RAFFINEMENT ALTARICA POUR L'ETUDE DE SYSTEMES A DIFFERENTS NIVEAUX DE DETAIL
ALTARICA REFINEMENT FOR HETEROGENEOUS GRANULARITY MODELS ANALYSIS

R.Bernard, S.Metge, F.Pouzolz
Airbus France
316, rte de Bayonne, M0151/4
31060 Toulouse

P.Bieber
ONERA
2, av. Edouard Belin, BP4025
31055 Toulouse Cedex 4

A.Griffault, M.Zeitoun
LaBRI
351, cours de la Libération
33405 Talence cedex

Résumé
Nous définissons une notion de raffinement pour les modèles AltaRica afin de permettre l'analyse de la sécurité de systèmes décrits à différents niveaux de détails. Nous montrons comment s'assurer automatiquement qu'un modèle raffine un autre à l'aide de l'outil de vérification Mec V. Nous présentons deux résultats théoriques, l'un permet de vérifier le raffinement en se focalisant sur un seul composant du modèle à la fois et l'autre caractérise le type d'exigences de sécurité qui sont préservées par le raffinement. Nous proposons d'utiliser ces résultats théoriques pour faciliter l'analyse la sécurité d'un ensemble de systèmes aéronautiques interconnectés.

Summary
We define a notion of refinement for AltaRica models in order to analyze the safety of systems described at various levels of detail. We describe a technique based on the Mec V model-checker to automatically check that one model refines another one. We present two theoretical results: the first one allows to verify refinement by focusing on one component of the model at a time, the second result defines the kind of requirements that are preserved by refinement. We propose to use these theoretical results in order to ease the safety assessment of a set of interconnected aeronautical systems.

Aircraft multi-system safety analysis

Aircraft design is based on a hierarchical functional description where top-level aircraft functions (functions such as: control aircraft yaw in flight, or reduce speed on ground) are linked with lower level system functions (functions such as: control the rudder, generate DC electrical power, display information for the pilot…). This description guides the functional hazard analysis that identifies situations associated with function misbehaviour that have a safety impact. These situations are called Failure Condition (FC), they are classified according to their severity. This leads to the definition and the allocation of safety requirements that have to be taken into account by system designers. In the context of Aircraft certification, safety specialists check system architecture proposals in order to verify that safety requirements defined for each Failure Condition are met.

During European R&D projects ESACS (see. [2]) and ISAAC (see. [3]), Airbus, LABRI and Onera proposed a method based on formal notations such as AltaRica to model system architectures and to study how failures propagate in such architectures. Associated analysis tools can be used to analyse a system architecture model in order to automatically search for all minimal combinations of failures leading to a given failure condition FC and to compute the probability of FC.

This approach was first applied with success to study standalone systems. But, systems tend to become more and more inter-dependent due to the use, for instance, of shared computing and communication equipments. Failures in one system could impact several other systems. One could imagine scenarios where each system satisfies the safety requirements that were allocated to it but aircraft level system requirements are not fulfilled when systems are connected together. In the framework of an internal project called CVB-RoSaS, Airbus performed case studies that interconnect various systems in order to study multi-system safety problems.

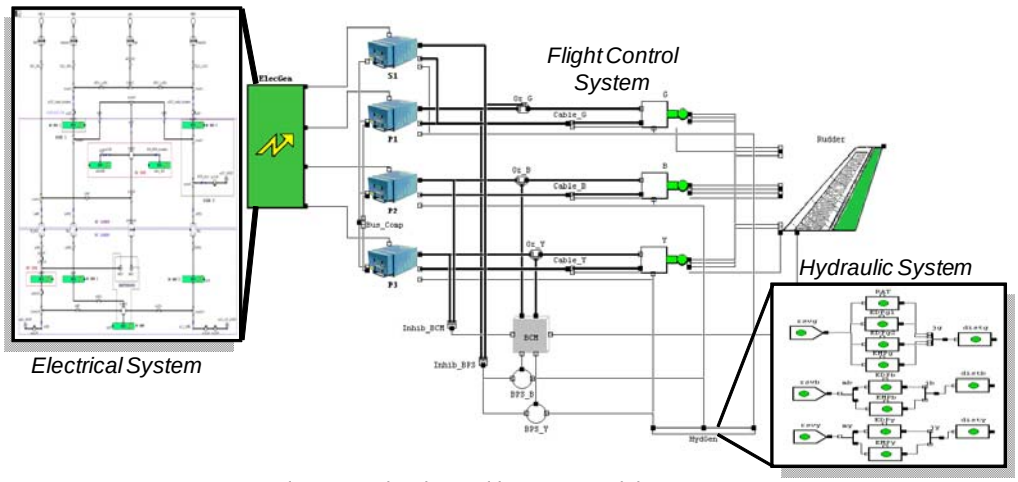

Fig. 1: An AltaRica multi-system model

The CVB-RoSaS project showed that it was rather easy to interconnect AltaRica system models developed independently in order to build a big multi-system model. Figure 1 shows the interconnection of models useful to study the safety of a Flight Control function such as the control of the Rudder. At the centre of the picture is shown the Rudder control architecture that uses computers, relays, servo-valves and sensors. On the left side of the picture the electrical system is shown, it is made of generators, transformers, circuit breakers, switches and bus bars. The right side of picture shows the hydraulic system architecture that is made of tanks, pumps, valves and distribution lines.

These three systems are interconnected. The electrical system provides power through the bus bars to both the hydraulic pumps and the flight control computers. The hydraulic distribution lines provide power to the servo-valves hat actuate the rudder. Furthermore, in emergency cases, hydraulic power is used to generate electrical power and to power a back-up module (BCM) that controls the rudder.

One lesson learnt from the CVB-RoSaS project is that it was interesting to interconnect system models that do not necessarily include all the details of their architectures. For instance, the model could focus on one particular system that is described at a fine level of detail and all other interconnected systems would be described at a coarser level of detail. This allows circumventing some limitations related with the complexity of the analysis of very detailed models. Furthermore, this approach would be consistent with the current safety process used by Airbus to deal with multi-system safety assessment. When studying one system, the safety analyst identifies a set of failure conditions of the systems in interface (called Dependent System Failures DSF) that are related with the FC of the system under study. Safety assessment results, such as minimal cut sets for a FC of the system under study, would detail failures of the system equipments but, for failures occurring in other systems, they would contain DSF such as loss of DC power on side 1 instead of detailed equipment failure modes such as short-circuit in electrical DC bus-bar named EXP1. This approach helps the analyst to focus on the architecture of the systems under study and to hide details about other systems.

One question related with the proposed approach is: what is the status of an analysis based on a high level model of a system in interface when this model is replaced with a model including more details? To answer this important question we have tried to define formally situations where it is safe to replace an AltaRica model by another one. We have investigated the notion of refinement of models that has been defined for formal methods applied to software design, such as the B-method [1]. In this context, designers progressively add details to an abstract model while preserving the properties that are enforced by the abstract model.

In the first chapter we explain basic aspects of the AltaRica language, then we describe model analysis tools focusing on the Mec V model-checker [14]. In the second chapter we introduce a notion of refinement for AltaRica models and we explain how to check that one model refines another using the Mec V model-checker. In the third chapter, we detail how to use refinement to perform safety assessment and in the fourth chapter we deal with the safety assessment of a set of interconnected systems that have different levels of detail.

AltaRica and Failure Propagation Models

1. AltaRica language

AltaRica is a formal language developed by the LaBRI (Laboratoire Bordelais de Recherche en Informatique), jointly with French industrial partners (especially Dassault Aviation and Elf Aquitaine…) in order to model safety critical systems.
Each component, called a node, is composed of the declaration of variables and events, and the definition of transitions and assertions. These concepts are illustrated by the following example. Component SWITCH has two positions: initially it is on position 1, it transmits on its output the primary signal `InSignal1` as long as it is received. When the primary signal is lost, an internal event changes definitely the switch position to position 2 in order to transmit the secondary signal `InSignal2`. As long as the primary signal is received, a failure may occur and the switch can be stuck on position 1. This behaviour is illustrated by the labelled transition system in figure 2.

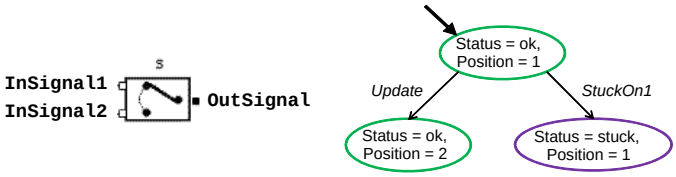

Fig. 2: SWITCH AltaRica node and associated Labelled Transition System

Given the domains {`Ok, Stuck`} called `StatusType`, and {`1, 2`} called `SwitchPos`, the AltaRica model of a SWITCH is presented in the following tables.
The state variable table declares that `Status` (resp. `Position`) is an internal variable of type `StatusType` (resp. `SwitchPos`) that is initially equal to `Ok` (resp. `1`). The value of a state variable can only be modified by transitions.

State	Type	Initial Value
`Status`	`StatusType`	`Ok`
`Position`	`SwitchPos`	`1`

Tab. 1: State variable of SWITCH

The flow table declares input/output/local variable(s), typed, but not initialised. They are used to model data exchanged with interfaced components.

Flow	Type	Direction
`InSignal1`	`Boolean`	`In`
`InSignal2`	`Boolean`	`In`
`OutSignal`	`Boolean`	`Out`

Tab. 2: Flow variables of SWITCH

The event table declares events and tells whether they are triggered internally by the system or externally by its environment. In the latter case, an occurrence probability law may be associated with the event. In the following all internal events will be named update.
The transition table contains the set of transitions describing events. A transition is composed of a guard (Boolean expression based on state variables and input/local flow variables, defining in which conditions the transition may be triggered), the name of the event labelling the transition, and the set of state variable new affectations. In the following table, a transition is associated to each event:

- event `StuckOn1` may only be triggered when component SWITCH is not already stuck and is on position 1.The new value of `Status` is `stuck`.
- event `update` may only be triggered when the component is on position 1, not stuck and does not receive the position 1 input signal anymore. The new value of `Position` is 2.

Event	Guard	New affectations	
		Status	Position
`StuckOn1`	`Status = Ok and Position = 1`	`Stuck`	-
`Update`	`Status = Ok and Position = 1 and not InSignal1`	-	2

Tab. 3: Transition table of SWITCH

The assertion table contains the set of output/local variables computation formulae. Computation formulae of an output variable may be based on state, input and local variables. Local variables are mainly used to factorise redundant expressions. The following table defines how the `OutSignal` output is computed. An `OutSignal` is sent (`OutSignal = true`) whenever a signal is received on the input corresponding to the current position of the switch otherwise no signal is transmitted (`OutSignal = false`).

Assertion	Case	Value
`OutSignal`	`Position = 1 and InSignal1`	`True`
	`Position = 2 and InSignal2`	`True`
	`Else`	`false`

Tab. 4: Assertion table of SWITCH

An AltaRica model, composed of several components linked together, is a node composed of sub-nodes and may be considered as a representation of a Labelled Transition System (LTS) of all possible behaviour scenarios. Each state of the LTS is a configuration, i.e. a function that associates a value with each state and flow variables. The initial state is first defined by initial value of components state variables. Then, assertions are computed and values are associated with outputs. These values are propagated to connected components hence values can be associated with inputs. In the LTS, two states are connected by an arrow labelled with an event name whenever, in the first state, the guard of the transition associated with that event is satisfied and, the second state is equal to the first state with state variables modified according to the assignments of the corresponding transition.
A scenario of the model is a path in the LTS that starts from the initial state and moves from state to state by selecting an arrow labelled with an event. If, from a current state, an arrow labelled with an update internal event exists, then external event cannot be selected from this configuration. This means that update internal events are given priority with respect to external ones.

2. AltaRica for failure propagation modeling

Most of the events of an AltaRica model, that describe failure propagation in a system, represent failure modes of equipment of the system architecture. These events are mainly stochastic events: probability laws can be associated to them and later be used to evaluate the enforced quantitative requirement. But some events may represent nominal actions such as monitoring or reconfigurations that we call internal events. Those actions are generally automatically performed in order to prevent the system to stay in an unsafe situation. From a modelling point of view, internal events are modelled as update internal events and interpreted as events that should not appear in safety analyses. As cascades of nominal actions are often possible, the modeller focuses on stable states were no internal event can be triggered.

The model described in figure 3, is made of two redundant generators `G1` and `G2`. They are linked to a switch `S` that behaves as described in the previous section. Each generator has one failure mode (`loss`), we assume that a generator is able to provide power unless it is lost.

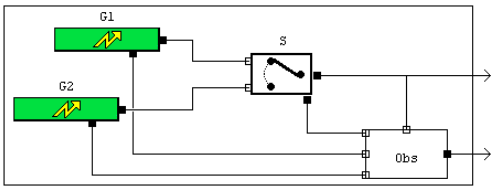

Fig. 3: Redundant generators with a switch

Consider the following scenario: in the initial state `G1` provides power and `S` is on position 1 so `S` delivers power, if `G1` is lost then as `S` is on position 1 `S` cannot deliver power but this not a stable state as internal update event `S.update` can be triggered and, in the resulting state, `S` is on position 2 so, if `G2` works correctly, `S` delivers power.

As explained in the introduction, the analyst identifies unsafe situations, called Failure Conditions (FC), and associates a classification and safety/reliability requirements (qualitative and quantitative) with each FC. The requirement is of the form "*If the FC has occurred then a combination of at least N failures occurred*". Table 5 gives the relation between the number N and the classification of the FC.

Classification	Catastrophic	Hazardous	Major	Minor	No Safety Effect
Qualitative requirement	3	2	2	1	0

Tab. 5: Classifications and corresponding qualitative requirements

In order to model FC, we define a component called Observer, gathering necessary data of involved components. This component gets an input per data and one Boolean output per FC, the feared situation being described into the FC output assertion. The observer is a virtual component because it generally does not exist in the actual system architecture.

To count failure modes that occurred, each component has to be enriched with a failure counter (initialized to 0 and incremented when a failure mode occurs), whose value is transmitted to the observer. The observer adds all individual counters to determine the system failure mode number. The requirement is then modeled as an observer output for which the assertion is the FC assertion completed with a criterion (see table 5) on the least number of failure modes that have to occur.

As we aim at modeling abstract failure modes, summarizing failure combinations for instance, the failure counter may be incremented by more than 1 for some failure modes. In the following of this paper, this increment will be called the weight of a failure mode.

The example in figure 3 illustrates the observer, failure counter and requirement modeling concepts.

Each component is communicating its failure counter to the observer component `Obs`. The `Obs` component is also receiving the output value of the switch, symbolizing if power is generated by the system. The requirement "loss of power generation implies that at least two failures have occurred" is described as a boolean output of `Obs` whose assertion is "`Req = (OutFromS or (TotalFailNumber > 1))`" (`OutFromS` is the boolean output received from S, `TotalFailNumber` is the sum of input failure counter values)

3. Model analysis means

Industrial tools provide services to perform the analysis of the model. Automatic and interactive simulations facilitate the exploration of scenarios of events potentially leading to a FC.

Probably the most useful tool is the interactive graphical simulation. The tool describes graphically the system model and its current configuration. In the current configuration, all external events that can be triggered are presented to the user. The user selects one event, then all internal events that can be triggered are automatically triggered and the new configuration is computed so the user can look at the consequences of each failure event.

Using the simulation, it is possible to validate with the system designer that the composition of nodes, separately developed, behaves consistently with the system actual behaviour. Once the AltaRica model has been validated, the simulation allows a quick understanding of the system and its reconfigurations in case of failure occurrence.

Automatic simulation can be used either to directly generate Minimal Size Sequences (MSS) of events leading to a FC or to generate a FT from which Minimal Cut Sets (MCS) can be computed with appropriate tools. The algorithm is described in [13].

Academic tools are also available and one of the most powerful is the model-checker Mec V [14]. Mec V interprets an AltaRica node as a LTS. Loading a node A creates two domains `A!c` (state set) and `A!ev` (transition label set), and two relations `A!t` (transition relation set) and `A!init` (initial state set). Mec V is able to compute relations over several LTS, manipulating states and transitions. Mec V specification logic is formally defined and permits to transcribe easily relations between LTS.
Given a set of variable, Mec V considers all possible combinations of values as the state set but some may be unreachable so the initial relations have to define what are the reachable states and the triggerable transitions.
In order to define relations that may be further directly applicable to any couple of nodes to be compared, we first defined a set of variables that have to be initially defined:

- `initA(s)`: the set of initial states in the first node,
- `reachA(s)`: the set of reachable states `s` of the first node,
- `transA(s,e,t)`: the set of transition in the first node where `s`, `e` and `t` are respectively the initial configuration, the label and the reached configuration of the transition.

The Mec V specification logic contains the fixed point operators (least and greatest fixed points), useful to compute state sets by induction. For instance, the set of reachable configurations of a node A is composed of the initial configuration and the configurations reached after a transition from an already reached configuration. In other words, a configuration *s* is reachable if it is an initial configuration or if there exist a reachable configuration *u* and an event *e* such that a transition labeled by *e* leads from *u* to *s*, also written in Mec V specification logic as:

```
ReachA(s) += initA(s) | (<u> (ReachA(u) & <e>(A!t(u,e,s))));
```

AltaRica Refinement

1. Simulation relation

AltaRica refinement is defined in order to be able to replace, in a model, an abstract node by a concrete one while the properties of the abstract node are preserved. To be able to replace a node by another one in an AltaRica model without problems, we have to maintain the connections between the node being replaced and other nodes in the model. So we require that input and output flow variables are identical in both the abstract node and the concrete node. We do not impose constraints on the names of the events of the two nodes. This would allow refining, for instance, a node by another node made of several sub-nodes.

The AltaRica refinement we are proposing is based, given the labelled transition systems (LTS) corresponding to two models, on establishing relations between reachable states of each LTS. We consider two relation classes:

- simulation (system *A* is simulated by system *A'* if any move between states in A_{LTS} is matched with a move between states in A'_{LTS}),
- bisimulation (one system simulates the other and vice-versa).

Furthermore, the user has to define what events are to be considered as observable in each node. This is used to define:

- strong relations where any event is observable and so each transition of *A* is transitions of *A'*,
- weak relations where some events are not observable, in that case we compare sequences of transitions composed of one observable event labeled transition and possibly several unobservable event labeled transitions.

We propose to use Mec V to verify refinement relation between two nodes. To define formally these relations we use a state equivalence relation, written `eqC(s,s')`in Mec V specification logic computes the set of state pairs (`s` is a state of the abstract node and `s'` is a state of the concrete node) such that both states associate the same values to the flow variables.

From a theoretical point of view, a model *A* is simulated by *A'* iff any state *s* of A_{LTS} is in simulation relation with a state *s'* of A'_{LTS}. Two states *c* and *c'* are in relation iff:

- they are equivalent according to relation `eqC`
- for any observable transition in A_{LTS} starting from *c* there is an observable transition in A'_{LTS} starting from *c'* such that reached states are in simulation relation.

The Mec V formula for relation `Sim` computing all state pairs that are in strong simulation relation is written as a fixed point equation (cf. last paragraph of AltaRica chapter):

```
Sim(c,c') -= eqC(c,c') & ([e][t] (transA(c,e,t) => <e'><t'>(transA'(c',e',t') & Sim(t,t'))));
```

Where `transA` defines transitions in node A and `transA'` defines transitions for node A'.

Once all pairs of states in strong simulation relation are computed, strong simulation between A and A' is verified if their initial states are in simulation :

```
A_isSim_byA'(x) := x = ([s](initA(s) => <s'> (initA'(s') & Sim(s,s'))));
```

According to relation `Sim` definition, if initial configurations are in strong simulation then all reachable configurations are also in strong simulation.

In the following of this section we give a classical example of vending machine models to illustrate the strong simulation relation. We consider two models M1 and M2 that have the same declarations: three actions (`Insert_coin`, `Get_coffee`, `Get_tea`), a state variable named `choice` over domain {`None,Coffee,Tea,Both`} (initialized to `None`), a state variable named `Drink` over the same domain (initialized to `None`), and a flow variable named `Drink_obtained` (equal to `Drink`); but have different behavior: in M1, once a coin is inserted, the user is able to select his drink whereas in M2, once a coin is inserted the vending machine selects and limits the available drink.

Event	Guard	New affectations	
		Choice	Drink
Insert_coin	Choice = None	Both	-
Get_coffee	Choice = Both	None	Coffee
Get_tea	Choice = Both	None	Tea

Event	Guard	New affectations	
		Choice	Drink
Insert_coin	Choice = None	Coffee	-
Insert_coin	Choice = None	Tea	-
Get_coffee	Choice = Coffee	None	Coffee
Get_tea	Choice = Tea	None	Tea

Fig. 4: transitions table of M1 (left) and M2 (right)

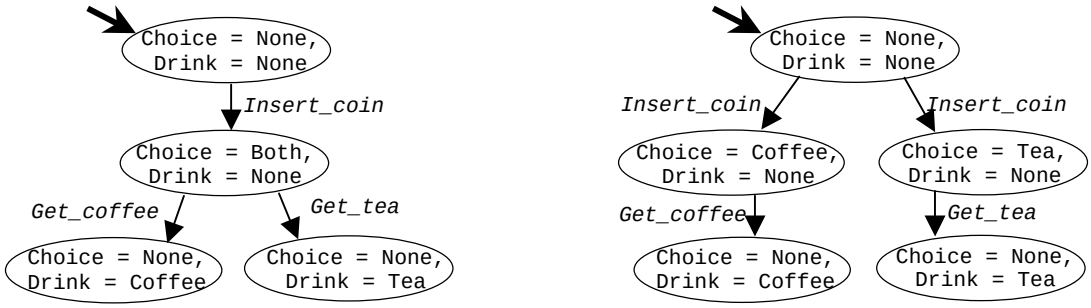

Fig. 5: Labelled Transition Systems of M1 (left) and M2 (right)

In this case the state equivalence relation `eqC` relates states of M1 and M2 that associate the same value to flow variable `Drink_obtained`. The corresponding Mec V formula is:

```
eqC(s: M1!c, s':M2!c) := ReachA(s) & ReachA'(s') & s.DrinkObtained = s'.DrinkObtained
```

Considering this equivalence criterion, any state of one of those LTS has an equivalent state in the other LTS. But regarding both LTS it is observable that:

- M2 is simulated by M1: from the initial state, for any transition selected in M2, it is possible to select in M1 a transition labeled with the same event,
- M1 is not simulated by M2: after the `Insert_coin` transition in M1, it is possible to select indifferently either `Get_coffee` or `Get_tea` whereas in M2, the `Insert_coin` transition leads to state where only one of those events may be selected.

Using Mec V, we could define a relation that would automatically compute state pairs such that M1 does not simulate M2.

2. Theoretical properties of the simulation relation

The following theorem found in [9] links the simulation relation of two LTS with the preservation of formulae that belong to the universal (ACTL*) and existential (ECTL*) fragments of Computational Tree Logic : if A1 is simulated by A2 then

- any formula belonging to ACTL* that is verified by A2 is also verified by A1,
- any formula belonging to ECTL* that is verified by A1 is also verified by A2.

The universal fragment of CTL* logic allows to express properties that are true on all sequences of events of the LTS. This fragment includes all formulae of Linear Temporal Logic that we use to formalize Safety Requirements (see [10]). So, in the following of this paper, we consider that a refined node A' preserves the properties of the abstract node A if A' is simulated by A.

G.Point established in [12] the following composition theorem: if two nodes A and A' are strongly bisimilar then node M, containing A, and node M', obtained by replacing A by A' in M, are strongly bisimilar.

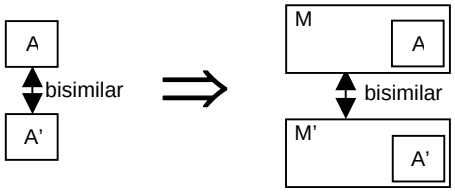

Fig. 6: Composition theorem illustration for strong bisimulation

Strong bisimulation is very restrictive because it does not tolerate any discrepancy between the initial abstract node and the refined node: adding new transitions (e.g. adding new sub-nodes induces new transitions) or deleting previous transitions (e.g. enriching guards may restrict the set of possible transitions) leads to non bisimilar nodes. So we considered that it could not be applied to Airbus industrial safety modeling problematic. We established a similar theorem for strong simulation relation: if A is in strong simulation relation with A', then M is in strong simulation relation with M', obtained by replacing A by A' in M.

These two theoretical results have an important practical importance. Suppose that M enforces a property F that belongs to ACTL* and that we replace the node A of M by A'. If A' is simulated by A then according to the second theorem, M' is simulated by M and according to the first theorem, M' also enforces property F. So, using the strong simulation relation, it is possible to replace a node by another one in a model while preserving the properties enforced by the initial model.

Refinement for safety analyses

Our goal is to apply the theoretical results defined in the previous sections in order to support the safety analysis of a system described at various levels of details. We suppose that the analyst was able to check that an abstract system model enforced its qualitative safety requirements. To achieve this, he could have generated the set of all minimal scenarios. This generation can be very time consuming. Then a more detailed model of the same system has to be analysed. To check efficiently that the new model enforces its safety requirements, we would prefer to verify that the new model refines the abstract model instead of recomputing the set of all minimum sequences for the new model. As the safety requirements we consider can be stated in the LTL logic, we know that if the concrete model is strong-simulated by the abstract model then the concrete model preserves the safety requirements of the abstract model.

We cannot directly use this approach with the models developed at AIRBUS due to the use of internal `update` events that should not be observed. So strong-simulation will not work as it is restricted to the comparison of models without unobservable events. In this case, we could either adapt the theoretical results for weak-simulation relations in order to deal with non-observable events or we could transform the LTS associated with a model using `update` events in order to remove all unobservable events and apply strong-simulation to the resulting LTS. We have explored the second approach.

In order to remove internal events and to observe only stable states, the observable LTS (LTS_{Obs}), composed of observables states linked by observable transitions, has to be computed from the initial LTS (LTS_{Ini}).

Reachable states of LTS_{Obs} are all the states in LTS_{Ini} where no internal event may happen. We connect directly any state in LTS_{Obs} with the observable state resulting from the longest sequence in LTS_{Ini} made of a failure mode transition and then any number of internal event transitions, this connection (depicted in blue in figure 7) is labeled with the name of the failure mode transition. Once this transformation is performed for the LTS corresponding to the two models under study, we can verify whether a strong simulation exists between the transformed LTS.

This property could be verified using a Mec V relation. We just need to define new relations `ReachObsA` and `transObsA` that compute the set of reachable observable states and transitions between observable states. Then we replace in the definition of `eqC` and `Sim` the instances of `ReachA` and `transA` with respectively `ReachObsA` and `transObsA` in order to obtain the new relation that will be used to verify refinement.

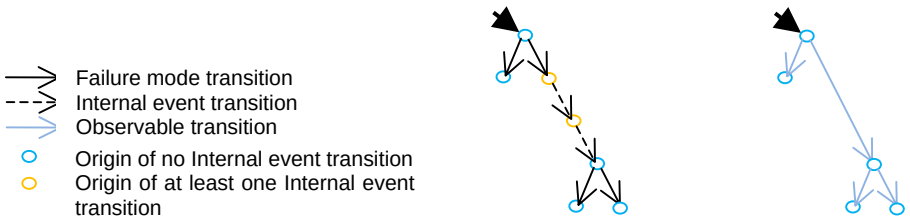

Fig. 7: from LTS to LTS_{Obs}

The following of this section presents an illustration of refinement verification. Given a power generation system composed of two DC sides, each side is concerned by a failure condition, "loss of DC power", that could be classified major. Regarding the table 5, failure combinations leading to each FC have to be at least of length 2.
The abstract model (see, figure 8) is composed of two nodes DC1 and DC2, representing respectively DC power on side 1 and side 2. Each node is composed of one boolean output nominally true and becoming false after an occurrence of a `loss` event whose weight is 2. The requirements, described in the observer component, are expressed as: the DC1 (resp. DC2) node output is true or the DC1 (resp. DC2) node failure counter is greater than 1.

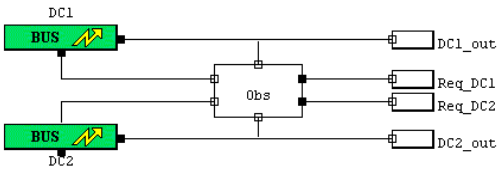

Fig. 8: Abstract model of power generation

Figure 9 presents two candidate architecture, both composed of two generators, Gen1 and Gen2, similar to DC nodes but with a loss failure event whose weight is equal to 1, and one or two switches (see AltaRica language) so that the power of generator may be provided to both DC outputs.

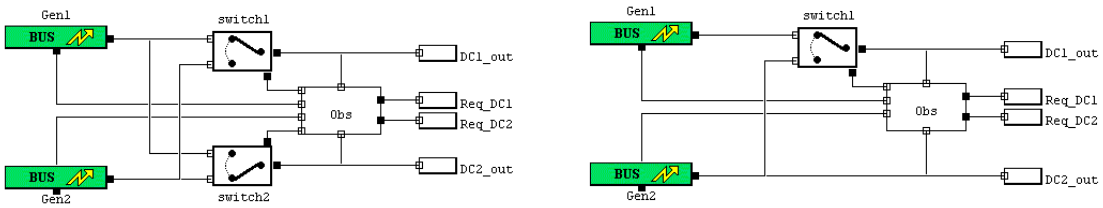

Fig. 9: Proposed architecture models M1 (left) and M2 (right)

The abstract model and both proposed architecture models have been written in AltaRica in order to perform a verification using Mec V. As M1 and M2 contain switches that have an `update` event, it is necessary to compute the observable LTS. All successive relations to compute the observable LTS and then to check if it is simulated by the abstract model are computed instantaneously. The results are that M1 is a refinement whereas M2 is not. Thanks to mec V, it appears that the loss of DC power on side 2 may be reached with a single failure, loss of Gen2, so the corresponding requirement is not fulfilled whereas this requirement was always fulfilled on the abstract model.

Multi-system safety assessment

In this last section we explain how we intend to use refinement in order to support the multi-system safety assessment approach that was described in the first section of this paper. We want to model systems in interface at various levels of detail and perform analysis combining systems models at different levels of detail. We defined 3 levels for models of interfaced system:

- At the most abstract level, the model describes the requirements and defines the interfaces, both in an interconnection perspective and in a requirement verification perspective. The inputs and the outputs are those considered by interfaced systems and conserved by refined models. The failure modes modelled are exactly the Dependent System Failure defined by the safety analyst.
- At intermediary level, the model of the system architecture is simplified in order to contain main components and to describe main reconfiguration logics. A failure mode may be a basic failure mode or symbolize a combination of sub-component failure modes.
- At the finest grain of detail level, the system architecture is fully detailed: the model contains all components and all reconfiguration logics are described. All failure modes are basic failure modes.

The model on figure 8 is an example of the most abstract level model of electrical system while model M1 on figure 9 is an intermediary level model of the same system. A fine level model is presented on the left part of the figure 1.

Performing a multi-system analysis, on the basis of a model where all systems are detailed, would certainly be very time-consuming as execution time of algorithms for the generation of minimal failure scenarios depend on the size of the models. Furthermore, the safety analyst interested in one system does not want to observe all the details of failures of systems in interface. In order to prevent those difficulties, we take benefit of the composition theorem related with the strong-simulation relation as illustrated on following figure:

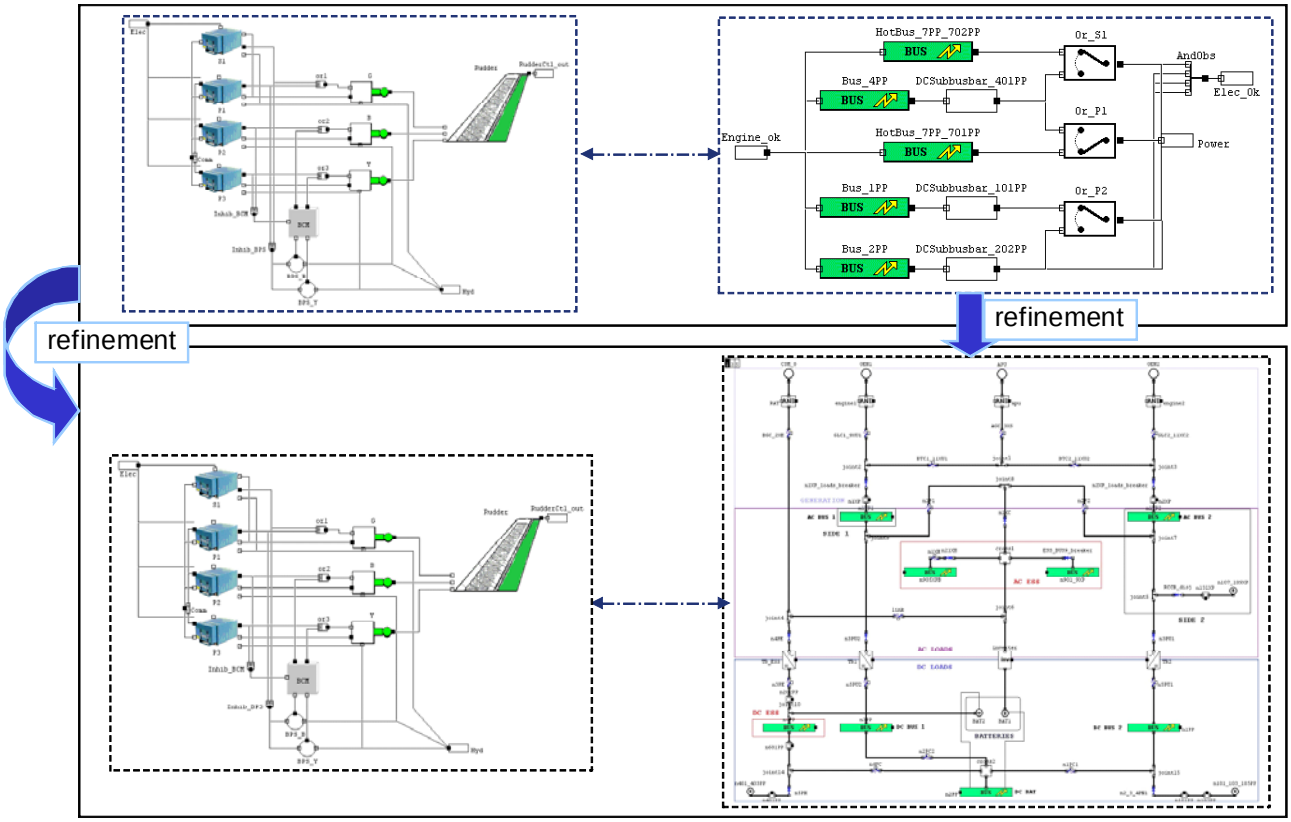

Fig. 10: Refinement application for multi-system safety analysis

The system is initially studied with a detailed model of the flight control system and an intermediary model of the electrical system. The electrical system model is based on a preliminary architecture and fulfils requirements elaborated by the flight control system. As the electrical system architecture becomes more precise, new intermediary electrical system models are built. Each new intermediary model has to refine the initial abstract model, up to the finest level of detail model. The verification that an electrical system model refines another one

guarantees that the requirements fulfilled on the initial multi-system model remains valid on the new multi-system model, avoiding the full requirement analysis. Following this approach, the safety analyst in charge of the flight control system would focus on the verification of the design of this system, the verification of the refinement relation between various versions of the electrical system should be performed by the team in charge of the safety assessment of the electrical system.

Conclusion

In this paper we have proposed to use strong-simulation as a candidate for a notion of AltaRica refinement. We explained how the Mec V model-checker can be used to verify that two AltaRica models are linked by this refinement relation. We showed that some theoretical properties of this relation could be used to assess the qualitative safety of systems described at different levels of details. Furthermore, we explained also how refinement can be used to deal with multi-system safety assessment.

Other works [5] [7] have also investigated the relations between refinement and safety. They often use refinement to relate an abstract model that is failure-free and a model with failures. Their goal is to prove that all failures in the second model are masked by appropriate mechanisms and consequently the model with failures is equivalent to the model without failures. Our use of refinement is different as we aim at relating models including failures.

One perspective for our studies is to check the effectiveness of this approach on a complete industrial-size case-study. We would like to compare two approaches used to assess whether a model is refined by another one : one that is based on computing and comparing sets of minimal scenarios and the approach we proposed in this paper that is based on using a model-checker to check the strong simulation relation between models. Another perspective is to assess whether quantitative safety assessment could also benefit from the refinement approach that we proposed.

References

[1] J.R Abrial, The B-Book, Assigning Programs to Meanings, Cambridge University Press, 1996.
[2] O.Akerlund et alter, ESACS : an integrated methodology for design and safety analysis, ESREL, 2003.
[3] O.Akerlund et alter, ISAAC, a framework for integrated safety analysis of functional, geometrical and human aspects, ERTS, 2006.
[4] A.Arnold, Systèmes de transitions finis et sémantique des processus communicants, Masson, 1992.
[5] R.Banach, R. Cross, Safety Requirements and Fault Tree using Retrenchment, Safecomp 2004.
[6] R.Bernard et alter, Experiments in model based safety analysis : flight controls, DCDS, 2007.
[7] J.Elmqvist, S. Nadjm-Tehrani, Safety-Oriented Design of Component Assemblies using Safety Interfaces, FACS 2006
[8] A.Griffault, S.Lajeunesse, G.Point, A.Rauzy, J-P.Signoret, P.Thomas, The AltaRica language, International Conference on Safety and Reliability, ESREL, 1998.
[9] O. Grumberg, D. Long, Model-checking and Modular Verification, ACM transactions on Programming Languages and Systems, vol. 16, n° 3, 1994.
[10] C.Kehren et alter., Vérification par model-checking d'un système électrique, Lambda-Mu, 2004.
[11] J.McDermid, D.Pumfrey, A development of hazard analysis to aid software design, COMPASS 1994.
[12] G.Point, AltaRica: Contribution à l'unification des méthodes formelles et de la sureté de fonctionnement, PhD Thesis, 2000.
[13] A.Rauzy, Modes automata and their compilation into fault trees, Reliability Engineering and System Safety, 78:1-12, 2002.
[14] A.Vincent, Conception et réalisation d'un vérificateur de modèles AltaRica, PhD Thesis, 2003.