Processus de réalisation d'AMDEC basé sur les outils OpenAltaRica

Sara Rachid, Emmanuel Clement, Nicolas Le Berre, Michel Batteux

Institut pour la Maîtrise des Risques
Sûreté de Fonctionnement - Management - Cindyniques

Processus de réalisation d'AMDEC basé sur les outils OpenAltaRica
Performance of FMECA Based on OpenAltaRica Tools

Sara RACHID
Thales LAS
Rungis, France
sara.rachid.e@thalesdigital.io

Emmanuel CLEMENT,
Nicolas LE BERRE
Thales DMS
Brest, France
emmanuel.clement@fr.thalesgroup.com
nicolas.leberre@fr.thalesgroup.com

Michel BATTEUX
IRT SystemX
Palaiseau, France
michel.batteux @irt-systemx.fr

***Résumé*—Cette communication présente une méthode permettant de mener des AMDEC sur des systèmes complexes, en se basant sur une modélisation AltaRica 3.0. Cette méthode permet d'exploiter le concept du MBSA dans le but d'améliorer la lisibilité, l'exactitude des résultats et les coûts d'une AMDEC.**

Mots-clés— Sûreté de fonctionnement, AMDEC, Systèmes complexes, OpenAltaRica

***Abstract*—This communication presents an approach which aims to perform FMECA on complex systems, based on AltaRica 3.0 modelling. This approach aims to invest the MBSA concept in improving the readability, the accuracy and the cost of an FMECA.**

Keywords— System Dependability, FMECA, Complex Systems, OpenAltaRica

I. INTRODUCTION

A. Contexte

L'analyse des modes de défaillance, de leurs effets et de leurs criticités (AMDEC) d'un système industriel complexe apparait comme une tâche difficile à réaliser. Cela pour deux raisons principales.

D'une part, la quantité des modes de défaillance à traiter peut rapidement atteindre plusieurs milliers, voire dizaines de milliers pour des systèmes de guerre électronique d'avion d'armes par exemple. Cette quantité, couplée au nombre d'effets systèmes opérationnels et de sécurité à analyser, entrainent une explosion des combinaisons à traiter.

D'autre part, trois « ingénieries » sont nécessaires pour réaliser sereinement une AMDEC. L'ingénierie de la sûreté de fonctionnement, premièrement, est garante de la méthode et pilote l'activité. L'ingénierie matérielle, quant à elle, doit apporter son expertise sur les aspects bas-niveau. Enfin, l'ingénierie système contribue à analyser les impacts « systèmes » et « opérationnels » des défaillances bas-niveau.

Le concept MBSA (Model-Based Safety Assessment), implémenté par AltaRica, remédie à ces limites par la création de modèles plus proches des architectures fonctionnelles et physiques des systèmes. Ces modèles sont plus lisibles, plus faciles à réaliser et à partager entre les différentes ingénieries. Ceci permet d'avoir des modèles de systèmes de plus en plus matures, grâce au partage et à la validation des hypothèses des différentes parties.

La méthode présentée ici décrit l'approche retenue pour la réalisation d'AMDEC dans l'outil MBSA libre d'utilisation et de diffusion System-Analyst ([2] et [3]), et mise en pratique à THALES Defense Mission Systems.

B. Objectif

Afin de gérer les contraintes de la méthode AMDEC citées, cette communication propose un processus de réalisation des analyses AMDEC sur les systèmes complexes, en se basant sur une modélisation AltaRica 3.0. Cette méthode permettra d'exploiter le concept du MBSA pour améliorer la lisibilité, l'exactitude des résultats et les coûts des analyses AMDEC.

C. Approche

Le processus de réalisation d'analyses de type AMDEC que nous présentons est articulé autour des étapes principales suivantes :

1. Analyse préliminaire des risques : cette étape permet d'identifier les évènements redoutés opérationnels et de sécurité du système étudié.

2. Phase de modélisation sous AltaRica 3.0 : cette étape permet de modéliser le système en suivant les formalismes des ingénieries matérielle et système sous un modèle regroupant différents éléments : ses constituants locaux au travers du formalisme de l'ingénierie matérielle, son architecture au travers du formalisme de l'ingénierie système, les modes de défaillance qui seront analysés, enfin

les stratégies de test et les effets intermédiaires et finaux à considérer.

3. Phase de simulation OpenAltaRica : cette étape permet de propager les défaillances par simulation automatique du modèle.

4. Génération automatique de l'AMDEC : cette étape permet par propagation de chacun des modes de défaillance d'identifier différents éléments : les effets intermédiaires et les effets finaux opérationnels et de sécurité, les modes dégradés et la criticité des effets finaux, les taux de couverture et de localisation par les tests, les modes de défaillance dangereux non détectés, enfin les défaillances de logiciels entrainant directement des événements redoutés.

Un cas concret est utilisé dans la communication pour illustrer les phases du processus. De plus, la communication apporte les résultats d'application du processus sur un cas industriel.

La suite de la communication se décompose comme suit. La deuxième partie présente la méthode AMDEC. La troisième partie décrit le langage de modélisation AltaRica 3.0 et le projet associé OpenAltaRica. La quatrième partie décrira la méthode proposée. Celle-ci sera illustrée sur un cas industriel en cinquième partie. La sixième partie indiquera les bénéfices et limites de la méthode proposée. Enfin, la dernière partie conclura cette communication

II. LA MÉTHODE AMDEC

A. *Définition*

Définie par la norme européenne Norme NF EN IEC 60812, la méthode AMDEC est un outil d'analyse dysfonctionnelle qui permet d'analyser les effets opérationnels et sécuritaires (safety) des défaillances de bas-niveau sur le système et leurs criticités. Cette méthode permet de recenser les modes de défaillance ayant les impacts les plus critiques sur l'opération du système. Cela permet de définir par la suite des actions correctives pour réduire les risques soulevés, comme la modification de l'architecture, la définition de points de tests, la définition d'actions de maintenances préventives, etc. Aussi, les résultats de l'AMDEC sont utiles dans les études de fiabilité prévisionnelle, de maintenabilité, de testabilité et de sécurité (safety).

B. *Procédure*

Selon qu'elle étudie les défaillances des composants physiques ou des fonctions, une AMDEC est menée selon deux approches différentes : « approche composant » et « approche fonctionnelle ».

La réalisation d'une AMDEC passe par les étapes suivantes :

- Définition du système : cette étape vise à obtenir une définition complète du système incluant : les profils d'utilisation du système à analyser, la définition des éléments qui seront étudiés (fonctions, composants physiques), leurs interdépendances, leurs interfaces externes selon chaque profil d'utilisation.
- Définition des modes de défaillance potentiels de chaque élément étudié (fonction ou composant physique).
- Evaluation de l'effet de chaque mode de défaillance au niveau de l'élément local et au niveau de l'opération globale du système et la sécurité.
- Attribution d'un niveau de sévérité au mode de défaillance étudié en fonction de l'effet le plus critique qu'il produit.
- Identification des méthodes de détection des modes de défaillance étudiés (tests intégrés, informations visuelles, etc.).
- Définition d'actions correctives pour gérer les modes de défaillance les plus critiques et suivi des résultats de l'application de ces actions.

Les résultats de l'analyse AMDEC sont généralement documentés sous forme de tableaux, comme suit :

TABLE I. EXEMPLE DE TABLEAU AMDEC

Item Id.	**Failure Mode**	**Failure Rate**	**Causes**	**Effects**			**Severity Class.**	**Actions**	**Remarks**
				Local Effect	***Next Higher Level***	***End Effect***			
Item #1	**FM #1.1**	$\lambda_{1.1}$	**…**	…	…	…	…	…	…
	FM #1.2	$\lambda_{1.2}$	**…**	…	…	…	…	…	…
Item #2	**…**		**…**	…	…	…	…	…	…

C. Limites

La méthode AMDEC forme un outil puissant pour l'analyse des modes de défaillance, grâce à sa nature systématique qui favorise le traitement exhaustif des modes de défaillance d'un système. Cependant, la méthode AMDEC a plusieurs limites qui rendent sa réalisation compliquée.

D'abord, comme la méthode exige d'analyser un par un les modes de défaillance, cela crée une forte dépendance entre le nombre de modes de défaillance du système et le temps de réalisation de l'AMDEC et son coût. Cette contrainte oblige dans plusieurs cas de réduire la complexité de l'analyse au détriment de ses objectifs de précision.

Deuxièmement, une AMDEC se base sur le principe de propagation d'un effet local au niveau global du système pour évaluer sa criticité. Ceci implique une réalisation et une maintenance difficiles et sujettes à erreurs pour des systèmes complexes de par leur nature (nombre de composants important, interconnexions fortes entre ces composants, enfin grand effort d'abstraction et maîtrise du système exigés).

L'analyse de l'effet d'une défaillance bas-niveau à un niveau plus global peut être humainement difficile à établir pour un système relativement complexe par son nombre de composants et leurs interconnexions.

De plus, le formalisme de réalisation et de documentation des résultats des analyses AMDEC est très éloigné de l'architecture fonctionnelle et physique du système analysé. Cela rend difficile son partage et sa validation avec les équipes d'« ingénierie matérielle » et d'« ingénierie système », ainsi que sa maintenance par la suite.

III. OpenAltaRica et AltaRica 3.0

A. AltaRica 3.0

AltaRica 3.0 est un langage de modélisation événementiel et orienté objet, dédié aux analyses probabilistes du risque de systèmes complexes [5]. Il permet de modéliser les systèmes à plus haut niveau que les formalismes classiques d'analyse du risque (arbres de faute, chaînes de Markov, réseaux de Pétri stochastiques, etc.), sans augmenter la complexité des calculs des indicateurs du risque. AltaRica 3.0 est la combinaison du cadre mathématique GTS, pour Guarded Transitions System ([8]), et du paradigme de structuration S2ML, pour System Structure Modeling Language ([6]).

Le cadre mathématique GTS permet de rendre compte des aspects comportementaux que l'on souhaite modéliser, c'est-à-dire la sémantique d'exécution. Cette exécution est similaire aux autres formalismes à événements discrets dans le sens où chaque fois qu'une transition est tirable, elle est planifiée et potentiellement tirée après un certain délai ([9] et [11]). Ces délais peuvent être déterministes ou stochastiques ; et AltaRica 3.0 fournit les délais usuels (du type exponentiel, Weibull, etc.) et des délais empiriques.

Le paradigme de structuration S2ML permet de construire les modèles de différentes manières. S2ML unifie les deux paradigmes de structuration dominants des langages de modélisation : l'orienté objet à classe et l'orienté objet à prototype. Il permet de modéliser suivant deux approches combinables. L'approche dite 'top-down', avec une vision au niveau du système, qui permet la réutilisation de schémas de modélisation, et donc utilisant le paradigme orienté objet à prototype. L'approche dite 'bottom-up', avec une vision au niveau des composants, qui permet la réutilisation de composants définis en bibliothèques, et donc utilisant le paradigme orienté objet à classe.

Le pouvoir d'expression du langage AltaRica 3.0 n'est seulement qu'une partie de la solution pour l'analyse du risque de systèmes complexes. Il est en effet nécessaire d'avoir des outils d'évaluation associés qui soient efficaces. En effet, l'évaluation pour la sûreté ou la fiabilité est un problème dit de complexité #Phard ([10]). Cela signifie que cette barrière de complexité n'est pas un problème de technologie, mais un problème théorique. Il est donc nécessaire de concevoir des outils d'évaluation implémentant efficacement les algorithmes de traitement. Le langage AltaRica 3.0 vient de ce fait avec un ensemble d'outils permettant de calculer différents indicateurs. Fig. 1 représente l'ensemble des outils d'évaluation associés au langage AltaRica 3.0. Il s'agit autant des outils graphiques permettant de concevoir et éditer des modèles AltaRica 3.0, ainsi que des outils permettant de les évaluer. Le simulateur pas-à-pas est l'outil permettant de réaliser des expériences virtuelles sur les modèles (vérification manuelle de modèles par exemple). Les deux outils de compilation : vers les arbres de faute et vers les chaînes de Markov (pas encore disponible) permettent de générer, à partir de modèles AltaRica 3.0, des modèles vers ces formalismes cibles. L'utilisation d'outils d'évaluation associés à ces formalismes est ensuite possible. Le générateur de séquences permet d'obtenir l'ensemble des séquences d'évènements amenant à un événement redouté considéré. Enfin, le simulateur stochastique permet d'obtenir des statistiques sur des exécutions aléatoires de modèles AltaRica 3.0. L'ensemble de ces outils est distribué au sein de la plateforme OpenAltaRica issu du projet du même nom.

B. Le projet OpenAltaRica

Le projet OpenAltaRica est un projet réalisé au sein de l'Institut de Recherche Technologique (IRT) SystemX et en partenariat avec Thales, Apsys, Safran et AltaRica Association. Ce projet, de 5 ans et terminé en septembre 2019, a produit l'implémentation de la plateforme de référence pour le langage AltaRica 3.0. Cette plateforme d'expérimentation est disponible sur le site du projet ([4]). Elle intègre l'ensemble des outils développés durant le projet : l'environnement intégré de modélisation et simulation AltaRica Wizard ([7]), le simulateur pas-à-pas, le simulateur stochastique ainsi que le compilateur vers les arbres de faute. Des travaux complémentaires sont encore en cours afin de compléter la plateforme actuelle par d'autres fonctionnalités ou outils.

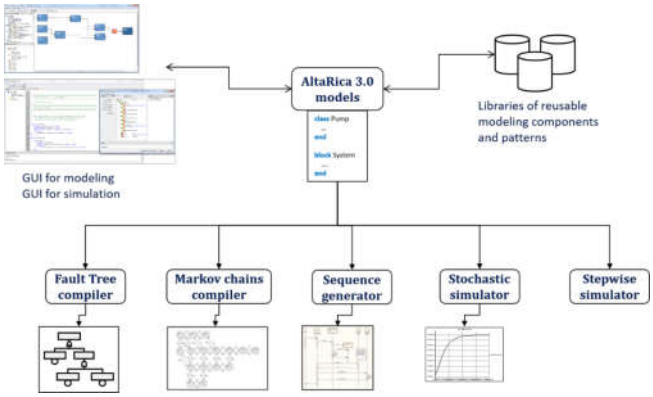

Fig. 1. Outils d'évaluation associés au langage AltaRica 3.0.

IV. DESCRIPTION DE LA MÉTHODE

A. *Généralités sur la méthode*

1) Principe général

Le processus proposé s'articule autour de trois phases principales. D'abord, l'analyse préliminaire des risques qui nous permettra de soulever les événements redoutés opérationnels et de sécurité du système.

Ensuite, la phase de modélisation qui permet de réaliser un modèle organique AltaRica 3.0 du système regroupant l'ensemble des éléments nécessaires pour l'étude. Ces éléments sont ses composants fonctionnels et/ou physiques, leurs interdépendances, les stratégies de test, les modes de défaillance et les événements redoutés à étudier.

Ensuite, la phase de simulation permet de propager les différents modes de défaillance, un par un, pour identifier les tests qu'ils déclenchent et les événements qu'ils impliquent.

Tout au long de cette partie, un cas applicatif simple sera employé pour illustrer les détails de chaque étape.

Après l'explication de la méthode, une analogie sera faite entre les éléments du modèle et ceux d'une AMDEC traditionnelle.

2) Cas applicatif

Le cas applicatif choisi consiste en trois niveaux hiérarchiques, comme représenté en Fig. 2 : le niveau du système, un niveau intermédiaire de deux blocs et un bas niveau constitué de cinq composants.

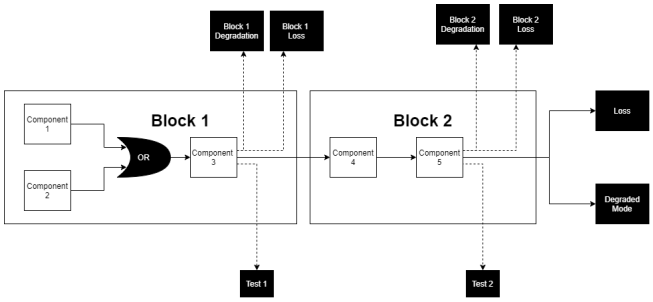

Fig. 2. Schéma du cas applicatif.

Deux effets finaux sont considérés au niveau système : le passage aux états de fonctionnement dégradés (ex : perte d'une voie de réception sur 10) et la perte opérationnelle totale du système.

Au niveau intermédiaire, on considère comme effets le passage aux états de fonctionnement dégradés et la perte totale opérationnelle pour chaque bloc.

Un test est instauré à la sortie de chaque bloc du niveau intermédiaire pour surveiller son état.

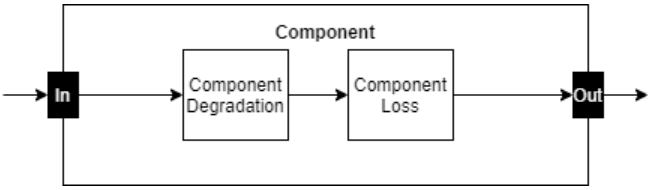

Fig. 3. Modes de défaillance considérés dans le cas applicatif

Pour simplifier l'étude du cas applicatif, on considérera que les composants bas niveau ont uniquement deux modes de défaillance potentiels, à savoir le mode « fonctionnement dégradé » (ex : dégradation d'une capacité de filtrage) et la perte totale de fonctionnement.

B. *Etapes de la méthode*

1) Analyse préliminaire des risques

Cette phase consiste à faire une analyse préliminaire des risques pour soulever les événements redoutés opérationnels et de sécurité potentiels du système étudié, que peut impliquer une défaillance de ses fonctions. L'objectif de l'AMDEC par la suite est de relier chaque défaillance bas-niveau aux événements redoutés système qu'elle peut produire.

2) Modélisation du système

a) Modélisation des flux : l'état d'un flux échangé entre des composants du système reflète la présence de défaillances dans les composants qu'il aurait parcourus, et la nature de ces défaillances. Par exemple, un flux parvenant d'un composant en fonctionnement dégradé aura une « qualité » dégradée. Pour représenter cette dépendance, les flux échangés sont représentés par des valeurs numériques parmi une liste de valeurs définies. Les valeurs de la liste renvoient à des types précis de défaillances, adaptés aux besoins des clients et aux objectifs de l'analyse. Fig. 4 présente la définition des valeurs des défaillances qui sera considérée pour le traitement du cas applicatif.

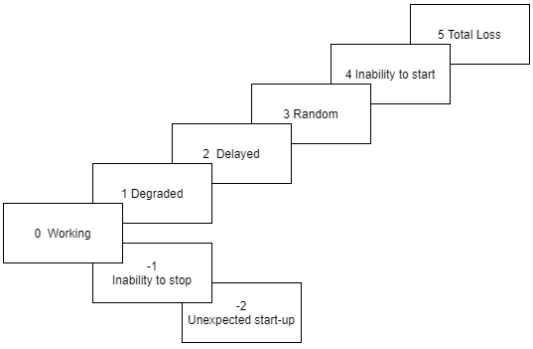

Fig. 4. Valeurs des défaillances du cas applicatif

b) Modélisation des modes de défaillance : pour calculer la sortie d'un composant, il est important de prendre en compte les entrées du composant et les modes de

défaillance du composant qui affectent cette sortie. Concrètement, un flux de sortie sera l'état le plus critique entre l'état de l'entrée et le mode de défaillance. En modélisation, le calcul de la sortie reviendra à comparer la valeur numérique de l'entrée et celle correspondante au mode de défaillance modélisé.

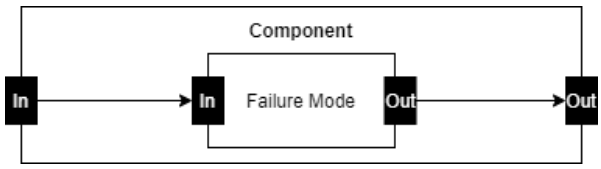

Fig. 5. Modélisation d'un mode de défaillance

Suivant la définition des modes de défaillance considérée (Fig. 4), la valeur de la sortie correspondra à la valeur maximale entre la valeur numérique de l'entrée et celle du mode de défaillance. Néanmoins, si les valeurs comparées sont toutes inférieures ou égales à 0 (fonctionnement normal, incapacité à s'arrêter, démarrage intempestif), la valeur numérique retenue sera alors la valeur minimale. Cette méthode de modélisation permet de considérer les défaillances de type « intempestives » qui se propagent différemment des modes de défaillances de type « dégradation ou perte ». Fig. 6 est la traduction, en AltaRica 3.0, de ce qui vient d'être présenté.

```
block Component
    // Component FLow Variables
    Integer in (reset = 0);
    Integer out(reset = 0);
    // Failure Mode Flow Variables
    Integer FM_in (reset = 0);
    Integer FM_out(reset = 0);
    //Failure Mode Numerical Value
    Integer FM_value (init = 8);
    //Failure Mode Current State
    Boolean FM_launched (init = false);
    //Failure Mode Event
    event FM_event (delay=exponential(1e-08));
    transition
        //Change of Failure Mode State
        FM_event: FM_launched == false -> FM_launched := true;
    assertion
        FM_in := in;
        //Output Value Definition
        FM_out := if FM_launched == false then FM_in
               else if max(FM_in, FM_value)>0 then max(FM_in, FM_value)
                    else min(FM_in, FM_value);
        out := FM_out;
end
```

Fig. 6. Modèle AltaRica 3.0 d'un composant

La modélisation des modes de défaillance, telle que présentée ci-dessus, a l'avantage d'être également adaptée à des contextes autres que l'AMDEC, qui peuvent exiger d'analyser des défaillances multiples. Ceci grâce à la considération des valeurs d'entrée qui constituent des mémoires des défaillances apparues en amont.

c) Modélisation des effets intermédiaires et finaux : L'apparition d'un effet à un niveau intermédiaire ou au niveau système est définie par le remplissage d'une ou plusieurs conditions sur les sorties des composants à ce niveau-là. Ainsi, la représentation d'un effet intermédiaire ou final sera assurée par des éléments d'observation AltaRica 3.0, nommés « observers » et représentés en Fig. 7, qui surveilleront les conditions impliquant l'apparition de cet effet. Concrètement, on attribue à un « observer » une équation booléenne définie par les sorties du système dont dépend l'effet modélisé. Quand l' « observer » prend la valeur « Vrai », ceci signifie l'apparition de l'effet correspondant.

```
//Definition of Block 1 Intermediate Effects
observer Boolean Degraded_block_1 = Block_1.Component_3.out == 3;
observer Boolean Loss_block_1 = Block_1.Component_3.out == 5;
//Definition of Block 2 Intermediate Effects
observer Boolean Degraded_block_2 = Block_2.Component_5.out == 3;
observer Boolean Loss_block_2 = Block_2.Component_5.out == 5;

//Definition of System Feared Events
observer Boolean Op_Loss = Block_2.Component_5.out == 5;
observer Boolean Degraded_Mode = Block_2.Component_5.out == 3;
```

Fig. 7. Eléments d'observation des effets en AltaRica 3.0

d) Modélisation des tests : un test, par définition, est un dispositif qui surveille l'état d'une ou plusieurs sorties. De la même façon que les effets, et comme le montre Fig. 8, les tests seront modélisés par des « observers », dont la valeur est celle d'une équation booléenne dépendante des sorties surveillées.

```
//Definition of tests
observer Boolean Test_1 =  Block_1.Component_3.out != 0;
observer Boolean Test_2 =  Block_2.Component_5.out != 0;
```

Fig. 8. Eléments d'observation des tests en AltaRica 3.0

e) Modélisation des portes logiques : les portes logiques « ET » et « OU » sont déployées dans ce contexte pour référer respectivement aux branchements fonctionnels en série et en redondance. Un branchement en série implique que toutes les entrées doivent fonctionner pour que la sortie fonctionne. Un branchement en redondance, en revanche, signifie qu'il suffit qu'une entrée fonctionne pour que la sortie fonctionne. Ainsi, comme la sortie dépend de toutes les entrées dans la porte « ET », elle prendra l'état de l'entrée la plus sévèrement défaillante. Pour la porte « OU », sa sortie exige le fonctionnement d'au moins une entrée. Elle prendra donc l'état de la sortie la moins défaillante. Dans le cas où les entrées comparées sont limitées à un fonctionnement normal et des défaillances de type « intempestives », la sortie de la porte « OU » prendra la défaillance intempestive la plus sévère. Comme le montre Fig. 9, cette représentation se traduit en AltaRica 3.0 par l'utilisation des fonctions « min » et « max ». Elle a l'avantage d'être indépendante du nombre d'entrées, et d'être par conséquent adaptée aux portes multi-entrées. Similairement à la modélisation choisie pour les modes de défaillance, cette modélisation des portes logiques permet de considérer les défaillances de type « intempestives » qui se propagent différemment des modes de défaillances de type « dégradation ou perte ».

```
block Component

    // Definition of the block variables
    assertion
        // OR and AND Logical Gates
        OR_out := min(OR_in_1, OR_in_2);
        AND_out := if max(AND_in_1, AND_in_2) > 0 then max(AND_in_1, AND_in_2)
                   else min(AND_in_1, AND_in_2);
    // Other assertions

end
```

Fig. 9. Modélisation des portes logiques en AltaRica 3.0

f) Modélisation des composants : pour les composants bas-niveau, ils peuvent être modélisés par des objets AltaRica 3.0 de type « block », qui regroupent leurs variables d'entrée/sortie et leurs modes de défaillance. (voir Fig. 6)

Il est toutefois possible, si les composants du système sont identiques, de définir une classe « Component » et de l'instancier, comme le montre Fig. 10. Ceci est le cas du cas applicatif présenté.

```
class Component
    // Component FLow Variables
    Integer in (reset = 0);
    Integer out(reset = 0);
    // Failure Mode Flow Variables
    Integer Loss_in (reset = 0);
    Integer Loss_out(reset = 0);
    Integer Degradation_in (reset = 0);
    Integer Degradation_out(reset = 0);
    //Failure Mode Numerical Values
    Integer Loss_value (init = 5);
    Integer Degradation_value (init = 3);
    //Failure Mode Current States
    Boolean Loss_launched (init = false);
    Boolean Degradation_launched (init = false);
    //Failure Mode Events
    event Loss_event (delay=exponential(1e-08));
    event Degradation_event (delay=exponential(1e-06));
    transition
        //Change of Failure Mode States
        Loss_event: Loss_launched == false -> Loss_launched := true;
        Degradation_event: Degradation_launched == false
                           -> Degradation_launched := true;
    assertion
        //Degradation Mode Changes
        Degradation_in := in;
        Degradation_out := if Degradation_launched == false then Degradation_in
              else if max(Degradation_in, Degradation_value)>0
                   then max(Degradation_in, Degradation_value)
                   else min(Degradation_in, Degradation_value);
        //Loss Mode Changes
        Loss_in := Degradation_out;
        Loss_out := if Loss_launched == false then Loss_in
              else if max(Loss_in, Loss_value)>0
                   then max(Loss_in, Loss_value)
                   else min(Loss_in, Loss_value);
        out := Loss_out;
end
```

Fig. 10. Définition de la classe "Component" en AltaRica 3.0

Pour les blocs du système des niveaux intermédiaires, ils seront composés des composants des plus bas niveaux, en plus des « observers » représentant les effets intermédiaires à chaque niveau.

g) Modélisation du niveau système : le système sera présenté par un modèle hiérarchique qui garde les détails des différents niveaux. Ceci est d'une grande importance parce qu'il garde la traçabilité entre les effets finaux du système et leurs causes aux niveaux les plus bas.

En considérant la classe « Component » définie ci-dessus, la modélisation du cas applicatif est donnée en Fig. 11.

```
block System
    // Definition of Block 1
    block Block_1
        //Definition of Block 1 Components
        Component Component_1, Component_2, Component_3;
        // Input/Output Variables of The OR Gate
        Integer OR_in_1(reset=0);
        Integer OR_in_2(reset=0);
        Integer OR_out(reset=0);
        // Flow Links Between Components
        assertion
            OR_in_1 := Component_1.out;
            OR_in_2 := Component_2.out;
            OR_out := min(OR_in_1,OR_in_2);
            Component_3.in := OR_out;
    end
    // Definition of Block 2
    block Block_2
        Component Component_4, Component_5;
        assertion
            Component_5.in := Component_4.out;
    end
    assertion
        Block_2.Component_4.in := Block_1.Component_3.out;

    //Definition of Block 1 Intermediate Effects
    observer Boolean Degraded_block_1 = Block_1.Component_3.out == 3;
    observer Boolean Loss_block_1 = Block_1.Component_3.out == 5;
    //Definition of Block 2 Intermediate Effects
    observer Boolean Degraded_block_2 = Block_2.Component_5.out == 3;
    observer Boolean Loss_block_2 = Block_2.Component_5.out == 5;
    //Definition of System Feared Events
    observer Boolean Op_Loss = Block_2.Component_5.out == 5;
    observer Boolean Degraded_Mode = Block_2.Component_5.out == 3;
    //Definition of tests
    observer Boolean Test_1 =  Block_1.Component_3.out != 0;
    observer Boolean Test_2 =  Block_2.Component_5.out != 0;

end
```

Fig. 11. Modélisation du cas applicatif en AltaRica 3.0

3) Simulation "pas-à-pas"

a) Généralités sur l'outil : le simulateur pas à pas permet de simuler la réalisation d'un événement et d'observer ses effets sur les différentes variables du modèle exécuté. A partir de ce principe de fonctionnement, nous simulons un par un les modes de défaillance du cas applicatif et observons les valeurs des « observers ». Il est important de remettre le système dans son état initial à chaque simulation, afin d'isoler l'effet de chaque mode de défaillance. Ceci revient à appliquer l'algorigramme de Fig. 12.

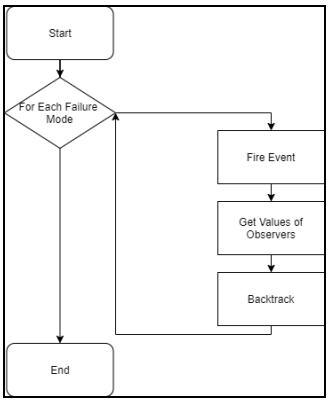

Fig. 12. Algorigramme de l'étude via la simulation pas-à-pas

L'utilisation du simulateur « pas-à-pas » est possible à partir de son interface graphique, comme à partir de son interpréteur de commandes. L'application de ce processus chez Thales s'appuie sur une chaîne outillée ([3]) qui automatise la génération de la séquence de commandes nécessaires pour simuler un système et son exécution par le simulateur.

Afin de simuler les modes de défaillance du cas applicatif, Fig. 13 présente les commandes utilisées, telles que :

- « print tr » : permet d'afficher les événements tirables du système avec leurs indices.
- « fire 0 » : permet de tirer l'événement d'indice 0.
- « print o » : permet d'afficher les valeurs actuelles des « observers ».
- « back » : permet d'annuler le dernier événement tiré.
- « quit » : permet de mettre fin à la simulation.

```
print tr
fire 0
print o
back
fire 1
print o
back
fire 2
print o
back
fire 3
print o
back
fire 4
print o
back
fire 5
print o
back
fire 6
print o
back
fire 7
print o
back
fire 8
print o
back
fire 9
print o
back
quit
```

Fig. 13. Commandes du simulateur pour le cas applicatif

Les résultats de l'exécution de ces commandes sont :

```
Fireable transition(s):
  [0]  Block_1.Component_1.Loss_event (delay = exponential( 1e-008 ))
  [1]  Block_1.Component_1.Degradation_event (delay = exponential( 1e-006 ))
  [2]  Block_1.Component_2.Loss_event (delay = exponential( 1e-008 ))
  [3]  Block_1.Component_2.Degradation_event (delay = exponential( 1e-006 ))
  [4]  Block_1.Component_3.Loss_event (delay = exponential( 1e-008 ))
  [5]  Block_1.Component_3.Degradation_event (delay = exponential( 1e-006 ))
  [6]  Block_2.Component_4.Loss_event (delay = exponential( 1e-008 ))
  [7]  Block_2.Component_4.Degradation_event (delay = exponential( 1e-006 ))
  [8]  Block_2.Component_5.Loss_event (delay = exponential( 1e-008 ))
  [9]  Block_2.Component_5.Degradation_event (delay = exponential( 1e-006 ))

Observer(s):
  Degraded_Mode = false
  Degraded_block_1 = false
  Degraded_block_2 = false
  Op_Loss = false
  Test_1 = false
  Test_2 = false

Observer(s):
  Degraded_Mode = false
  Degraded_block_1 = false
  Degraded_block_2 = false
  Op_Loss = false
  Test_1 = false
  Test_2 = false

Observer(s):
  Degraded_Mode = false
  Degraded_block_1 = false
  Degraded_block_2 = false
  Op_Loss = true
  Test_1 = true
  Test_2 = true

Observer(s):
  Degraded_Mode = true
  Degraded_block_1 = true
  Degraded_block_2 = true
  Op_Loss = false
  Test_1 = true
  Test_2 = true

Observer(s):
  Degraded_Mode = false
  Degraded_block_1 = false
  Degraded_block_2 = false
  Op_Loss = true
  Test_1 = false
  Test_2 = true

Observer(s):
  Degraded_Mode = true
  Degraded_block_1 = false
  Degraded_block_2 = true
  Op_Loss = false
  Test_1 = false
  Test_2 = true

Observer(s):
  Degraded_Mode = false
  Degraded_block_1 = false
  Degraded_block_2 = false
  Op_Loss = true
  Test_1 = false
  Test_2 = true

Observer(s):
  Degraded_Mode = true
  Degraded_block_1 = false
  Degraded_block_2 = true
  Op_Loss = false
  Test_1 = false
  Test_2 = true
```

Fig. 14. Résultats de la simulation pas-à-pas

4) Génération automatique de l'AMDEC

A partir des données récoltées de la simulation « pas-à-pas » du modèle système, nous traçons la liaison entre les modes de défaillance bas-niveau, et les effets intermédiaires et finaux qu'ils produisent.

De plus, nous retenons pour chaque mode de défaillance la liste des tests, qu'il déclenche, qui composent sa « signature de test » correspondante. A partir de cette information, nous pouvons retirer les tests servant à la détection et à l'isolation de chaque mode de défaillance. Ces éléments sont utiles, notamment pour les analyses de type « AMDEC orientée testabilité ». La référence [1] élabore davantage sur cet aspect.

Ainsi, la simulation « pas-à-pas » du modèle permet de remplacer la partie la plus coûteuse de l'AMDEC, qui est l'étude de la propagation des modes de défaillance au niveau global.

Il est essentiel, afin d'obtenir une AMDEC complète, de coupler les informations obtenues aux caractéristiques statiques des modes de défaillance qui sont : leurs causes, effets locaux et taux de défaillance.

Tout comme la phase de simulation, cette phase est également assurée par une chaîne outillée automatisée ([3]) chez Thales, afin d'avoir une génération d'AMDEC automatique.

La Table II montre un exemple de tableau AMDEC généré par la méthode proposée, avec les colonnes bleues représentant les informations obtenues directement de la simulation du modèle système.

C. *System-Analyst : une implémentation de la méthode*

Par ailleurs, la méthode proposée dans cette communication est implémentée dans l'outil logiciel System-Analyst, un outil d'analyses de sûreté de fonctionnement basé sur le MBSA. C'est un logiciel libre de diffusion et d'utilisation, dédié à l'analyse des risques, qui permet de réaliser des modèles de systèmes complexes par l'intermédiaire d'une interface complètement graphique. Il se couple aux outils OpenAltaRica pour modéliser un système, réaliser des AMDEC, des analyses de disponibilité, de testabilité, des arbres de défaillance, etc.

Pour réaliser la méthode présentée, System-Analyst part de modèles graphiques du système créés par l'utilisateur, et traduits en arrière-plan en des modèles AltaRica 3.0. A la demande, l'outil génère une AMDEC du système avec une transparence du déroulement des phases du processus.

TABLE II. TABLEAU AMDEC RÉSULTANT

Item Id.	**Failure Mode**	**Failure Rate**	**Causes**	**Effects**			**Detection Means**	**Severity Class.**	**Actions**	**Remarks**
				Local Effect	***Next Higher Level***	***End Effect***				
Comp 1	Loss	$\lambda_{1.1}$	…	Comp 1 Loss	None	None	None	…	…	…
	Degraded mode	$\lambda_{1.2}$	…	Comp 1 Degradation	None	None	None	…	…	…
Comp 2	Loss	$\lambda_{2.1}$	…	Comp 2 Loss	None	None	None	…	…	…
	Degraded mode	$\lambda_{2.2}$	…	Comp2 Degradation	None	None	None	…	…	…
Comp 3	Loss	$\lambda_{3.1}$	…	Comp 3 Loss	Loss of Block 1	Operational Loss	Test 1 Test 2	…	…	…
	Degraded mode	$\lambda_{3.2}$	…	Comp 3 Degradation	Degraded Block 1	Degraded System	Test 1 Test 2	…	…	…
Comp 4	Loss	$\lambda_{4.1}$	…	Comp 4 Loss	Loss of Block 2	Operational Loss	Test 2	…	…	…
	Degraded mode	$\lambda_{4.2}$	…	Comp 4 Degradation	Degraded Block 2	Degraded System	Test 2	…	…	…
Comp 5	Loss	$\lambda_{5.1}$	…	Comp 5 Loss	Loss of Block 2	Operational Loss	Test 2	…	…	…
	Degraded mode	$\lambda_{5.2}$	…	Comp 5 Degradation	Degraded Block 2	Degraded System	Test 2	…	…	…

V. APPLICATION SUR UN CAS INDUSTRIEL

A. *Présentation du cas industriel*

La méthode a été appliquée sur un système de l'industrie aéronautique, représenté par un modèle organique dont les dimensions sont données en Table III.

TABLE III. DIMENSIONS DES ELEMENTS DU CAS INDUSTRIEL

	Eléments	Nombre d'éléments
Système	***Niveaux hiérarchiques***	7
	Composants bas niveau	*756*
	Modes de défaillance bas niveau	*1152*
	Tests intégrés	*22*
Effets considérés	***Effets finaux***	19
	Effets intermédiaires	45

B. *Résultats*

La modélisation du système a été réalisée dans un temps estimé à 6 semaines de travail. Le traitement automatique du modèle par la suite a généré une AMDEC complète de ses modes de défaillance dans un temps de 10 minutes.

La réutilisation de ce modèle pour d'autres études de sûreté de fonctionnement peut économiser 80% de leur temps de réalisation.

VI. BÉNÉFICES ET LIMITES

A. *Bénéfices*

Le déploiement du concept du MBSA, et de la plateforme OpenAltaRica dans la réalisation d'AMDEC, tel que défini dans la communication, apporte plusieurs gains par rapport à la méthode AMDEC traditionnelle.

D'abord, en se conformant aux formalismes et outils utilisés par les ingénieries autres que la sûreté de fonctionnement, la méthode proposée facilite le partage d'information et la validation des données par l'ensemble des ingénieries, et sa maintenance ultérieure. De plus, ce rapprochement avec les formalismes des autres ingénieries crée la possibilité d'attribuer à l' « ingénierie matérielle » la tâche de livraison d'un modèle préliminaire du système.

Deuxièmement, cette méthode permet de capitaliser et de gérer en configuration les analyses de bas niveaux qui sont souvent communes à d'autres systèmes, produits ou projets industriels.

Troisièmement, le traitement automatique assure une propagation des effets locaux dans le modèle jusqu'au niveau système et permet ainsi d'analyser exhaustivement tous les cas de défaillances et de réduire la possibilité d'erreur humaine.

Quatrièmement, l'utilisation d'un modèle hiérarchique qui représente les différents niveaux du système, permet de garder la traçabilité du chemin de propagation des modes de défaillance bas niveau jusqu'au niveau système. Absents des synthèses d'AMDEC traditionnelles, ces détails sont importants pour appuyer les conclusions de l'AMDEC.

Cinquièmement, cette méthode réduit sensiblement la durée et donc le coût de cette analyse, comme la simulation des modes de défaillance par le simulateur « pas-à-pas » prend significativement moins de temps que l'analyse à la main.

Finalement, le modèle du système utilisé dans la méthode proposée, est réalisé sous AltaRica 3.0, qui représente un formalisme bien documenté et par nature réutilisable dans différents contextes de la sûreté de fonctionnement. Ceci facilite la lisibilité et la maintenance des modèles d'une part, et permet d'autre part d'obtenir un modèle de système réutilisable dans d'autres études. Cette interopérabilité visée entre les études de la sûreté de fonctionnement permet d'avoir des modèles de systèmes de plus en plus matures, grâce au partage et à la validation des hypothèses entre les études.

B. *Limites*

Comme explicité précédemment, la méthode présentée dans cette communication apporte des bénéfices importants par rapport à la méthode traditionnelle de génération des AMDEC, hérités principalement de la puissance des technologies OpenAltaRica. Cependant, la méthode présentée reste limitée par les limites de la définition d'une AMDEC. La considération de la dimension temporelle, malgré la capacité d'AltaRica 3.0 à l'implémenter, ne peut être intégrée dans la méthode présentée puisqu'elle dépasse la définition d'une AMDEC.

La taille des modèles à simuler dans OpenAltaRica peut également former une limite de cette méthode. En revanche, compte tenu des expérimentations de la méthode réalisées jusqu'à présent, cette taille limite indéterminée semble être suffisante pour les besoins industriels actuels..

La rentabilité économique de la méthode présentée par rapport à la méthode traditionnelle d'AMDEC se base principalement sur la facilité de réutilisation des modèles du système et des sous-composants qu'elle offre. Ainsi, dans le cas où le besoin de réutilisation de ces éléments est absent, la rentabilité de cette méthode devient relativement réduite.

VII. CONCLUSION

Cette communication présente un processus de réalisation d'une AMDEC basée sur les outils OpenAltaRica.

Ce processus s'articule autour de quatre étapes principales. D'abord, une analyse préliminaire des risques est menée pour soulever les événements redoutés opérationnels et de sécurité du système. Ensuite, la phase de modélisation vise à créer un modèle du système regroupant les éléments nécessaires pour le traiter. Le modèle est ensuite traité par le simulateur « pas-à-pas » d'OpenAltaRica pour observer les effets des modes de défaillance du système. Finalement, un post-traitement automatisé des résultats de simulation du système permet de générer une synthèse de l'étude.

Parmi les principaux bénéfices de ce processus, on retrouve l'élimination de l'erreur humaine par l'implémentation d'un traitement totalement automatisé des modèles système.

Également et en complément des résultats de l'AMDEC, ce processus produit un modèle système AltaRica 3.0 réutilisable dans les autres contextes de sûreté de fonctionnement pris en compte par OpenAltaRica.

Comme d'autres processus innovants déployant OpenAltaRica, ce processus vise à assurer un haut niveau d'interopérabilité entre les études de sûreté de fonctionnement, par l'utilisation d'un modèle système unique.

REFERENCES

[1] E. Clement, D. Riera and M. Batteux. "Analyse des performances de diagnostic d'un système au moyen d'une modélisation AltaRica 3.0". Actes du congrès Lambda-Mu 21. Reims, France. October 2018.

[2] https://www.system-analyst.fr/

[3] S. Breton, P. Le-Com, T. Thomas and E. Clement. "System-Analyst – Un outil MBSA pour l'analyse des risques, libre de diffusion et compatible avec Arbre-Analyste et OpenAltaRica". Actes du congrès Lambda-Mu 21. Reims, France. October 2018.

[4] https://www.openaltarica.fr/

[5] M. Batteux, T. Prosvirnova, and A. Rauzy. "AltaRica 3.0 in 10 Modeling Patterns". In *International Journal of Critical Computer-Based Systems*. Inderscience Publishers. Vol. 9, Num. 1–2, pp 133–165, 2019.

[6] M. Batteux, T. Prosvirnova, and A. Rauzy. "From Models of Structures to Structures of Models". In *IEEE International Symposium on Systems Engineering (ISSE 2018)*. Roma, Italy. October, 2018.

[7] M. Batteux, T. Prosvirnova, and A. Rauzy. "AltaRica Wizard : un environnement intégré de modélisation et simulation pour AltaRica 3.0". Actes du congrès Lambda-Mu 21 (actes électroniques). Reims, France. October, 2018.

[8] M. Batteux, T. Prosvirnova, and A. Rauzy. "Altarica 3.0 assertions: the why and the wherefore". Journal of Risk and Reliability. Professional Engineering Publishing. September, 2017.

[9] C. G. Cassandras, and S. Lafortune. "Introduction to Discrete Event Systems". New-York, NY, USA: Springer, 2008.

[10] L. Valiant. "The Complexity of Enumeration and Reliability Problems". SIAM Journal of Computing, vol. 8, nbr. 3, pages 410-421. 1979.

[11] A. Zimmermann. "Stochastic Discrete Event Systems". Springer-Verlag Berlin Heidelberg, 2008.

[12] S. RACHID. "Elaboration d'une méthode d'analyse de la testabilité basée sur la modélisation dynamique des systèmes". Rapport de stage de fin d'études 2019, ISTIA, l'école d'ingénieurs de l'université d'Angers, France, unpublished.