Toward a validation process for model based safety analysis

R. Adeline[1], J. Cardoso[2], P. Darfeuil[1], S. Humbert[1], C. Seguin[3]

1: TURBOMECA, BP N°6, 64511 BORDES (romain.adeline ; pierre.darfeuil ; sophie.humbert@turbomeca.fr)
2: ISAE, 10 Avenue Edouard Belin, 31055 TOULOUSE (janette.cardoso@isae.fr)
3: ONERA, 2 Avenue Edouard Belin, 31055 TOULOUSE (christel.seguin@onera.fr)

Abstract: Today, Model Based Safety Analysis processes become more and more widespread to achieve the safety analysis of a system. However and at our knowledge, there is no formal testing approach to ensure that the formal model is compliant with the real system. In the paper, we choose to study AltaRica model. We present a general process to well construct and validate an AltaRica formal model. The focus is made on this validation phase, i.e. verifying the compliance between the model and the real system. For it, the proposed process recommends to build a specification for the AltaRica model. Then, the validation process is transformed to a classical verification problem between an implementation and a specification. We present the first phase of a method to verify the compliance between the model and the specification.

Keywords: Model Based Safety Analysis, validation, formal model, AltaRica

1. Introduction

Model-Driven Engineering (MDE) is a methodology which aims to base the development of a system on the creation, the refinement and the integration of models. MDE was created to increase productivity by, for example, simplifying the design or promoting communication between the different teams working on a same project.

If at the beginning, MDE was focused on the development of software systems, its applications are nowadays more spread and deal for example with the achievement of safety analyses. Thus, where MDE is generally focused on the automatic generation of code (e.g. C, C++), Model Based Safety Analysis (MBSA) aims to provide a model to automatically perform classical safety analyses such as Fault Tree Analysis (FTA) or Failures Modes and Effects Analysis (FMEA) ([8]).

Concerning this MBSA, several works ([9]) propose to assess the safety of systems by allowing simulation, the automatic generation of fault trees or searching automatically failure scenarios leading to undesired events. Amongst all the languages, we have chosen to use AltaRica which was designed to formally describe the functional and dysfunctional behaviour of a system.

In a general way, the process to study the behaviour of a model can be divided into three main parts:

- *Modelling activity.* System components and their behaviours are described in an adapted formal language. Methodologies and guidelines can be written to give rules and best practices. These guidelines aim at both helping the model design and encouraging the reusability of models (by , for example, standardizing information implemented in the model);
- *Validation of the model.* This step ensures that the model is a valid abstraction of the real system (correct hypotheses and same behaviours). Without this step, the next one can be useless;
- *Verification of the specification.* We check that the model holds the system's specifications.

In this paper, we deal with the second point. Indeed, although the problem of validation of AltaRica models is not well address in the literature, this validation seems to be essential in order to check if safety analyses are generated from erroneous or incomplete models.

The paper introduces the first steps of a validation process for AltaRica models. Section 2 presents an overview of the classical safety analysis process. With showing some limits of this kind of analyses, we present the motivation for the use of MBSA. Then, the AltaRica language is described in section 3. Sections 4 to 7 are devoted to the presentation of the validation process for AltaRica models. In the sections 6 and 7, the work is focused on the unit validation, i.e. the validation of the AltaRica component library. Finally, last section presents a conclusion of our work and futures works.

2. From classical safety analysis to MBSA

2.1 Definitions

Before describing the classical safety analyses, some definitions, strongly inspired from [1], are introduced.

- Failure: the inability of an item to perform its intended function.
- Failure condition: Condition with an effect on the system and its users, caused by one or several

failures. It depends on both operational and environmental conditions.

- Failure mode: the way in which the failure of an item occurs.

2.2 Classical safety analyses

Safety engineering ensures that the safety requirements (extracted from international standards) are held by the system considering all potential failure modes of each component. In this purpose, safety studies aim to define the safety requirements and then ensure that the system fulfils its required properties. In industrial practice, we can distinguish the following types of safety analyses.

- Assessment of qualitative requirements. The objective is to demonstrate that no combination of events with less than N individual failures leads to the failure condition (N depends on the severity of the failure condition).
- Assessment of quantitative requirements. The objective is to compute the occurrence probability of failure condition.

To perform the safety analyses, safety engineers traditionally use the Failure Modes and Effects Analysis (FMEA) and the Fault Tree Analysis (FTA) ([8]).

Building a FMEA consists in identifying all the potential failure modes of each system component and analysing their local and global effects on the system.

A FTA is a top down approach which illustrates the way in which low level component failures or events contribute to the global system failure condition. Thus, an FTA begins with a defined failure condition and breaks it down into basic failure modes identified in the FMEA. One strong advantage of the FTA is its simple formalism. After doing the breaking down, the FTA become a logic gate network. In this network, the failure condition is described by a logic equation from which we can extract failure scenarios. From this logic equation, it is even possible to compute the occurrence probability of the failure condition during the considered mission.

2.3 Toward Model Based Safety Assessment

Although FMEA and FTA are classical methods, several limits can be seen.

- The size and the complexity of current industrial systems grow. They become highly reconfigurable and performing the identification of failure scenarios without model can be error prone;
- Because a fault tree describes only one failure condition, it can be heavy to build all fault trees for all failure conditions;
- In case of modification of the system, several fault trees would have to be updated;
- Even if the formalism of fault trees allows an easy computation of qualitative and quantitative results, this formalism is different from the representation of the system. The fault trees can be difficult to read for someone outside the safety domain, especially when the number of terminal events is important.

We think these limits can be overcome by performing the safety analysis activities on formal model of the system under development. Instead of building one fault tree for each failure condition, we provide a formal model describing both the nominal system behaviour and the dysfunctional behaviour of the system. On this model, several failure conditions can be studied. The failure scenarios as well as the FTA could be automatically generated.

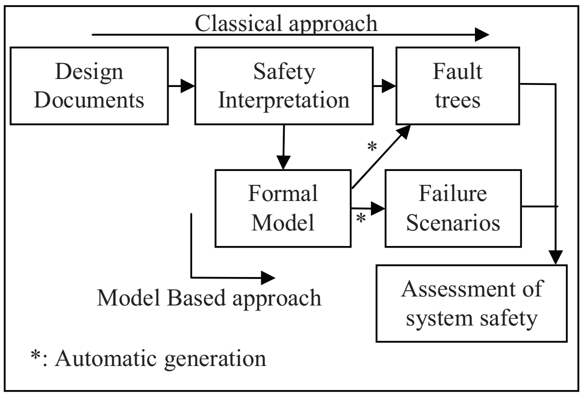

Figure 1 : The two approaches

2.4 Toward a rigorous process for MBSA

So, MBSA can help to tackle several limits shown by the classical approach. But, today, the modelling process of these formal models is based, as for the classical approach, on informal sources.

Indeed, FMEA and FTA are based on informal sources such as regulations (for the identification of failure conditions and requirements), design documents and field experience. Thus, they are highly dependant on the skill and also on the interpretation of safety engineers in charge of the analysis. Moreover, the validation process of FMEA and FTA is mainly based on 1) the verification by experts that no failure mode is forgotten in the FMEA, 2) that the effect of the failure modes are well described and 3) that all failure scenarios are considered in the FTA. Thus, validation process remains a mere proofreading and is all but formal.

Today, the same report can be made about the model based approach.

To solve this problem, the paper presents a global process for the rigorous AltaRica modelling of a system. The focus is made on the validation of these AltaRica models.

3. The AltaRica language

Amongst all the languages available in the literature to perform MBSA, we have chosen to use AltaRica [2], a formal modelling language, developed at LaBRI to describe both functional and dysfunctional behaviour of a system. It allows representing the failure propagations in an industrial system.

The language is carried out by the tool Cecilia™ OCAS (for example) which provides a graphical interface to design models and allow analysing them by different ways such as simulation, automatic generation of minimal cuts (i.e. shortest scenarios leading to the failure condition) or sequences (i.e. ordered cuts).

AltaRica language is hierarchical and compositional. Each component is described by a mode automaton [3]. The basic unit to model a system component is called a "node" and is composed with three different parts: 1) the declaration of variables and events; 2) the definition of transitions; 3) the definition of assertions.

Each component has a finite number of flow variables and state variables. Flow variables are the inputs and the outputs of the node: they are the links between the node and its environment. State variables are internal variables which are able to memorize current or previous functioning mode (for example, failure mode). In our models, these variables (flow and state) are either Boolean or enumerated. Then, each node owns also events which modify the value of state variables. These events are phenomenon such as a failure, a human action or a reaction to a change of one input value.

The transitions describe how the state variables are modified. They are written such as "G(s,v) |- E ->s_" where G(s,v) is a Boolean condition on state s and input variables v, E is the event and s_ is the effect of the transition on state variables. If the condition G is true, then event E can be triggered and state variables are modified as described in s_.

The assertions describe how output variables are constrained by the input and state variables.

These concepts are illustrated by the following example.

```
node component
  flow
       input: {ok, low, null}: in;
       output: {ok, low, null}: out;
  state
       ST = {ok, degraded, lost};
  init
       ST:=ok;
  event
       Degradation, Fail;
  trans
       ST=ok |- Degradation -> ST:= Degraded;
       ST=ok or ST=degraded |- Fail -> ST =lost;

  assert
      output = case { ST=ok : input,
       ST=degraded and input=ok: low,
       else null} ;
edon
```

This component has one *input* and one *output* variables both ranging over the domain {ok, low, null}, one state variable *ST* ranging over the domain {ok, degraded, lost} and two events *Degradation* and *Fail*. At the initial instant, the node is in state "ok". The event Degradation (respectively Fail) describes a failure which leads the node into the state "degraded" (respectively "lost"). "Degradation" (respectively "Fail") can be triggered only if the node is in state "ok" (respectively in the state "ok" or "degraded"). The assertion means that the output value is equal to the input one if the node is in state "ok". The output value is equal to "low" if the state is "degraded" and the input value is "ok". In every other case, the output value is "null".

To build the model of the global model, several nodes are interconnected.

Thus, failures are propagated via nodes by their inputs and outputs. Failures can also be propagated by synchronizations which simulate the failure of several components at the same time. Hierarchy of nodes can be used to model complex components and build the model of the global system. Once the global model is obtained, the AltaRica model allows analysing failure condition. Different tools can calculate, for example, minimal cut sets or the occurrence rate of a failure condition.

4. Validation in literature

Validation of classical safety analyses is mainly based on several proofreadings (by engineers and/or experts). Based on formal models, we believe that the new approach can benefit from a more elaborated validation process.

A lot of works aim at ensuring that a formal model meets a given specification ([4], [5]). Such a work is used to verify that the real system (pictured by the model) meets its specification and its requirements.

But one strong hypothesis under this kind of work is that the model behaviour is in accordance with the behaviour of the real system… which is, according to us, far from obvious! Thus, we believe that it is essential to verify that the two behaviours (model and real system) are coherent before verifying that some properties are hold by the model.

Testing compliance between a specification and the model is dependant from both the specification form and the model language. The following of this section presents two techniques extracted from literature to verify the compliance of the model with the real system.

4.1 Step-by-step simulation

For the validation of the model, one of the most overwhelmed methods is based on the step-by-step simulation of the model. This method ensures, for tested scenarios, that the model behaves as the real system. Nevertheless, this method is not perfect.

- Due to its nature, this method is not exhaustive. Indeed, an exhaustive simulation is often not feasible (at human scale) due to the large number of scenarios playable by the model.
- We don't have, to our knowledge and for AltaRica model, a way to measure the quantity of model covered by the test set. In the same way, we don't know which part of the model is tested by the test set (and which part is not).

In literature, works introduce different methods to generate relevant test cases from a given specification. For example, [4] presents a formal testing method developed for Statechart. The paper identifies, from a Statechart specification, a set of test cases to verify the compliance of an implementation and its specification.

4.2 Model Checking

To validate the model, another method advocated by literature is the model checking. The goal is to test if a property is held by the system. For it, the property is stated as a temporal logic formula. Then the model checker (i.e. the tool for applying model checking) tests if this property is valid in any state of the model. When a formula is not valid, the tool produces a counter example giving a scenario which violates the property. The counter-example can be useful to correct and update the model.

So, if model checking allows verifying properties on a model, it allows, in particular, verifying the conformity of the model in comparison with the real system. But some difficulties; described above, can be highlighted.

- Properties can be difficult to find and to formalize by engineers and experts;
- For a property, the model-checker provides only one counter-example.

Typically, the kind of property verified on the formal model is properties that the real system has to hold. For example, we can verify that no single failure leads to the loss of the system. But for our application, we will have to verify properties that the real system holds. So identifying properties needs a great experience and an excellent knowledge of the system and its behaviour.

For AltaRica language, previous works have developed a platform between AltaRica and the model checker SMV [10].

5. A validation process

5.1 High level view of the process

MBSA is clearly a promising approach to overcome limits shown by classical safety analyses. But, to use MBSA in industrial domain, we believe that it's very important to have a method of validation of the model. For this purpose, we envisage two major phases: 1) guiding the modelling phase to prevent some errors and 2) validating the models to detect and eliminate remaining errors. The first subject is not the point here. Details can be found in [6]. Here, the focus is done on the validation phase of an AltaRica model. According to us, a high level view of the general validation process can be pictured as below.

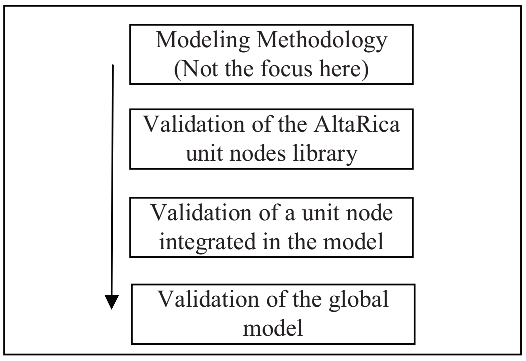
Modeling Methodology (Not the focus here)
Validation of the AltaRica unit nodes library
Validation of a unit node integrated in the model
Validation of the global model

Figure 2 : High level validation process

5.2 Validation process proposal

We introduce in this part a general method for the three levels of validation described above (unit, integrated and global). The general strategy of this method is to:

- Build a schematic representation of the system under study (this representation will play the role of a specification);
- Verify that the AltaRica model is compliant with this representation.

Thus, we transform the validation problem of an AltaRica model into a "classical" verification problem between a specification and an implementation. In the following, we call SFPM (Specification of Failure Propagation Model) the specification of the AltaRica model and AIFP (AltaRica Implementation of the Failure Propagation model) the implementation of the AltaRica model.

At this moment, the careful and cautious reader won't do any assumptions on the form of the SFPM. Indeed, according to us, the form of the SFPM will be different depending on the level of validation (unit, integrated and global).

The proposed process can be divided into two steps:

Concerning the SFPM:

- Define the SFPM from informal documents;
- Define the coverage criteria of the SFPM. What do we want to test on the SFPM? (the criteria will differ depending on the form of the SFPM and the goal of the validation);
- From both the SFPM and its coverage criteria, we generate test cases which cover these criteria.

Concerning the AIFP:

- Define coverage criteria on the AIFP. What do we want to test on the AIFP? (i.e. defining coverage criteria on AltaRica models);
- Play the test cases extracted from the SFPM on the AIFP. Checking of outputs and behaviours (state transfer);
- Check on the AIFP which part of the AIFP is covered by the SFPM test cases according to the AIFP coverage criteria;
- Identify AIFP parts uncovered by SFPM test cases;
- Generate test cases to cover these uncovered AIFP parts;
- Play these test cases and check for outputs and behaviours.

To well understand this process let us explicit two particular points of uncertainty in the process. First the SFPM might not be complete (i.e. it might not contain all of the real system's behaviour). Indeed, this SFPM arises from human work and might not contain all possible execution path of the AIFP. So, giving coverage criteria on the SFPM, generating test cases and playing them on the AIFP give confidence in the methodology but are not sufficient. AIFP can embed more information and more behaviours than the SFPM; some of them are true, some other are wrong behaviour and have to be detected and annihilated. Giving coverage criteria on AIFP prove that the model has been "sufficiently" tested. One other advantage of testing AIFP is that we can possibly complete the SFPM afterwards.

We believe that this general process can be applied and adapted to each kind of validation (unit, integrated and global). Depending on the validation level, parts of the process will change; in particular, the form of the SFPM will be different, different coverage criteria will be defined. In the following, our work has been focused on the unit validation (i.e. the validation of an AltaRica node).

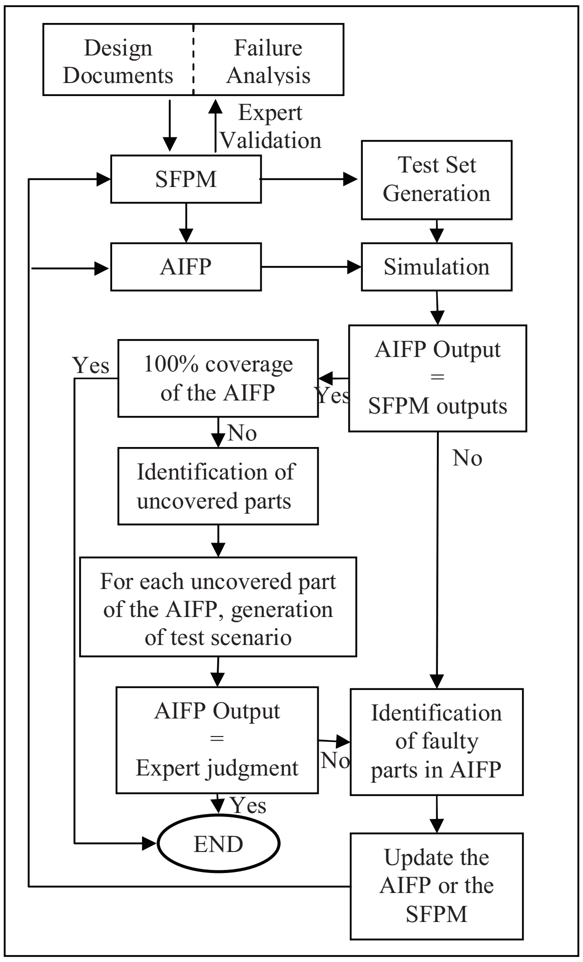
Design Documents
Failure Analysis
Expert Validation
SFPM
Test Set Generation
AIFP
Simulation
100% coverage of the AIFP
AIFP Output = SFPM outputs
Yes
Yes
No
No
Identification of uncovered parts
For each uncovered part of the AIFP, generation of test scenario
AIFP Output = Expert judgment
Identification of faulty parts in AIFP
No
Yes
END
Update the AIFP or the SFPM

Figure 3 : A proposal for the validation process

6. Construction of the SFPM

In this section, we deal with the definition and the building of the Specification of Failure Propagation

Model (SFPM) for an AltaRica node (for the unit validation).

In a general way and to justify the existence of SFPM, we consider that building a first representation of the component's behaviour (even if it is incomplete but was validated by the experts) is a prerequisite to build the AltaRica node and help at its validation.

6.1: SFPM Requirements

Before dealing with the form and the chosen formalism for this SFPM, we present three major requirements for this SFPM.

- The SFPM should depict correctly the real component behaviour;
- The SFPM has to be understandable by experts;
- The SFPM should be as formal as possible.

Concerning the first point and according to previous section, the SFPM is a critical point for the validation of AltaRica models. Indeed, as a specification, the SFPM is the reference of the AltaRica model, i.e. if the SFPM contains errors, the AltaRica model will contain errors too. Thus, the SFPM should be a correct depiction of the real system behaviour.

About the second point and because the SFPM have to be validated by experts, the SFPM needs to be written down in an easily understandable form. About this point, we agree that validating the SFPM by a mere expert proofreading is rather strange. However, we believe that having a schematic representation of the system is a prerequisite to build and validate the AltaRica model. Moreover, the focus is made here on the unit validation, i.e. the validation of a component of the AltaRica model. So, the size of the SFPM will be limited and, for the expert, easier to validate than the global AltaRica model.

The last requirement is that the SFPM has to be as formal as possible. Indeed, SFPM is the starting point for the generation of test cases. To allow a systematic generation of these test cases, having a formal (at least a semi-formal) description of the SFPM will be useful.

Taking into account these three requirements, the following subsection proposed a form for this SFPM.

6.2: SFPM building

To build this SFPM, a variety of methods / formalisms can be found in literature to picture and formalize the behaviour of a system. This includes, for example, decision tables (which output for which inputs?), Petri networks, Markov chains, Finite State Machines, UML diagrams, mode automata. The general purpose of this SFPM is to be a kind of bridge between informal documents (design documents, safety analysis or / and failure analysis documents) and the AltaRica model. For practical reasons and because of the AltaRica language, we choose, a Mode Automata like form. We are conscious that it is irrelevant to ask an analyst for a complete mode automaton. We suppose here that the analyst can write little parts of mode automata and existing tools can compute these parts into a complete mode automaton.

To achieve that, we propose to:

- Model the inputs / outputs of the component with a block diagram;
- Model the states and transitions with a classical state – transition UML diagram;
- Add to this state-transition diagram the assertions, i.e. the value of the outputs depending on the state and the input. The assertions will be written as ITE (If Then Else).

On the Figure 4, (a) is the block diagram describing the inputs, the outputs and their types. On the example, the inputs and the outputs are enumerated. The figure (b) describes the three states "Ok", "Degraded" and "Lost" of the automaton, the transitions "Fail" and "Degradation", and the value of the output in each state. Two remarks can be made:

- If a guard of a transition depends on the value of one or several inputs, the state – transition diagram will picture input change as transition,
- A state of the automaton won't be automatically pictured by one AltaRica state variable. Here, we will use one AltaRica state variable with three values {Ok, Degraded, Lost}.

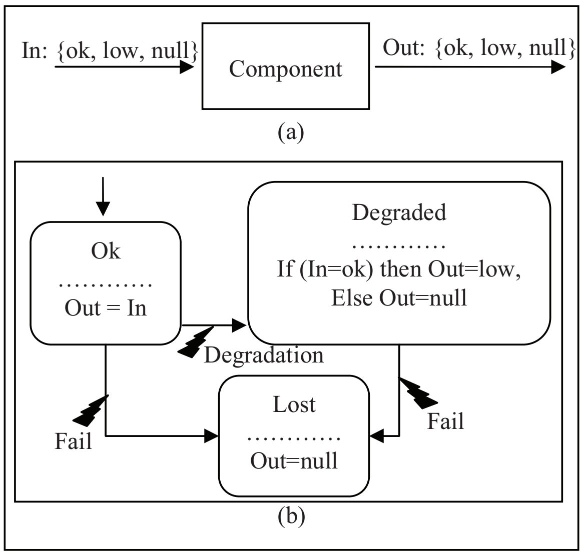

Figure 4 : Proposal for the SFPM

We believe that these kinds of diagrams are sufficiently clear, explicit and comprehensible to be validated by experts. Taking the hypothesis that we can transform these diagrams into a mode automaton, these diagrams will be considered as a reference (i.e. a specification) for both the implementation of the component model (modelling activities) and the validation of the implemented model. Thus, our problem can be considered as a classical problem of verification that an implementation is correct with respect to its specification.

6.3: AIFP building

From the SFPM, the AIFP is manually built. Concerning our example, the SFPM represented on Figure 4 corresponds to the AltaRica model presented in section 3.

7. Verification of the AIFP

At this stage, we have at our disposal the SFPM and the AIFP. We want now to validate the AIFP according to the process described in section 5.

7.1: SFPM and AIFP coverage criteria

First, we have to define the objectives of the verification. So, we must define what we want to test on both SFPM and AIFP, i.e. the coverage criteria on SFPM and AIFP.

<u>SFPM coverage</u>: On the SFPM, the coverage criteria are:

- the coverage of all states;
- the coverage of all transitions;
- the coverage of the assertions. For the assertions and because they are written under "If Then Else" form, we can choose classical coverage criteria of the DO178B [7]. Here, we choose the Condition / Decision coverage.

<u>AIFP coverage</u>: Defining coverage criteria for the AIFP consists in defining criteria for an AltaRica model.

Our coverage criterion is to cover the transitions and the assertions of the AltaRica model.

About the transitions, let us remind that AltaRica transitions are written under the form "G(s,v) |- E ->s_". We transform this form into "If (G(s,v) and E) then s_". The meaning is identical. But, written under "If Then Else" form, we can apply classical coverage criteria defined in DO178B. We choose the condition / decision coverage:

- Condition coverage: a condition is a simple Boolean expression, i.e. it can not be broken down into a simpler Boolean expression; the condition coverage means that each condition has to be evaluated to both true and false.
- Decision coverage (or branch coverage): the entire Boolean expression has to be evaluated to both true and false.

The AltaRica assertions are written under "If Then Else" form. So again, we can apply coverage criteria of the DO178B. We choose also the condition / decision coverage.

7.2: Preliminary for AIFP verification

Here, we want to directly measure the effectiveness of the test strategy on the AltaRica model. We present thus a way to measure the coverage of the model by introducing flags into the model.

<u>Measure of the coverage of the AltaRica transitions:</u>

To measure the coverage of the AltaRica transitions, we follow the following step.

- We transform AIFP transitions according to the condition coverage of the DO178B, i.e. transforming "A or B |- E → s_" into two transitions "A |- E → s_" and "B |- E → s_". Thus, each transition has a unique condition, i.e. the guard of each transition can not be broken into a simpler Boolean expression;
- We add to the AltaRica model one Boolean state variable ST_T_i per modified transition;
- We initialize them to "false";
- We transform transition such as:

  "A |- E → s_ & ST_T_1:=true"

  "B |- E → s_ & ST_T_2:=true";
- We add an observer to the AltaRica model. The observer is true if all ST_T_i are true. That means that all transitions have been covered.

<u>Measure of the coverage of the AltaRica assertions:</u>

To measure the effective coverage of the AltaRica assertions, we follow the following step.

- We transform AIFP assertions according to the condition coverage of the DO178B, i.e. transforming:

```
    "case {  A or B: V1,
             else V2}"
  Into
    "case {  A: V1,
             B: V1,
             else V2}".
```

- We add an integer local variable named LOC with the same form of the output variable:

```
“LOC = case {   A: 1,
                B: 2,
                else 3}”
```

- We add to the AltaRica model one boolean state variable ST_A_i per assertion
- We initialize them to “false”
- We add one transition for each assertion: “LOC=i and ST_A_i=false |- Dirac(0) → ST_A_i=true” where Dirac(0) is an instantaneous event
- We add an observer to the AltaRica model. The observer is true if all ST_A_i are true: that means that all assertions have been covered.

7.3: Verification of the AIFP

The first objective is to verify that the AIFP satisfies the properties of the SFPM. According to the SFPM coverage criteria (cf. 7.1); it consists in detecting, in the AIFP, missing states, transitions or assertions presents in the SFPM.

The second objective is to verify that there is no unwanted information in the AIFP. According to the AIFP coverage criteria, it consists in detecting extra states, transitions or assertions in the AIFP.

Verification of the AIFP: SFPM criteria

For verifying the properties of the SFPM on the AIFP, the first step is to generate a set of test cases that satisfies the SFPM coverage criteria. Then, those test cases are simulated on the AIFP.

Generation of the test cases: From the SFPM, we generate automatically test cases. Written as mode automata, our approach uses different works of the literature ([4], [5]).

The general principle of our approach is:

- Use the state-transition diagram of the SFPM (of the mode automata) to generate a first set of test cases which cover all states and transitions.
- Use the assertion part of the SFPM to complete the first set of test cases in order to cover the assertions. For it, we can use works about the coverage of imperative code ([7]).

Simulation of the test cases on the AIFP: Once these test cases obtained, we simulate them on the AIFP. It consists in checking if the actual output is the expected one (defined in the SFPM) and if after each transition, the state transfer is correct. To achieve this task, a great help is given by the step-by-step simulation tool of Cecilia™ OCAS. Indeed, thanks to this tool, every state and every output are observable. So checking for correct output and correct state transfer is a very easy task. If an error occurred (wrong output / wrong value of a variable state), the model is updated and tested again. When no error is claimed by the verification, we can go further on the process.

Moreover, during the simulation, we check, on the AIFP, for the potential activation of the flag ST_T_i and ST_A_i.

Verification of the AIFP: AIFP coverage

Previous subsection has tested the AIFP according to SFPM coverage criteria. The objectives were to discover missing states, transitions or assertions in the AIFP (with respect to the SFPM). Here, our objective is to detect extra states, transitions or assertions in the AIFP. Potentially, these extra information can be wrong (i.e. errors in the AIFP), or true (i.e. errors in the SFPM).

So, in this section, we want to generate test cases to test the AIFP according to AIFP coverage criteria.

For it and in the previous subsection (during the simulation of test cases extracted from the SFPM), the analyst has checked for the activation of the flags ST_T_i and ST_A_i. Here, we suppose that the analyst in charge of the verification knows the list of all defined flags, i.e. he knows all defined ST_T_i and ST_A_i.

So, our objectives here are the followings:

- Analysing which parts of the AIFP are not tested by the test cases extracted from the SFPM;
- Generating test cases to cover these parts.

About the first point, the analyst knows the list of all existing flags and the list of already activated flags (by the tests extracted from the SFPM). So, identifying the list of non-activated flag (the list of untested parts of the AIFP) is trivial.

Then, for each non-activated flag, we use the sequence generator tool of Cecilia™ OCAS to identify scenarios which activate these flags (for each flag, we obtain a list of scenario which activates this flag). Then, we choose the minimal list of scenarios which activates all non-activated flags. This minimal list corresponds to the test cases set.

Then, for each of these test cases, we play it on the AIFP (i.e. on the AltaRica model) and we observe the output and the state transfer. Here, to know if they are correct, our reference is an expert judgement.

In a general way and all along the process, if an error is detected, we have to check if the error is due to the SFPM or the AIFP. If it is due to the AIFP, we update the AIFP and have to play again the test cases extracted form the SFPM (and following step). If the error is due to the SFPM, the SFPM has to be updated and the total process has to be re-started. According to us, when we have no errors at the end of the process, the validation of our AltaRica node is finished.

9. Conclusion

In this paper, we outlined several important aspects of the design process of an AltaRica model (and more generally of a formal model). Amongst these aspects, the most important are 1) having a specification model and 2) transform the validation problem into a verification problem between the implementation and the specification.

The first point, not presented in details in the paper, consists in transforming a design from an informal description into a detailed and formal (at least semi-formal) specification usable for both the modelling and validation activities.

The second point, which is the subject of the paper, is to check that the implementation (i.e. the AltaRica model) is consistent with this specification. Coverage criteria are given and a process is described to generate automatically test cases which cover (with respect to the defined coverage criteria) the AltaRica model.

However, we argued that validation of AltaRica model is needed but we base this validation on a specification model validated by experts. Works will be described in [6] to present a process for the modelling activities which can be an input for a rigorous building of this specification. Also and for future study, a rigorous way to transform the specification into the implementation could be considered.

10. References

[1] Society of Automotive Engineers: "*ARP4754: Certification considerations for highly integrated or complex aircraft systems*", SAE international, 1996.

[2] G. Point: "*Contribution à l'unification des méthodes formelles et de la sûreté de fonctionnement*", PhD Thesis, LaBRI, University of Bordeaux, France, 2000.

[3] A. Rauzy: "*Mode Automata and their compilation into fault trees*", Reliability Engineering and System Safety, 78:1-12, 2002.

[4] K. Bogdanov and M. Holcombe: "*Statechart testing method for aircraft control systems*", Software testing, verification and reliability, 11:39-54, 2001.

[5] H. Ural: "*Formal methods for test sequence generation*", Computer communications, v.15 n.5, p.311-325, 1992.

[6] R. Adeline, J. Cardoso, P. Darfeuil, S. Humbert, and C. Seguin: "*Toward a methodology for the AltaRica modelling of multi-physical systems*", in preparation, ESREL 2010, Rhodes, Greece, 2010.

[7] RTCA - EUROCAE: "*DO-178B / ED-12: Software considerations in airborne systems and equipment certification*", 1992.

[8] A. Villemeur: "*Reliability Availability Maintainability and Safety Assessment*", John Wiley & Sons Ltd, 1992.

[9] A. Joshi, M. Whalen, M. Heimdahl: "*Model-based safety analysis final report*", NASA contractor report, NASA/CR-2006-213953, 2006.

[10] C. Kehren, C. Seguin, P. Bieber, C. Castel, C. Bougnol, J-P. Heckmann, S. Metge: "*Advanced Multi-System Simulation Capabilities with AltaRica*", Proceedings of the International System Safety Conference, 2004.

11. Glossary

MDE:	Model-Driven Engineering
MBSA:	Model Based Safety Analysis
FTA:	Fault tree Analysis
FMEA:	Failure Modes and Effects Analysis
SFPM:	Specification of Failure Propagation Model
AIFP:	AltaRica Implementation of Failure Propagation model