PRELIMINARY DESIGN OF FUTURE RECONFIGURABLE IMA PLATFORMS - SAFETY ASSESSMENT

Pierre Bieber[1], Julien Brunel[1] , Eric Noulard[1] , Claire Pagetti[1] , Thierry Planche[2] , François Vialard[2]
[1] ONERA, Toulouse, France, [2] Airbus France, Toulouse, France

Keywords: *IMA, reconfiguration, safety assessment*

Abstract

The next generation of IMA platforms should include reconfiguration capabilities in order to limit the effect of hardware failures on aircraft operational reliability. In this paper, we describe the safety assessment of a preliminary design of a reconfigurable IMA platform. The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7 / 2007-2013) under Grant Agreement n° ACP7-GA-2008-211439.

1 Introduction

The trend for modern military and civilian aircrafts is to support aircraft applications with an Integrated Modular Avionics (IMA) platform. The current generation of IMA (IMA-1G) replaces separate and dissimilar equipments, with fewer common processing modules, sharing the necessary communication links. The number of processing units in the A380 is half that of previous generations. Reductions in airline operating costs are expected to be significant. Indeed, the decrease of the number of computers and cables (for power supply or communication) contributes to the reduction of the aircraft weight that leads to a better fuel consumption efficiency. The reduction of the number of types of equipments will help the airline to buy and store less types of spare parts, this should lead to maintenance savings.

A typical IMA platform is described in Fig. 1. Its hardware architecture is made of a set of computing modules (numbered 1 to 5) that are connected to communication switches (labelled S1 to S4). Computation Processing Modules (CPM) are grouped into clusters, such that all computing modules in a cluster are connected to the same communication switch.

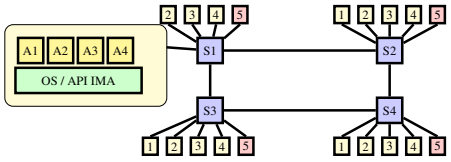

Fig. 1 Architecture of Reconfigurable IMA Platform

Avionics applications (labelled A1 to A4) are hosted in the partitions running on the computing modules. Data flows exchanged by applications hosted on different computing modules are transmitted through communication switch paths that connect the two computing modules.

1.1 Reconfiguration purposes

The SCARLETT project[1] funded by the European Union and led by Thales as a coordinator is currently studying the next generation IMA. Among other objectives, the project aims at adding reconfiguration capabilities to the IMA-1G platform. Fig. 1 shows in red the new spare computing modules that are added to the platform architecture. Reconfigurable IMA should

[1] http://www.scarlettproject.eu/, Point of contact: didier.hainaut@fr.thalesgroup.com

be able to change the configuration of the platform by moving applications hosted on a faulty computing module to spare computing modules. The main goals to be achieved by the reconfigurable IMA platform are:

1. *Improve the operational reliability of the aircraft while preserving current levels of Safety.*

2. *Avoid unscheduled maintenance and associated costs.*

3. *Limit the impact of reconfiguration on certification practices and effort.*

Aircraft systems have to enforce stringent safety requirements that address the effects of failures on the life of passengers. To fulfill these requirements a minimum level of redundancy is associated with an application on the basis of the severity of the effects of its loss. For instance, three occurrences of an application managing cabin air pressure would be required because loss of cabin pressurization is catastrophic whereas no occurrence of an application managing in-flight entertainment is required as it is considered as a comfort application whose loss has no safety effects.

Operational reliability addresses the effect of failures on economical aspects of flight operations. One source of improvement is to decrease the number of flight delays or cancellations caused by faulty computing modules. Before each flight, the health status of all equipments is assessed in order to check whether for all applications the correct level of redundancy is available. If this is not the case the aircraft cannot be used (it is NOGO). It should be possible to restore the minimum level of redundancy by moving the applications running on the faulty module to a non-faulty one. This should also help to defer maintenance operations until the aircraft has reached an appropriate location.

Other works, such as [Ell97, See96, SH95, AFAL07, SSE+10], have investigated reconfigurable avionics platforms.

1.2 Safety assessment objectives and methodology

Aircraft manufacturers have to show compliance with international regulations using means that have been accepted by the certification authorities. This includes showing that safety requirements are enforced, establishing the predictability of communication and computing real-time performances and developing software and hardware according to strict development guidelines.

The safety experts must prove that the addition of any new system does not degrade the safety of the aircraft. The purpose of the paper is to formally show that the preliminary design of the reconfigurable IMA-2G satisfies its safety requirements. To reach this objective we have applied a methodology, summarized in figure 2, which is decomposed as follows:

1. **Selection of high level safety requirements**: A list of high level safety requirements that the reconfiguration mechanisms shall enforce is produced. This list would include requirements such as *the loss of reconfiguration has a minor safety effect*;

2. **Hazard Analysis (HA)**: The objective of this analysis is to study the effect of failures of elements of the reconfiguration system architecture. Practically, this architecture is represented as a set of interacting functions. For each combination made of one function and a failure mode of interest, a dysfunctional behaviour is associated with the function. Its effect on reconfiguration and on the platform is then analysed. If the effect exceeds the one expected by the safety requirement, there must be a mitigation mean so that the effect remains acceptable;

3. **Safety validation**: The goal of this step is to verify that the architecture of the reconfiguration system enforces the safety requirements. The idea is to formally model the functional and dysfunctional behaviours of the reconfigurable IMA-2G us-

ing the AltaRica language. Safety requirements are also formally described in the language as a particular component called the *observer*. From this model, sequence generation tools automatically generate failure scenarios leading to potentially unsafe situations and verify that safety requirements are satisfied.

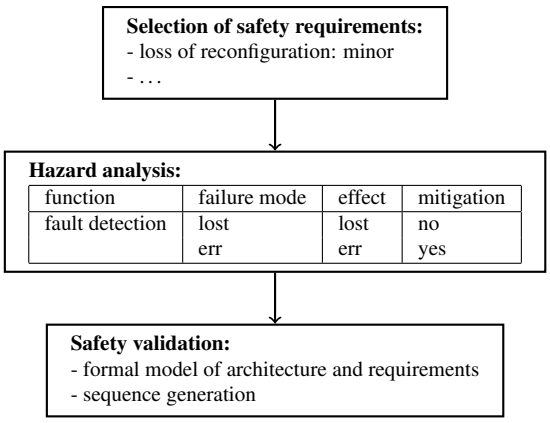

Fig. 2 Methodology for the safety assessment

1.3 Outline of the paper

In the following section, we present the reconfiguration process and architecture. Then, we describe, in section 3, safety requirements and, in section 4, hazard analysis attached to the reconfiguration system. In section 5, we prove the analysis to be correct by introducing a formal model of the architecture and by using automated sequence generation techniques.

2 Specification of the reconfiguration

The role of the reconfiguration consists in determining when a reconfiguration can occur and in performing a *correct by construction* modification of configurations in order to reach a better and safe state. When a computing module fails, a reconfiguration can be launched if this failure has an operational reliability impact, meaning that the aircraft becomes NOGO. The objective of the reconfiguration is to load the applications hosted on the failed module on a free spare module. The execution of a reconfiguration consists of a sequence of monitored steps. If any step goes wrong, the reconfiguration is stopped and the platform is left in a safe configuration.

2.1 Functional architecture

More precisely, the reconfiguration system is decomposed in several functions as shown in Fig. 3. A nominal reconfiguration is the application of the functions in the increasing order. The *reconfiguration sequencer* is a central element since it controls all other functions and verifies that each step is correct. A reconfiguration can be seen as 4 phases:

- **Failure detection phase:** The first step is the detection of a failure of a module which is made by the function *fault detection*. The *module test* applies some trouble shooting on the module to verify that there is a permanent failure. If the failure is confirmed, the *deactivate CPM* shuts down the failed module. In that way, the module does not emit any longer on the network;
- **Selection of a valid configuration:** the *configuration selector* stores a graph of validated platform configurations. Given the current configuration and the failed module, the function provides an available spare which can host the software of the failed module;
- **Execution of a reconfiguration:** this is done in three steps. The *activate SPR* function turns on the selected spare. The *load network* function updates the routing table of the switch for adding the spare. The *load SPR* function downloads the software on the spare;
- **Verification activities:** the *reconfiguration monitor* function monitors that the network and the spare download the correct configuration. This is done after each downloading step. The *configuration checker* verifies at the end of the downloading that

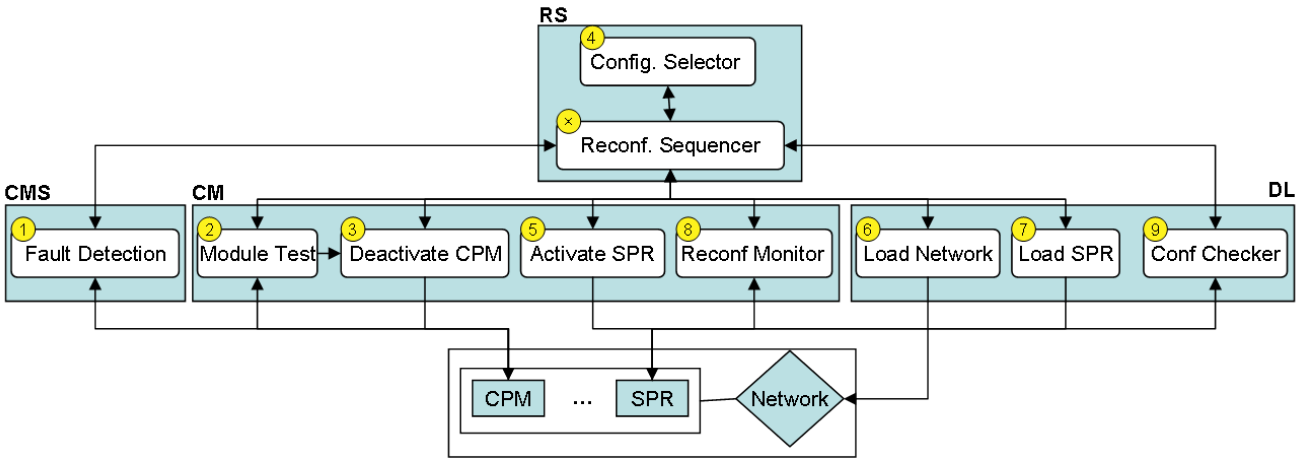

Fig. 3 Functional architecture

the spare has loaded the expected software. This is the last check and if every went fine, the reconfiguration sequencer returns to a waiting state until the next reconfiguration.

2.2 Allocation on the platform

The functions are supported by four computing modules (that are considered to be non-reconfigurable). The CMS (Centralized Maintenance System) hosts the *fault detection* functions. The RS (Reconfiguration Supervisor) hosts the *reconf. sequencer* and *conf. selector* functions. The CM (Cabinet manager) hosts the *module test*, *deactivate CPM*, *activate SPR* and *reconf. monitor*. The DL (Data loading) hosts the *load network*, *load SPR* and *conf. checker* functions.

This allocation, illustrated Fig.3, causes common failures as if a module is lost all hosted functions are lost. For this reason, we will try to avoid hosting on the same module both a function and its mitigation function.

3 Safety requirements for reconfiguration mechanisms

The nominal behaviour of SCARLETT reconfiguration is to migrate functions hosted on CPM C towards spare S whenever C is lost and the spare module is operative. We consider two main failure conditions related with reconfiguration:

- **Loss of reconfiguration**: reconfiguration is not performed when it is needed. For instance, CPM C is lost, a spare module is operative and the reconfiguration function is not able to migrate functions hosted on C towards the spare S. Functions hosted by C are lost until maintenance on C is performed.

- **Erroneous reconfiguration**: reconfiguration is performed incorrectly leading to the erroneous behaviour of functions supported by the platform. A possible scenario could be that the CPM C is lost, the reconfiguration function incorrectly migrates functions hosted on C towards a faulty spare S. Functions hosted by C could behave erroneously until the erroneous reconfiguration is detected and corrected.

The assessment of the severity of the failure conditions defined in the previous section depends of the severity of the loss or erroneous behaviour of the functions supported by the platform. Obviously, if the platform only supports functions whose loss or erroneous behaviour has no safety effect then any reconfiguration failure condition will have no safety effect. For IMA-1G, the platform supported functions whose loss or erroneous behaviour has a severity that goes from NSE (No Safety Effect) up to HAZ (Hazardous). In the forthcoming IMA-2G, it could

be the case that the platform supports functions whose loss or erroneous behaviour belongs to the CAT (Catastrophic) class.

Consequently we consider that the loss of reconfiguration is NSE. Functions hosted on the faulty CPM are lost and are not recovered after reconfiguration, hence loss of reconfiguration has no effect on these functions. And we consider that Erroneous reconfiguration is HAZ to CAT. Erroneous reconfiguration can lead to the undetected erroneous behaviour of several functions hosted by the platform whose most severe classification is considered to be HAZ to CAT.

The previous classification of the severity of Failure Condition is focused on the safety impact of reconfiguration failures. According to the classification, reconfiguration loss should have moderate safety requirements whereas erroneous reconfiguration should have more stringent safety requirements. As one goal of the reconfiguration mechanism is to enhance the operational reliability of the platform, it is worth noticing that frequent loss of reconfiguration would be in contradiction with the improvement of the reliability of the platform.

So we consider that the reconfiguration system architecture shall enforce the following qualtitative safety requirement: "*A single failure shall not lead to the failure condition loss or erroneous reconfiguration*". As the design of the reconfiguration system is still in progress, we think that it is premature to associate quantitative safety requirements with the defined failure conditions.

4 Hazard analysis

In this section, we analyse the effect on reconfiguration of the individual failure of the 10 functions listed section 2. For each function, we analyse the effect of two basic failure modes: loss (the function does not provide its nominal output when expected) and erroneous (the function provides an incorrect output).

We also study how other functions might mitigate the effect of reconfiguration function deviations. Mitigation is needed in order to enforce the qualitative safety requirement stating that "no single failure shall lead to erroneous or lost reconfiguration". The effect of the deviation of a single function that leads to "Loss of reconfiguration" is acceptable because we have to take into account the fault of the CPM module that triggers the reconfiguration. So, when we add the sub-function deviation and the CPM fault, we obtain a double failure that leads to the failure condition. This is consistent with the safety requirement. Hence we do not study mitigation means for the function deviations that lead to the loss of reconfiguration. We do investigate mitigation for the function deviations leading to erroneous reconfiguration.

Because of the lack of space, we only detail functions *Fault detection* and *Load SPR*. Then we give the conclusion of the hazard analysis as a table of failure modes and associated effects.

Function *Fault Detection* nominal behaviour is to inform the reconfiguration sequencer that a CPM module has failed. When this function is lost, the CPM module fault is not detected, and reconfiguration is not triggered. This leads to the loss of the reconfiguration. When this function behaves erroneously, a non-faulty module is incorreclty detected faulty. Without mitigation this could lead to an incorrect reconfiguration. In the preliminary design, this is mitigated by *Module test* function that should disagree on the CPM health status leading to the loss of reconfiguration.

Function *Load SPR* nominal behaviour is to load the application software on the spare module. When this function is lost the application software is not loaded on the spare. This leads to the loss of reconfiguration. When this function is erroneous, an incorrect software could be loaded on the spare. Without mitigation, this could lead to an erroneous reconfiguration. Function *Reconf Monitor* should detect the incorrect configuration and *Reconf sequencer* should stop the reconfiguration, this results in the loss of reconfiguration.

The following table summarizes the results of the Failure Hazard Analysis we have described in the previous section. The table is organized in four columns:

- **Function**: name of the considered function.
- **Mode**: Failure mode considered, either lost or err (for erroneous).
- **Effect**: Most severe effect of the deviation of the considered function on the reconfiguration function (lost means Loss of reconfiguration, err means Erroneous reconfiguration),
- **Mitigation**: Name of the mitigation function when needed.

Function	Mode	Effect	Mitigation
Fault Detection	lost	lost	Not needed
	err	spurious	Module Test
Module Test	lost	lost	Not needed
	err	lost	Not needed
Configuration Selector	lost	lost	Not needed
	err	err	Reconf Monitor
Deactivate CPM	lost	err	Reconf Monitor
	err	lost	Not needed
Load Network	lost	lost	Not needed
	err	err	Reconf Monitor
Activate Spare	lost	lost	Not needed
	err	err	Load Spare
Load Spare	lost	lost	Not needed
	err	err	Reconf Monitor
Reconf Monitor	lost	lost	Not needed
	err	lost	Not needed

5 Safety validation

From the specification and hazard analysis, we make a formal model of the reconfiguration architecture. The objective is to formally analyse the behaviour of the system under the failures of its sub functions. Whatever is the combination of these failures, we must prove that the safety requirement *A single failure shall not lead to the failure condition loss or erroneous reconfiguration* is satisfied.

5.1 Formal model of the reconfiguration system

We have chosen to model the reconfiguration on a cluster. The platform is composed of the equipments participating to the reconfiguration (RS, CMS, DL and CM) plus 5 reconfigurable modules and 2 spares connected by a switch. We have not represented the switch in the model.

We make the hypothesis than no more than one failure occurs at a time. For instance, if a CPM fails, there is no failure during the reconfiguration. But, it may be the case that when a CPM fails, several equipments are already failed.

5.1.1 *The AltaRica language*

The AltaRica language [APGR99] was defined in the 90's to help the analyse of the dependability of systems. It is based on extended finite automata which can exchange values of specific variable (named *flow variable*) and which can be synchronised (synchronised product or broadcast). The idea is to describe the failure modes of a component as different states of an automaton.

As an example, let us explain the modeling of the *fault detection* function. The automaton is the product of the two automata drawn in Fig. 4.

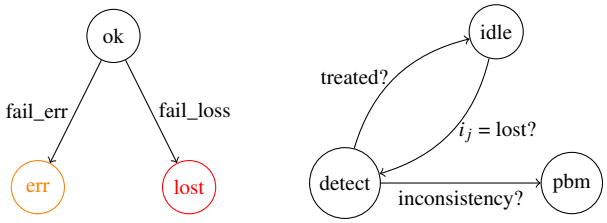

Fig. 4 Model of the *fault detection* function

On the left, the automaton shows the nominal mode *ok* and the two failure modes *lost* and *err*. Initially, the function is *ok*. If a failure occurs, the function enters in a failure mode. This is represented by a transition labelled by an event. On the right, the behaviour during a detection is schematized. This automaton applies for both modes *ok* and *err*. Indeed, if the function is *lost*, it means that it is blocked and nothing happens. Thus, it remains in the state *idle*. In the nominal mode *ok*, the detection is correct, while in the state *err*, the failed detected module is a wrong one. More formally, the *fault detection* function is modeled with:

- two local variables: $ds \in \{ok, lost, err\}$ represents the mode and $s \in \{idle, detect, pbm\}$ represents the functional behaviour;
- five inputs i_j with $j \in [1,5]$ and $i_j \in \{ok, lost, silent\}$ which represent the state of

the five CPMs. If $i_j = ok$, it means that the CPM is healthy, if $i_j = lost$, it means that the CPM is failed and finally, if $i_j = silent$, it means that the CPM is shut down;

- the input *treated* is emitted by the reconfiguration sequencer when the failure has been treated by a module test or a reconfiguration. If *treated=true* and the function still detects a failed CPM, that is $i_j = lost$, it means that the reconfiguration system is in an inconsistency state;

- the transitions depicted in Fig. 4. For instance, the transition to reach the failure mode *loss* is written

```
ds=ok |- fail_loss -> ds:= lost;
```

The transitions of the right automaton are triggered by internal events, corresponding to the functional behavior of the function. For instance,

```
ds!= lost and i1 = lost and s = idle
     |- to_detect -> s := detect;
```

the internal event *to_detect* is automatically fired as soon as the guard of the transition is true. It represents that fact that a failed module is detected and that the reconfiguration process starts;

- an output variable *num_failed_CPM*: it corresponds to the failed detected module. Its value is computed by the following predicate.

```
num_failed_CPM = case {
   ds=ok and s!=pbm and i_j=lost: i_j,
   ds=ok and s!=pbm and not i_j=lost : 0,
   ds=lost : 0,
   s=pbm : 6,
   ds=err and not (i_j=lost) : 0,
   else 6 - i_j}
```

In the nominal mode, if no CPM is failed the value is 0 and otherwise it is the number of the failed CPM; if the function is lost, it does not detect anything and the value is always 0; if the function has detected an inconsistency it remains in a deadlock state emitting the value 6; if the function is erroneous and a CPM is lost, the value is wrong with $6 - i_j$ instead of i_j.

5.1.2 *Model of the reconfigurable platform*

We model in AltaRica using the tool Cecilia OCAS [Sys07] developed by Dassault. Each component is described by an automaton as shown previously and the automata are assembled together. The assembly is drawn in Fig. 5.

We recognise the cluster composed of 5 CPMs and 2 spares; and the different functions involved in the reconfiguration process. The component *CM_pwr_management* regroups the functions *module test*, *deactivate CPM* and *activate SPR*. The components are connected with links which represent the data-flow. A link has always a white box extremity and a black box extremity. The white boxes represent inputs while the black boxes the outputs. For instance, there is a flow between *RS_sequencer* and *CMS_fault_detection* for the variable *treated* with a black box on *RS_sequencer*.

5.2 Formal analysis

Cecilia OCAS offers the possibility to generate all the failure sequences that lead to a given situation, and in particular to a failure condition. This is typically used to ensure that a failure condition never occurs, or to know exactly in which cases it may happen.

5.2.1 *Observer component*

We first have to specify the failure condition with an AltaRica component, called an *observer*. The failure condition we are interested in corresponds to the loss of reconfiguration. This can be specified by the following statement: *some application is hosted neither by its module nor by a spare module*. This is indeed the case when the module that hosts a software fails, and the platform is not reconfigured.

An observer has no state variable, and no event. It only has input variables (in order to observe the system current configuration), output variables (in order to describe the failure condition in terms of inputs) and possibly local variables to help the formulation of the failure condition.

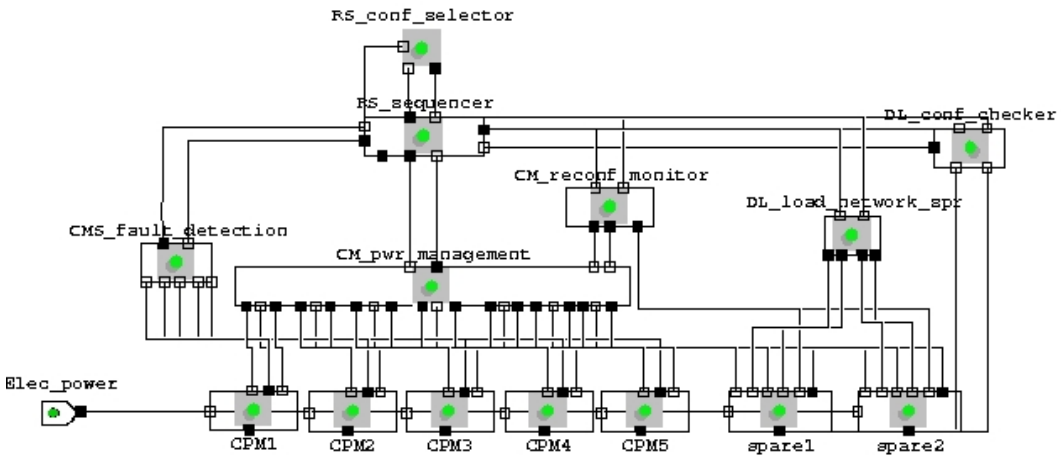

Fig. 5 Representation in AltaRica

Our observer has

- five inputs a_i, with $i \in [1,5]$, taking their value in $[0,5]$, which represent the application running on CPM i. $a_i = j$ (with $j > 0$) means that the application j is running on CPM i. $a_i = 0$ means that CPM i hosts no application.
- two inputs s_1, s_2 taking also their value in $[0,5]$, which represent the application running on the two spares
- five local variables o_i, with $i \in [1,5]$ that ease the formulation of the failure condition. o_i means that the application i is hosted either by its CPM, or by one of the two spares. In practice, we have for $i \in [1,5]$:

```
o_i = (a_i = i or s_1 = i or s_2 = i)
```

- one output *Out* which is false whenever the failure condition holds

```
Out = (o_1 and o_2 and o_3 and o_4 and o_5)
```

Notice that *Out* is true whenever all the five applications are hosted by some computational resource (CPM or spare), and false whenever at least one of the applications is not hosted by any computational resource, which corresponds to our failure condition. So, we generate sequences that lead to $Out = false$.

5.2.2 *Example of sequence generation*

For this, we parametrize OCAS sequence generator with the maximal length of sequences we want to generate, and with the kind of sequences we are interested in (minimal cuts, combinations, permutations). The tool then produces all the sequences in a specified file.

We give as an example the following sequence that is generated for the above-mentioned observer (situation: $Out = false$, maximum length: 2).

```
{'CMS_fault_detection.fail_error', 'CPM1.fail'}
```

This sequence corresponds to a failure of the fault detection function, leading to an erroneous behaviour, following by a permanent failure of CPM1. Let us see how this sequence of events leads to our failure condition. After the failure of CPM1, the fault detection function does as if another CPM had failed. The sequencer then asks for confirmation, gets a negative answer and then stops reconfiguration. So in the end the application that was hosted by CPM1 is not hosted by any operational computational resource.

5.2.3 *Applications*

The sequences can be analysed with respect to four aspects.

First, it is possible to check the consistency between the model and the Hazard analysis that

was performed. We can check that each function whose failure leads to the loss of reconfiguration, according to the hazard analysis table, appears in a length 2 sequence in combination with the failure of a CPM . We can also check that functions that, once mitigated, lead to loss of reconfiguration appear in length 3 sequences in combination with a failure of the mitigation function and a CPM failure.

Secondly, we can check that the qualitative requirements are enforced so that no sequence is made of a single failure event. The analysis of our model found that the erroneous behaviour of function *Deactivate CPM* could potentially lead to the loss of all CPM. This indicates that the Cabinet Manager is highly critical and that is should be duplicated in order to enforce the safety requirements that were selected.

Thirdly, we can analyse the segregation requirements of the architecture. We can extract from the sequences the pairs of functions that need to be segregated. We consider that a faulty function and a mitigation function should be segregated, they should not be hosted on the same computer because the erroneous behaviour of this computer could lead to the erroneous behaviour of both the faulty function and the mitigation function. In that case, the safety requirement stating that no single failure shall lead to lost or erroneous reconfiguration would no longer be enforced. The analysis found a problem related with the Cabinet Manager (CM) computer because the function *Deactivate CPM* and its mitigation *Reconf. Monitor* both run on CM. Due to the criticality of the CM/Power Management application, it could be the case that this computer has sufficient internal redundancy such that erroneous behaviour of the CM is not caused by a single event. In that case, no mitigation would be needed to limit the effect of power management deviations.

Finally, we can analyse the allocation of DAL levels to functions. According to SAE ARP 4754 [SAE10], a Development Assurance Level (DAL) is associated with each item. This level guides the methods to be applied in the implementation of the item. The allocation is based on the classification of the most severe Failure Condition (FC) that can be partially caused by the item fault. The DAL level of functions leading to erroneous reconfiguration should be A as these functions contribute to a FC that is potentially classified CAT. As these functions do not contribute alone to the Failure Condition, the revised ARP4754a has established rules that authorize to downgrade the DAL level of functions contributing to a FC. The rule that can be applied for double function failures leading to a CAT failure condition when these two functions are segregated is that at least one function is allocated DAL A and the level of the second can be decreased down to level C. We extracted the combinations of double failures from the generated sequences and checked that the DAL allocated to them was consistent with ARP4754a rules

6 Conclusion

The Hazard analysis and formal modelling of SCARLETT reconfiguration mechanisms were conducted in order to provide a first assessment of the safety impact of the deviation of the main functions. The deviations (function loss and function error) that were considered are theoretical. They need to be validated with respect to the actual behaviour of the reconfiguration mechanisms. We intend to use the SCARLETT demonstrators in order to test whether the theoretical deviations are possible, whether the effects are similar to what was imagined and whether the mitigation means that were proposed are sufficient to block the effects that were considered.

The formal safety models enabled to explore exhaustively and automatically the effect of multiple failures of items inside and outside (electrical power) the reconfiguration system. It should also be possible to easily compare various allocations of the functions onto existing system functions (Cabinet Manager, Reconfiguration Supervisor, CMS, DLCS, CC, ...).

Another activity that should be investigated in the future is the safety assessment of other reconfiguration scenarios that were described in [BNP+09] that deal with topics as migrating ap-

plications from a CPM to a Spare that does not belong to the same cluster, or migrating applications on spare partitions of other CPMs.

References

[AFAL07] Luis Almeida, Sebastian Fischmeister, Madhukar Anand, and Insup Lee. A dynamic scheduling approach to designing flexible safety-critical systems. In *EMSOFT '07: Proceedings of the 7th ACM & IEEE international conference on Embedded software*, pages 67–74, New York, NY, USA, 2007. ACM.

[APGR99] André Arnold, Gérald Point, Alain Griffault, and Antoine Rauzy. The altarica formalism for describing concurrent systems. *Fundamentae Informaticae*, 40(2-3):109–124, 1999.

[BNP+09] Pierre Bieber, Eric Noulard, Claire Pagetti, Thierry Planche, and FranÃğois Vialard. Preliminary design of future reconfigurable ima platforms. *In 2nd Workshop on Adaptive and Reconfigurable Embedded Systems (APRES'09), SIGBED Review*, 6(3), October 2009.

[Ell97] Steve M. Ellis. Dynamic software reconfiguration for fault-tolerant real-time avionic systems. In *Microprocessors and Microsystems, Proceedings of the 1996 Avionics Conference and Exhibition*, volume Volume21, Issue 1, pages 29–39, July 1997.

[SAE10] SAE. Aerospace recommended practices 4754a - development of civil aircraft and systems, 2010. SAE.

[See96] Kenneth A. Seeling. Reconfiguration in an integrated avionics design. In *Digital Avionics Systems Conference, 1996., 15th AIAA/IEEE*, pages 471–478, Oct 1996.

[SH95] Chak Sriprasad and Michele Harvey. Dynamic software reconfiguration using system-level management. In *Digital Avionics Systems Conference, 1995., 14th DASC*, pages 336–, Nov 1995.

[SSE+10] Tobias Schoofs, Peter Schmitt, Christian Engel, Eric Jenn, and Rodrigo Coutinho. "enhanced dispatchability of aircrafts using multi-static configurations". In *European Real-Time Systems and Software (ERTSS)*, 2010.

[Sys07] Dassault System. Module OCAS: Analyse système par arbre de défaillance. manuel utilisateur ocasv4.3, 2007.

6.1 Copyright Statement