Tiered Model-Based Safety Assessment

Kevin Delmas, Christel Seguin, Pierre Bieber

Tiered Model-Based Safety Assessment

Kevin Delmas, Christel Seguin, and Pierre Bieber

ONERA/DTIM Université de Toulouse, 2 av. E. Belin, 31055 Toulouse, France
firstname.lastname@onera.fr

Abstract. Processes and techniques used for assessing the safety of a complex system are well-addressed by safety standards. These standards usually recommend to decompose the assessment process into different stages of analysis, so called tiered safety assessment. Each analysis stage should be performed by applying recommended assessment techniques. To provide confidence in the correctness of the whole analysis, some verification techniques, usually traceability checking, are applied between two stages. Even if the traceability provides some confidence in the correctness of the decomposition, the following problems remains How to model the system behaviours at each stage of safety assessment? How to efficiently use these stages during the design process? What is the formal relationship between these modelling stages? To tackle these problems, we propose a way to specify, formalize and implement the relations between assessment stages. The proposal and its pros & cons are illustrated on a Remotely Piloted Aircraft System (RPAS) use-case.

1 Introduction

The development of smart transport systems is classically addressed by tiered processes (for instance V cycle) where product specifications are successively refined until the final implementation phase.

The components of these systems may be subject to various kinds of faults which may result in unacceptable safety issues. The safety effects of the potential faults must therefore be carefully analysed and their acceptability assessed. Achieving such activities with a monolithic safety assessment may be tedious, error-prone, and would fail to provide confidence in the system dependability.

Standards like ARP-4754 [17] (aeronautics domain) provide a safety process used successfully in the industrial domain to perform the safety assessment of complex systems. One key of its success is the tight coupling of the development and safety processes, that is in each development tier the relevant safety assessment are processed and fed the next development steps with analyses results. A safety process therefore generates analyses made at various level of granularity and addressing various kinds of systems and components.

In addition to classical methods like fault trees or Markov chains (available in [18]), newer approaches like Model Based Safety Assessment (MBSA) can be used to perform the mentioned analyses. Altarica [2] is one of the most successful safety modelling and assessment language used in an industrial context [12].

Besides the modelling and assessment ease brought by these approaches, ensuring the consistency and traceability of safety analyses is still a prominent problem.

We claim that, like MBSE can been used [20] to tackle traceability and consistency problems during development, the MBSA offers a suitable environment to produce consistent and traceable safety analyses for complex systems.

The contribution of this paper is twofold, first it provides a methodology to model the relations between development tiers, second it formalises these relations with Altarica to benefit from the automated safety assessment. The remainder of the paper is organised as follows: 1. the classical safety process and its resulting analyses are succinctly presented (section 2) and the need of traceability and consistency is motivated; 2. our modelling methodology and its implementation using Altarica are presented (section 3); 3. the benefit of our methodology (section 4) is demonstrated on a simplified RPAS use-case; 4. eventually the related work on safety analysis of complex systems is detailed (section 5).

2 Safety process and MBSA

Complex systems are usually developed using a tiered process, that is some design and validation activities are performed at each stage of development and then fed the subsequent stages. Safety activities are performed throughout the design process. Designers can rely on classical formalisms (for instance listed in the ARP-4761 [16]). As identified by [14], the classical formalisms like fault-trees or Markov chains embrace an architecture-agnostic modelling. Hence, a tedious abstraction work must be achieved by the safety engineers to derive the safety models out-of the system specification. Furthermore, adapting the safety models after an evolution of the system design may be cumbersome and error-prone. Architecture-aware formalisms, like component fault trees [8], finite automata [4], mode automata [2] or hierachic safety assessments [13] have been introduced to overcome the limitations of classical formalisms. Architecture-aware formalisms provide a way to define the dysfunctional behaviour of entities called *components* that are instantiated and connected to build the architecture of a *system*. Ultimately this interconnection of components can be analysed by automatic solvers like [15,3], this kind of analyses is the so-called Model-Based Safety Assessment.

2.1 Reminder on MBSA and Altarica

Amongst the possible formalisms, Altarica [2] is one of the most popular and successfully applied MBSA language in both academic and industrial fields. Since Altarica is a formal language, its behaviour can be simulated and automated safety assessment (like [15]) can be performed. Therefore, the underlying language used in the models presented in this paper is Altarica and their analysis is performed by the tool Cecilia-OCAS [5] from Dassault Aviation.

A system modelled with Altarica is a set of interconnected components, these connections are considered as constraints over the possibles values of the inputs and outputs of the components. These components therefore own the following elements: 1. *flow variables* the inputs and outputs that are used to

interface the node with its environment; 2. *state variables* the internal variables that can encode node's functional or dysfunctional state (*e.g.* failure modes); 3. *events* the elements used to trigger the transitions amongst node's states, note that these events can be deterministic (*e.g.* reconfiguration events) or probabilistic (failure events). The node's functional and/or dysfunctional behaviour is defined by the following relations between states, flows and events: 1. *transitions* encode the possible state evolutions, each transition written $g \vdash e \rightarrow a$ informally means "when the guard g (condition over the current state and the value of the flow variables) is true when the event e is triggered then the action a is performed (assignment of state variables)"; 2. *assertions* encode the constraints between the possible values of flow and state variables. In the sequel we assume that the output flows are defined by the inputs flows and the state variables (*dataflow* restriction).

2.2 Reminder on the safety assessment process

To ensure the dependability of complex systems, the safety assessment process must be tailored to the development process. The standards like ARP4754 and ISO26262 promote a safety assessment process where various safety activities are performed throughout the development process.

Generally, a safety plan can be seen as an application of the following safety assessment pattern at various levels:

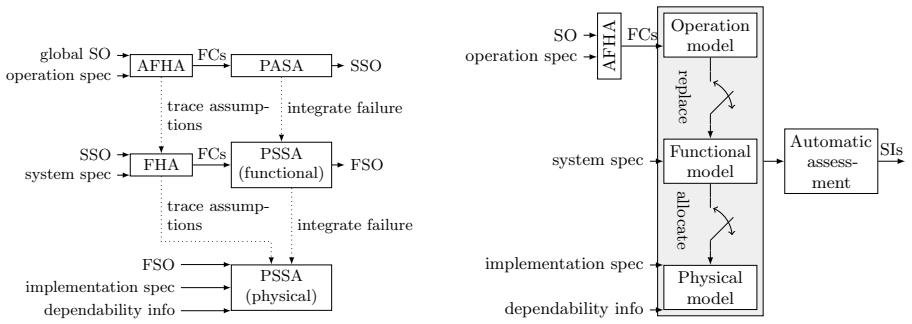
Hazard Analysis (HA) Identify the conditions, in a given context, that may rise safety issues so-called *failure conditions* (FC) and allocate *safety objectives* (called SOs) to these failure conditions commensurate with the hazard's severity (or the previous safety objectives). These safety objectives are bounds over safety indicators (called SIs) such as the acceptable minimal number of root failures of a FC, the upper bound of a FC occurrence rate.

Safety Assessment (SA) Assess the proposed architecture against the objectives and derive from this assessment the safety objectives that must be met by the subsequent architectures designed during the development process.

For instance, during the aircraft specification definition an Aircraft Level, Functional Hazard Assessment (AFHA) is performed and provides the failure conditions and their severity. The *context* is specified through a set of assumptions that must be traced and ultimately confirmed to ensure the validity of the analysis.

Once the aircraft's sub-systems are known and their dependencies identified, then a Preliminary Aircraft Safety Assessment (so-called PASA) is performed. This analysis provides the appropriate safety objectives that must be fulfilled by the aircraft's systems.

The applicant must then demonstrate the fulfilment of the PASA safety objectives on each system with various assessments performed throughout the system's development process. In the sequel we will adopt the following three-stage development process (used in [7]): 1. operational: considering the system failures and their impact in its operational environment (AFHA/PASA); 2. functional: considering the safety impact of function failures on the system's environment (FHA/PSSA



(a) Classic safety assessments (b) Tiered Model-Based Safety Assessment

Fig. 1: Safety assessment processes

functional); 3. physical: considering the safety impact of implementation failures on the function or system environment (PSSA physical).

2.3 Relations between safety assessments

When moving from one stage to another one, failures of the former architecture will depend on the failures of the components of the new architecture. The figure 1a identifies the *relations* between assessment performed during the safety activities. The explicit relations (plain arrows) are the data exchanged by the analysts to perform the safety assessments. For instance, performing the FHA of a system needed some knowledge of the system specification and the safety objectives allocated to assessed system. Some other relations are implicit (dotted line), since they fall under the expertise of the analyst to properly deriving and tracing some piece of information between stages. For instance, the analyst must identify the failure conditions (expressed over functions) that are related to a system failure mode for which a safety objective has been allocated.

According to the figure 1a, we can identify two kinds of implicit relations:

Allocations express the dependencies between components of two architectures, such as the the resource dependencies created by mapping functions on physical components.

Replacements link some parameters of two assessments performed at various stages, typically the environment assumptions between AFHA and FHA.

The validity of the global safety assessment relies on the identification and tracing of these relations. The prominent threats to validity are:

Traceability If the analyst is not able to trace the dependencies or the links between assessments' parameters, the demonstration of the global safety objectives can be compromised by inconsistencies between safety assessments.

Composability The assessments performed throughout the design process are dedicated to the identification and assessment of precise kinds of failures *e.g.* operational specification failures for PASA. The fulfilment of the global objective is achieved by a composition of the safety assessments. Some piece of information of an assessment may be considered during subsequent assessments. If the assessment framework does not provide some mechanisms to handle such dependencies between assessments, the analyst must handle them manually ; that may be error-prone and time-consuming.

Maintainability The safety assessment is more likely to be an iterative process than a linear process. The analyst needs to efficiently reflect the evolution of the design on the safety assessments. Because of the system's complexity the safety impact of a design choice may spread way beyond the considered architecture, so handling system's evolution manually is error-prone and can compromise the whole safety assessment.

To tackle these problems, we present an MBSA approach extended with a formal modelling of the relations between the safety assessments built throughout the design process, so-called Tiered Model-Based Safety Assessment.

3 Formalisation of the tiered safety assessment with Altarica

We introduce the notion of Tiered Model-Based Safety Assessment providing a formal modelling of the relations between the safety assessments identified in the previous section.

3.1 Overview of the approach

The formalisation provides a convenient way to express replacement and allocation relations between two architectures . Thanks to this modelling, the relations between safety assessments (identified in the figure 1a) can be greatly reduced and boils down to the ones depicted in the figure 1b.

The formalisation mainly addressed the relations PASA / functional PSSA and functional PSSA / physical PSSA. The former can been seen as a replacement relation where the failure modes of the systems are expressed as failure conditions over the functions of the systems. The latter is an allocation relation introducing the dependencies of functions on physical items. Thanks to MBSA, these relations can be modelled as constraints between the models of various architectures and the safety assessment can be delegated to an automatic solver. Since the relations between architectures will be formally modelled, the failure conditions contributing to the global hazards identified in the AFHA does not need to be further refined in subsequent architectures. One just enables the relations that must be considered according to the assessment level and performs its assessment (whatever the level) on the failure conditions of the operational level. Therefore the safety objective allocation phase and the adaption of system failure as function failure conditions (FHA) are not needed any more. Beyond this simplification, the proposed process provides a solution to the following threats of validity:

Traceability By formalising the relations between the models, the analyst will need to identify and express them to perform the safety assessments. So the relations between safety assessments and the assumptions made on the impact of component failures on another architecture are natively traced.

Composability The analyst can perform an assessment using the information dispatched over several models since the relations are formally expressed. The relations can then be activated to perform a safety assessment of the aggregation of the safety knowledge contained in the models.

Maintainability Eventually the safety impact of any evolution of an architecture (at any level) is automatically considered during safety assessment since its impact will spread through the relations between models.

3.2 Relation specifications

To properly integrate the notion of relation in the safety assessment models, one must provide a formal definition of such relations.

Definition 1 (Notations and Modelling assumption). *Let us consider that the behaviour of a component c is described by a set of state transitions T_c and an output function σ_c providing the output failure modes according to the inputs valuations and the current state. Let the valuation of a variable x be denoted by $V[x]$ and the Cartesian product of the valuations of a set X of variables be denoted by $V[X] = \prod_{x \in X} V[x]$. Let $V_{x \mapsto a}$ be the extension of V with $x \in X$. To introduce the definitions of the relations we assume that if a component c can fail then its possible failure modes are encoded by a unique state variable S of type FM containing the failure modes and a mode encoding the correct state denoted by ok which is the initial value of S.*

Allocation The purpose of an allocation relation is to consider the resource dependencies of a component, in addition to its own failures. Let c be the initial component modelling the component without considering the allocation dependencies. Adding an allocation relation can be seen as a transformation of c into another component a. Let R be an input of a providing the failure mode of the component's resource, then the transitions remains unchanged, thus $T_c = T_a$ and when the resource dependencies are fulfilled ($R = ok$) then for any valuation V of I and state s we have $\sigma_a(V_{R \mapsto ok}, s) = \sigma_c(V, s)$.

Replacement The purpose of a replacement relation is to replace the spontaneous occurrence of a failure mode (encoded in S) by a some function over the component failures of another architecture. So let us consider the initial component c modelling the component without considering the replacement relation. Replacement can be seen as a transformation of c into another component r. Let R be an input of r providing the new failure mode then the transitions of c involving S assignment must be no more fireable in r for any valuation of the state and inputs. Moreover for any state s and valuation V of $I \cup R$ such that $s = V[R]$ then $\sigma_r(V) = \sigma_c(V[I], s)$. Note that σ_r does not depends on S any more since it is totally replaced by R.

3.3 Modelling relations

The formalisation of the relations is founded on dependency modelling through ALTARICA flows. These flows carry the information gathered from one architecture model to another one. The integration of the information in the targeted architecture model is achieved by flow connection to the standard interface of the definition 2.

Definition 2 (Relation interface). *Let c be a component, FM be its failure modes then the following inputs must be provided by c:*

Activation (A) *is a boolean input enabling the failure mode transitions.*
Resource (R) *is an input of type FM providing the failure mode of the underlying resources used by c.*

From the specification of the replacement and allocation relations, one can transform any components c satisfying the assumptions of the section 3.2 into a component c' that can be used to encode the replaced and allocated version of c. The activation of the desired relation is based on the A and R inputs of the interface.

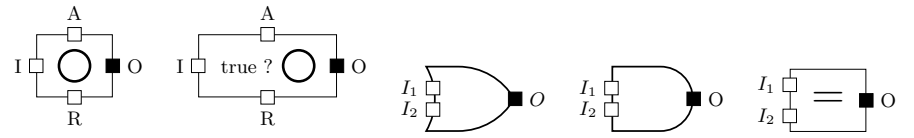
Definition 3 (Interface implementation). *Let c be a component, T_c be its transition set, σ_c its output function, T_E be the set of transitions containing an assignment of S, V be a valuation of $I \cup R$ and s be the current state. The transition set $T_{c'}$ and output function $\sigma_{c'}$ of the adaptation c' of c implementing the interface can be defined as follows:*

$$T_{c'} = \{g \wedge \neg A \vdash e \to a | g \vdash e \to a \in T_E\} \cup (T_c \setminus T_E)$$
$$\sigma_{c'}(V,s) = \begin{cases} \sigma_a(V,s) & \text{if } A \\ \sigma_c(V[I], V[R]) & \text{otherwise} \end{cases}$$

No relation If the component is not linked by a replacement nor an allocation relation then its internal failures (S) and inputs (I) only impact its outputs, so the activation input should be set to *true* and the resource input should be *ok*. Since A is always true then in any transition of c', $g \wedge A \Rightarrow g$ so $T_{c'} = T_c$. Moreover, for any valuation V of $I \cup R$ and state s we have $\sigma_{c'}(V,s) = \sigma_a(V,s)$, in addition when $R = ok$ we also have $\sigma_a(V[I]_{R \mapsto ok}, s) = \sigma_c(V[I], s)$ so $\sigma_{c'}(V,s) = \sigma_c(V[I], s)$ holds.

Replacement When a component failure modes are replaced, A must be set to *false* and R connected to the replacement function. Since $A = false$, for all transition $g \vdash e \to a \in T_{c'}$ containing an assignment of S we have $\neg A \Rightarrow \neg g$ hence the transitions encoding the failure evolution of c are not fireable. Furthermore, let $\sigma_r(V) = \sigma_c(V[I], V[R])$ then we have $\sigma_{c'}(V,s) = \sigma_r(V)$ so for any state s and valuation V of $I \cup R$ such that $s = V[R]$ we have $\sigma_{c'}(V,s) = \sigma_r(V) = \sigma_c(V[I], s)$.

Allocation When component resource dependencies are considered A must be set to *true* and R connected to the allocation function. Since $A = true$, we have $T_c = T_{c'}$ and the output function is $\sigma_{c'}(V,s) = \sigma_a(V,s)$. By definition of σ_a for any state s, valuation V of $I \cup R$ where $V[R] = ok$ we have $\sigma_a(V,s) = \sigma_c(V[I], s)$ so $\sigma_{c'}(V,s) = \sigma_c(V[I], s)$ holds.



(a) Generic Block (b) Alarm Block (c) Best operator (d) Worst operator (e) Equal operator

Fig. 2: Standard components

```
node generic_block
 flow //interface flows
  I : FM : in; //input FM
  O : FM : out; //output FM
  R : FM : in; //alloc or replace FM
  A : bool : in; //activate
 state //component FM
  S : FM;
 event //one event per fm in FM
  e_{fm_1}, ..., e_{fm_n};
 init //initially component is ok
  S := ok ;
 trans
 //failures are permanent
  S = ok and A ⊢ e_{fm_1} → S := FM_1;
  ...
  S = ok and A ⊢ e_{fm_n} → S := FM_n;
 assert
  //output FM is the worst of
  //R, S and I
  O = worst(I, worst(R, S));
end
```

Listing 1.1: ALTARICA code of the generic Block

3.4 Standard components

The ALTARICA models presented in the remainder of the paper are build on top of a library of generic components providing the interface of the definition 2. The *fallible* components of the figures 2a and 2b are named *blocks* (graphically discriminated by an internal circle) and provides the relation interface. Conversely the *infallible* components of the figures 2c and 2d, named *operators*, does not provide the relation interface. The operator best of figure 2c (resp. worst of figure 2d) provides the lowest (resp. greatest) failure mode amongst I_1 and I_2 according a total order $<$ over the failure modes. A possible definition and concretisation of the generic block is provided by the example 1.

Example 1 (Interface implementation). The generic block is generic over the set of failure modes (FM) and thus the ALTARICA code 1.1 must be concretised with a given failure mode set to obtain the ALTARICA model of this block. As requested by the definition 2 the initial state is *ok*. To fulfil the *Transition* constraints for allocation and replacement, the transition's guards complies to the definition 3. The function worst is used whatever the value of A *i.e.* $\sigma_{c'} = \sigma_a = \sigma_c$. Nevertheless, when A is true we have $\sigma_{c'} = \sigma_a$ and when A is false we know that $S = ok$ so $\sigma_{c'}(I, R, S) = worst(I, worst(R, ok)) = worst(I, R) = \sigma_c(I, R)$. So this implementation complies to the definition 3.

In the sequel we will consider that blocks own the following generic failure modes: the block does not provide its intended behaviour (called *lost*); the block provides an erroneous behaviour (called *err*). A system can then be a concrete block where $FM = \{ok, err, lost\}$.

3.5 Decomposing analyses

A safety assessment considering only the component failures of a specific architecture is obtained by deactivating all the components of the other architectures and building replacement relations. The only contributors to the high-level hazards will be the component failures of the target architecture. Through the replacement relation, the analyst will benefit from the failure propagation modelled in the higher level architectures to perform its safety assessment.

If the analyst want to consider the failure of several architecture levels simultaneously then the considered components must be activated and an allocation relation must be defined between the considered levels. For instance, such an analysis on functional and physical levels can provide the combination of function specification and physical failures that may contribute to top level hazards.

4 Safety assessment of an RPAS system

Let us illustrate the modelling framework on a simplified remotely piloted aircraft system (RPAS) [1]. The drone's mission is to inspect an infrastructure locate in a pre-defined evolution zone. Since some populated areas located nearby, the drone should not fly, land nor crash outside the evolution zone.

If one wants to use such system, the hazards inherent to the RPAS must be identified and their likelihood demonstrated as acceptable. To do so, the hazards should be identified out-of the failure modes of the top level functions of the RPAS that are *Control Flight i.e.* stay in the evolution zone and *Abort Flight i.e.* detect the conditions where motors must be cut off.

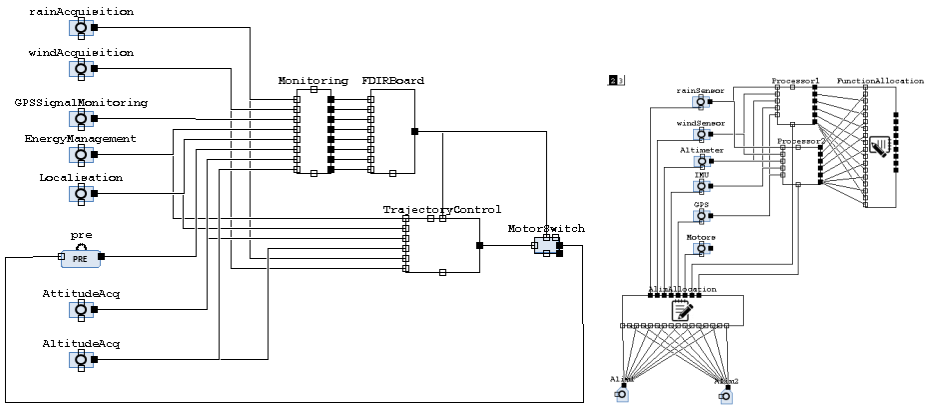
The severity of a failure condition is classified using a severity scale derived from the ARP4754, here let us only consider *Catastrophic* as a potential ground or in-flight collision leading to one or several fatalities and *Hazardous* as a controlled crash in a predefined zone without stringent access control. The simplified AFHA of the table 1 provides an assessment of the safety impacts.

In addition, the AFHA must provide the safety objectives attached to these failure conditions, in the remainder of this paper we consider that Catastrophic failure conditions must not be reached by single failures. We will not illustrate the safety assessment based on quantitative measure. Note that one can easily perform such quantitative assessment with the minimal cutsets computed for each architecture. Consequently we will assess only the failure condition *Fly away without flight abortion capability* (CAT) that could lead to collision with vehicles or other aircraft.

4.1 Operational-level assessment

The RPAS is constituted of a Flight Controller System (FCS) managing the flight plan and the trajectory of the drone. The Flight Termination System (FTS) monitors the FCS and can reconfigure the FCS to mitigate its failure, the ultimate action of the FTS is to trigger a controlled crash to avoid a fly-away. The identified

[1] available at `www.onera.fr/sites/default/files/274/IMBSA2019code.zip`



(a) Functional Architecture (b) Physical Architecture

Fig. 3: Functional and physical architectures

failure conditions are encoded as observers over the FCS and FTS systems as follows: 1. a fly-away occurs when the FCS provides an erroneous control of the drone and the FTS is not able to trigger a controlled crash; 2. a crash in the zone occurs when the FCS is lost or if the FTS triggers a controlled crash. The minimal cutsets of these failure conditions has been generated automatically by Cecilia-OCAS:

$$MCS = \{\{FCS.err, FTS.lost\}\}$$

So at this stage the no single failure mode requirement for Catastrophic failure condition is fulfilled. Nevertheless the latter result holds if no common mode of failures are added during functional and physical architecture design.

4.2 Functional-level analysis

The functional architecture is depicted by the figure 3a wherein *Acquisition functions* acquire flight parameters and monitor adversary conditions; *Monitoring* acquires data are checked by independent alarms; *TrajectoryControl* functions controlling the drone from flight parameters and control mode; *MotorSwitch* cutting motors' power supply if the flight termination mode is selected.

Function	Failure	Context	Consequences	Severity
Control Flight	*err*	cannot abort flight	Crash outside evolution zone	**Catastrophic**
		can abort flight	Crash inside evolution zone	**Hazardous**
	lost	-	Crash inside evolution zone	
Abort Flight	*lost*	cannot control flight	Crash outside evolution zone	**Catastrophic**
		can control flight	No safety effect	**NSE**
	err	-	Crash inside evolution zone	**Hazardous**

Table 1: Simplified AFHA of the RPAS

The last node called **FDIRBoard** encodes the on-board safety policy that selects the control mode according to the alarm states. The selection rules are coded as an ALTARICA automaton selecting the control mode according the alarm states. More precisely, at any time flight termination is chosen when the attitude **or** trajectory are not correct, otherwise emergency landing is chosen if the rain **or** wind **or** altitude **or** energy are not correct, otherwise hovering mode is chosen in case of loss of GNSS **or** localization, otherwise the mission mode is selected.

To perform the safety assessment, the analyst must replace the operation failure modes by some failure conditions over the functional architecture. To achieve that we saw that the component must be deactivated and the new failure mode must be provided trough R. The replacement relation considered is 1. the trajectory state provides the state of the FCS; 2. when the trajectory or the attitude estimation is not correct then the flight termination must be triggered and the switch should cut the motor otherwise the FTS does not works properly.

Thanks to the replacement relation, the analyst can compute the following cutsets integrating the safety knowledge of the functional architecture. The result shows that the functional architecture does not integrate common mode of failure for the Catastrophic failure conditions. The analyst can then allocate these functions on physical resources.

$$MCS = \left\{ \begin{array}{ll} \{GPSSignalMonitoring.err, & TrajectoryControl.Pilot.err\}, \\ \{Localisation.err, & TrajectoryControl.Pilot.err\}, \\ \{MotorSwitch.lost, & AttitudeAcq.err\}, \\ \{MotorSwitch.lost, & TrajectoryControl.Pilot.err\} \end{array} \right\}$$

4.3 Physical-level analysis

The physical architecture shown by the figure 3b is composed of two processors executing the software, sensors, motors, and two power supply channels.

Each acquisition function is allocated both on a processor and on a sensor. The monitoring, control mode selection, trajectory control and abort flight are implemented as software executed on the processors. The trajectory management additionally depends on the motor to control the drone's trajectory. Let us consider that the analyst wants to consider only one processor and power supply channel in the physical architecture. The safety assessment considering this allocation relation can be assessed by computing the new cutsets, for the sake of readability we display the cutsets containing only physical failures.

$$MCS = \{\{Alim1.err\}, \{Processor1.err\}, \{Alim1.lost, IMU.err\}\{Processor1.lost, IMU.err\}\}$$

Allocating all components on the same power supply and processor produces a common mode of failures identified by the minimal cutsets generator. The analyst must reconsider its allocation relation to avoid such a single point of failure. For instance, allocating the monitoring, FDIR and MotorSwitch on the second processor and the other software on the first one. Hence the second processor should be powered by the second power supply. The validity of the reallocation

is assessed by recomputing the minimal cutsets:

$$MCS = \left\{ \begin{array}{ll} \{Alim2.err, Alim1.err\}, & \{Alim2.err, IMU.err\}, \\ \{Alim2.err, Processor1.err\}, & \{Alim2.lost, Alim1.err\}, \\ \{Alim2.lost, IMU.err\}, & \{Alim2.lost, Processor1.err\}, \\ \{Processor2.err, Alim1.err\}, & \{Processor2.err, IMU.err\} \\ \{Processor2.err, Processor1.err\}, & \{Processor2.lost, Alim1.err\}, \\ \{Processor2.lost, IMU.err\}, & \{Processor2.lost, Processor1.err\} \end{array} \right\}$$

5 Related work

Tiered safety assessment processes propose to decompose the global task in several easiest sub-tasks to master the analysis of complex systems. Each sub-task uses a specific model for an analysis which is focussed on an abstract system view or a more detailed subpart. The issue is to ensure the maintainability, traceability and composability of all these models and analyses.

We explored in this paper the use of a unique model which can progressively integrate several models, while keeping possible the analyses of the model subparts at the relevant granularity level. We used of course the composition and hierarchy features of AltaRica. However, this is not enough. Safety models are not limited to structures: they encompass more or less sophisticated failure propagation logics. So our main contribution was to clarify the logical dependencies between the subparts of interest and to show how they can be encoded to ease the model update and its tiered analysis. A difficulty was to do it in a way which preserves the analysis tractability and the results readability for all tiers of the process.

This is rather original. Indeed, the mainstream idea of the literature is to handle the maintainability, traceability and composability of all the sub-models and analyses by characterizing the relations between analyses made at different design stages, more or less formally and outside the models.

5.1 Relation through refinement

The approaches propose to consider a complex system as a *layered system*. For instance in [19], a framework of safety modelling is for layered safety mechanisms is implemented using event-B [1]. The notion of layer can encompass various meaning, in [19] and [9], a layer is a model of a safety mechanism handling failures either locally, or by using dedicated safety mechanisms (sub-layers) or by invoking more general-purpose safety mechanism (up-layer). Instead of performing safety analyses for each layer of safety mechanisms and handling manually the relations between them, the authors of [9] propose to formalise the layer hierarchy with the notion of *refinement* of event-B. Using such a framework enables the designers to formalise the behaviour of fault-tolerance mechanisms and to perform a global formal analysis for some fault-tolerance properties.

Another notion of layer is exploited in [6] as a way to represent several *abstraction* stages. The proposed framework is based on component fault trees where the

user can define abstract component fault trees. A notion of *concretisation* can then be used to provide a realisation of a specification. The framework assists the analyst be providing automatic consistency checks.

As shown by the presented works, providing a formal notion of refinement is a way to ensure the maintainability, the traceability and the composability of the assessment. Nevertheless, the refinement preserve logical properties and it does not offer any guaranty on the preservation of probabilities. Our approach does not pretend to solve this issue. However, the replacement and allocation enable a quick computation of cutsets and associated probabilities.

5.2 Relation through synchronisation

The approach of [12] addressed the safety assessment of tiered system by focusing their effort on the formalisation of the allocation between architectures produced at various design stages. The authors propose to use the MBSA to model the dysfunctional models of these architectures in a single model. The allocation is then formalised through the notion of synchronization provided by the mode automaton formalism (more information on synchronisation can be found in [14]).

Allocation through synchronisation is an efficient and light way to model the dependencies between components of various architectures. Nevertheless, an allocation dependency can be considered as an arbitrary complex function of resource failures, for instance the dependency of a function on its implementation resource can be a set of resource failure combinations. Unfortunately the synchronisation language expressivity limits the modelisable dependencies. Moreover, the synchronisation are not *oriented*, so it is not possible to encode that a failure event is caused by a combination of failure events (as a safety analyst may want to represent the resource dependency). Usual dependencies like common causes are not sufficient to cover the full spectrum of dependencies of an allocation.

5.3 Relation through traceability

Another kind of approaches like [11] relies on the traceability between the safety and design process. In this work the analysis and design phases are modelled as UML elements and the relations between them are explicitly modelled. The analyst is then able to link the designed architecture to the corresponding safety model. The traceability between the safety and design is used to ease the manual checking.

The approach of [7] proposes to analyse complex system using MBSA. The idea is to decompose the system's architecture at various levels straightforwardly linkable to the design phase of the system. At each level the failure conditions and the dependencies between the components of the architecture are modelled. Note that a model may embed some information from an upper level. Furthermore, the failure conditions expressed at a given level must be refined at subsequent analysis levels. The traceability and consistency between models is then manually handled. Some methods like [10] can be used to link component failures and then compare the minimal cutsets produced by the safety assessments.

Such approaches suffer from the following issues: 1. the integration of some information of an upper level architecture is tedious and can generate some inconsistencies between models if the information is not integrated properly; 2. without formal modelling of the relations the maintainability is not addressed; 3. manual handling of the traceability for complex system can be the source of inconsistencies between models and assessments. Nevertheless, our approach can be seen as an extension of the approach of [7] wherein the architectural models are formally linked through allocation and replacement relations.

6 Conclusion

Summary The tiered safety assessment is recommended by the safety standards to master the complexity of assessment of complex systems. This recommendation is currently implemented by performing separately complementary fault tree analysis or failure mode and effect analysis and by tracing in documents the links between hypothesis or results provided by each analysis. When possible, a sub-fault tree replaces a leaf when design details are given, e.g. after allocating physical resources to a function. However, it is not easy to maintain the traceability links between all these data when the analysis of a new component is added or when hypotheses are modified. This paper identified the replacement and allocation relation used in multi-staged safety assessment. It formalises the meaning of these relations and shows how they can be implemented with Altarica. The practical interest of the approach is illustrated on a RPAS case study.

Limitations and Future works The relations considered in our approach always trace a safety knowledge to a higher architecture level. Nevertheless, the behaviour of an architecture can be needed to model the failure propagation in a lower-level architecture. This kind of relations needs to be specifically address since it can considerably enhance the accuracy of the safety assessment on complex systems. Moreover, the proposed modelling approach provides a way to build a monolithic model containing various levels of safety knowledge. Consequently, the automatic safety assessment does not benefit from this modelling paradigm, that may lead to poor assessment performance. A solution would be to develop a solver considering the modelling paradigm to enhance the efficiency of the assessment.

Acknowledgment This work is part of the Phydias french study which is granted by the DGAC to study drone safety.

References

1. J.-R. Abrial. *Modeling in Event-B: System and Software Engineering.* Cambridge University Press, 2010.
2. A. Arnold, G. Point, A. Griffault, and A. Rauzy. The altarica formalism for describing concurrent systems. *Fundamanta Informaticae*, 40(2-3):109–124, 1999.
3. B. Bittner, M. Bozzano, R. Cavada, A. Cimatti, M. Gario, A. Griggio, C. Mattarei, A. Micheli, and G. Zampedri. The xsap safety analysis platform. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 533–539. Springer, 2016.

4. B. Bittner, M. Bozzano, A. Cimatti, and G. Zampedri. Automated verification and tightening of failure propagation models. In *AAAI*, pages 907–913, 2016.
5. Dassault. *Cecilia OCAS framework*, 2014.
6. D. Domis, K. Höfig, and M. Trapp. A consistency check algorithm for component-based refinements of fault trees. In *IEEE 21st International Symposium on Software Reliability Engineering, ISSRE 2010, San Jose, CA, USA, 1-4 November 2010*, pages 171–180, 2010.
7. J.-L. Farges, C. Saurel, C. Seguin, F. Deschamp, A. Favre-Bonté, A. Ruaudel, A. Desfosses, and M. Laval. Addressing safety assessment of autonomous robot operation and design with model based safety assessment. In *Lambda Mu 21 ≪Maîtrise des risques et transformation numérique: opportunités et menaces≫*, 2018.
8. B. Kaiser, P. Liggesmeyer, and O. Mäckel. A new component concept for fault trees. In *Proceedings of the 8th Australian workshop on Safety critical systems and software-Volume 33*, pages 37–46. Australian Computer Society, Inc., 2003.
9. L. Laibinis and E. Troubitsyna. Fault tolerance in a layered architecture: a general specification pattern in b. In *Proceedings of the Second International Conference on Software Engineering and Formal Methods, 2004. SEFM 2004.*, pages 346–355. IEEE, 2004.
10. O. Lisagor, M. Bozzano, M. Bretschneider, and T. Kelly. Incremental safety assessment: Enabling the comparison of safety analysis results. In *28th International System Safety Conference (ISSC)[submitted to]*, 2010.
11. F. Mhenni, J.-Y. Choley, N. Nguyen, and C. Frazza. Flight control system modeling with sysml to support validation, qualification and certification. *IFAC-PapersOnLine*, 49(3):453–458, 2016.
12. M. Morel. Model-based safety approach for early validation of integrated and modular avionics architectures. In *International Symposium on Model-Based Safety and Assessmemt*, pages 57–69. Springer, 2014.
13. Y. Papadopoulos and J. A. McDermid. Hierarchically performed hazard origin and propagation studies. In *Computer Safety, Reliability and Security*, pages 139–152. Springer, 1999.
14. T. Prosvirnova. *AltaRica 3.0: a Model-Based approach for Safety Analyses*. PhD thesis, Ecole Polytechnique, 2014.
15. A. Rauzy. Mathematical foundations of minimal cutsets. *Reliability, IEEE Transactions on*, 50(4):389–396, 2001.
16. SAE. Aerospace Recommended Practices 4761 - guidelines and methods for conducting the safety assessment process on civil airborne systems and equipment, 1996.
17. SAE. Aerospace Recommended Practices 4754a - Development of Civil Aircraft and Systems, 2010.
18. A. Villemeur. *Reliability, availability, maintainability and safety assessment*. John Wiley & Sons, 1992.
19. I. Vistbakka, E. Troubitsyna, and A. Majd. Multi-layered safety architecture of autonomous systems: Formalising coordination perspective. In *2019 IEEE 19th International Symposium on High Assurance Systems Engineering (HASE)*, pages 58–65. IEEE, 2019.
20. M. Zeller, D. Ratiu, and K. Höfig. Towards the adoption of model-based engineering for the development of safety-critical systems in industrial practice. In *Computer Safety, Reliability, and Security - SAFECOMP 2016 Workshops, ASSURE, DECSoS, SASSUR, and TIPS, Trondheim, Norway, September 20, 2016, Proceedings*, pages 322–333, 2016.