Two formal methodologies of Model-Based Safety Assessment for Fault Tree Analysis

Track 3: Product reliability assessment and management of 2023 7th International Conference on System Reliability and Safety (ICSRS)

Isabella Lanzani
Department of Electronics,
Information and Bioengineering (DEIB)
Politecnico di Milano
Italy, Milano
isabella.lanzani@polimi.it

Riccardo Scattolini
Department of Electronics,
Information and Bioengineering (DEIB)
Politecnico di Milano
Italy, Milano
riccardo.scattolini@polimi.it

Enrico Zio
Energy department
Politecnico di Milano
Milan, Italy
MINES Paris-PSL
Centre de Recherche sur les
Risques et les Crises (CRC)
Sophia Antipolis, France
enrico.zio@polimi.it

Alessandro Cimatti
Fondazione Bruno Kessler
Povo, Trento, Italy
cimatti@fbk.eu

Marco Bozzano
Fondazione Bruno Kessler
Povo, Trento, Italy
bozzano@fbk.eu

Stefano Tonetta
Fondazione Bruno Kessler
Povo, Trento, Italy
tonettas@fbk.eu

***Abstract*—During the design of safety-critical systems, the automatic estimation of the reliability of a proposed architecture could be a valuable asset. In the aerospace sector, according to ARP4754A [17] and ARP4761 [16] standards, the design development process must be performed in parallel with the safety assessment process. The practical reason is that an architecture that does not comply with safety requirements must be modified accordingly as soon as possible. This paper illustrates two existing techniques that can enable the automatic generate fault tree evaluations from an architectural model. A discussion over their advantages and their possible industrial implementation is provided, alongside a practical case study.**

***Index Terms*—Safety analysis, FTA, MBSA, AltaRica, xSAP**

Acronyms

AHRS	Attitude and Heading Reference System
CCA	Common Cause Analysis
CMA	Common Mode Analysis
FCS	Flight Control System
FEM	Failure Effect Modelling
FHA	Functional Hazard Analysis
FLM	Failure Logic Modelling
FMEA	Failure Modes and Effects Analysis
FTA	Fault Tree Analysis
MBSA	Model-Based Safety Analysis
MCC	Most Critical Condition
MCSs	Minimal Cut Sets
PSSA	Preliminary System Safety Assessment
SSA	System Safety Assessment
TLE	Top Level Event

Leonardo Helicopters, as part of *PNRR n°352*

I. Introduction - Safety in Avionics

The world of avionics is full of necessary complexities (e.g. redundancy, mitigation, detection...) due to the high risk of application. Strict standards regulate and limit this risk, guided by the classic safety criterion 'the higher the risk, the lower the probability of occurrence'. The ARP4761 [16] is the current pillar of this standardisation process: it regulates the type of assessment accepted by certification agencies and formally explains how to guarantee the safety of the system.

Following the well-known V-cycle (Figure 1), a safety process begins with Functional Hazard Analysis (FHA) and continues with Preliminary System Safety Assessment (PSSA) and System Safety Assessment (SSA), which constitute the two cores of verification and validation, respectively. The safety process starts with an initial assessment at the aircraft level, which lays the foundation for the specifications that the final product must have. In this paper, the focus will be on the system level, for the safety evaluation at Flight Control System (FCS) level.

FHA consists in the tabular examination of the system functions to identify fault conditions and classify them according to their severity. This is the beginning of the overall safety assessment process. The PSSA and SSA have a similar structure and the main difference between them is the objective.

The PSSA is a systematic evaluation of a proposed architecture and its implementation, based primarily on the severity of the failures defined in FHA. PSSA must determine the

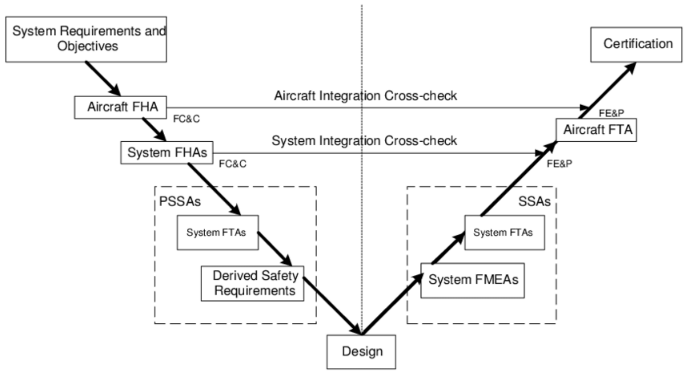

System Requirements and Objectives
Certification
Aircraft Integration Cross-check
Aircraft FHA
FC&C
FE&P
Aircraft FTA
System Integration Cross-check
System FHAs
FC&C
FE&P
PSSAs
SSAs
System FTAs
System FTAs
Derived Safety Requirements
System FMEAs
Design

Fig. 1: Safety Process V-cycle

safety requirements for all elements of the architecture. It is intended to “guide the design” and is an iterative process that begins early in the system design process. This is the time when protection mechanisms and architectural attributes are considered and implemented, so its results are extremely important (e.g. in dialogue with suppliers).

On the other hand, the objective of SSA is to demonstrate that all relevant safety objectives have been achieved, validating the results of PSSA. The SSA is a detailed examination of the safety-relevant aspects of the system: once completed, it verifies that the system is ready to be certified.

Both SSA and PSSA are based on Fault Tree Analysis (FTA) (ARP5761 considers other methods instead of FTA, but this is the most widely used method). The FTA is a top-down analysis with the objective of calculating the probability of a given system failure condition, called top-event, by analysing the combination of component failures, called basic-events, that contribute to its occurrence. The ultimate goal is to demonstrate compliance with severity requirements. However, FTA is nothing more than an (intelligent) reinterpretation of the safety-focused system architecture, and this work moves in the direction of reducing the waste of time, labour and error generation that manual calculation of FTA generates.

This article will compare two formal methods for automatic fault tree calculation. The comparison will highlight their characteristics in order to propose a formal process that encompasses both methodologies within the safety V-cycle. Furthermore, using a case study, an explanation of the semantics and behaviour of the two tools is proposed.

In the remainder of the paper a classification of current Model-Based Safety Analysis (MBSA) techniques is presented in Section II, with a focus on two existing methods (AltaRica and xSAP). In Section III, the case study is explained. In Section IV, the two strategies considered are analysed and, finally, in Section V, conclusions and consideration over possible future works are drawn.

II. MBSA

The term MBSA refers to all those approaches that, by using a model to determine the reliability properties of the system, aid safety assessments. There is a kind of misuse of this term which, for the sake of clarity, is limited here to formal methods that create models translatable into a FTA. However, there is not a strict topological equivalence between such models and FTA, since models could be much more detailed (e.g. they could take into consideration dynamic aspects of the system that FTA does not consider).

A classification of the different MBSA techniques was made in 2011 by Lisagor and Kelly [9] and has since then been reffered to in numerous articles [5] [6].

It divides the methodologies by *model provenance* and *semantics of component dependencies*.

The model at the centre of MBSA is divided between two extremes: the design model and an autonomous safety model built for the purpose. The first extreme ensures greater consistency with the design model. Furthermore, the development and safety processes can share common modelling environment, languages and tools. The second extreme allows for a thorough representation of unintended dependencies between components that the Nominal Model does not consider. Hybrid models contain features of both methods: they start from the architecture of the design model, but instruct safety engineers to characterise the behaviour of individual components for safety assessment purposes.

The second distinction involves Failure Logic Modelling (FLM) and Failure Effect Modelling (FEM). In FLM, components exchange only failure modes: they propagate failure information expressed via Boolean or string-like constructs (e.g. “no pressure supplied” for a pump that fails to pressurise a downstream water flow). Models using these constructs can easily be driven to create an FTA. The main disadvantage of this approach is the difficulty in reusing the components for a different contexts. The components are characterised with respect to an implicit design intent. A more realistic representation of components is the FEM, in which components exchange abstract flows of ’real’ physical entities (e.g. mass or energy). Fault information is more descriptive than the previous Boolean type labelling and reuse of components is possible. A major disadvantage is that for FTA computation, more complex algorithms are needed. There is also a hybrid category in this classification axis wherein intentional interactions are modelled with FEM, while unintentional ones with FLM.

The article focuses on two different methodologies for calculating FTA: AltaRica and xSAP. In Figure 2, they are represented according to the previously mentioned two-dimensional classification. AltaRica allows both hybrid and FEM semantics.

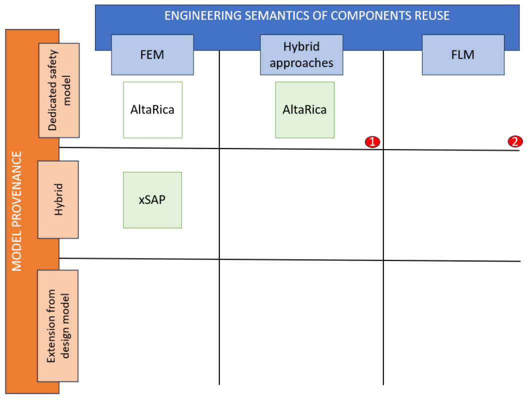


Fig. 2: Classification of the considered techniques

Both methodologies originated from aerospace projects and are therefore particularly suitable for solving problems specific to this sector (e.g. common cause failures).

In the paper cited above [9], it is argued that the PSSA could be performed from positions labelled 1 or 2 (Figure 2) and, thus, AltaRica fulfils the specific requirements of the PSSA.

It is shown [8] that model checking techniques can be used in the verification phases, producing good results in aiding the SSA process phase.

As will be explained in Section V, since xSAP is able to replicate an AltaRica model, a proposed workflow with these two tools can cover the entire V safety cycle.

A. *AltaRica*

AltaRica is a safety modelling language first proposed in 1999 by Point and Rauzy. Since then, it has undergone a series of modifications up to the latest version we considered in this work, $AltaRica3.0$.

AltaRica is a high-level modelling language dedicated to probabilistic safety analyses. Prior to any evaluation, models are ‘flattened’ and their semantics are represented in terms of systems of protected transitions, to generalise classical safety formalisms, like reliability block diagrams or Markov chains [11]. In AltaRica, components are characterised by events, state variables and flow variables: a transition between states is triggered by events, whereas flow variables are calculated from the current state.

An example of an AltaRica language specification for a sensor Attitude and Heading Reference System (AHRS) is shown in Figure 3. In this, the Object Oriented prone specification, that allows the definition of hierarchical and reusable components, is evident. “Class LossErrCom” contains the skeleton of all components that could fail in the event of a loss or erroneous event. Following the same criterion, any class defined in this way can be instantiated several times in the system definition. Looking at the language, the “Class AHR” instantiates the state variable s, and the flow variables $valid$ and $date$, and initialises them. The events $loss$ and err are defined, together with their probability. In the ‘transition’ instruction, their triggering changes the state of the system. Given this state scenario, the ‘assertion’ instruction uniquely

```
class LossErrComp
    State s (init=WORKING);
    parameter Real Lambda=(1);
    parameter Real Lambda2=(1);
    event loss (delay=exponential(Lambda));
    event err (delay=exponential(Lambda2));
transition
    loss : s==WORKING -> s:=F_LOSS;
    err : s==WORKING -> s:=F_ERR;
end

class AHRS
Boolean sen_valid(reset=false);
Real sen_data(reset=0.0);
Real In (reset=0.0);
    extends LossErrComp (Lambda =1.16e-5, Lambda2 =1.13e-7);
assertion
sen_valid := if(s!=F_LOSS) then true else false;
sen_data  := if (s==WORKING) then In else 6;
end
```

Fig. 3: AltaRica block for AHRS inertial sensor

defines the changes of the flow variables with respect to the state. The finite state automaton of the component AHRS are depicted in Figure 4. After the model has been flattened, it

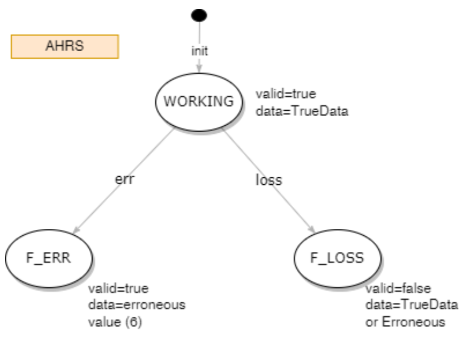


Fig. 4: State transition formalism for AHRS

can be compiled as a Boolean equation in the .opsa format (Open Probabilistic Safety Analyst, an ‘ad hoc’ format by Rauzy [12]), which can be read by many fault tree compilers, such as *Arbre Analyst* [3].

B. *xSAP*

The xSAP safety analysis platform provides several functionalities for finite and infinite state synchronous transition systems. In particular it supports the definition of failure modes based on customisable libraries, an automatic model extension system, the generation of safety analysis artifacts and their probabilistic evaluation. The analysis performed by this tool is logically equivalent to an exhaustive search through the activation of all possible permutations of failure modes. The permutations that lead to the satisfaction of the fault condition expression are reported as Minimal Cut Sets (MCSs).

xSAP is used in the following workflow [1]:

Nominal Behaviour Modelling The Nominal Modeldescribes the behaviour of a given system when everything works as expected (no faults). The model is written in the SMV language [7].

Model Extension Model extension is performed to enrich the Nominal Modelwith the specification of possible faults that may influence the system behaviour. The result, the extended Nominal Modelwith the defect behaviour, is called the Extended Model. The model extension can be performed manually or automatically. In the second scenario, the one used in this work, the model extension is performed automatically by xSAP. Similar to the nominal model, the Extended Model is written in the SMV language.

Safety Evaluation The objective of safety evaluation is to assess the robustness of a given system in the presence of faults. Safety assessment of critical systems is typically performed in parallel with system design to ensure that the system meets the safety requirements necessary for its deployment and use. Key techniques in this area are FTA, Failure Modes and Effects Analysis (FMEA) and Common Cause Analysis (CCA).

The most important aspect of xSAP is the implementation of a family of routines for such analyses, based on state-of-the-art model checking techniques. These include algorithms based on BDD, SAT and SMT. Using these routines, it is possible to validate the actual behaviour of the system as well as create ad-hoc counterexamples for both nominal and Extended Models of the system. The models created increase the complexity and representativeness of the system itself.

Considering the same model example AHRS, the nominal and its failure effect instructions are shown in Figure 5, for the Nominal Model and Figure 6 for the Failure Extension Instructions.

```
MODULE AHRS(d_env)
VAR
  d_AHRS : real;
  v_AHRS : boolean;
ASSIGN
  init(d_AHRS) := 0;
  next(d_AHRS) := d_env;
ASSIGN
  init(v_AHRS) := TRUE;
  next(v_AHRS) := v_AHRS;
```

Fig. 5: xSAP module for AHRS, from Nominal Model

```
EXTENSION OF MODULE AHRS

     SLICE sensor_loss AFFECTS v_AHRS WITH

       MODE AHRS_loss{1.16e-5} : Permanent Inverted(
                  data input << v_AHRS,
                  data varout >> v_AHRS,
                  event failure);-- >> fev_AHRS_loss);


     SLICE sensor_err AFFECTS d_AHRS WITH


       MODE AHRS_err{1.13e-7} : Permanent  ErroneousByValue(
                   data input << d_AHRS,
                   data varout >> d_AHRS,
                   event failure);-- >> fev_AHRS_err);
```

Fig. 6: Failure Extension Instruction for AHRS

The fault specification contains two required fault models, one for fault effects and the other for fault dynamics. Fault Effects specifies how the affected part of the system model changes due to the presence of the faults, whereas Fault Dynamics specifies how the presence of the fault changes over time (e.g. if it is a permanent fault or if it can be repaired). In the example, Fault Effects for *loss* is “Inverted” (where the $valid$ signal *boolean* is forced from 1 to 0 to simulate sensor loss) and for *erroneous* is “ErroneousByValue” (where the $data$ signal *real* is forced to a value other than the defined value). For both faults the dynamic fault is set to ‘Permanent’ (to indicate that it is a non-repairable fault). The fault construct is hierarchical: each module can have many slices, which show the variables subject to the defined faults, and each slice can have multiple modes, which describe how a specific fault mode affects the defined variable. Probabilistic indicators are also represented in the mode definition.

There is a strong difference with respect to AltaRica for fault conditions that by their nature do not produce a predefined output, such as ErroneousByValue. In xSAP, the effect of the fault, due to the capabilities of the model checker, can result in any one of an infinite number of (erroneous) values. In AltaRica this multiplicity of values cannot be expressed, as the flow variables are set to a specific value following the ‘assignment’ code segment.

III. System and Model

The case study considers the safety-relevant aspects of an avionic voter for AHRS inertial data, in the architecture depicted in Figure 7.

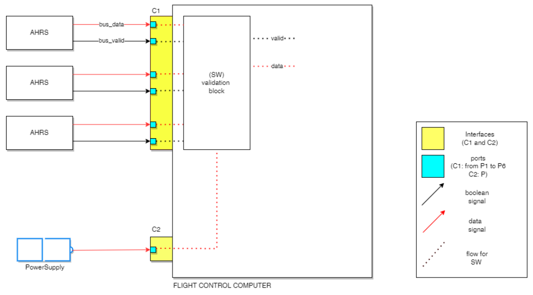


Fig. 7: Voter Schematic

An attempt was made to realistically represent the hardware components, in order to show how, due to the inherent modular nature of MBSA, it is possible to save time in the calculation of safety constructs.

This case study, although simple, is defined to contain the main peculiarities of the aerospace sector, in particular redundancies and fault detection algorithms.

The voter strategy consists, firstly, in discarding the AHRS signals whose $valid$ is zero, then in comparing the remaining $data$ in order to evaluate a coherent value from the sensor measurements. Consistency is achieved if two measures signals show the same numerical value. Both $valid$ and $data$ signals, are transmitted to the voter via busses connected to the FCS through the ports of an I/O interface (called C1). Similarly, the battery that powers the FCS is connected to a port of another I/O interface (called C2). Of course, these ports are not the only signals entering the FCS, but they are the ones relevant to the case at hand.

Software specifications are irrelevant in this type of analysis: software flaws are present in the FTA only as random internal flaws, unrelated to the specific algorithm that they implement. Safety-critical software undergoes a completely different certification procedure, and the curious reader is referred to the analysis of DALs and the process described in the standards Do178 [14] and DO245 [13].

In order to focus on hardware component, this case study does not contain random software failures (so it is assumed that the voter logic is completely reliable). The components that can fail are depicted in Figure 7 and listed in the following table I. The certificate authority states its safety requirements in terms of "Probability of Failure per Flight Hour" [16], as indicated in Table I by FH.

Component	Instance Identifier	Basic Events	
		Fault Typed	***Failure rate [FH]***
AHRS	a	Loss of AHRS	1.16e-05
		Erroneous of AHRS	1.13e-07
Power	PS	Loss of power	5.33e-07
Bus	B	Loss of (valid) channel	1.82e-07
I/O Interface	P	Loss of port	2.61e-5
	C	Loss of interface	4.56e-7

TABLE I: List of Faults

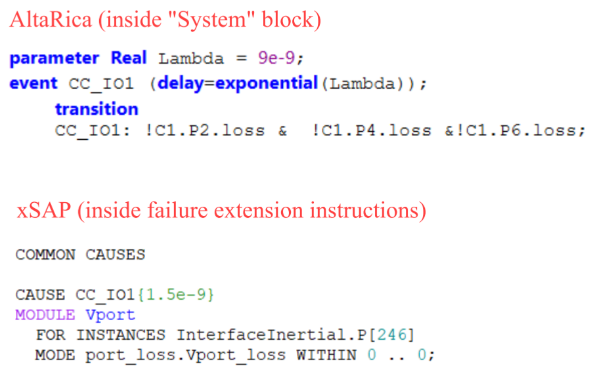
a) Modelling assumptions: Busses and I/O interfaces only fail in the event of a 'loss', which is realistically justified by the ARINC429 [4] protocol, the worldwide aerospace industry standard. In this protocol, transmitted and received signals strictly follow bit-code rules and any corrupted signal is detected by the receiver-end due to the rule violation. Furthermore, when detected, the protocol provides a specific bit indicator for the incorrect state, here represented in the valid fault signal (set to 0 in case of loss). To model this safety-relevant feature, in both languages, the *data* and *valid* signals have been separated into a Boolean and a real variable, respectively, allowing for a more intuitive representation.

A note can be made about the 'interface loss' fault: it is defined as the simultaneous loss of all ports connected to that I/O interface: it is modelled as a common cause fault that simultaneously triggers the loss faults of all affected ports. Figure 8 shows the semantics of this specific common cause. In AltaRica, common causes are defined in the 'System' (main) block, which calls all component events that are triggered by it. In xSAP, there is a specific function in the fault instruction file to describe common causes.

b) Top Level Events: The first Top Level Event (TLE) considered for this case study is the loss of inertial data:

$$Loss\ of\ voter\ :=\ \neg\texttt{voter\_valid}$$

The loss represents a fault condition of which the system is aware: this is done according to two-out-of-three logic


AltaRica (inside "System" block)

```
parameter Real Lambda = 9e-9;
event CC_IO1 (delay=exponential(Lambda));
    transition
    CC_IO1: !C1.P2.loss &  !C1.P4.loss &!C1.P6.loss;
```

xSAP (inside failure extension instructions)

```
COMMON CAUSES

CAUSE CC_IO1{1.5e-9}
MODULE Vport
  FOR INSTANCES InterfaceInertial.P[246]
  MODE port_loss.Vport_loss WITHIN 0 .. 0;
```

Fig. 8: Common Cause declaration for AltaRica and xSAP

(signals are only compared when their validity is set to true, and discarded otherwise), and only when the comparison is sufficiently consistent, the $voter\_valid$ signal set to TRUE and the inertial data entrusted to the FCS. Whatever set of faults triggers this condition, the system reacts to this fault by warning the crew.

$$Erroneous\ of\ voter\ :=\ \texttt{voter\_valid} \wedge \texttt{voter\_data}\ \neq\ TrueData$$

The top erroneous event is more subtle: it implies that everything works normally for the system. This particular failure only occurs when several signals fail in an erroneous top event while showing the exact same numerical value (with $valid$ set to true). The severity associated to this second TLE is greater than the previous one. $\texttt{voter\_valid}$ is defined as follows:

$$\texttt{voter\_valid} := if(C2.P.Out == PowerOn\ and$$

$$(P\ or\ S_1\ or\ S_2\ or\ S_3))$$
$$then\ true\ else\ false$$

$voter\_data$ previously defined is composed by a Primary (P) and the three Secondaries (S_1, S_2, S_3): these instances indicate the three favourable possible outcomes of the comparison.

$$P = ((C1.P2.Out\ and\ C1.P4.Out\ and\ C1.P6.Out)$$
$$and\ (C1.P1.Out\ =\ C1.P3.Out\ or$$
$$C1.P1.Out\ =\ C1.P5.Out\ or\ C1.P3.Out\ =\ C1.P5.Out))$$
$$S_1 = (C1.P2.Out\ and\ C1.P4.Out\ and$$
$$C1.P6.Out\ =\ false)\ and\ C1.P1.Out\ =\ C1.P3.Out)$$
$$S_2 = (C1.P2.Out\ and\ C1.P6.Out\ and$$
$$C1.P4.Out\ =\ false)\ and\ C1.P1.Out\ =\ C1.P5.Out)$$
$$S_3 = ((C1.P4.Out\ and\ C1.P6.Out\ and$$
$$C1.P2.Out\ =\ false)\ and\ C1.P3.Out\ =\ C1.P5.Out))$$

The $voter\_data$ signal exiting the voter is defined as follows:

$$\texttt{voter\_data} :=$$
$$if\ (C2.P\ and\ C1.P1.Out\ =\ C1.P3.Out)\ then\ C1.P1.Out$$
$$if\ (C2.P\ and\ C1.P3.Out\ =\ C1.P5.Out)\ then\ C1.P3.Out$$
$$if\ (C2.P\ and\ C1.P1.Out\ =\ C1.P3.Out)\ then\ C1.P1.Out$$
$$if\ S_1\ then\ C1.P1.Out$$
$$if\ S_2\ then\ C1.P1.Out$$
$$if\ S_3\ then\ C3.P1.Out\ else\ NaN$$

Note that internal checks and detection logic should not use the '$RealData$' signal, relying only on internal information (e.g. cross-references), whereas the higher event definition (e.g. erroneous) may do so.

IV. Comparative Analysis

In this Section, a comparative analysis over the two formal methods is proposed. The criteria for this comparison are: numerical results, ease of modelling, and the goodness of the tree structures produced, the validation capabilities over both the model and the artifacts, and lastly, the computational demand. Final considerations are proposed at the end of the Section.

A. Numerical Results

Given the previously defined model we can analyze the artifacts obtained by the two tools. The comparison for the first TLE, the loss, is straightforward if we consider only loss failures, and the resultant probability is the same from both models: 9.93^{-07}. In this scenario, both tools produce the MCSs represented in Table II. The case become more complex

N°	Order	Probability [FH]	Components
1	1	5.33^{-7}	PS.loss
2	1	4.56^{-7}	CC_IO1
3	2	6.81^{-10}	C1.P2.loss C1.P6.loss
4	2	6.81^{-10}	C1.P2.loss C1.P4.loss
5	2	6.81^{-10}	C1.P4.loss C1.P6.loss
6	2	3.03^{-10}	C1.P4.loss a1.loss
7	2	3.03^{-10}	C1.P6.loss a1.loss
8	2	3.03^{-10}	C1.P2.loss a2.loss
9	2	3.03^{-10}	C1.P2.loss a3.loss
10	2	3.03^{-10}	C1.P6.loss a2.loss
11	2	3.03^{-10}	C1.P4.loss a3.loss
12	2	1.34^{-10}	a1.loss a2.loss
13	2	1.34^{-10}	a1.loss a3.loss
14	2	1.34^{-10}	a2.loss a3.loss
15	2	4.75^{-12}	B2.loss C1.P4.loss
16	2	4.75^{-12}	B2.loss C1.P6.loss
17	2	4.75^{-12}	B4.loss C1.P2.loss
18	2	4.75^{-12}	B6.loss C1.P2.loss
19	2	4.75^{-12}	B4.loss C1.P6.loss
20	2	4.75^{-12}	B6.loss C1.P4.loss
21	2	2.11^{-12}	B4.loss a1.loss
22	2	2.11^{-12}	B6.loss a1.loss
23	2	2.11^{-12}	B2.loss a2.loss
24	2	2.11^{-12}	B2.loss a3.loss
25	2	2.11^{-12}	B6.loss a2.loss
26	2	2.11^{-12}	B4.loss a3.loss
27	2	3.31^{-14}	B2.loss B4.loss
28	2	3.31^{-14}	B2.loss B6.loss
29	2	3.31^{-14}	B4.loss B6.loss

TABLE II: Minimal Cut Sets for loss TLE

when adding the erroneous basic event of AHRS components (second row of Table I). Recalling previous boolean expressions, voter's loss can also result from the numerical comparison of erroneous signals. This can be produced from the same xSAP model without any additional effort, just by adding the failure instruction *ErroneousByValue* discussed in Section II-B. AltaRica, instead, requires the user to explicit the numerical flow variable in case of erroneous (referring to Figure 3, where assertion imposes $data$ variable from AHRS to be equal to 6 outside the nominal state). In this way, by nesting the same AHRS component in the main, two erroneous faults will displace the same output value and the loss will not be reported: it is necessary to modify the model (e.g. by creating two models, where one avoids the use of nested components to represented this particular case).

The Figure 9, depicts the TLE described by "Erroneous of voter" TLE, previously defined, as shown by the xSAP fault tree viewer.

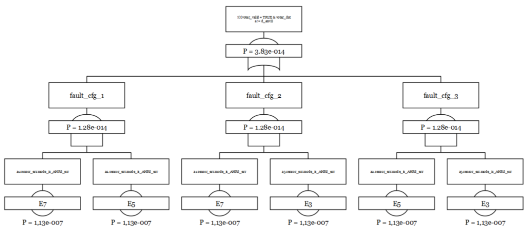

!(!(voter_valid = TRUE & voter_data != d_env))
P = 3.83e-014
fault_cfg_1
fault_cfg_2
fault_cfg_3
P = 1.28e-014
P = 1.28e-014
P = 1.28e-014
a1.sensor_err.mode_is_AHRS_err
a2.sensor_err.mode_is_AHRS_err
a1.sensor_err.mode_is_AHRS_err
a3.sensor_err.mode_is_AHRS_err
a2.sensor_err.mode_is_AHRS_err
a3.sensor_err.mode_is_AHRS_err
E7
E5
E7
E3
E5
E3
P = 1,13e-007
P = 1,13e-007
P = 1,13e-007
P = 1,13e-007
P = 1,13e-007
P = 1,13e-007

Fig. 9: xSAP - FTA for erroneous of inertial data

In xSAP, the value of the failed variable (in this case, the erroneous inertial sensor) is not defined numerically: it is the model checker algorithm underlying the construction of the tree that automatically finds that equal pairs of erroneous values recreate the top event (as expected). The calculation for this case led to 48 MCSs of order 2 and 2 of order 1. Using the same functionality as the model checker, it is possible to demonstrate this and create a numerical simulation that reproduces this behaviour (e.g. the first and second sensors fail by producing an erroneous inertial data of -1). xSAP decides itself which (erroneous) numerical values to inject to obtain the fault condition: in the case of a loss it presentes two different numerical values, in the case of an erroneous two identical fault values. Modeling something similar in AltaRica is more difficult if we want to maintain the same model for both TLE. A practical suggestion is to create different AltaRica models of the same plant, to highlight the correct fault combinations for each TLE.

B. Modelling

Modelling in AltaRica is intuitive: the language is explained in the AltaRica Association's manuals [18]. Going into more detail, structural operations allow for an optimal use of the object oriented classes [10]:

- *Composition* allows the creation of hierarchies of nested components
- *Inheritance* extends elements of one class into another class (used in Figure 3, with command *extends*)
- *Aggregation* incorporates blocks and objects to make them belong to different branches of a hierarchical model

xSAP only maintain the first of the previous operations. Moreover, modelling the Nominal Model could be challenging, due to the explicit FEM semantics of the components. The fault instructions, on the other hand, are rather simple and adding faults to the model is intuitive. The libraries allow a detailed description of failure events (e.g. conditional failures allow the creation of specific situations based on the state of the system). The Extended Model created in this way is more realistic (as highlighted in the previous section IV-A). It depends on the objective of the study whether the added-value behind this characterisation is effective.

C. Tree generated

The goodness of the representation of fault trees is discussed as an indicator of performance: one might assume that the probabilistic results, which can be extracted from any tree representation, are at the heart of the assessment, while others force the argument on its readability, usually out of industrial pragmatism. Since the two tools are able to produce equivalent trees, but different in their graphical form, the focus of this Section should be in which one is producing the greater added-value from its representation.

AltaRica automatically generates an .opsa file [12], that can be transformed into a graphical fault tree (with, for example, *Arbre Analyst* [3]). This tree recalls the nomenclature of the model architecture, so it can be used to check model implementation. Unfortunately, as the complexity of the system increases, using the automatically generated tree to check the validity of the model becomes rather difficult. In the example provided, readingTLE for loss was straightforward, since the algorithms recalled the definition of primary and secondaries mid variables. However, FTA erroneous was much more difficult to read.

The solution in xSAP is a fault tree already optimised to show MCSs. Therefore, it is quite easy to read but it is almost impossible to use it to validate the model. In xSAP, graphical representation of the tree can be visualized using .py scripts provided in the xSAP platform.

The appreciation of one FTA structure over the other is goal-oriented.

D. Validation of the model

Since the probabilistic results are calculated automatically, the problem of validating the model and assessing its correctness is crucial. In AltaRica, this is possible both from the tree itself (although it is rather difficult) and from the interactive simulator. The latter shows how a series of user-selected events change the system variables and whether or not they reach the main event. This is a useful tool, but complex models may be rather difficult to evaluate with this tool, given the large number of combinations of events to be checked manually.

Validation in xSAP is performed using the functionality of the model checker: firstly, the software automatically checks whether the defined main event can be obtained from the Nominal Model itself (the main event is in fact a malfunction that cannot occur if the system behaves normally). Furthermore, the user can validate both the Nominal Model and the Extended Model, testing the expected behaviour with CLT and LTL expressions. Thus, in xSAP it is possible to validate the properties and requirements of both models using the model checker functionality. It can be a long procedure, and results are intrinsically dependent on the performed tests. Two examples of this are here provided:
$LTLSPEC\ A\ =\ F(X\ C1.P2.Out\ =\ FALSE$
$\rightarrow\ voter\_valid\ =\ FALSE$ In this case, the model checker is able to create a counterexample (showing that this expression is not an invariant of the system) confirming that the loss of a channel is not sufficient to produce a loss.
$LTLSPEC\ B\ :=\ F(X\ (C1.P2.Out\ =\ FALSE\ \&$
$C1.P6.\ =\ FALSE\ )$
$\rightarrow voter\_valid =\ FALSE$

In this other case, instead, the model checker found no counterexample for this scenario, showing that the loss of two channels results in the loss TLE.

E. Validation of the FTA

Validating the correctness of the FTA means discussing how to prove that the artifacts are indeed representative of the model. Indeed, this is a difficult topic: without using external software, it is not possible to guarantee that the internal computational steps are correct. It is an open point for both tools and, for now, validation of FTA can only be done manually from both artifacts.

It can be worth mentioning that xSAP can use the model checker's calculation capability to provide proofs of its correctness. In particular, it automatically creates an inductive invariant expression to evaluate that if all MCSs are blocked, it is impossible to reach TLE. This supports the creation of artifacts with respect to the model.

F. Performance Evaluation

Analysing the tools and their use, some conclusions can be drawn. Firstly, it can be observed that the proposed approaches are quite similar to each other. In fact, the AltaRica and xSAP results show many similarities in their operation: the description of components, the Boolean definition of the top event, the possibility of reusing components and a similar semantic language. To understand which is the best way to use them in the security process, an evaluation of their performance was carried out. The performance indices used are: Expressiveness, ease of modelling, goodness of representation FTA and validation capability.

Criteria	**AltaRica**	**xSAP**
Expressiveness	Medium	High
Easiness in modelling	High	Medium
FTA goodness	Architecture related	Readable
Model validation capabilities	Low	High

TABLE III: Performance Criteria Evaluation

The main comment between on tools is an obvious trade-off between expressiveness and ease of modelling. AltaRica is generally more intuitive, but more constrained: this is a consequence of the rigid semantics of flow and state variables. It can, nevertheless, be used to produce reliability indicators for some specific cases in order to guide the design process (as suggested by ARP standards). Altarica finds its best use in the PSSA phase: the security requirements that can significantly alter the architecture are limited and the possibility of building a model for each would quickly yield appreciable results.

Instead, model checking analyses, such as the one produced by xSAP, could truly help the verification phase of the safety process (SSA). The creation of a unique model, extended with all the relevant failure modes, could be used for an exhaustive numerical evaluation over the effects of those failure injections in the system, ensuring to consider all the specific numerical values that contribute to the current TLE.

Moreover, due to the similarities between the two languages, it has been shown that it is possible to translate an AltaRica model into an extended version of the xSAP model checker [2]. This translation step is not (yet) automatic, but could be implemented in future work.

By invoking the Safety Process standard, the validation and verification aspects can be bridged: a formal process between these two tools, with AltaRica for the first part and, when the architecture has been finalised, xSAP for the second can certainly be instantiated.

V. Conclusions and Future Work

The increasing complexity of today's systems shows that the manual production of reliability artifacts is becoming more and more prohibitive and formal techniques such as the ones represented here are going to become mandatory at some point. SAE International addressed this issue and promised a new release of the standard ARP4761, the ARP4761A [15], in order to allow for MBSA techniques to be implemented in industrial practice.

There are many promising directions for future work. Investigating ways to increase complexity of application and not escalating computational effort is one of them. And in this same direction, how more complex cases could be addressed with formal methods. Another direction to take in this field is to study the problem of synthesis for diagnosability, i.e. finding subsets of the available sensors that are sufficient to guarantee diagnosability and possibly minimise a cost function.

From a more industrial point of view, future work could involve several avenues. Firstly, it can be said that neither AltaRica nor xSAP guarantee a readable fault tree, and the possibility of improving algorithms in this respect is an open research topic. Certification is a critical element that must be considered in the avionics sector: the software that produces official results for the certifying body must itself be certified. Today, this is not the case for either AltaRica or xSAP. Another practical aspect that both tools could consider is the integrability of trees with tree viewers actually used in industrial practice (e.g. Isograph).

Acknowledgment

We would like to warmly thank Michel Batteux, of the AltaRica Association, who provided useful comments that improved the quality of this work.

References

[1] Marco Bozzano, Alessandro Cimatti, Marco Gario, David Jones, and Cristian Mattarei. Model-based safety assessment of a triple modular generator with xSAP. *Formal Aspects of Computing*, 33, 2021. Publisher: Springer.

[2] Marco Bozzano, Alessandro Cimatti, Oleg Lisagor, Cristian Mattarei, Sergio Mover, Marco Roveri, and Stefano Tonetta. Safety assessment of AltaRica models via symbolic model checking. *Science of Computer Programming*, 98, 2015.

[3] Emmanuel Clement. Arbre Analyste website. https://www.arbre-analyste.fr/en.html. [Online; accessed 3-Juy-2023].

[4] Condor Engineering. *ARINC protocol Tutorial Manual*. 2010.

[5] Simon Gradel, Benedikt Aigner, and Eike Stumpf. Model-based safety assessment for conceptual aircraft systems design. *CEAS Aeronautical Journal*, 2022. Publisher: Springer.

[6] Sohag Kabir. An overview of fault tree analysis and its application in model based dependability analysis. *Expert Systems with Applications*, 77, July 2017.

[7] FBK Fondazione Bruno Kessler. The SMV specification language. https://nusmv.fbk.eu/NuSMV/papers/sttt$_j/html/node7.html$.

[8] Rahul Krishnan and Shamsnaz Virani Bhada. Integrated System Design and Safety Framework for Model-Based Safety Assessment. *IEEE Access*, 10, 2022.

[9] Oleg Lisagor, Tim Kelly, and Ru Niu. Model-based safety assessment: Review of the discipline and its challenges. In *The Proceedings of 2011 9th International Conference on Reliability, Maintainability and Safety*. IEEE, 2011.

[10] Tatiana Prosvirnova. AltaRica 3.0: a Model-Based approach for Safety Analyses. 2014.

[11] Tatiana Prosvirnova, Michel Batteux, Pierre-Antoine Brameret, Abraham Cherfi, Thomas Friedlhuber, Jean-Marc Roussel, and Antoine Rauzy. The AltaRica 3.0 project for model-based safety assessment. *IFAC proceedings volumes*, 46(22), 2013. Publisher: Elsevier.

[12] Antoine Rauzy. *XFTA An Open-PSA Fault Tree Engine*. 2012.

[13] RTCA. *DO-254/ED-80: Design Assurance for Airborne Electronic Hardware*. 2000.

[14] RTCA. *DO-178C/ED-12C: Software Considerations in Airborne Systems and Equipment Certification*. 2012.

[15] SAE. ARP4761A: Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment . https://www.sae.org/standards/content/arp4761a/. [Not yet released].

[16] SAE. *ARP 4761: Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment*. 1996.

[17] SAE. *ARP 4754A: Guidelines for Development of Civil Aircraft and Systems*. 2010.

[18] Tatiana Prosvirnova, Michel Batteux, and Antoine Rauzy. *AltaRica 3.0 Language Specification*. 2020.