does not reflect the actual system where some components are initially stopped and are started when they are needed. Another option is to construct an activation property of the form $\neg unex\_comb$, where $unex\_comb$ is the disjunction of all conjuncts representing an unexpected combination. The conjunct representing combination $\{c1.failure, c2.failure, ..., \neg c'1.A, \neg c'2.A, ...\}$ would be equal to $c1.failure \wedge c2.failure \wedge ... \wedge \neg c'1.A \wedge \neg c'2.A \wedge ...$. Once again, showing that this activation predicate eliminates the unexpected combinations is trivial. The main benefit of this option is that such an activation property could be automatically generated by a tool. But the major drawback of this option is that it does not make a difference between activation properties that should be satisfied by the engine control, the pump control or the RAT control. So we prefer a third option that consists in defining activation properties that are related to one component (EMPy, EMPb, RAT) or to a small set of components (Eng1 and Eng2).

The activation property for the hydraulic system act_ok is defined by the conjunction $act\_engine \wedge act\_EMPy \wedge act\_EMPb \wedge act\_RATb$. Activation properties are of the form $act\_cnd \Rightarrow c.A$: "*whenever act_cnd is true component c is activated*".

The activation condition of EMPb is that the aircraft is in flight or it is on ground and at least one engine is active.

$act\_EMPb : (\neg on\_ground \vee (on\_ground \wedge (Eng2.A \vee Eng1.A))) \Rightarrow EMPb.A$

The activation condition of RATb is that the aircraft is in flight and both engines are lost.

$act\_RATb : (\neg on\_ground \wedge \neg Eng1.O \wedge \neg Eng2.O) \Rightarrow RATb.A$

The activation condition of EMPy is that the aircraft is on ground.

$act\_EMPy : on\_ground \Rightarrow EMPy.A$

The activation condition of both engines is that the aircraft is in flight.
$act\_engine : \neg on\_ground \Rightarrow (Eng1.A \wedge Eng2.A)$;

We use a model-checker to prove that all unexpected combinations are eliminated by the actions of the control sub-system. Altarica is connected to the MEC model checker [2]. We did not use this tool because it has strong limitations on the size of systems that it can handle, a new version of the MEC model-checker that should correct these limitations is under development at University of Bordeaux. In the meanwhile, we used Cadence Lab SMV model-checker [10]. Altarica models are automatically translated into SMV input language. We first prove that activation properties eliminate all unexpected combinations, then we show that activation properties are enforced by the control sub-system model.

Elimination of unexpected combinations should be proved by establishing that the following formula is valid: $\Box(act\_ok \Rightarrow \neg unex\_comb)$. Our first attempt to prove this formula failed, because it was possible that both engines were stopped on ground, and as the aircraft was on ground the RAT would not be activated. So unexpected combination $\{\neg Eng1.A, \neg Eng2.A, \neg RATb.A\}$ could not be not eliminated. To eliminate this combination we have to assume that at least one engine is active, this was established by adding in the guard of the *eng_cntrl.stop* action aconstraint related to the status of the other engine.

To check that activation properties are enforced we first attempted to prove that $\Box(act\_ok)$ is valid (i.e. at every time step activation properties hold). But, we failed to prove this property as, due a pilot or automatic logic reaction delay, EMPy can only be started one time-step after the aircraft is on ground. So the correct verification goal is :
$\Box(\neg act\_ok \Rightarrow \circ act\_ok))$ i.e. at every time step, if the activation properties do not hold then they hold at the next time step.

3.3 Result synthesis

Finally we have to combine the various results we have obtained so far in order to prove that the hydraulic system meets its safety requirements. On one hand we have checked using fault-tree analysis tools that, under the assumption that the unexpected combinations are eliminated, the physical sub-system enforces the non-temporal safety requirements. On the other hand we have proved using a model-checker that the activation properties eliminate the unexpected combinations and the control sub-system guarantees that the activation properties hold.

The synthesis of these verification results is performed thanks to a set of inference rules that preserve the validity of Linear Temporal Logic formulae (see [16]):

- **Necessitation rule** : if non-temporal formula f is valid in any state of the system then $\Box f$ is also valid,
- **Strengthening rule** : if formulae $\Box(g_0 \Rightarrow (f_1 \Rightarrow f_2))$ and $\Box(f_0 \Rightarrow g_0)$ are both valid then formula $\Box(f_0 \Rightarrow (f_1 \Rightarrow f_2))$ is also valid

– **Next rule** : if formulae $\Box(f_0 \Rightarrow (f_1 \Rightarrow f_2))$ and $\Box(\neg f_1 \Rightarrow \circ f_1)$ are both valid then formula $\Box f_0 \Rightarrow \Box(\neg f_2 \Rightarrow \circ f_2)$ is also valid.

We consider that the fault-tree analysis provides a proof that the following non-temporal formula is valid
$\neg unex\_comb \Rightarrow (2\_failures \Rightarrow \neg TotalLoss)$

So, by applying Necessitation rule, we derive that the following formula is valid:
$\Box(\neg unex\_comb \Rightarrow (2\_failure \Rightarrow \neg TotalLoss))$

By model-checking we have shown the validity of:
$\Box(act\_ok \Rightarrow \neg unex\_comb)$ So by applying Strengthening rule, we obtain the validity of:
$\Box(act\_ok \Rightarrow (2\_failures \Rightarrow \neg TotalLoss))$

This equivalent to formula :
$\Box(2\_failures \Rightarrow (act\_ok \Rightarrow \neg TotalLoss))$

Finally, we also proved by model-checking the validity of:
$\Box(\neg act\_ok \Rightarrow \circ act\_ok)$

By applying Next rule, we show the validity of :
$\Box 2\_failures \Rightarrow \Box(Totaloss \Rightarrow \circ\neg TotalLoss))$

Hence we can conclude that the safety requirement is guaranteed by the hydraulic system.

4 Concluding Remarks

High level languages dedicated to safety analysis usually deal either with static hierarchical view of complex systems, in order to support fault tree analysis (see [8, 7]) or with the dynamic part (see [5] for instance). Altarica language provides high level concepts that allow concise descriptions of both static and dynamic parts of complex systems. We shown how we used these features to model the "static physical part" and the "dynamic control part" of the hydraulic system of the A320 Aircraft.

We explained how each part can be analysed by the most adapted tools, namely a fault tree generator for the static part and a model-checker for the dynamic one. This allows to overcome the limitations of the existing tools. On one hand, the available fault tree generator computes automatically classical static fault trees, that gather all causes of an unexpected event but that do not distinguish the event order. On the other, model-checkers can tackle controllers and event order. Another possibility (see [11] for instance) is to use the model-checker to interactively generate fault-tree. When a model-checker cannot check a formula, it usually computes a counterexamples that shows a scenario leading to a state where the formula does not hold. So a model-checker could be used to find scenarios that are counter-example of the negation of a temporal failure condition. However, the use of a model-checker does not guarantee that all scenarios that cause a failure condition will be found (this problem is studied in ESACS project [6]).

Finally, the proposed approach seems adapted to the assessment of the early definition of a system architecture. Indeed, physical components are first identified during the architecture design. Knowing the safety requirements and the physical features of the architecture, fault tree analysis allows the derivation of requirements that constraint the design of system controllers. Then the designed controllers can be assessed by model-checking. We shown the approach feasibility for the earliest system models, with poor knowledge on failure modes. The experiments are pursued to assess the approach interest and tractability for more detailed models.

Acknowledgements

This work was partially supported by EU GROWTH programme project ESACS (Enhanced Safety Assessment of Complex Systems).

References

1. A. Griffault A. Arnold, G. Point and A. Rauzy. The Altarica Formalism for Describing Concurrent Systems. *Fundamenta Informaticae*, 34, 1999.
2. P. Crubillé A. Arnold, D. Bégay. *Construction and Analysis of Transition Systems with MEC*. World Scientific Publishers, 1994.
3. Analogy. About saber mixed-signal simulator. "http://www.analogy.com/products/simulation/about_saber.htm".
4. Groupe ARALIA. Computation of prime implicants of a fault tree within aralia. *Reliability Engineering and System Safety*, 1996. Special issue on selected papers from ESREL'95.
5. G.E. Apostolakis C.J. Garrett, S.B. Guarro. The dynamic flowgraph methodology for assessing the dependability of embedded software systems. *IEEE systems, man and cybernetics*, 25(5), 1994.
6. ESACS Consortium. Enhanced Safety Assessment of Complex Systems. "http://www.cert.fr/esacs".
7. P. Fenelon. Towards integrated safety analysis and design. *ACM Applied Computing Review*, 1994.
8. P. Fenelon and J. MacDermid. Integrated techniques for software safety analysis. In *IEE colloquium on Hazard Analysis*, 1992.
9. MathWorks. The mathworks - simulink. "www.mathworks.com/products/simulink/".
10. K.L. McMillan. *The SMV language*. Cadence Berkeley Labs, 1999.
11. G. Staalmarck O. Akerlund, S. Nadjm-Tehrani. Integration of Formal Methods into System Safety and Reliability Analysis. In *17th International System Safety Conference*, 1999.
12. Society of Automotive Engineers. Aerospace recommended practice (arp) 4754,certification considerations for highly integrated or complex aircraft systems.
13. Society of Automotive Engineers. Aerospace recommended practice (arp) 4754,guidelines and methodsfor conducting the safety assessment procees on cibil airborne systems and equipment.
14. A. Rauzy. Modes automata and their compilation into fault trees. *Reliability Engineering and System Safety*, 2002.

15. Esterel Technologies. Scade suite for safety-critical software development. "www.esterel-technologies.com/scade/".
16. A. Pnueli Z. Manna. *Temporal Verification of Reactive Systems - Safety*. Springer Verlag, 1995.