Integration of Commonalities in the Paradigm of Model-Based Safety Analysis in Aerospace

Lanzani Isabella[a*], Luca Uliano[b], Riccardo Scattolini[a]
[a]Politecnico di Milano, Milano, Italy
[b]Leonardo Helicopter Division, Samarate, Italy

Abstract: The need for rapid and accurate preliminary safety assessments in the development of complex systems, particularly in the aerospace sector, is undeniable. This urgency is due to the need to make informed decisions on architecture changes at the earliest possible stage. To meet this challenge, Model-Based Safety Analysis (MBSA) approaches are emerging as key tools. These approaches allow the automatic generation of Fault Tree Analysis (FTA) and other reliability calculations. This paper focuses on the integration in a MBSA frame of Common Mode Analysis (CMA), which addresses potential commonalities between components. Following the guidelines of ARP4761A, the presented procedure allows for the direct elicitation of such independence requirements. Minimal Cut Sets (MCS) are automatically computed from automatically computed from a safety model of the system, expressed in a suitable language such as AltaRica. The process allows for design decisions, such as determining the minimum number of components required to meet industrial safety standards. Among other independence assertions, the problem of common causes arising from shared common resources is addressed, and an optimal allocation solution is proposed. All results are examined throughout a practical study in which two flight control system architectures are compared: this comprehensive evaluation of their safety characteristics aims to simulate a realistic industrial scenario.

Keywords: Model-Based Safety Analysis , Failure Propagation Models , Common Mode Analysis , Independence Requirements , AltaRica , Aerospace Safety Process .

ACRONYMS

Abbreviation	Definition		
A/C	Aircraft	IRS	Inertial Reference System
A/D	Analog To Digital Converter	MC	Markov Chain
ADH	Air Data Heading System	MCC	Most Critical Condition
CMA	Common Mode Analysis	MBSA	Model-Based Safety Analysis
DAL	Development Assurance Level	MCS	Minimal Cut Set
DD	Dependence Diagram	OLE	Ok Loss Erroneous (failure propagation type)
ESM	Extended System Models	PRA	Particular Risk Analysis
FTA	Fault Tree Analysis	PSSA	Preliminary System Safety Assessment
FPM	Failure Propagation Models	RA	Radar Altimeter
GPS	Global Positioning System	RDC	Remote Data Concentrator
GTS	Guarded Transition Systems	(S)FHA	(System) Functional Hazard Assessment
IFR	Instrumental Flight Rules	SM	State Machine
		SSA	System Safety Assessment

1. INTRODUCTION

Accepting malfunction's numerical probabilities often derives from assessing multiple systems based on the assumption that faults are independent.

In safety-critical applications, like aerospace, it is necessary to ensure that such independence between certain components exists or that the risk associated with their dependence is deemed acceptable[1]. Hence, assessments to identify common causes for faults or malfunctions are mandatory during both Preliminary System Safety Assessment (PSSA) and System Safety Assessment (SSA).

Especially being able to cover the early development stage, where the system architecture has not been defined yet, can produce enormous benefits in terms of saving time and costs. Following the recommended guidelines, the article aims to support the generation of Common Mode Analysis (CMA),

during this stage. CMA's primary objective here is the elicitation of independence requirements, to anticipate safety-critical aspects of the proposed architecture.
Apart from eliciting them, the scope of the study will be to automatically assert importance metrics and independence assertions for each independence determined to be necessary. This will provide the designer with the set of requirements together with all the safety considerations that can be obtained at this stage, to be used to support the next iterations of the development itself. This iterative interaction between the processes of safety and design will be further enhanced by integrating the study within a Model-Based Safety Analysis (MBSA) frame.

This article is structured as follows: Section 2 will describe the aerospace safety process. The focus of this Section will be MBSA and CMA during Preliminary System Safety Analysis (PSSA). Failure Propagation Models (FPM) are described together with the advantages of their application. Two auxiliary software, AltaRica Wizard and DalCulator, used for MBSA and CMA respectively, are described here.

Section 3 will define the proposal to automatically elicit independence requirements and relevant indicators, namely important metrics and independence assertions, extracted from each requirement.

Section 4 will provide a realistic case scenario in which two airborne architectures are being compared, following the aerospace safety process. Results are driven from this scenario.

Finally, conclusions and future works can be found in Section 5.

2. AEROSPACE SAFETY PROCESS

Safety-critical systems, such as aeronautical technologies, must guarantee high-reliability standards to reduce the hazards of using their applications. Figure 1 illustrates the safety assessment process. Its assessments are presented according to the stages of development at which they occur: concept development, preliminary and detailed design (development) - highlighted in red - and integration and verification [1].

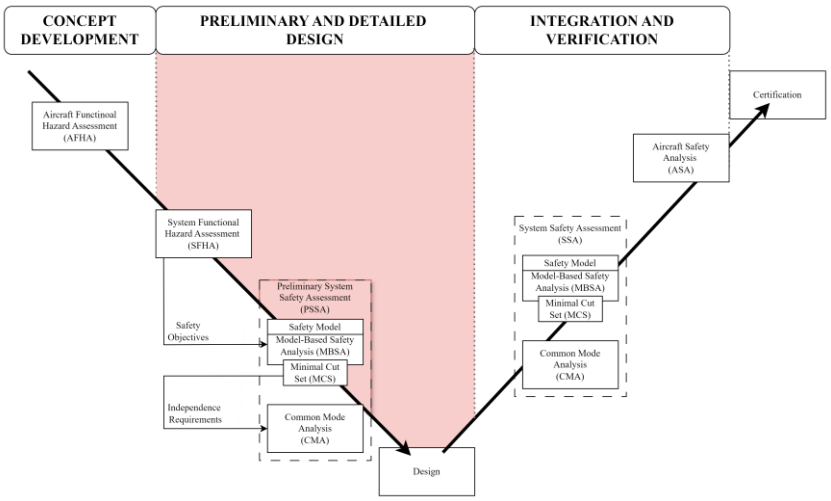

Figure 1 – Safety assessment process [1]

In the early phases of the second stage, in which this research is carried out, the objective is to produce requirements: here not all decisions have been taken and the focus will progressively shift from a high-level point of view into a lower level, or, consistently, it will shift from a preliminary system description into a detailed one. The requirements generated from the safety assessment process in this phase must provide valuable insight into critical aspects of the avionics architectures.
To fully cover the design development, the safety analysis must be flexible to trace these changes and produce as rapid as possible indicators to guide the designer into an architecture that can comply with said requirements. The result is an iterative flow between two separate and different domains of knowledge: system safety and system design.

The design development stage begins with input from the Aircraft Functional Hazard Assessment (AFHA) - from the concept development stage - which consists of the identification and evaluation of potential hazards associated with the aircraft, regardless of the details of its design or implementation [1]. The AFHA is used as input to develop the System Functional Hazard Assessment (SFHA, FHA in short), whose objective is to decompose the aircraft's high-level functions and failure conditions into those of the system.

The Preliminary System Safety Assessment (PSSA) ensures that the proposed architecture meets the probabilistic targets of the FHA (henceforth referred to simply as 'safety objectives'). The main goal of the PSSA is to determine how component failures can lead to the system-level failure conditions identified by the FHA, and to provide both quantitative and qualitative indicators of compliance with safety objectives (probability budget, DAL allocation…). In this article, as shown in Figure 1, this analysis is carried out using MBSA techniques [1]. It is worth mentioning that MBSA is not the only possible assessment to carry out PSSA, as common alternatives in industrial applications use Fault Tree Analysis (FTA), Markov Chains (MC), or Dependability Diagrams (DD). Section 2.1 will detail the PSSA process and especially MBSA techniques.

Another important output of PSSA is the identification of independence requirements that must be evaluated broadly, in terms of design, sharing of mutual resources, installation, and other criteria. Some of these criteria drive the design process and provide the designer with useful information. Other criteria were not deemed relevant in this part of the process as they entail a highly defined system description (e.g. installation procedure, physical location) or detailed information about the final components (e.g. supplier name and batch). Section 2.2 will detail the CMA process.

With misuse of notation, in the article, systems are made by components and not items.

The PSSA safety analysis is iterative and becomes more detailed as development progresses. At the end of the iterations, the final architecture is scrutinized against all the safety-related requirements collected, and its safety is assured through the integration and verification phase. At this stage, System Safety Analysis (SSA) verifies that the proposed final version of the system meets all the requirements that have been generated. SSA is carried out using the same analysis as PSSA. It is not part of this article as the focus is on bridging the main PSSA assessments: MBSA and CMA.

2.1. PSSA - MBSA

Aerospace systems are known to deal with complex components, deeply embedded systems, reconfiguration procedures, and redundancies. Quantifying with numerical probability the reliability of different architectures is not trivial. Model-Based Safety Analysis (MBSA) was conceived to support these computations.

More than one classification of the different types of MBSA has been made, among which the distinction based on the generation of safety models is particularly useful [4].

In this article, focus is on techniques that rely on the use of formal methods for the description of a standalone Failure Propagation Model (FPM). Examples of software are AltaRica Wizard and xSAP [2]. A FPM describes what failure modes can originate in a system component and how these failure modes are propagated into the system [3]. This safety model is evaluated against a specific scenario, generally a failure condition from the SFHA, to automatically derive the combination of failure events that trigger this condition. Compared with alternative methods of aerospace (e.g. FTA, MC), the benefits of FPM are various: given a library of components, a safety model is easier to build, it requires less time, and can be quickly updated if changes happen in the architecture. In truth, FPMs alone can be more assertive than other assessment analyses (e.g., static FTA), since they allow for dynamic considerations. Moreover, FPMs can be simulated and tested, to confirm their consistency with the real system. In the aerospace field, the use of MBSA for certification means has been strongly supported for many years, and it is now included in the aerospace recommended practices [1].

Other than standalone FPM, MBSA also contains Extended System Models (ESM) that, as the name suggests, are system models enriched with fault models [4, 5]. These models rely on heavy numerical simulations and "nominal" case descriptions. Nevertheless, especially if physical interactions play a significant role, using them to get extensive information about the system's behaviour leads to extremely high computational effort and risks incomplete coverage of possible effects [6].

Due to these considerations, AltaRica 3.0, a standalone FPM modelling language from the AltaRica Association, was chosen for this research [8].

AltaRica 3.0

AltaRica is a high-level object-oriented modelling language dedicated to (probabilistic) safety analyses. AltaRica models can be directly transformed into FTA or used to automatically compute the list of MCSs against a specific failure scenario [7].

The scope of the article is only to give an intuitive grasp of the possibilities of this language, without delving into its complexities. Readers interested can find resources to understand better [8, 12].

An example of the AltaRica language is provided in Figure 2.
AltaRica allows for the instantiation of state variables, flow variables, and events. Firing a specific "transition" will change the state variables of the specific class. Then, flow variables are uploaded accordingly to the "assertion" instructions. The semantics of transitions and assertions are based on Guarded Transition System (GTS) [8].

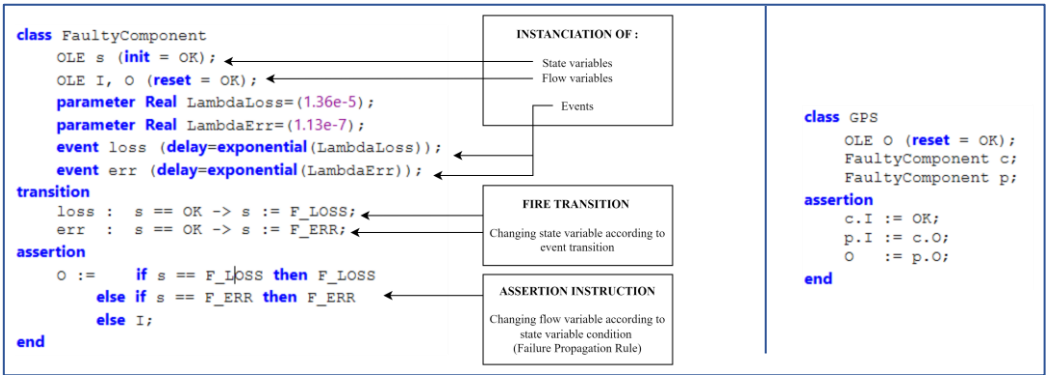

Figure 2 - AltaRica model for *"FaultyComponent"* and *"GPS"*

In Figure 2, the class *"FaultyComponent"* is defined: events *"loss"* and *"err"* will trigger a change in the internal State Machine (SM) defined by *"s"* (see Figure 3) to *"F_LOSS"* or *"F_ERR"* respectively. The flow variable *"O"* is then propagated through other parts of the model to show how component failures affect the system. The domain *"OLE"*, used to instantiate both state variable and flow variables, describes three possible outcomes (*"OK"*, *"F_LOSS"* and *"F_ERR"*). *"GPS"* is built by reusing this semantic for the definition of an internal failure of the core (*"c"*) and a failure of the port (*"p"*).

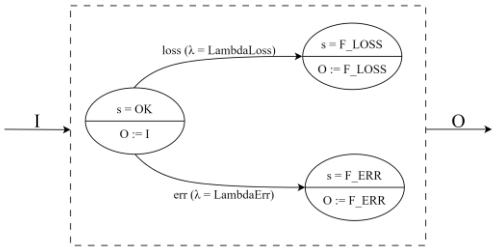

Figure 3 – SM for *"FaultyComponent"*

NUMBER	ORDER	PROBABILISTIC CONTRIBUTION	EVENTS			
1	2	9.79E-07	ss1.adh.c.err	ss2.adh.c.err		
2	2	1.11E-04	ss1.adh.c.err	ss2.adh.c.loss		
3	2	9.79E-07	ss1.adh.c.err	ss2.adh.p.err		
...	...	...	...			
410	3	9.69E-10	ss1.adh.p.err	ss3.gps.c.err	ss4.gps.c.err	
411	3	1.10E-07	ss1.adh.p.err	ss3.gps.c.err	ss4.gps.c.loss	
412	3	9.69E-10	ss1.adh.p.err	ss3.gps.c.err	ss4.gps.p.err	
413	3	1.10E-07	ss1.adh.p.err	ss3.gps.c.err	ss4.gps.p.loss	
...	...	...	...			
579	4	1.40E-06	ss1.adh.p.loss	ss2.adh.p.loss	fcs2.io_fcs.csens1.p2.loss	ss4.ra.c.err
580	4	1.40E-06	ss1.adh.p.loss	ss2.adh.p.loss	fcs2.io_fcs.csens1.p2.loss	ss4.ra.p.err
...	...	...	...			

Figure 4 – List of Minimal Cut Sets

The list of MCSs can be extracted from the safety model (Figure 4). From it, it is possible to highlight the concept of hierarchization: being AltaRica an object-oriented language, a class can be defined in terms of other classes. For example, looking at the 410th MCS, "ss4.gps.c.err" uniquely defines the core failure of the GPS sensor contained in the Sensor Suite 4 (SS4), which fails in erroneous; similarly, in the 413th MCS, "ss4.gps.p.loss" represents its port failing in loss.
The order of the MCS (Figure 4) is the number of events contained in each specific MCS.

Regarding common failure modes, AltaRica has been implemented with the possibility to fire several failure events from different instances simultaneously. In industrial practice, common modes are not associated with probabilities due to the impracticality of finding a quantitative probability for them, for this reason, the analysis explained here will not make use of this feature.

Instead, in industry, common causes of failure appear separately during CMA.

2.2. PSSA – CMA

The complete evaluation of all critical pairs of components is part of CMA. Critical pairs are defined here as all pairs of component's failure modes that are inside a MCS (or equivalently, are under the same "AND" gate of FTA) of most severe failure conditions. Indeed, a dependence between critical components makes the probabilistic computation from the MCS mathematically unreliable and, hence, non-compliant with safety objectives. To prevent this, the list of MCS is examined, critical pairs are extracted and for each, a thorough examination is carried out to avert the risk of a commonality that would escalate the failure condition. These examinations are used to comply with independence requirements.

Other than the quantitative validation of the probability, independence requirements are needed to guide design decisions, as the allocation of shared equipment or resources.
DalCulator, used in this research for the optimal allocation of resources, is briefly described in the following paragraph.

DalCulator
DalCulator is an open-source software developed by ONERA's MBSA team, here used to compute the optimal allocation for foreseen shared resources, concerning independence requirements [9].

For instance, let's take a single MCS where the objective is minimizing the number of resources while avoiding single points of failure due to their allocation:

MCS_i = {ss1.gps.p.err , ss2.gps.p.loss , ss3.gps.p.err},
the resulting computed allocation could be:

- {ss1.gps.p} → r_1
- {ss2.gps.p , ss3.gps.p} → r_2

Where the minimal number of resources computed is equal to two.

User-defined constraints can be given as input to personalize the independence analysis computation. In the experiment of Section 4, *Coloc* constraint was used to force the allocation of a set of components under the same resource.

In addition to deriving functional independence requirements, DalCulator can support the decision process for Development Assurance Level (DAL) and Failure Probability Budget [10, 11], which are not addressed in this research.

3. PROPOSAL

Figure 5 represents the article proposal. The horizontal axis represents the tool used to support the analysis, while the vertical axis represents a division in the procedural step envisioned. The results for both PSSA-MBSA and PSSA-CMA are shown in yellow.

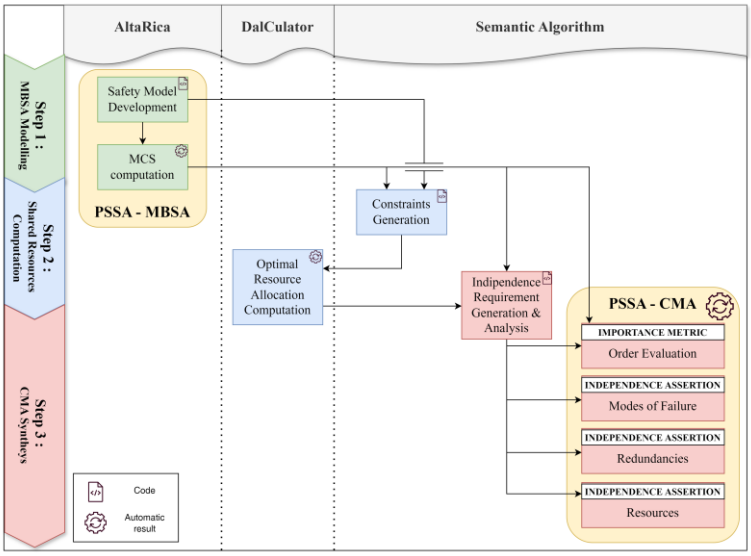

Figure 5 - Process scheme

Step 1 – MBSA modelling

The safety model (FPM) is created in a suitable MBSA environment, such as AltaRica, which allows a model description of the safety-related features of the system and automatic generation of the MCS.

Modelling rules are applied to extend the model information:

1. Identification of functional redundancies (defined later in this Section) using shared naming conventions: instances of such components are named with the same unambiguous pattern followed by a different number (e.g. fcs1, fcs2).

2. Component's modes of failure are limited only to "loss" and "erroneous": this information will appear in the MCS (e.g. ss1.gps.c.loss and ss1.gps.c.err, for a core failure of the first GPS sensor) and will be further exploited.
3. Any transmission of data between separate components takes place through a port; ports are always named "p" and if there are several ports at the same level in the hierarchy, they are distinguished by a number (e.g. ss1.gps.p1, ss1.gps.p2).

The main output of this phase is the list of MCSs automatically produced. It can be used to process safety objectives, with the FPM itself. Hence, this first output of the process represents PSSA - MBSA.

Step 2 – Shared resources allocation

Step two entails the implementation of commonalities due to shared resources.
Resource is a very broad term. It is used here to describe those components that are not part of the safety model due to their low functional importance or different system ownership. However, they are components on which several system elements depend, and which can become a source of common failure modes. The objective is to address these criticalities as early as possible to make informed decisions on architectural changes. For this step, DalCulator Independent Analysis was used (explanation in Section 2.2).
For each considered resource:

1. An automatic generation of constraints (relying on the hierarchical naming convention of the safety model and on the basic knowledge of the resource itself) is performed.
2. DalCulator receives as input the constraints set and the MCSs to compute the optimal allocation.
3. The resulting allocation is forwarded to the next step of the analysis to be processed.

Constraints generated in experiment described in Section 4 are representative of the aerospace field. Nevertheless, the same concept can be applied to different fields and systems.

Step 3 – CMA synthesis

The algorithm will start by eliciting the independence requirements by searching through the MCS the critical pairs. For each elicited requirement, one importance metric and three independence assertions are investigated.

Relying on industrial expertise, the importance metric was used to sort the requirements.
The importance metric is defined as the minimum order of the MCS in which the pair of components is present (Table 1). This provides an immediate insight into the criticality of a possible common failure between the two components.

Table 1. Importance Metric Outcome

Outcome 1	The pair is found in a MCS of order 2	*A common failure will create a single point of failure, mitigations are required*
Outcome 2	(Else) the pair is found in a MCS of order 3	*A common failure will not create a single point of failure, mitigations are suggested*
Outcome 3	(Else) the pair is found in a MCS of order 4 or more	*A common failure between these components is not immediately dangerous*

Independence assertions are defined here as qualitative support material for compliance with the requirement. More precisely, they are early assumptions that test, under some criteria, the independence between the two considered events; they are not meant to be rigid assertions nor a means of compliance with the requirement. Instead, independence assertions give valuable insights into the robustness of the system under analysis and may be used in the future SSA–CMA.

1. The first independence assertion is the "Redundancy Assertion". Functionally redundant components are defined here as components performing the same functionality; indeed, two components created for the same purpose could share similar features and, thus, common failures. Due to MBSA characteristics, it was possible to automatically assess this functional redundancy assertion by building the model with recurring name patterns on such components. Hence information about redundancies can be directly inferred from the list of MCSs.
2. The second independence assertion is the "Failure Mode Assertion". During PSSA it is suggested to start by limiting the failure propagation to "loss" and "erroneous" for each fallible component. It can be

reasonably assumed that no internal common failure can generate a "loss" in one component and an "erroneous" in another.

3. Third independence assertion will inform whereas the pair under analysis is sharing the resource or not. Having covered this assertion in Step 2, they will not be included in the tables in this section.

Table 2. Independence Assertion Outcome

Claim	**Question**	**Analysis outcome**	
Redundancy	Is the pair functionally redundant?	YES	*Pairs are inherently functionally redundant: common modes must be carefully considered*
		NO	*Pairs are not inherently functionally redundant: there is no reason to suspect a common mode exists between them*
Failure Mode	Does the pair fail with similar modes of failure?	YES	*Components fail similarly, both as Loss or both as Err*
		NO	*Components fail differently, one Loss and the other Err*

4. EXPERIMENT

A comparison is made between two architectures (Figure 6), representing two possible designs of an avionic control system, to be installed in an aircraft. The main functionality of these systems is to control actuators and provide stabilization to the system, processing the data received from the sensors.

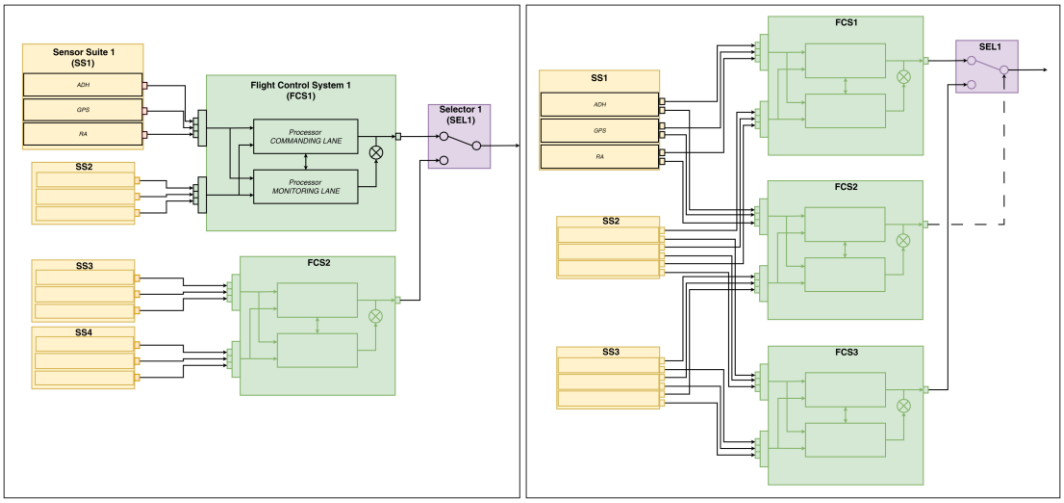

Figure 6 – Proposed architectures

In yellow, Sensor suites (SSs), contain an Air Data and Heading sensor (ADH), a Global Positioning System (GPS), and a Radar Altimeter (RA): they measure and transmit flight data upstream.

In green, the flight control systems (FCS) receive this data. They consist of two processors, one on the monitoring lane and one on the command lane. The logic implemented is 'Fail Silent' (cessation of data output): if the monitoring lane finds an inconsistency between its calculations and those of the command lane, it forces a loss of the output signal.

Each FCS performs a consolidation procedure with the data received from the sensor suites. Taking a conservative stance, each control unit receives the pack of signals, processes them inside the COM-MON pattern, and propagates an OLE signal representing the outcome of the consolidation.

- A loss signal (F_LOSS) is propagated if FCS does not receive sufficient information to perform all consolidations safely.
- An erroneous signal (F_ERR) is propagated if FCS receives enough misleading information on any consolidation and relies on it.
- A correct signal (OK) is transmitted if neither of the above scenarios is true.

In purple, a Selector (SEL) allows for continuing the operations in simplex logic, with a reduction in safety margins, whereas a failure is detected.

Selector changes slightly between the two architectures:

- In left-hand side of Figure 6, it detects a failure of the first FCS to revert to the second FCS.

- In right-hand side of Figure 6, it uses the output from the central FCS2 as an arbiter between the primary FCS1 and secondary FCS3 (indeed, the central output is never being propagated to the actuators).

The failure conditions tested in both architectures are the "Loss of Flight Data Transmission" and the "Erroneous of Flight Data Transmission". They are represented in the extract of the FHA in Table 3. Each failure is described inside the safety model by the same observer: the OLE variable output of the selector.

Table 3. FHA extraction

ID	Failure Condition	MCC	Severity	Description	AltaRica Observer
01_Loss	Loss of Flight Data Transmission	IFR	HAZ	The autopilot stops working. The pilot is forced to engage without any knowledge of the environment.	Output from selector = F_LOSS
02_Err	Erroneous of Flight Data Transmission		CAT	The pilot or autopilot could misbehave due to the acting failure and lose A/C stability	Output from selector = F_ERR

4.1. Step 1 – MBSA modelling

Table 4. MCS summary

	First Architecture DUPLEX		Second Architecture TRIPLEX	
	01_Loss	***02_Err***	***01_Loss***	***02_Err***
Number of MCS	2808	4588	9896	118606
Number of MCS (order 1)	0	2	3	0
Number of MCS (order 2)	675	135	743	93
Number of MCS (order 3)	1404	2212	4365	4237
Number of MCS (order 3>)	729	2241	4788	122936

Table 4 shows that both architectures present single points of failure. The one from first architecture (under 02_Err) can be explained by the duplex nature itself, incapable of detecting an erroneous via a comparison between two misleading sources in the worst-case scenario of consistent errors (erroneous signals representing the same value). The ones from the second architecture (under 01_Loss) are trickier: they depend on SS2 connected to FCS1 and FCS3. An erroneous from that source will generate a loss signal on both FCSs, due to "Fail Silent" implemented logic, which will not be isolated.

CAT failures with single points of failure are not acceptable for civil conventional aircrafts. Therefore, if this had been the scope of the first architecture, the development process could have stopped there (as clear from FHA in Table 3). Instead, it is assumed that the adopted certification standard for the A/C system does not require this criterion.

4.2. Step 2 – Shared resources computation

Given the nature of the experiment, the following problem was defined: data from sensors needs to pass through an A/D converter, and suppliers are suggesting MIMO converters, able to process multiple signals.

Constraints are automatically produced by the following considerations: only signals transmitted from sensors to FCS need an A/D converter and shared signals must be allocated to the same A/D converter. The algorithm automatically processes the information from the safety model in AltaRica, relying on naming patterns, and producing the set of constraints. DalCultator then uses MCS and constraints to compute the optimal allocation, enforcing the absence of single points of failure due to resource allocation.

Table 5 contains an example of the constraints computed. Numerical results are in Table 7.

Table 5. Extract of automatically derived requirements

First architecture	Second Architecture
Coloc('fcs1.io_fcs.csens1.p1','ss1.adh.p')	*Coloc('fcs1.io_fcs.csens1.p1','ss1.adh.p1')* *Coloc('fcs2.io_fcs.csens1.p1','ss1.adh.p2')*

4.3. Step 3 – CMA synthesis

Table 6. Independence Analysis Extract

TARGET	REQUIREMENT & ANALYSIS
fcs1.cc1.com.bit AND fcs2.cc1.com.bit	**Requirement 1: Components fcs1.cc1.com.bit AND fcs2.cc1.com.bit shall be independent**
	A common failure will create a single point of failure, mitigations are required
	Pairs are inherently functionally redundant: common modes must be carefully considered
	Components fail similarly, both as Loss or both as Err
	…
fcs2.cc1.com.lru AND fcs2.cc1.xmon	**Requirement 307: Components fcs2.cc1.com.lru AND fcs2.cc1.xmon shall be independent**
	A common failure will not create a single point of failure, mitigations are suggested
	Pairs are not inherently functionally redundant: there is no reason to suspect a common mode exists between them
	Components fail differently, one Loss and the other Err
	…
ss2.adh.c AND ss2.rada.c	**Requirement 1466: Components ss2.adh.c AND ss2.rada.c shall be independent**
	A common failure between these components is not immediately dangerous
	Pairs are not inherently functionally redundant: there is no reason to suspect a common mode exists between them
	Components fail similarly, both as Loss or both as Err

Table 6 shows an extract of the importance metric and independence assertions defined in Section 3.

4.4 Results

Table 7 - Independence Analysis Summary

Indicator	Description	First Architecture		Second architecture	
GENERAL	Number of independence requirements	657		1607	
	Number of failure events (from MCS)	49		64	
IMPORTANCE METRIC	Number of independence requirements related to a MCS of order 2	628		601	
	Number of independence requirements related to a MCS of order 3	29		865	
	Number of independence requirements related to a MCS of order 4 or higher	0		141	
INDEPENDENCE ASSERTIONS	Number of A/D converters from optimal allocation	4		6	
	Number of critical pairs not functionally redundant	574	87.4%	1398	87.0%
	Number of critical pairs who fail similarly as Loss or Erroneous	651	99.1%	1514	94.2%

A summary of the architecture comparison can be found in Table 7.

The increase in complexity of the second architecture results in 950 more independence requirements to be considered than those from the first. Indeed, the list of failure events from the second architecture shows 15 more than the first architecture. However, it could be argued that the two architectures are comparable from a single point of failure avoidance perspective, as shown by the results of the importance metric analysis. It is worth reminding that the second architecture shows no single point of failure under the CAT failure condition, making it more appealing from a safety perspective.

The results of the optimal allocation of A/D converters show that choosing the second architecture implies two more converters, to deal with the larger number of transmissions needed.

The same percentage of requirements can be argued independently of the functional redundancy assertion, but in absolute terms, the second architecture has 209 requirements related to functional redundancies, while the first has 83. Finally, 93 requirements from the second architecture can be reasonably argued to be independent due to different failure modes, while the same assertion applies to only six requirements in the first architecture.

A safer architecture comes at the cost of more requirements: this analysis aids the designer in choosing the architecture that best complies with the appropriate regulations.

5. CONCLUSIONS

The article presents an approach for integrating MBSA results to support other parts of the security process, namely the essential management of independence requirements.

In the authors' opinion, the most important feature here addressed is the possibility to automatically and freely address criticalities and foreseen unsafe aspects related to dependencies between components, from the very first parts of the preliminary design development. Highlighting critical pairs, sorting them using qualitative indicators (e.g. the minimal order of the MCS in which they appear), and addressing independence assertions, are beneficial features that will surely speed up CMA during the verification phase. This is especially true when dealing with large numbers of requirements, as common in industry.

The ability to calculate an optimal allocation based on the DalCulator is another valuable feature that supports the design decision process when selecting equipment. Moreover, the flexibility of using naming conventions directly from the safety model enhances the usefulness of MBSA techniques, maximizing the extraction of relevant information.

Future work aims to add metrics and assertions to cover the analysis as much as possible, including probabilistic consideration. We are curious to carry out similar analyses also in the verification phase. The flexibility of DalCulator in different sharing resource scenarios will be investigated, as well as its other functionalities for DAL and budget allocations.

Acknowledgements for funding organizations

This work was partly funded by Leonardo Helicopter Division.

Acknowledgments

We'd like to thank the ONERA RIME Team: Kevin Delmas, Pierre Bieber, Christel Seguin, Tatiana Prosvirnova, Sergio Pizziol. Thank you for giving us support to DalCulator and for many helpful comments. We would like to thank Enrico Zio for his endless interest in this journey.

References

[1] Society of Automotive Engineers. ARP4761A: Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment. 2023.

[2] I. Lanzani, R. Scattolini, E. Zio, A. Cimatti, M. Bozzano, and S. Tonetta. Two formal methodologies of Model-Based Safety Assessment for Fault Tree Analysis. International Conference on System Reliability and Safety (ICSRS), 376–383, 2023.

[3] R. Bernard, J.-J. Aubert, P. Bieber, C. Merlini, and S. Metge. Experiments in model based safety analysis: Flight controls. IFAC Proceedings Volumes, 40, 43–48, 2007.

[4] S. Gradel, B. Aigner, and E. Stumpf. Model-based safety assessment for conceptual aircraft systems design. CEAS Aeronautical Journal, 1–14, 2022.

[5] A. Joshi and M. P. Heimdahl. Model-based safety analysis of simulink models using SCADE design verifier. Computer Safety, Reliability, and Security: 24th International Conference, 24, 122–135, 2005.

[6] P. Hönig, R. Lunde, and F. Holzapfel. Model based safety analysis with smartIflow. Information, 8, 7, 2017.

[7] T. Prosvirnova and A. Rauzy. Automated generation of minimal cut sets from AltaRica 3.0 models. IJCCBS, 6, 50, 2015.

[8] T. Prosvirnova. AltaRica 3.0: a model-based approach for safety analyses. PhD Thesis, Ecole Polytechnique, 2014.

[9] ‘DalCulator: https://github.com/onera/dalculator/releases/tag/v2.0.0

[10] P. Bieber, R. Delmas, and C. Seguin. DALculus – Theory and Tool for Development Assurance Level Allocation. Computer Safety, Reliability, and Security, 43–56, 2011.

[11] K. Delmas, L. Chambert, C. Frazza, and C. Seguin. Optimization of Development Assurance Level Allocation. Digital Avionics Systems Conference (DASC), 1–10, 2023.

[12] M. Batteux, T. Prosvirnova, and A. Rauzy. AltaRica 3.0 in ten modelling patterns. Int. J. Critical Computer-Based Systems, 9, 133-165, 2019.