Stratégies de modélisation AltaRica de la propagation de défaillances dans les systèmes dynamiques

Tatiana Prosvirnova, Christel Seguin, Christophe Frazza, Estelle Saez, Mathilde Machin, Xavier de Bossoreille, Jean Gauthier, Pierre Darfeuil, Frédéric Deschamp

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

Stratégies de modélisation AltaRica de la propagation de défaillances dans les systèmes dynamiques

Strategies for modelling failure propagation in dynamic systems with AltaRica

Tatiana Prosvirnova[1], Christel Seguin[1], Christophe Frazza[2], Estelle Saez[3], Mathilde Machin[4], Xavier de Bossoreille[4], Jean Gauthier[5], Pierre Darfeuil[6], Frédéric Deschamp[7]

[1] *ONERA/DTIS, Université de Toulouse, 31000 Toulouse, France ; tatiana.prosvirnova@onera.fr; christel.seguin@onera.fr*

[2] *SATODEV, 25 rue Marcel Issartier, 33700 Mérignac, France ; christophe.frazza@satodev.fr*

[3] *IRT Saint-Exupéry, B612 • 3 rue Tarfaya • CS 34436, 31405 Toulouse cedex 4, France ; estelle.saez@irt-saintexupery.com*

[4] *APSYS Airbus, 37 Avenue Escadrille Normandie Niemen, 31700 Blagnac, France ; xavier.debossoreille@apsys-airbus.com; mathilde.machin@apsys-airbus.com;*

[5] *Dassault Aviation, 54 AV Marcel Dassault, 33700 Mérignac, France; jean.gauthier@dassault-aviation.com*

[6] *Safran Helicopter Engines, Avenue Joseph Szydlowski, 64510 Bordes; pierre.darfeuil@safrangroup.com*

[7] *LGM, Euclide B4 - ZAC St Martin du Touch, 1 Rue Emmanuel Arin, 31300 Toulouse, France ; frederic.deschamps@lgm.fr*

Résumé — Le langage AltaRica a été conçu pour faciliter la modélisation de la propagation de défaillances et les analyses de sécurité dans les systèmes complexes. Ainsi, il permet plus particulièrement de modéliser la dynamique fonctionnelle (changement de mode de contrôle, reconfiguration de matériels…) et dysfonctionnelle (cascades de pannes, pannes cachées…) des systèmes.

L'objectif de cette communication est de donner des guides pour exploiter au mieux cette capacité de modélisation de la dynamique. Les guides proposés ont été développés par un panel d'experts safety classique et MBSA (Model Based Safety Assessment) dans le cadre du projet S2C (System & Safety Continuity) des IRTs Saint Exupéry et System X. Leur usage a pu aussi être illustré lors de la modélisation AltaRica du système de freinage avion qui a été réalisée pour l'annexe MBSA de l'ED-135 / ARP4761A.

Mots-clefs — Model Based Safety Assessment, MBSA, systèmes dynamiques, modèles de propagation de défaillances, AltaRica, système de freinage ARP4761.

I. Introduction

Le langage AltaRica [2] a été conçu pour faciliter la modélisation de la propagation de défaillances et les analyses de sécurité dans les systèmes complexes. Ainsi, il permet plus particulièrement de modéliser la dynamique fonctionnelle (changement de mode de contrôle, reconfiguration de matériels…) et dysfonctionnelle (cascades de pannes, pannes cachées…) des systèmes.

L'objectif de cette communication est de donner des guides pour exploiter au mieux cette capacité de modélisation de la dynamique. Les guides proposés ont été développés par un panel d'experts safety classique et MBSA (Model Based Safety Assessment) dans le cadre du projet S2C (System & Safety Continuity) des IRTs Saint Exupéry et System X. Leur usage a pu aussi être illustré lors de la modélisation AltaRica du système de freinage avion qui a été réalisée pour l'annexe MBSA de l'ED-135 / ARP4761A [1].

Le groupe de travail S2C a extrait du retour d'expérience MBSA des participants, des écueils couramment rencontrés et des stratégies de modélisation adoptées pour surmonter les difficultés. L'article se concentre sur un phénomène dynamique potentiellement problématique : le contrôle continu d'un processus physique avec des boucles d'asservissement.

Le guide de modélisation s'efforce de donner pour ce cas :

– un exemple d'un système très simple et d'un évènement redouté qui illustre la nécessité de la modélisation ;

– les erreurs de modélisation habituelles de ce type de système et les dysfonctionnements qui en résultent ;

– les bonnes pratiques de modélisation sous la forme de stratégies de modélisation AltaRica ;

– les résultats des analyses qualitatives (simulations, recherche des causes des évènements redoutés) qui donnent confiance dans le comportement, modélisé selon les stratégies ;

– les hypothèses générales dans lesquelles les stratégies de modélisation sont valables.

La modélisation des systèmes dynamiques en sûreté de fonctionnement a principalement été étudiée pour évaluer les

indicateurs probabilistes habituels de fiabilité ou de sécurité (voir par exemple [18]). Cet article caractérise les stratégies de modélisation des systèmes dynamiques qui permettent également le calcul des séquences d'événements conduisant aux évènements redoutés. Ce type de modélisation est en voie de normalisation dans l'aéronautique et l'article contribue également à clarifier les choix de modélisation faits pour traiter le système de freinage des avions dans le futur document ED-135 [1].

Le reste de cet article est organisé comme suit. La section 2 décrit l'étude de cas. La section 3 donne un aperçu des travaux connexes, du langage de modélisation AltaRica et de ses outils d'évaluation. La section 4 présente les problèmes soulevés par la modélisation de la propagation des défaillances des systèmes avec asservissement et discute des différentes stratégies pour les résoudre. La section 5 présente l'illustration de ces problématiques sur le système de freinage avion qui a été réalisé pour l'annexe MBSA de l'ED-135 [1]. La section 6 conclut cet article et donne quelques perspectives.

II. DESCRIPTION DE L'ETUDE DE CAS

Afin d'illustrer comment traiter la modélisation de la propagation des défaillances des systèmes avec des boucles d'asservissement, considérons un exemple simple illustré à la Fig.1.

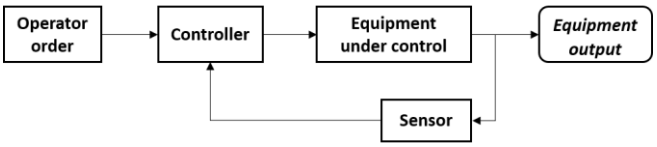

Figure 1: Étude de cas : un équipement asservi.

Dans cet exemple, nous considérons un système composé d'un équipement asservi et d'un contrôleur qui construit une commande à partir des informations fournies par un capteur et de l'ordre initial envoyé, par exemple, par un opérateur. Le capteur acquiert des données sur la sortie de l'équipement et les envoie au contrôleur, qui est utilisé pour commander l'équipement. Cet exemple est une boucle d'asservissement simplifiée, et nous pouvons facilement remplacer l'équipement par une vanne ou un actionneur. Nous considérons que tous les composants (ordre de l'opérateur, contrôleur, équipement asservi et capteur) ont deux modes de défaillance :

– fail_loss : entraîne la perte du composant ;
– fail_err : entraîne un comportement erroné du composant.

Du point de vue de la sécurité, les évènements redoutés évalués sont les suivants :

– FC1 : Perte de la sortie de l'équipement ;
– FC2 : Sortie de l'équipement erronée.

La sortie de l'équipement est surveillée par le capteur qui envoie les informations acquises au contrôleur. La sortie de l'équipement dépend des données d'entrée de l'équipement. Le contrôleur calcule un ordre réévalué à partir de ses deux entrées (l'ordre de l'opérateur et les informations acquises par le capteur) et commande l'équipement en fonction de cet ordre. Les défaillances des composants, les sorties et les effets correspondants du système sont décrits dans le Tableau 1.

Composant	**Mode de défaillance**	**Effets sur la sécurité**
Order (Operator order)	fail_loss fail_err	Cela conduit à la perte de contrôle et à la perte de la sortie de l'équipement. Entraîne une commande erronée de l'équipement et une sortie d'équipement erronée.
Equipment (under control)	fail_loss fail_err	Entraîne une perte de la sortie de l'équipement. Entraîne une sortie de l'équipement erronée. La donnée erronée est utilisée par le capteur.
Sensor	fail_loss fail_err	Entraîne la perte de l'acquisition du capteur envoyée au contrôleur, ce qui entraîne la perte de la sortie de l'équipement. Entraîne une information erronée de l'acquisition du capteur, conduisant à une sortie d'équipement erronée.

Tableau 2: Modes de défaillance des composants et leurs effets sur la sécurité.

L'équipement de sortie, sur la Fig.1, est un artefact de sécurité, un observateur des évènements redoutés. Pour calculer la commande et contrôler l'équipement, le contrôleur a besoin de la sortie de l'équipement asservi envoyée par le capteur. Avant même de commencer la modélisation, on peut identifier que le système modélisé est une boucle d'asservissement : l'entrée du contrôleur dépend de la sortie du capteur qui dépend elle-même de la sortie du contrôleur.

La modélisation des systèmes avec une boucle d'asservissement en utilisant l'approche classique de l'arbre de défaillance peut conduire à des équations circulaires. Si l'arbre de défaillance est structuré en suivant strictement les dépendances des différentes entrées et sorties du système, il y aura une logique circulaire dans l'arbre de défaillance produit.

En pratique, la plupart du temps, les équations circulaires dans les arbres de défaillance sont résolues par les analystes, qui font des hypothèses sur le comportement du système et adaptent une stratégie de modélisation pour éliminer les équations circulaires de l'arbre de défaillance. Néanmoins, lorsqu'une équation circulaire apparaît dans un arbre de défaillance, il est toujours utile d'analyser les impacts possibles de la simplification effectuée pour la résoudre.

La modélisation de systèmes avec une boucle d'asservissement à l'aide de langages de modélisation de haut niveau supportant l'approche MBSA (Model Based Safety Assessment) peut conduire à des problèmes similaires. Différentes stratégies pour les résoudre sont discutées dans les sections suivantes.

III. TRAVAUX CONNEXES

A. *Modèles de propagation des défaillances statiques et dynamiques*

Les formalismes de modélisation de la propagation des défaillances peuvent être divisés en deux catégories : les modèles combinatoires (par exemple, les arbres de défaillance ou les blocs diagrammes de fiabilité) et les modèles à événements discrets (par

exemple, les chaînes de Markov ou les réseaux de Petri stochastiques généralisés).

Les modèles combinatoires sont des modèles statiques, c'est-à-dire que tous les événements sont supposés être indépendants et peuvent se produire dans n'importe quel ordre. En d'autres termes, l'ordre d'apparition des événements n'a aucune influence sur l'apparition des évènements redoutés. Dans ce cas, les modèles sont évalués en résolvant des systèmes d'équations booléennes pour calculer des ensembles de coupes minimales (MCS) et des indicateurs probabilistes (par exemple, la probabilité d'occurrence de l'évènement redouté).

Des algorithmes de résolution efficaces ont été développés pour les modèles statiques [16], ce qui permet d'évaluer des modèles à l'échelle industrielle. Les modèles à événements discrets peuvent être statiques, et dans ce cas, l'ordre d'apparition des événements n'a aucune influence sur l'état résultant. Mais ce n'est pas toujours le cas. On dit qu'un modèle à événements discrets est dynamique s'il existe au moins un couple de séquences constitué des mêmes événements et aboutissant à des états différents.

Pour les modèles dynamiques, l'ordre d'apparition des événements est important. Les modèles statiques à événements discrets peuvent être évalués par la génération et la résolution d'équations booléennes. Pour les modèles dynamiques à événements discrets, la compilation en équations booléennes n'est pas toujours possible et peut perdre des informations. Dans ce cas, il est possible de générer des séquences d'événements menant aux évènements redoutés. La génération de séquences permet d'explorer partiellement les scénarios de défaillance du modèle. Notez que le temps de calcul augmente considérablement par rapport à la génération d'équations booléennes et à leur résolution.

Il existe différents langages de modélisation de haut niveau supportant l'approche MBSA. Parmi eux, nous pouvons citer AltaRica (AltaRica LaBRI [2], AltaRica DataFlow [7], AltaRica 3.0 [4]), Figaro [8], SAML [9], HiP-HOPS [12], Component Fault-Trees [11], les réseaux de Petri stochastiques généralisés à prédicats implémentés dans GRIF [17]. Cette liste n'est pas exhaustive.

Parmi les formalismes de modélisation cités, HiP-HOPS et Components Fault-Trees sont des formalismes combinatoires et permettent de créer des modèles statiques. Pour modéliser la propagation des défaillances des systèmes avec des boucles d'asservissement en utilisant ces formalismes, l'analyste doit faire des hypothèses sur le comportement du système afin de créer un modèle statique. AltaRica, Figaro, SAML et les réseaux de Petri sont basés sur des modèles à événements discrets et permettent de décrire des modèles statiques et dynamiques. Le contrôle continu d'un processus physique avec une boucle de rétroaction est un phénomène dynamique. Différentes stratégies peuvent être adoptées pour la modélisation de la propagation des défaillances des systèmes avec des boucles d'asservissement. Certaines d'entre elles sont présentées et discutées en utilisant AltaRica DataFlow dans la suite de cet article.

B. Langage de modélisation AltaRica

AltaRica est un langage de modélisation textuel de haut niveau, formel et spécifique à un domaine, dédié à l'analyse de la sécurité, créé à la fin des années 90 [2]. AltaRica est un langage centré sur les événements. Le comportement des composants est décrit au moyen de machines à états. L'état d'un composant est représenté par des variables (appelées variables d'état) et leurs valeurs. Les changements d'état sont possibles quand, et seulement quand, un événement se produit. L'occurrence d'un événement met à jour les valeurs des variables, par le déclenchement d'une transition : un triplet < garde, événement, post-condition >, où une garde est une expression booléenne construite sur les variables, où l'évènement peut représenter un mode de défaillance ou une action fonctionnelle stochastique ou déterministe, et une post-condition est une instruction qui modifie la valeur des variables d'état. AltaRica distingue deux types de variables : les variables d'état et les variables de flux. Les variables d'état ne peuvent être modifiées que par le déclenchement de transitions. Les variables de flux sont utilisées pour modéliser les informations circulant entre les nœuds d'un modèle. Leur valeur est calculée à partir de la valeur des variables d'état grâce à un mécanisme décrit au moyen de ce qu'on appelle l'assertion.

Lorsqu'une transition est déclenchée, sa post-condition est d'abord exécutée pour calculer la valeur des variables d'état, puis l'assertion est exécutée pour calculer la valeur des variables de flux.

Le comportement des composants est décrit à l'intérieur des nœuds. Les nœuds peuvent être assemblés en hiérarchie, leurs flux d'entrée et de sortie peuvent être connectés et leurs transitions peuvent être synchronisées. Les nœuds peuvent être stockés dans les bibliothèques de composants réutilisables et sont réutilisés par instanciation, comme dans les langages de programmation structurés. AltaRica est un langage asynchrone : une seule transition peut être déclenchée à la fois. Cependant, il offre un mécanisme pour synchroniser les événements. Par exemple, les défaillances de cause commune, les équipes de réparation partagées, les diffusions peuvent être représentées au moyen de synchronisations.

Il existe trois versions du langage de modélisation AltaRica :

– AltaRica LaBRI, la première version du langage développé par le LaBRI [2] ;

– AltaRica DataFlow, la deuxième version du langage implémentée dans plusieurs outils industriels [7] ;

– AltaRica 3.0 mis en œuvre dans la plate-forme OpenAltaRica par l'association AltaRica et l'IRT SystemX [4].

Modèles acausal et causal

La principale différence entre les trois versions du langage est la sémantique de l'assertion (calcul des valeurs des variables de flux).

Dans la première version du langage, AltaRica LaBRI, l'assertion est un ensemble de contraintes. Il n'y a pas de variables d'entrée ou de sortie et de cette façon, il est possible de représenter des modèles acausaux. Chaque fois qu'une transition est déclenchée, l'action de la transition est d'abord exécutée, puis un ensemble de contraintes (l'assertion) est résolu pour calculer les valeurs des variables de flux. Les contraintes ont un grand pouvoir expressif. Cependant, en général, la résolution des contraintes implique de multiples itérations de calcul et peut être très consommatrice de ressources. Un ensemble de contraintes peut avoir plusieurs solutions acceptables, ce qui donne un modèle non déterministe. En outre, il se peut qu'il n'y ait pas de solution. Le cas échéant, le modèle initial est incorrect.

Afin de pouvoir évaluer des modèles à l'échelle industrielle, une deuxième version, AltaRica DataFlow, a été créée, réduisant le pouvoir expressif de l'assertion [7]. Sa sémantique est basée sur les Automates de Mode [14]. Dans cette version du langage, l'assertion est un ensemble d'assignations DataFlow. Chaque variable de flux est assignée une seule fois dans le modèle et il n'y a pas de définitions circulaires. Ainsi, il est seulement possible de représenter des modèles causaux. L'ordre d'exécution des assignations DataFlow est calculé une seule fois lors de la compilation du modèle. Lorsqu'une transition est déclenchée, la valeur des variables d'état est d'abord calculée, puis la valeur des variables de flux est calculée en exécutant les affectations DataFlow une seule fois, ce qui est plus efficace que la résolution des contraintes.

La sémantique de la troisième version du langage, AltaRica 3.0, est définie en termes de systèmes de transitions gardées [3]. Pour pouvoir modéliser facilement certains types de systèmes bouclés, par exemple des réseaux ou des systèmes électriques, AltaRica 3.0 introduit le concept d'affectation bidirectionnelle. Si x et y sont des variables, x:=:y est une affectation bidirectionnelle, qui est équivalente à deux affectations : x:=y, y:=x. L'assertion est une instruction, où chaque variable peut être définie dans plusieurs assignations et il peut y avoir des définitions circulaires. Après chaque déclenchement de transition, l'action de la transition est d'abord exécutée, puis la valeur des variables de flux est calculée par

la résolution en points fixes de l'assertion. En général, la résolution en points fixes de l'assertion consomme plus de ressources que le calcul des affectations DataFlow, mais moins que la résolution des contraintes.

Il convient de noter que les boucles rencontrées dans les réseaux de communication ou les systèmes électriques sont différentes des boucles de commande à rétroaction présentées dans la section 2 et que la résolution des points fixes ne résoudra pas les problèmes de modélisation. Les techniques de modélisation présentées dans cet article doivent être utilisées dans ce cas.

AltaRica DataFlow

Dans la suite de cet article, nous nous concentrons sur AltaRica DataFlow. Il est utilisé comme langage de description de plusieurs outils de modélisation industriels : Cecilia Workshop (Dassault Aviation, Satodev), Simfia V3 et SimfiaNeo (Apsys).

De nombreuses expériences à l'échelle industrielle ont été menées avec cette version du langage [5, 6, 13]. La plus significative est l'utilisation d'un modèle MBSA pour l'évaluation de la sécurité du système de commandes de vol dans le cadre de la certification du Falcon 7X de Dassault Aviation.

Un ensemble d'outils d'évaluation efficaces a été développé pour cette version du langage, notamment un compilateur d'arbres de défaillance [14], un générateur de séquences d'événements critiques et un simulateur pas à pas. La définition de sémantiques temporelles et stochastiques d'AltaRica DataFlow a permis de développer un compilateur de chaînes de Markov [15], un modelchecker probabiliste [19] et un simulateur Monte Carlo [10].

IV. Modelisation de l'etude de cas et analyse avec AltaRica DataFlow

A. *Questions soulevées par la modélisation de la propagation des défaillances des systèmes avec des boucles d'asservissement*

La Figure 2 montre une représentation graphique du modèle AltaRica DataFlow de l'étude de cas. Le comportement de chaque composant est représenté par un nœud AltaRica. Le nœud **Sensor** représente le comportement du capteur, le nœud **Order** - le comportement de la commande de l'opérateur, le nœud **Equipment** - le comportement de l'équipement étudié, les nœuds **AllControlInputs** et **Control** représentent le comportement du contrôleur. Le nœud **EquipmentOutput** est un artefact de sécurité : c'est un observateur de l'état de la sortie de l'équipement qui modélise les évènements redoutés.

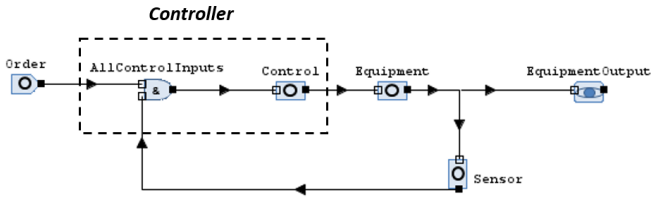

Figure 2 : Représentation graphique du modèle AltaRica DataFlow de l'étude de cas.

L'état interne de chaque nœud est représenté par trois valeurs : {*ok*, *lost*, *err*}, où *ok* représente le comportement nominal, *lost* représente la perte du composant et *err* représente le comportement erroné du composant. Les nœuds **Order**, **Control**, **Sensor** et **Equipment** ont deux événements *fail_err* et *fail_loss*, qui représentent les modes de défaillance de ces composants. Le noeud **AllControlInputs** est un noeud logique, il n'a pas d'état interne.

Si au moins l'une des entrées est *err*, alors la sortie est *err*. Sinon, si au moins l'une des entrées est *lost*, alors la sortie est *lost*. Sinon, la sortie est *ok*.

Le détail des nœuds AltaRica est donné dans le Tableau 2.

Composant	Flux d'entrée / sortie	Variables d'état	Transitions	Assertions
Order	Input: N/A Output: O Type: *{ok, lost, err}*	*S ∈{ok, err, lost}* Initially *ok.*	S=*ok* |- *fail_loss* -> S:= *lost*; S=*ok* |- *fail_err* -> S:= *err*;	O=S;
AllControl-Inputs	Inputs: I1, I2 Output: O	N/A	N/A	O = case*{* I1=*err* or I2=*err* : *err*, I1=*lost* or I2=*lost* : *lost*, else *ok }*;
Control Sensor Equipment	Input: I Output: O Type: *{ok, lost, err}*	*S ∈{ok, err, lost}* Initially *ok.*	S=*ok* |- *fail_loss* -> S:= *lost*; S=*ok* |- *fail_err* -> S:= *err*;	O =case *{* S=*ok*: I, S=*lost*: lost, else *err }*;
Equipment-Output	Input: I Output: O Type: *{ok, lost, err}*	N/A	N/A	O=I;

Tableau 2 : Cadre de modélisation en AltaRica DataFlow.

Le modèle AltaRica DataFlow donné en Fig. 2 n'est pas correct car l'assertion de ce modèle n'est pas DataFlow. On dit qu'il y a un cycle d'équations dans l'assertion s'il existe une variable de flux qui dépend d'elle-même dans l'assertion. En d'autres termes, il y a une définition circulaire dans l'assertion. En pratique, les cycles d'équations peuvent être détectés lors d'une compilation grâce au graphe de dépendance de l'assertion. Si le graphe de dépendance de l'assertion a des cycles, alors il y a un cycle dans les équations de l'assertion.

Ainsi, dans notre exemple, il existe un cycle dans les équations de l'assertion, la sortie du nœud **Équipement** dépend de lui-même. Pendant la compilation du modèle, une erreur est détectée (voir par exemple la Figure 3), laquelle doit être corrigée. Il existe différentes stratégies pour résoudre ce problème. Elles sont présentées ci-après.

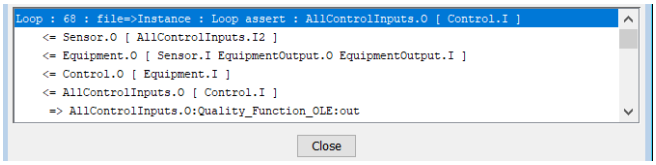

Figure 3 : Exemple d'une erreur : cycle dans l'assertion.

B. *Solution « casser la boucle »*

La solution de simplification ou "casser la boucle" modifie le modèle afin de résoudre le cycle d'équation en "cassant" la boucle d'asservissement du système, et en s'assurant que l'analyse du modèle de sécurité est toujours représentative du système étudié. La plupart du temps, il est nécessaire d'ajouter des hypothèses et des explications afin d'atteindre cet objectif. Par exemple, au lieu d'analyser la boucle d'asservissement illustrée à la Fig.2, nous pouvons choisir d'effectuer l'analyse sur un modèle avec une boucle d'asservissement simplifiée comme illustré à la Fig.4.

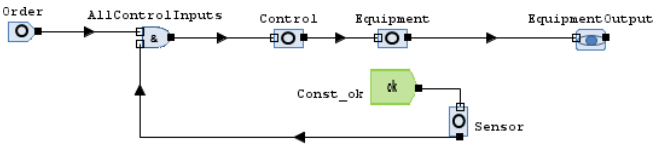

Figure 4 : Illustration de la solution « casser la boucle ».

Dans notre exemple, la boucle d'asservissement existante est "cassée" à l'aide d'un artefact de sécurité représenté en vert sur la Fig.4. Le but du nœud **Const_ok** est d'utiliser le même nœud **Sensor** que celui défini précédemment. C'est une consigne numérique qui envoie une entrée *ok* au **Sensor**. Ceci est équivalent à l'utilisation d'un nœud **Sensor** sans entrée. Le point clé de cette solution est de s'assurer que l'analyse effectuée avec le modèle simplifié est aussi représentative que celle effectuée avec le modèle complet.

Validation et analyse du modèle Cette solution est valable si le modèle simplifié est représentatif du système étudié malgré la simplification. Notez que pour atteindre cet objectif, dans certains cas, il peut être nécessaire de fournir une analyse supplémentaire à la sortie du modèle. Dans l'exemple, nous considérons que l'analyste de sécurité qui décide de casser la boucle va vérifier que les coupes obtenues par "casser la boucle" sont représentatives de la boucle d'asservissement. C'est le cas, comme le montre le Tableau 3.

Coupes pour la FC1 : Perte de la sortie de **Equipment**	Coupes pour la FC2 : Sortie de **Equipment** erronée
Control.fail_loss Order.fail_loss Euipment.fail_loss Sensor.fail_loss	Control.fail_err Order.fail_err Equipment.fail_err Sensor.fail_err

Tableau 3 : Coupes pour la solution « casser la boucle ».

En particulier, nous vérifions que les défaillances du **Sensor** (*fail_loss* et *fail_err*) conduisent aux évènements redoutés, comme prévu (la perte du **Sensor** conduit à la perte de la sortie de **Equipment** et un **Sensor** erroné conduit à une sortie de **Equipment** erronée). Par ailleurs, les effets des défaillances des autres composants sont inchangés. En effet, les défaillances **Order**, de **Control** ou de **Equipment** (*fail_loss* et *fail_err*) conduisent directement aux défaillances correspondantes de la sortie de **Equipment**. Dans cet exemple, l'état du **Sensor** (*ok* ou *failed*) a un effet direct. Dans le cas d'un **Sensor** erroné, l'évènement redouté FC2 : Sortie d'**Equipment** erronée" est directement atteinte.

Contre-exemple : dans le cas de l'ajout d'une consolidation entre les deux entrées de **AllControlInputs** (une entrée *ok* et l'autre *err* entraîne la perte de la sortie), la solution "casser la boucle", proposée dans cet exemple, n'est pas valable. Elle conduit à la perte de l'information capturée par le capteur. Dans ce cas, un **Order**, un **Equipment** ou un **Control** erroné conduit alors à la perte de la sortie de **Equipment** (FC1) et non à une sortie erronée de **Equipment** (FC2). Il est encore possible de "casser la boucle" en reliant directement les entrées **AllControlInputs** à la sortie **EquipmentOutput** et en cassant la boucle juste avant le nœud **Control**. Néanmoins le modèle résultant est très éloigné du système initial.

Avantages et inconvénients de l'approche L'intérêt de cette approche est de résoudre le cycle d'équation dans l'assertion en utilisant une modélisation statique. Par rapport à la modélisation dynamique, la modélisation statique permet de réduire le temps de calcul des coupes. De plus, il est possible de générer des équations booléennes à partir de modèles statiques sans perte d'information. Dans ce cas, ce choix de modélisation peut permettre de résoudre de grands modèles à l'échelle industrielle. Enfin, pour les modèles statiques, le calcul des probabilités est simple.

Néanmoins, la simplification du modèle conduit à un modèle plus proche de ce que le spécialiste de la sécurité a à l'esprit (et qu'il pourrait écrire en utilisant l'approche par arbre de défaillance) que de la description initiale du système. Les modélisateurs peuvent choisir de faire en sorte que le modèle ressemble au système, par exemple en ajoutant des artefacts graphiques ou des liens "vides". Dans ce cas, ils introduisent une cohérence artificielle entre les modèles de sécurité et du système, ce qui peut entraîner des malentendus et des erreurs futures. Seule la sortie située juste avant la "cassure de boucle" est affectée par tous les modes de défaillance. Par conséquent, cette approche n'est valable (en termes de jeu de coupes résultant) que lorsque la sortie de la boucle d'asservissement (ici **EquipmentOutput)** n'entraîne pas une reconfiguration de la boucle. Lorsque la boucle d'asservissement a plusieurs sorties (vers les FC ou vers les autres parties du modèle), certaines informations peuvent manquer. Dans notre exemple, si la sortie de la boucle d'asservissement était une entrée pour une surveillance **Monitoring** positionnée après le **Sensor**, ce nouveau nœud ne verrait pas l'impact des défaillances de l'**Equipment** comme illustré à la Fig. 5.

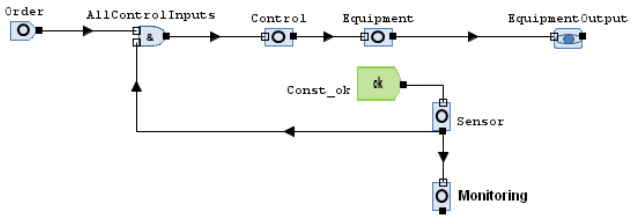

Figure 5 : Contre-exemple de la solution « casser la boucle ».

Cette approche est efficace lorsque la boucle peut être "cassée" avant les nœuds (ici le **Sensor**) qui n'affectent le système que lorsqu'ils sont défaillants (ici l'état *ok* ou *failed*). Son utilisation est limitée lorsque les composants sont impliqués dans la description fonctionnelle du système (par exemple dans la surveillance).

C. La solution du « Dirac »

La solution du « Dirac » introduit un artefact de sécurité pour gérer le cycle d'équations. Elle permet de modéliser toutes les dépendances entre les flux de sortie et d'entrée en introduisant une variable d'état. Afin de résoudre le cycle d'équation dans l'exemple illustré à la Fig. 6, nous introduisons cet artefact de sécurité par le biais de l'unité de modélisation appelée **FeedbackDelay**.

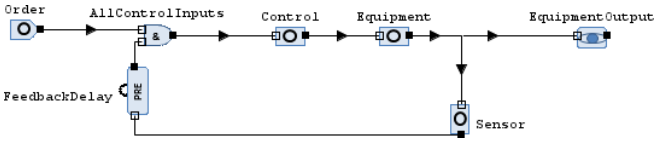

Figure 6 : Illustration de la solution « Dirac ».

L'unité de modélisation **FeedbackDelay** contient :

– Une variable d'état *prev_val* (valeur précédente) qui est initialement à ok ;
– Deux variables de flux : *I* et *O* ;
– Une assertion : *O = prev_val* ;
– Un événement déterministe immédiat appelé *update*, associé à la distribution de probabilité Dirac(0) ;
– Une transition *not(I = prev_val) |- update -> prev val:=I ;* qui permet de supprimer la dépendance de flux direct entre la valeur de sortie (*O*) du noeud et sa valeur d'entrée (*I*) et d'introduire une dépendance entre l'entrée *I* et la variable d'état *prev_val*.

La transition définie peut être lue comme suit : « lorsque la condition (la valeur d'entrée *I* est différente de la valeur d'état *prev_val*), l'événement déterministe *update* est instantanément déclenché (car il suit une loi Dirac(0)). En conséquence, *prev_val* est affecté à la valeur courante de *I*. En raison de l'assertion, la sortie *O* prend immédiatement la même valeur, ce qui entraîne la propagation du mode de défaillance.

L'introduction d'une variable d'état mise à *ok* initialise le problème à résoudre lorsqu'aucune défaillance n'est déclenchée. Cela résout le cycle d'équation pour l'état initial. A ce stade, il est intéressant de noter que la variable d'état introduit un « effet mémoire » sur la transition. En effet, la valeur d'état de *prev_val* ne

changera que lorsque les conditions de transition seront remplies. C'est pourquoi l'artefact de modélisation que nous avons présenté est souvent appelé délai. Il ne s'agit pas d'un temps quantitatif (qui serait par exemple mesuré en secondes) mais d'un temps séquentiel, c'est-à-dire l'ordre dans lequel les différentes mises à jour (*update*) se produisent.

Validation et analyse du modèle Les coupes minimales calculées pour le modèle donné Fig. 5 sont les mêmes que celles obtenues pour la solution précédente donnée dans le Tableau 3. La solution du « Dirac » ne nécessite pas de validation spécifique, à l'exception de la validation locale de la composante dédiée. Le simulateur pas à pas peut être utilisé pour valider le comportement du modèle. En outre, nous pouvons également souligner qu'une attention particulière doit être apportée lorsque plusieurs transitions instantanées Dirac(0) sont introduites dans le modèle et qu'elles peuvent être déclenchées en même temps.

Avantages et inconvénients de l'approche Le modèle proposé est très proche du système étudié. Il permet une représentation très proche de celle du système. Par conséquent, il sera plus facile de valider le modèle avec les ingénieurs système. Il sera également plus facile d'utiliser ce modèle pour communiquer avec les autres parties prenantes ou pour capturer le comportement du système.

L'introduction d'une transition déterministe peut conduire à un modèle dynamique. Dans ce cas, le solveur de l'outil générera toutes les séquences de défaillances possibles menant aux événements de plus haut niveau, alors que pour un modèle statique, il suffirait de générer toutes les combinaisons de défaillances (c'est-à-dire l'ensemble des coupes) ou, encore mieux, de résoudre directement une équation booléenne. Par conséquent, le temps de calcul devient plus important que pour un modèle statique. Au pire, pour les très grands systèmes (notamment ceux rencontrés à l'échelle industrielle), ce temps de calcul peut être un point bloquant.

De plus, lorsque plusieurs transitions déterministes instantanées de type Dirac(0) cohabitent, il faut gérer leur synchronisation ou la priorité de leur déclenchement. Cela ajoute de la complexité au modèle.

D. La solution « double-flux »

La solution du "double flux" repose sur l'ajout de flux artificiels pour traiter les dépendances dans le modèle. Comme le montre la Figure 7, les dépendances entre les variables sont modélisées par deux chemins différents.

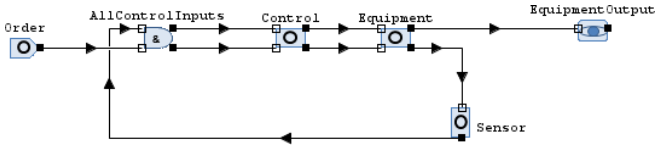

Figure 7 : Illustration de la solution « double flux ».

Tout d'abord, les modes de défaillance de tous les composants sont « collectés » par les flux allant de **Order** au **Sensor** (chemin inférieur). Dans cette voie inférieure, la sortie de **AllControInputs** ne dépend pas de la sortie du **Sensor**. Ensuite, **AllControInputs** possède une deuxième sortie. Chaque sortie de **AllControInputs** est liée à une entrée (voir Tableau 4).

Validation et analyse du modèle La validation de cette approche est la même que celle discutée pour l'approche "casser la boucle". Elle est nécessaire pour démontrer que le modèle est représentatif du système modélisé. Des analyses supplémentaires peuvent être nécessaires pour justifier ce choix. Les coupes sont conformes à celles attendues (les mêmes que celles données dans le Tableau 3) et valident les sorties du modèle.

Avantages et inconvénients de l'approche L'intérêt de cette approche est de résoudre le cycle d'équation en utilisant une modélisation statique. Ce choix permet ainsi de réduire le temps de calcul du modèle. De plus, le calcul des probabilités est simple. Cette approche est utilisable dans le cas de boucles d'asservissement avec plusieurs sorties (lorsque plusieurs composants en aval dépendent de la sortie de la boucle d'asservissement). Cette approche est celle qui nécessite le plus d'artefacts de sécurité, ce qui rend le modèle et les justifications plus lourds. En conséquence, elle est surtout utilisée pour les boucles locales, avec peu de composants impliqués.

Composant	**Flux d'entrée / sortie**	**Variables d'état**	**Transitions**	**Assertions**
All-Control-Inputs	Input: *I1, I2* Output: *O1, O2*	N/A	N/A	*O1 = I1 ; O2 = I2 ;*
Control	Input: *I1, I2* Output: *O1, O2* Type: *{ok, lost, err}*	*S ∈{ok, err, lost}* Initially *ok.*	*S=ok |- fail_loss -> S:= lost; S=ok |- fail_err -> S:= err;*	*O1 =case { S=ok: I1, S=lost: lost, else err }; O2 =case { S=ok: I2, S=lost: lost, else err };*
Equipment	Input: *I1, I2* Output: *O1, O2* Type: *{ok, lost, err}*	*S ∈{ok, err, lost}* Initially *ok.*	*S=ok |- fail_loss -> S:= lost; S=ok |- fail_err -> S:= err;*	*O1 =case { S=ok: I1, S=lost: lost, else err}; O2 =case { S=ok: I2, S=lost: lost, else err };*

Tableau 4 : Cadre de modélisation AltaRica DataFlow pour la solution « double flux ».

V. APPLICATION DES STRATEGIES DE MODELISATION AU SYSTEME DE FREINAGE D'UN AVION

A. Description du système

Le système modélisé est le système de freinage des roues d'un avion (WBS), et la fonction associée est « Freiner les roues pendant la phase d'atterrissage ou la phase de décollage (train d'atterrissage sorti) », qui est une sous-fonction de « Freiner l'avion au sol ». A noter que cette fonction de freinage sur les roues est complémentaire des modes de freinage aérodynamiques (ailerons, spoilers, aérofreins…) et de l'inversion de poussée des moteurs.

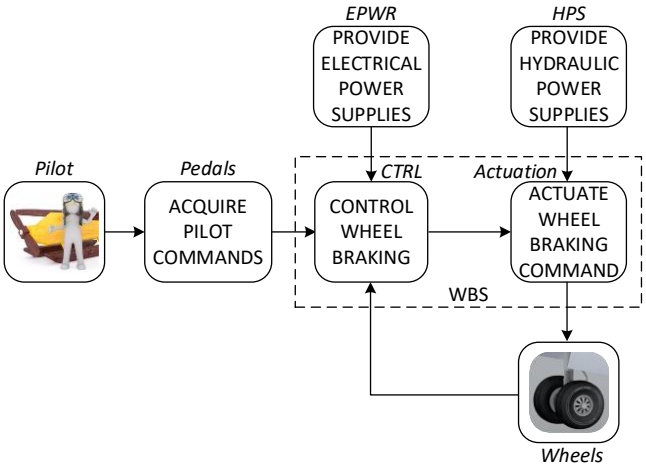

Figure 8 : Représentation de la fonction « Freinage des roues ».

La fonction de freinage des roues peut être comprise comme le montre la Fig. 8, depuis l'ordre du pilote jusqu'à l'action sur les roues, en tenant compte des systèmes de ressource électriques et hydrauliques.

A ce stade, une boucle d'asservissement est déjà envisagée : les capteurs permettant de mesurer la vitesse de rotation de chacune des roues envoient leur mesure aux contrôleurs, non seulement pour évaluer l'efficacité du freinage mais aussi pour prévenir du blocage des roues (détection du glissement, fonction anti-dérapage).

B. Description du modèle AltaRica Dataflow

Un modèle détaillé a été réalisé en AltaRica DataFlow, en l'occurrence avec l'outil Cecilia Workshop de Dassault-Aviation. Il est basé sur une architecture, une logique de fonctionnement et des modes de reconfiguration décrits dans l'annexe Q9 de l'ED-135 [1].

Pour des contraintes de taille, le modèle est présenté en deux parties illustrées dans les Figures 9 et 10.

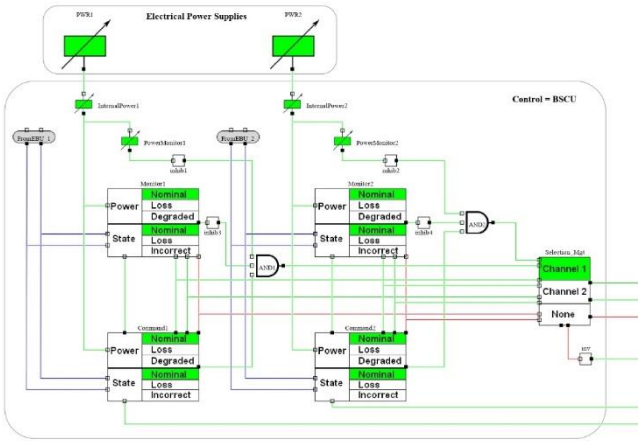

Figure 9 : Modèle de freinage, partie commande.

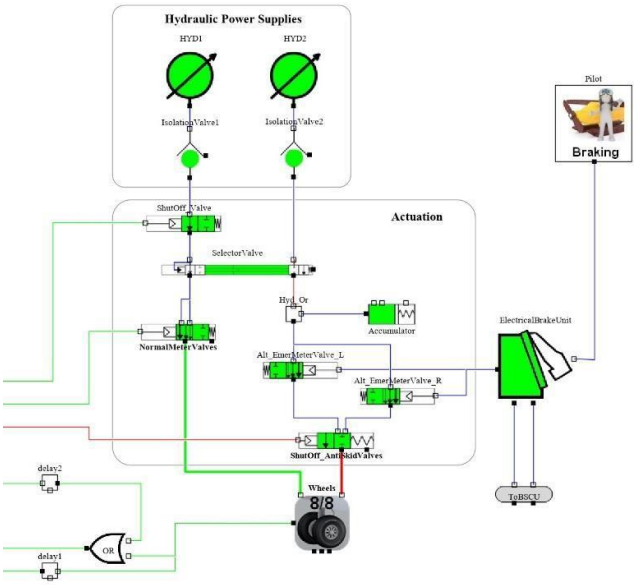

Figure 10 : : Modèle de freinage, partie actionneurs.

Il faut noter que les différents éléments visualisés ne sont pas obligatoirement des éléments élémentaires : ils peuvent contenir plusieurs éléments, selon le principe des poupées gigognes.

L'élément **Wheels** regroupe ainsi, par exemple, les huit roues et les huit capteurs associés, chacun ayant ses propres modes de défaillance.

C. Description de la stratégie de gestion des boucles d'asservissement adoptée

La solution adoptée est celle du « Dirac ». Ainsi, deux artéfacts de modélisation **delay1** et **delay2** comportent des évènements déterministes instantanés de type Dirac(0), tel qu'illustré dans la Figure 11.

L'artéfact **delay1** gère directement la boucle d'asservissement. A noter que la logique de gestion des capteurs de roues a été déportée dans l'élément **Wheels**, alors qu'elle est fonctionnellement réalisée par la partie commande. En effet, cela permet, au niveau du modèle, de gérer une boucle d'asservissement au lieu d'en gérer huit (une par roue), rendant le modèle beaucoup plus complexe et introduisant des problématiques à la synchronisation évoquée des Dirac(0). Bien évidemment, le comportement global du modèle a été testé et validé au regard du comportement attendu.

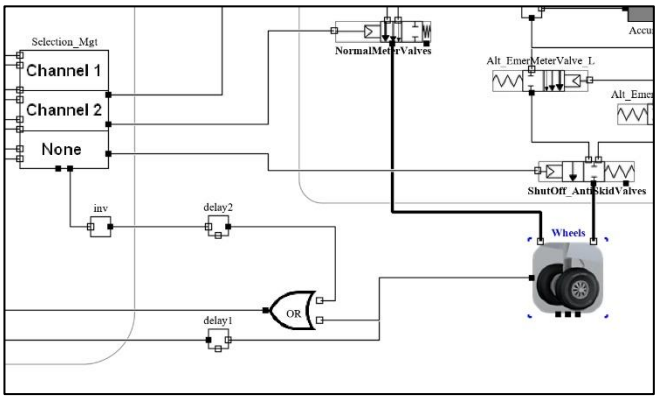

Figure 11 : Artéfacts de modélisation delay1 et delay2 de type « Dirac ».

L'artéfact **delay2**, quant à lui, trouve son fondement dans la séquence attendue de reconfiguration. En effet, deux évènements concurrents étaient déclenchables en même temps, mais l'ordre choisi par le logiciel de simulation ne correspondait pas aux spécifications du système. Cet artéfact permet de donner la priorité à l'évènement voulu.

Les artéfacts **delay1** et **delay2**, malgré de fait qu'ils comportent des Dirac(0), ne sont pas pour autant concurrents, i.e. ne sont pas déclenchables en même temps. Il n'a donc pas été utile de gérer leur priorité ou leur synchronisation.

L'intégralité du modèle est disponible sous différents formats (exports Cecilia, Docbook, Word et AltaRica) à l'adresse suivante : https://satodev.com/nos-produits/cecilia-workshop/ .

D. Analyse globale des choix effectués

Finalement, le système de freinage des roues d'un avion est un système asservi. Un premier choix de modélisation aurait pu être de « casser les boucles » au niveau des capteurs de roues, et ce sans perte de résultat au regard des analyses de sécurité, comme démontré précédemment.

Mais la volonté de représenter les modes de reconfiguration dans le modèle MBSA et donc de pouvoir simuler l'aspect fonctionnel, avec tous les avantages que cela comporte en termes de communication, de compréhension du système et de validation du modèle, a abouti à des problématiques de gestion de boucles d'asservissement.

Malgré la complexité apparente du système et par conséquence potentiellement celle du modèle MBSA, des choix de modélisation judicieux permettent de rendre ce dernier plus simple, donc plus facilement compréhensible et plus facilement maintenable. Certes cela engendre quelques pertes de représentativité localement mais permet néanmoins de garantir une bonne représentativité globale du modèle ainsi que le comportement dysfonctionnel.

VI. Conclusion et perspectives

Dans cet article, nous avons présenté différentes stratégies qui peuvent être utilisées pour représenter les boucles d'asservissement avec le langage de modélisation AltaRica DataFlow. Toutes les solutions proposées ont leurs avantages et leurs inconvénients.

Nos travaux futurs se concentreront sur l'identification d'autres types de problèmes de modélisation soulevés dans le domaine de la modélisation de la propagation dynamique des défaillances et sur la définition de stratégies de modélisation pour ces problèmes.

Des tests de performance plus poussés et des tests de robustesse pourront être également réalisés. Ces travaux ont permis également de définir des axes d'amélioration des performances du générateur de séquences.

References

[1] Eurocae ed-135 guidelines and methods for conducting the safety assessment pro-cess on civil airborne systems and equipment

[2] André Arnold, Alain Griffault, Gérald Point, and Antoine Rauzy. "The AltaRica Formalism for Describing Concurrent Systems", in Fundamenta Informaticae. IOS Press. Vol. 34, pp 109–124, 2000.

[3] M. Batteux, T. Prosvirnova, and A. Rauzy AltaRica 3.0. "Assertions: the Why and the Wherefore". *Journal of Risk and Reliability*. Professional Engineering Publishing. Vol. 231, Num. 6, pp 691–-700, September, 2017. doi: 10.1177/1748006X17728209

[4] M. Batteux, T. Prosvirnova, and A. Rauzy Altarica 3.0 in 10 modeling patterns. International Journal of Critical Computer-Based Systems 9(1–2), 133–165 (2018). https://doi.org/10.1504/IJCCBS.2019.098809

[5] Bernard, R., Aubert, J.J., Bieber, P., Merlini, C., Metge, S.: Experiments in model- based safety analysis: flight controls. In: Faure, J.M. (ed.) Proceedings of IFAC workshop on Dependable Control of Discrete Systems. pp. 43–48. Curran Associates, Inc., Cachan, France (June 2007), iSBN 9781617389948

[6] Bieber, P., Blanquart, J.P., Durrieu, G., Lesens, D., Lucotte, J., Tardy, F., Turin, M., Seguin, C., Conquet, E.: Integration of formal fault analysis in assert: Case studies and lessons learnt. In: Proceedings of 4th European Congress Embedded Real Time Software, ERTS 2008. SIA (electronic proceedings), Toulouse, France (January 2008), code R-2008-01-2B04

[7] Boiteau, M., Dutuit, Y., Rauzy, A., Signoret, J.P.: The altarica data-flow language in use: Assessment of production availability of a multistates system. Reliability Engineering and System Safety 91, 747–755 (2006)

[8] Bouissou, M., Bouhadana, H., Bannelier, M., Villatte, N.: Knowledge modelling and reliability processing: presentation of the figaro modelling language and asso- ciated tools. In: Proceedings of Safecomp'91 (1991)

[9] Güdemann, M., Ortmeier, F.: A framework for qualitative and quantitative model- based safety analysis. In: Proceedings of 12th High Assurance System Engineering Symposium. pp. 132–141 (2010)

[10] Khuu, M.: Contribution à l'accélération de la simulation stochastique sur des modèles AltaRica Data Flow. Thèse de doctorat, Université de la Méditerranée (Aix-Marseille II) (2008)

[11] Mohrle, F., Zeller, M., Hofig, K., Rothfelder, M., Liggesmeyer, P.: Automated compositional safety analysis using component fault trees. In: 2015 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW). pp. 152– 159. IEEE (2015)

[12] Papadopoulos, Y., Walker, M., Parker, D., Rude, E., Hamann, R., Uhlig, A., Gratz, U., Lien, R.: Engineering failure analysis and design optimization with hip-hops. Engineering Failure Analysis 18, 590–608 (2011)

[13] Quayzin, X., Arbaretier, E.: Performance modeling of a surveillance mission. In: Proceedings of the Annual Reliability and Maintainability Symposium, RAMS'2009. pp. 206–211. IEEE, Fort Worth, Texas, USA (January 2009), iSBN 978-1-4244-2508-2

[14] Rauzy, A.: Mode automata and their compilation into fault trees. Reliability En- gineering and System Safety 78, 1–12 (2002)

[15] Rauzy, A.: An experimental study on iterative methods to compute transient so- lutions of large markov models. Reliability Engineering & System Safety 86(1), 105–115 (2004)

[16] Rauzy, A.: Probabilistic Safety Analysis with XFTA. AltaRica Association, Les Essarts le Roi, France (2020)

[17] Signoret, J.P., Dutuit, Y., Cacheux, P.J., Folleau, C., Collas, S., Thomas, P.: Make your petri nets understandable: Reliability block diagrams driven petri nets. Reliability Engineering & System Safety 113, 61–75 (2013). https://doi.org/https://doi.org/10.1016/j.ress.2012.12.008

[18] Signoret, J.P., Leroy, A.: Reliability assessment of safety and production systems : analysis, modelling, calculations and case studies / Jean-Pierre Signoret and Alain Leroy. Springer Series in Reliability Engineering Ser, Springer, Cham, Switzerland (2021)

[19] Teichteil-Königbuch, F., Infantes, G., Seguin, C.: Epoch probabilistic model- checking. In: Model Based Safety Assessment Workshop. Toulouse, France (2011)