THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par **l'Institut Supérieur de l'Aéronautique et de l'Espace**
Spécialité : Systèmes informatiques et systèmes embarqués –
STIC réseaux, télécoms, systèmes et architectures

Présentée et soutenue par **Romain ADELINE**
le **14 mars 2011**

Méthodes pour la validation de modèles formels pour la sûreté de fonctionnement et extension aux problèmes multi-physiques

---

JURY

Mme Karama Kanoun, présidente
Mme Janette Cardoso, directrice de thèse
Mlle Sophie Humbert
M. Ioannis Parissis, rapporteur
M. Antoine Rauzy, rapporteur
Mme Christel Seguin, co-directrice de thèse

École doctorale : **Aéronautique - Astronautique**

Unité de recherche : **Équipe d'accueil ISAE-ONERA MOIS**

Directrice de thèse : **Mme Janette Cardoso**
Co-directrice de thèse : **Mme Christel Seguin**

THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

délivré par **l'Institut Supérieur de l'Aéronautique et de l'Espace**

Spécialité : Sûreté de Fonctionnement

---

par **Romain ADELINE**

14 Mars 2011

Méthodes pour la validation de modèles formels pour la Sûreté de Fonctionnement et extension aux problèmes multi-physiques

---

École doctorale	: **Aéronautique - Astronautique**
Unité de recherche	: **ONERA - DCSD**
Présidente du jury	: **Mme. Karama KANOUN**
Rapporteurs	: **Mr. Ioannis PARISSIS** : **Mr. Antoine RAUZY**
Directrice de thèse Co-directrice de thèse	: **Mme. Janette CARDOSO** : **Mme. Christel SEGUIN**
Encadrants	: **Mr. Pierre DARFEUIL** : **Mlle. Sophie HUMBERT**

Table des matières

Table des figures

Liste des tableaux

Liste des abréviations

AADL	Architecture Analysis & Design Language
AEV (NC)	Arrêt En Vol (Non Commandé)
AMDE	Analyse des Modes de Défaillance et de leurs Effets
ARP	Aerospace Recommanded Pratices
CCA	Common Causes Analysis
DAL	Development Assurance Level
E/S	Entrée / Sortie
EASA	European Aviation Safety Agency
FHA	Functional Hazard Analysis
FND	Forme Normale Disjonctive
FTA	Fault Tree Analysis
LaBRI	Laboratoire Bordelais de Recherche en Informatique
MBSA	Model Based Safety Assessment
MISSA	More Integrated System Safety Assessment
OCAS	Outil de Conception et d'Analyse Système
ONERA	Office National d'Études et de Recherches Aérospatiales
PSSA	Preliminary System Safety Analysis
SADT	Structured Analysis and Design Technic
SAE	Society of Automotive Engineers
SCADE	Safety Critical Application Development Environment
SCR	Software Cost Reduction
SdF	Sûreté de Fonctionnement
SMV	Symbolic Model Verifier
SSA	System Safety Analysis

Remerciements

Fin de l'étape.

... Voilà le 14 mars 2011 passé depuis quelques semaines, quelques mois... Le verdict est connu : je suis docteur... Ce n'est pas rien... Ce n'est pas tout non plus même si la proximité de la chose me rend encore un peu « ému » à l'évocation du résultat de trois années qui resteront probablement à jamais gravées dans ma mémoire... Pourquoi cette section ? Au-delà du simple respect de la tradition (ce qui déjà la justifierait de façon suffisante), le temps nécessaire est je crois faible pour se rendre compte que les personnes rencontrées au cours de cette étape majeure (elle l'est pour moi en tout cas) ne sont très certainement pas totalement étrangères à cela ...

Le constat précédent étant établi, l'exercice restant à faire, bien qu'extatique et d'importance, est difficile... A tel point que les mots satisfaisants se font rares et que le brouillon me servant de support commence sérieusement à se noircir de nombreuses ratures... Mais, au risque de me répéter, l'enjeu est d'importance : rendre hommage aux différents acteurs m'ayant épaulé, soutenu, inspiré ou secondé durant ces trois années de thèse. Et ils sont nombreux...

Plusieurs questions se posent alors dans ma tête... Comment n'oublier personne ? Dans quel ordre remercier ceux envers qui j'aurai éternellement une dette ? Comment faire passer, via l'écriture, des sentiments qui précipitent tout autour de moi ? ... Plusieurs stratégies me traversent l'esprit, aucune d'entre elles ne paraissant surclasser les autres...

Je commence donc par ne pas choisir l'originalité (vous me le pardonnerez) en remerciant celle qui a accepté de présider mon jury de thèse et qui restera à jamais la personne m'ayant annoncé le verdict : Karama Kanoun. Je remercie également Ioannis Parissis et Antoine Rauzy qui m'ont fait l'honneur d'accepter de rapporter le travail effectué durant ces trois ans. Merci !

Toujours en laissant de coté l'originalité, j'adresse mes remerciements à mes quatre encadrants Janette Cardoso, Christel Seguin, Sophie Humbert et Pierre Darfeuil (l'ordre n'a ici aucune espèce d'importance). Et avant même de commencer, je sais que les mots resteront insuffisants pour décrire l'ensemble des émotions que je souhaiterais faire passer. Tant de choses se sont passées au cours de ce projet commun. Des hauts... Des bas aussi soyons honnêtes mais gardez ces mots pour preuve de mon éternelle reconnaissance... Merci. Merci pour votre disponibilité.
Merci Janette d'avoir été présente, même à des milliers de kilomètres, même dans la dernière ligne droite où des questions, loin de l'aspect technique, se sont multipliées ... Christel... Merci pour tout ! Travailler avec toi a été une véritable chance, et un vrai bonheur. J'ai trouvé dans ta conception de la recherche ainsi que dans ta façon d'expliquer et de transmettre les idées de multiples sources d'inspiration et surtout de courage... Merci Sophie pour m'avoir, j'en suis sûr, compris et supporté lorsque, à de (trop ?) nombreuses reprises, je confondais besoins avec envie et autonomie avec indépendance. J'ai tant appris à vos cotés... Merci Pierre pour ta patience durant nos longues et animées discussions. Ta culture de la SdF et ton excellence (non non je n'en fais pas trop) sont et resteront pour moi un modèle à suivre et surtout à atteindre... Merci !
Je n'oublie pas ici Jean-Marc Bosc et Charles Castel avec qui j'ai eu la chance de commencer ce travail. Jean-Marc, merci de m'avoir fait confiance et d'avoir cru en moi ! Charles, je n'ai pas oublié nos discussions de début de thèse et qui trouvent dans le manuscrit beaucoup de répercussions directes ou indirectes. Merci à vous deux.

Mes pensées vont également à Rafael Gouriveau qui m'initia il y a de cela quelques années sur Besançon à la Sûreté de Fonctionnement. Rafael, je te dois beaucoup ! En plus de m'avoir présenter la discipline d'une façon m'ayant donné envie de poursuivre, c'est aussi toi qui un jour de 2007 au téléphone, m'a proposé un exercice auquel je n'avais absolument pas pensé ne serait ce qu'une seconde : faire une thèse ! Quelle idée ? Quelle idée ! Et si j'en ai c'est vrai choisi une différente de celle que tu proposais, je n'oublie pas que c'est aussi grâce à toi que je suis là !

Toujours sans stratégie particulière concernant l'ordre des remerciements, merci à tous les membres et doctorants de l'ONERA... En particulier aux « anciens », *i.e.* ceux m'ayant vu débuté l'aventure début 2008. Merci donc (je vais essayer de n'oublier personne) à Manu, aux Greg$_s$, à Ju, Nico, Stéphane, José, Sophie, Yoko, Jean-Charles. Merci aux « jeunes » que j'ai eu la chance de côtoyer lors de mes passages sur Toulouse. Merci également à l'ensemble des permanents du DCSD. Merci également à Pierre Bieber pour son soutien et ses conseils à de nombreuses reprises. Un merci particulier à mon « frère d'armes »Jean-Charles... J'admire ton courage à entreprendre un tel projet dans une situation professionnelle et personnelle établie. J'ai moi aussi énormément apprécié nos discussions dont les propos perdraient sans doute à être exposés ici...
Adresser un merci aux « autres » me permet d'être exhaustif... Je me note dans un coin de réutiliser l'astuce...

Sur Pau maintenant, merci à Turbinou ainsi qu'à tous les TM. En particulier au département DT/NAV, à ceux ayant gravités autour et à ceux qui y gravitent encore ! Merci à Guy Grienche et à Jean-Marc Bosc de m'avoir permis de faire parti du département et du service. Merci à Jean-Luc Thouvenot de m'avoir gardé :) Merci au service DT/NAV/SdF d'avoir contribué à ce que mes travaux se déroulent dans une ambiance parfois tout sauf studieuse :) Merci des nombreux « C'est bien de chercher... Mais à des moments, faut trouver aussi » entendus lors des au moins aussi nombreuses (et c'est pas peux dire !) pauses café... Merci donc à Sylvain, Aurélie, Christophe, Sophie, Céline, Jérôme, Linda, Greg, Lionel, Aymeric, Pierre et Sophie (ceux du début), Pascal, Gérald, José, Vincent, Antoine, Max, Manu, Bruno et aux autres (là encore je me dis que l'oublie est une possibilité.). Merci pour votre soutien.
Merci également au stagiaires rencontrés durant ces trois ans. Merci à Sandra, Jérôme, Julien, Clément, Martin, Cédric et Xavier pour leurs bonnes humeurs ! En particulier, un gigantesque merci aux deux derniers sans lesquels le travail présenté ici serait moins que ce qu'il est actuellement. Merci à eux pour le travail effectué, pour leurs idées, pour leurs propositions... Sur ce coup là, je vous en dois une les gars !
Merci également à Christophe Colette et Jean-Luc Fréalle pour avoir relus ce qu'il y avais à relire, souvent dans la précipitation. Merci pour votre disponibilité.
Enfin, merci et bon courage à « ma petite soeur d'armes » Aurélie... Courage ! C'est tellement bon à la fin que le jeu en vaut la chandelle ! Merci pour ton calme qui restera une source d'inspiration (et sans doute un peu de jalousie aussi) pour moi...

Toujours sur Pau, merci à mes deux préférées : Noelle et Mélanie... ou Mélanie et Noelle comme vous préférez ! Quelle chance de vous avoir connu les filles ! Noelle V. comédienne de talent et fille fantastique qui s'ignore beaucoup trop souvent ! Nos texas brownies après le ciné, nos rock sur du Ska-P et une infinité d'autres choses me manquent terriblement ... Tu me manques terriblement... Ah Ah !
Mélanie, ma partenaire préférée... La seule en fait... Qui peut prétendre rivaliser avec toi ? Hein ? Merci pour tout, merci de m'avoir écouté lorsqu'il m'est arrivé de ne pas aller et lorsque les doutes étaient plus présent que n'importe quelle autre chose... Merci pour toutes ces danses endiablées ! Tu me manques ! ! ! :)
Je vous aime les filles, vous le savez j'espère ! ! !

Pour poursuivre, comment ne pas avoir une énorme pensée pour tous les lapins ! Je vous aime mes petites boules de poil ! Dites présent ? Juju, Béran, Camille, Paka, Alice, Julie, Amélie, Vincent, Bruno, Mathieu, Guillaume Gui, Guillaume Laguite, Seve, Antoine, Mika, Kevin... Merci également à tous les autres ! Merci pour tous les excellents moments que j'ai passé en votre compagnie ... Vous êtes énormes ! Et même si je suis souvent très chiant, même si je ne bois pas

que du jus de carotte, même si j'ai une sainte horreur des jeux de société, je vous aime... Et ça, quand on y pense, c'est quand même pas mal ! :)

Merci aux autres ! Elodie... quelle chance j'ai eu de venir au Hoogaerden un mercredi soir et de rencontrer un groupe de filles complètement déjantées... Au milieu, toi :) Merci pour toutes ces soirées qui nous donnait l'occasion de se raconter nos vies imparfaites... Je t'adore ! Marina, nos concerts sur Toulouse me manquent déjà ! Merci à Bucheron, Xavier, Seb et Cht'i... Pour l'ensemble de notre œuvre... et parce que c'est vous les gars ! Merci à Fab, à sa famille et à tous les montbrunais pour m'avoir accueilli ponctuellement et fait monter sur les tables ! Merci également à Sylvain et Hélène pour leurs soutiens lors de mon passage à Rhodes... et pour la petite excursion du coté de Faliraki et aux thermes de Kalithea ! :)

Enfin, merci à ceux ayant toujours été là... qui m'ont construit... grâce auxquels je me suis construit... mes origines... mon socle... mon roman... ma famille. Toujours à me pousser vers la réalisation de mes rêves de jeunesse, merci Maman d'avoir toujours eu plus de confiance en moi que je n'en avais moi même. Merci pour ton courage. Merci d'avoir su transformer les chemins parfois encombrés que nous voulions prendre en routes toujours plus confortables. Merci d'avoir toujours été là... Merci pour tout !
Merci à celui m'ayant accompagné sur quelques uns de ces chemins. Merci à mon frère. Merci de m'avoir relevé lorsque plus rien ne semblait avoir d'importance. Même loin, merci d'être là.
Merci également à tous les autres. Grands parents, oncles et tantes, cousins, cousines... Merci d'être là ! Un grand merci particulier à Bénédicte et Mario. Merci pour tout ce que vous avez fait pour moi depuis quelques années.

Merci à vous tous qui m'avez soutenu et supporté... Quand on y pense, c'est très méritoire !

Pour finir, une pensée particulière pour quelqu'un que je n'oublie pas... Papa, j'aimerais savoir ce que tu penses de tout ça...

A mon père...

Introduction

L'objectif de cette thèse, initiée par Turbomeca (Groupe Safran) et réalisée en collaboration avec l'ONERA (Toulouse) et l'ISAE, a été de proposer des méthodes se voulant assister la réalisation et la validation de modèles formels supportant les études de sécurité de turbomoteurs d'hélicoptères.

Pourquoi ?

La question semble légitime... Afin d'appréhender dès le commencement et de manière pertinente les problématiques qui seront abordées dans ce mémoire, il est important (et sans doute même indispensable) de prendre un certain recul par rapport aux raisons ayant initiées les travaux présentés ici.

« Pour comprendre un système, il faut ... s'en extraire. »

Bernard Werber - L'empire des anges

Une notion à la mode ? ... Celle de « système »... Tout aujourd'hui contient ou utilise « des systèmes »... On parle en effet de systèmes informatiques, systèmes sociaux ou encore de systèmes industriels. Toujours plus important en taille, toujours plus complexe au niveau de leurs fonctionnements, les systèmes sont aujourd'hui confrontés à de nombreux défis : comment les développer ? Comment les réaliser ? Comment en prévenir les risques ? ...

Pour faire face à la complexité des systèmes à traiter et des problèmes à résoudre, l'intuition ne peut bien souvent suffire. Il est alors nécessaire de décrire cette complexité en construisant des représentations du comportement des systèmes considérés. Si dans un premier temps, ces représentations se limitaient à être des représentations mentales, une discipline (que l'on nomme « ingénierie dirigée par les modèles ») a pour objectif d'aider l'ingénieur à relever les défis énoncés en proposant la formalisation de ces représentations mentales. Le terme « modèle » sera dès lors souvent employé pour désigner ces représentations. L'ingénierie dirigée par les modèles se propose alors, pour répondre aux problématiques liées à la conception des systèmes, de placer les modèles au centre de la réflexion.

« The first step in solving any problem is to understand it. We often propose solutions to problems that we do not understand and then are surprised when the solutions fail to have the anticipated effect. »

Nancy Leveson

Le contexte très général des travaux décrits dans ce mémoire est celui de l'évaluation de la sécurité d'un système complexe critique aéronautique... plus particulièrement d'un moteur d'hélicoptère, ce qui explique le terme *aéronautique* de manière triviale et le terme *critique* puisque l'hélicoptère transportant des passagers, une défaillance de son moteur (où d'un de ses moteurs) peut avoir des conséquences dramatiques. Un turbomoteur peut aussi être considéré comme un

système complexe puisque constitué de différents sous-systèmes de natures physiques différentes (mécanique, hydromécanique, informatique...) en interaction les uns avec les autres.

Concernant le terme « évaluation de la sécurité » et puisque le moteur est un système critique, le motoriste se doit de garantir un niveau de sécurité acceptable de ses produits. Pour définir le terme « acceptable », des autorités dites de certification imposent au motoriste de répondre à un certain nombre d'exigences que doit satisfaire le moteur avant de pouvoir être monté sur la cellule de l'hélicoptère. Pour démontrer la satisfaction de ces exigences, des études de sécurité visent, entre autres, à analyser les chemins de propagation de défaillance à l'intérieur du système considéré. À l'heure actuelle, ces chemins de propagation sont représentés explicitement grâce à des analyses de type arbre de défaillance. Au cours des dernières années, l'augmentation du nombre d'exigences demandées par les autorités étant aller de paire avec l'augmentation de la complexité des turbomoteurs, ce type d'analyses est de plus en plus couteux, en temps et en ressources, à appliquer pour évaluer la sécurité de ces systèmes.

Pour aller au delà des difficultés rencontrées, il est possible d'utiliser des modèles formels compositionnels qui représenteront les chemins de propagation de manière implicite : le modèle peut être considéré comme une interconnexion de « boîtes » élémentaires décrivant comment une défaillance peut se propager dans un composant donné ; les chemins de propagation globaux sont obtenus par assemblage de ces « boîtes » élémentaires. Parmi les avantages immédiats de l'approche compositionnelle (modulaire), citons par exemple la possibilité de capitaliser les analyses déjà effectuées en permettant la réutilisation des « boîtes » d'ores et déjà disponibles. Citons également, toujours grâce à la réutilisation possible des « boîtes » déjà modélisées, un gain au niveau de la cohérence des analyses d'un turbomoteur à un autre (*e.g.* notation et termes utilisés dans les analyses). Parmi les langages implémentant une telle philosophie, nous avons utilisé le langage AltaRica développé à l'origine par le LaBRI en collaboration avec plusieurs industriels et académiques. Doté d'outils tels qu'un simulateur interactif (permettant de simuler un ou plusieurs scénarios sur le modèle et d'observer son comportement) ou un générateur automatique d'arbres de défaillance, le langage AltaRica a été conçu pour modéliser formellement le comportement fonctionnel et dysfonctionnel d'un système.

« Ce n'est point dans l'objet que réside le sens des choses, mais dans la démarche. »

Antoine de Saint-Exupéry - Citadelle

La complexité des systèmes considérés mêlée au saut dans l'inconnu que constituait l'utilisation d'une telle approche, des méthodologies de modélisation ont été nécessaires. En effet, au delà des objectifs à atteindre (qui sont les mêmes qu'auparavant, *i.e.* évaluer la sécurité du système considéré), il a fallu définir dans cette thèse l'ensemble des activités à mener pour réaliser ces objectifs. Concernant le langage AltaRica, diverses études ont déjà montré la pertinence de l'approche et se sont déjà intéressées à la mise en place de méthodologies de modélisation pour des systèmes électriques, hydrauliques ou informatiques. Les résultats obtenus ayant été jugés encourageant, les travaux décrits ici étendent dans un premier temps les travaux précédents à la mise en place de modèles multi-systèmes et multi-physiques. Pour cela l'approche restera classique et comportera deux principales étapes. La première phase est une phase d'abstraction du comportement où l'ingénieur en charge de la modélisation abstrait de la réalité les aspects pertinents du système étudié qui lui permettront de répondre à ses problématiques. Nous proposons lors de cette abstraction de mettre en place une *spécification* de chacune des « boîtes élémentaires » qui constitueront le modèle AltaRica. La seconde phase est une phase d'implémentation où il s'agira de formaliser en langage AltaRica le résultat de l'abstraction (décrit dans la spécification).

Une fois l'implémentation réalisée, il est alors possible de vérifier la cohérence et / ou la complétude du modèle formel obtenu. Pour ajuster, affiner ou éventuellement corriger le modèle, des activités de revue peuvent être effectuées par des ingénieurs ou des experts. Pour évaluer

l'efficacité de ces activités de revue, nous suggérons une mesure caractérisant de manière rigoureuse le degré de revue d'un modèle atteint lors de la simulation d'un jeu de scénarios. Pour cela et inspiré des travaux de la communauté « génie logiciel », des critères de couverture directement applicables à un modèle AltaRica ont été proposés. Le cycle « abstraction - implémentation - validation » est alors un cycle itératif où au fur et à mesure des erreurs rencontrées, l'ingénieur améliore à la fois l'abstraction du système réel (la spécification) et le modèle AltaRica. Le degré de revue des modèles et le taux de résultats positifs (résultats prévus par la spécification = résultat fourni par le modèle) donnera alors une indication sur la concordance entre le comportement du système réel et celui du modèle.

« Pour progresser, il ne suffit pas de vouloir agir, il faut d'abord savoir dans quel sens agir. »
Gustave Le Bon - Hier et demain

Ce mémoire présente l'ensemble des travaux réalisés pour supporter ce processus d'ingénierie des modèles de propagation de défaillances et pour le valider sur un cas d'étude. Il est composé de cinq chapitres : les deux premiers ont pour vocation d'introduire et / ou de rappeler différentes notions utiles à la compréhension de la suite ; les deux suivants détaillent les travaux effectués durant cette thèse ; le dernier se veut expliciter quelques enseignements pouvant être tirés de notre expérience.

Chapitre 1 : Concepts généraux autour de la SdF

Ce chapitre permet de présenter la notion de Sûreté de Fonctionnement et quelques concepts généraux lui tournant autour. Étant donné notre positionnement dans le domaine aéronautique, nous présentons le processus utilisé pour évaluer la sécurité d'un système aéronautique. Nous finissons le chapitre par la présentation des analyses classiques utilisées lors de ce processus.

Chapitre 2 : Modélisation formelle de systèmes

Ce chapitre débute par la présentation de la notion de modèle et se poursuit par la description du langage formel AltaRica, crée pour évaluer la Sûreté de Fonctionnement d'un système. Nous présentons et détaillons sa syntaxe ainsi que sa sémantique formelle fondée sur la notion d'automate de mode. Nous présentons les outils associés au langage AltaRica et décrivons dans la dernière section de ce chapitre d'autres formalismes supportant les analyses de Sûreté de Fonctionnement.

Chapitre 3 : Vers une méthodologie unifiée de modélisation AltaRica de systèmes physiques

Ce chapitre présente la première contribution de cette thèse : une méthodologie de modélisation unifiée permettant la représentation des propagations de défaillance au sein et entre des systèmes appartenant à des domaines physiques différents. Nous commençons ainsi par présenter une vision globale du processus ayant pour objectif l'abstraction du comportement d'un système. Puis, nous explicitons l'application de cette méthodologie à différents sous-systèmes physiques d'un turbomoteur (mécanique, hydromécanique, informatique...).

Enfin, nous proposons de formaliser les résultats issus de cette abstraction sous la forme d'une spécification de modèle AltaRica. Nous présenterons à cette occasion différents formalismes candidats à supporter l'écriture d'un tel document et en privilégierons un.

Chapitre 4 : Processus pour la validation de modèle AltaRica

Ce chapitre présente la deuxième contribution de cette thèse : un processus de validation de modèles AltaRica. Ce processus étant fondé sur la simulation de scénarios de test sur le modèle construit, ce chapitre est l'occasion de définir différentes notions et différents termes relatifs au test et à là couverture de test.

Pour s'intéresser au processus à proprement parler, celui-ci est en premier lieu diviser en trois étapes : la validation unitaire (*i.e.* la validation des « boîtes » élémentaires), la validation de l'intégration des « boîtes » les unes avec les autres et la validation du modèle global. L'accent est mis sur la validation unitaire. Des critères de couverture sont proposés afin d'identifier les portions du modèle testés par les scénarios (et les portions de l'étant pas) et de quantifier quelle proportion du modèle est effectivement couverte par les tests. Un début de réflexion est présenté concernant les phases de validation faisant suite à celle unitaire.

Chapitre 5 : Cas d'étude et retour sur l'approche

Ce chapitre présente quelques résultats numériques sur l'application des processus présentés au chapitre 3 et au chapitre 4. Quelques avantages et quelques difficultés liés à l'introduction de modèles AltaRica dans le processus Sûreté de Fonctionnement sont présentés et discutés.

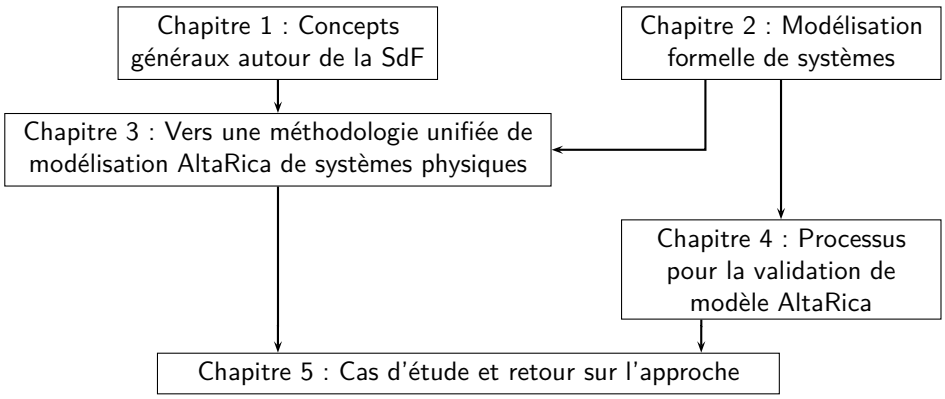


Schéma de lecture

Chapitre 1

Concepts généraux autour de la SdF

Sommaire

Cette partie a pour vocation d'introduire, de rappeler et/ou de présenter certains concepts d'ordre général autour de la Sûreté de Fonctionnement et de ces études.

Nous tenterons par exemple de présenter cette notion de Sûreté de Fonctionnement, ses origines, ses objectifs et ses méthodes. Nous présenterons la notion de défaillance que nous serons amenés à utiliser dans la suite de ce mémoire. Étant donné notre positionnement dans le domaine aéronautique, nous présenterons également le processus classique de développement et d'évaluation de la sécurité d'un système aéronautique. Enfin et avant de préparer la transition vers la section suivante, nous présenterons des méthodes couramment utilisées lors de l'évaluation de la Sûreté de Fonctionnement d'un système : les arbres de défaillance, les graphes de Markov et les Réseaux de Petri.

1.1 Vers la notion de Sûreté de Fonctionnement

1.1.1 Une philosophie : prévoir le pire...

Avant de parler de Sûreté de Fonctionnement, un état d'esprit relativement satisfaisant est celui dicté par les lois de Murphy (lois pouvant potentiellement être davantage connues sous le nom de « loi de l'empoisonnement maximum »). Le principe de telles lois est on ne peut plus simple : tout ce qui peut aller mal ira mal, souvent, au moment le plus inopportun et le plus préjudiciable... En particulier :

- si plus d'une chose peut aller mal, celle qui ira mal est celle qui peut avoir la conséquence la plus catastrophique ;
- un défaut caché se révèlera dans les pires circonstances possibles ;
- toute tâche qui peut être accomplie d'une manière incorrecte sera un jour accomplie de cette manière.

Prédire le pire est en général ce qu'il y a de mieux... à défaut de plus conservatif. La vision peut sembler assez fataliste mais elle a l'immense avantage d'induire un instinct de prudence vis-à-vis des dangers[1] auxquels toute entité est confrontée.

Pour appuyer cette vision des choses, Sun Tse exprimait dans son Art de la guerre que « celui qui, prudent, se prépare à affronter l'ennemi qui n'est pas encore ; celui là même sera victorieux. Tirer prétexte de sa rusticité et ne pas prévoir est le plus grand des crimes ; être prêt et en dehors de toute contingence est la plus grande des vertus ». Ce discours, à l'origine présenté pour des motivations belliqueuses, exprime de manière assez juste le bénéfice que l'on peut tirer à identifier les dangers avant que ceux-ci n'apparaissent.

Une attitude naturelle est alors de vouloir identifier ces dangers, d'en prévenir les causes et d'en mitiger les conséquences... Une attitude naturelle est alors de vouloir maîtriser les risques qui nous entourent.

1. Situation, condition ou pratique qui comporte en elle-même un potentiel à causer des dommages.

1.1.2 Notion de risque

Cette section vise à introduire la notion de risque et expliciter pourquoi maîtriser les risques est de nos jours une activité indispensable. Intuitivement tout d'abord, le risque peut être défini comme l'exposition du système considéré à un ou plusieurs dangers. Lorsque le système et le danger se rencontrent, c'est l'accident !

Selon l'ARP 4754 (Aerospace Recommanded Practice) [59] (section 1.4), le risque est « la fréquence (probabilité) d'une occurrence et le niveau de gravité associé ». Le risque peut donc être vu comme une mesure de l'effet d'un aléa sur une entité.

$$Risque = Probabilité \times Gravité$$

.

Cette notion de risque est donc un concept intrinsèque à tout système. En effet, tout système est susceptible d'être potentiellement soumis à un ou plusieurs aléas entraînant la perte des objectifs fixés préalablement. Les conséquences de l'occurrence d'un aléa peuvent donc être catastrophiques de plusieurs points de vue : matériel, économique, environnemental.... Maîtriser ces risques doit donc être une préoccupation majeure de tous ceux impliqués dans le cycle de développement d'un système.

Une autre justification du besoin de maîtriser les risques peut être trouvée dans la notion de confiance à accorder dans le produit utilisé. Cette notion de confiance paraît fondamentale aujourd'hui tant l'utilisation d'un système, d'un produit est dépendante de l'image qu'il renvoie. Au cours du temps, la recherche de la sécurité a évolué mais est restée une attitude naturelle de l'homme. [7] montre que si pendant longtemps, la notion de risque fut considérée comme une fatalité divine, on préfère aujourd'hui considérer un risque maîtrisé et ainsi revendiquer et profiter du droit à la sécurité. La notion de risque est donc fatalement liée à la notion de confiance, indispensable de nos jours pour convaincre de l'utilisation sûre d'un système.

Les concepts de risque et de maîtrise des risques sont donc sujets à plusieurs interprétations selon le domaine dans lequel on se place. On pourra citer notamment et de manière non exhaustive les risques financiers, les risques environnementaux ou les risques industriels. Dans chacun de ces domaines les enjeux et les problématiques liés à la maîtrise des risques différent profondément. Nous nous focaliserons ici sur les risques d'un système aéronautique de transport. Dans un tel système et selon le document CS-25 1309 [28], les critères d'évaluation du risque d'une occurrence sont :

- le degré de réduction des marges de sécurité de l'aéronef ;
- le degré d'augmentation de la charge de travail de l'équipage (en particulier du ou des pilotes) suite à l'occurrence ;
- la possibilité de subir des dommages matériels plus ou moins importants ;
- la possibilité d'avoir des blessures plus ou moins graves pouvant aller jusqu'au décès.

Nous utiliserons ces différents critères d'évaluation de risque lors de l'identification des exigences de sécurité dans la section 1.4.2.

1.1.3 Sûreté de Fonctionnement

... Prédire le pire est en général ce qu'il y a de mieux... L'objectif ultime de la Sûreté de Fonctionnement pourrait donc être de tout prévoir, tout imaginer des dysfonctionnements potentiels d'une entité. Ambitieux...

Pour préciser le terme de Sûreté de Fonctionnement donnons différentes définitions complémentaires.

- Une première, issue de la norme NF X60-319 [6], définit la Sûreté de Fonctionnement comme « l'ensemble des aptitudes d'un bien à accomplir une fonction requise (ou sa mission) dans des conditions données, pendant une durée donnée sans dommage pour lui même et son environnement ».
- Une seconde, plus large et issue de [65], définit la Sûreté de Fonctionnement comme « la science des défaillances ; elle inclut leurs connaissances, leurs évaluations, leurs prévisions, leurs mesures et leurs maîtrises ».
- Une dernière, extraite de [42], exprime l'idée selon laquelle « la Sûreté de Fonctionnement n'est pas un but en soi, mais un moyen : des démarches, des méthodes, des outils et un vocabulaire. La Sûreté de Fonctionnement n'est que du bon sens organisé et systématique. Maîtriser les risques est une attitude naturelle que chacun pratique ; mettre en œuvre la Sûreté de Fonctionnement, c'est professionnaliser cette attitude, l'optimiser, l'expliciter ».

La Sûreté de Fonctionnement s'applique alors dans tout le cycle de vie d'une entité, de la phase prévisionnelle (expression des besoins, conception, réalisation) à la phase opérationnelle (exploitation, maintenance, fin de vie). Tout au long de ce cycle de vie, les objectifs poursuivis sont multiples : évaluer et améliorer la fiabilité de l'entité, améliorer le processus de fabrication, améliorer la maintenance, améliorer la sécurité, répondre à des exigences réglementaires... Pour répondre à ces objectifs, plusieurs grandeurs se veulent évaluer la Sûreté de Fonctionnement d'une entité : la Fiabilité, la Maintenabilité, la Disponibilité, la Sécurité. Ces grandeurs constituent alors les grandeurs FMDS (RAMS en anglais pour Reliability, Availability, Maintenability, Safety). En ce qui nous concerne, la plus intéressante sera la sécurité (au sens de l'innocuité) du système et de son environnement vis-à-vis des défaillances potentielles d'un système.

Enfin, la troisième définition nous permet de faire le lien avec le domaine de l'aéronautique. En effet, lorsque l'on parle de « démarche » ou de « méthode », il est aisé de faire le lien avec les différents standards aéronautiques [59, 60] se voulant guider les phases de conception et d'évaluation de la sécurité d'un système aéronautique. Suivre ces guides permet de s'assurer que l'on utilise des méthodes reconnues par les autorités de certification (il est cependant admis que l'on puisse utiliser d'autres approches que celles préconisées par ces standards).

1.2 Quelques définitions

Il sera présenté dans cette section diverses définitions permettant de mieux comprendre et de mieux appréhender les futures lignes de ce mémoire. On commence ainsi par présenter les notions de système, de système complexe et d'hétérogénéité de système. À cette occasion, nous focalisons l'objet d'une sous-section sur quelques caractéristiques majeures traitant du comportement d'un tel système. Nous donnons également et à la suite de cela quelques définitions de termes utilisés dans le domaine de la Sûreté de Fonctionnement de manière courante.

1.2.1 Système et système complexe

1.2.1.1 Système

Étymologiquement, un système est un assemblage d'éléments divers. À partir de là et de la généralité de cette définition, il nous sera difficile d'en donner une définition simple et rapide. Cette dernière position est appuyée par [39] selon lequel le terme « système » appartient à la gamme des mots polysémiques et peut donc voir sa définition varier selon le domaine considéré. Puisqu'il demeure toutefois utile de fournir une définition du terme, nous en donnerons deux. Ainsi, [25] décrit un système comme « un ensemble d'éléments en interaction dynamique, organisé en fonction d'un but ». [65] définit un système comme « un ensemble déterminé d'éléments discrets interconnectés ou en interaction ». La première définition met alors l'accent sur la finalité ou le

but poursuivi par le système. La seconde permet d'identifier d'une part, le fait que le périmètre du système doit être identifiable, d'autre part, que les composants du système n'ont pas besoin d'être connectés pour être en interaction (et le système n'est alors pas la simple somme de ses sous-systèmes et/ou de ses composants).

1.2.1.2 Notion de complexité de système

Ludwig von Bertalanffy souligne de manière plutôt habile dans [10] que « la définition des systèmes comme ensemble d'éléments en interaction est si générale et si vague qu'on ne peut en tirer grand-chose ». Étant donné la généralité de ce concept, nous convenons aisément que lui adjoindre l'adjectif « complexe » n'aidera nullement le lecteur à préciser ce concept. D'autre part, la littérature abonde de concepts différents et plus ou moins bien définis.

Étymologiquement, le terme complexe peut se décomposer en « con plexus » signifiant « avec entrelacement ». Nous donnerons ici une définition issue de [35] selon laquelle les systèmes complexes « impliquent de nombreux composants qui interagissent dynamiquement à plusieurs niveaux ou échelles, lesquels exhibent des comportements communs pour différents types de systèmes, à différentes échelles et dans différentes disciplines ». Une autre définition, différente mais complémentaire, est donnée par l'ARP 4754 [59] et décrit la complexité comme « un attribut d'une fonction, d'un système ou d'un composant donnant à leurs opérations, à leurs modes de défaillance où à l'effet de leurs défaillances un caractère difficile à appréhender sans l'aide de méthodes analytiques ».

Cependant et sans cette complexité, les méthodes analytiques classiques telles que la simulation stochastique resteraient sans doute suffisantes pour appréhender un système. Cependant, l'appréhension d'un système complexe pose de nombreuses difficultés de compréhension (flou, incertain, imprévisible, ambigu, aléatoire, non déterministe...). Ainsi, et en accord avec [25], la complexité justifie le besoin que nous avons de posséder une représentation du système. Paradoxalement, cette complexité pouvant rendre difficile l'obtention d'une telle représentation, il nous est nécessaire de posséder des méthodologies supportant cette activité de modélisation.

Remarque : Notons ici que le terme « complexe » ne fait pas référence à la taille du système considéré. Pour définir une taille importante, il est utilisé dans la littérature ([40]) le terme de « complication ».
Pour présenter la différence de concept, reprenons un exemple issu de la même référence explicitant la différence entre un mur de briques et un puzzle. Il y est présenté qu'il n'est pas formellement plus difficile de monter un mur de 10 000 briques qu'un mur de 100 briques ; seul le temps de réalisation de la tâche augmente. On parlera de complication. Par contre, réaliser un mur de brique est formellement plus facile que la réalisation d'un puzzle car on doit tenir compte de la variété des interactions entre les éléments du système. On parlera de complexité.

1.2.1.3 Hétérogénéité des systèmes

Au delà de cette notion de complexité de système qui rend compte de l'interaction entre les différents constituants d'un système, une notion intéressante et importante est celle d'hétérogénéité des systèmes.

En effet, il est souvent mis en œuvre au sein d'un système industriel (*e.g.* un système aéronautique de transport) différents métiers devant coopérer pour accomplir la fonction d'un système.

Exemple : Sans rentrer dans les détails, on imagine aisément qu'un avion ou un hélicoptère est constitué de systèmes mécaniques, hydrauliques, électriques, de communication... Ces différents sous-systèmes mettent alors en jeu des technologies et des métiers différents et doivent être interfacés et intégrés au sein de l'aéronef.

Au sens large et selon [40], l'hétérogénéité d'un système concerne des choses aussi diverses que les substances des composants (matériels, logiciels...), les métiers mis en jeu, la sémantique des flux informationnels entre les constituants du système, la granularité de ces flux... Maitriser et comprendre cette hétérogénéité est alors un pré-requis à la représentation d'un système (nous traiterons cette notion de représentation d'un système au chapitre 2).

1.2.1.4 Quelques caractéristiques majeures d'un système

Nous présentons ici les caractéristiques majeures des systèmes qui devront être prises en compte dans cette représentation. Ces caractéristiques sont très fortement inspirées de celles présentées dans [65] :

- la structure du système :
  - les sous-systèmes et/ou les composants du système, leurs rôles et leurs fonctions,
  - les connexions et les interactions entre ces composants / sous-systèmes,
  - leurs localisations (fonctionnelles et/ou géographiques) dans le système ;
- les fonctions du système, des sous-systèmes ou des composants (fonctions principales et secondaires) ;
- les conditions de fonctionnement du système, de ses sous-systèmes et de ses composants :
  - les états de fonctionnement du système et de chacun de ses composants,
  - les conditions de dysfonctionnement du système et de chacun de ses composants,
  - les changements de configuration du système et de chacun de ses composants ;
- l'environnement du système :
  - les autres systèmes en interface avec le système d'étude,
  - les opérateurs humains potentiellement en interaction avec le système,
  - l'environnement du système (poussière, humidité, ondes électromagnétique...) ;
- les conditions d'exploitation du système :
  - les dispositifs de surveillance (alarme, barrière de sécurité...),
  - les possibles interventions sur le système (maintenance, action opérateur suite à une alarme...).

1.2.2 Vocabulaire [59]

Définition 1.1 : Défaillance

Une *défaillance* est « l'incapacité d'un élément à assurer sa fonction prévue ». ■

En tant que tel, le terme « défaillance » ne doit être appliqué, si l'on souhaite rester rigoureux, qu'à la notion de fonction. En effet, en considérant la définition ci-dessus, une défaillance est

entendue comme un état de l'entité et rien ne nous permet de faire la distinction entre une défaillance et la présence simultanée de plusieurs défaillances. Lorsque le système n'est plus à même de réaliser une ou plusieurs de ses fonctions, nous le qualifierons de « défaillant ».

Définition 1.2 : Mode de défaillance

Un *mode de défaillance* est « la manière dont la défaillance d'un élément se produit ». En d'autres termes, le mode de défaillance désigne l'évolution conduisant à un système défaillant. Plusieurs modes de défaillance différents peuvent ainsi conduire à une même défaillance. ■

Définition 1.3 : Condition de défaillance

Une *condition de défaillance* est une situation caractérisée par son impact, sur l'aéronef et ses occupants, à la fois direct et indirect, engendrée par ou à laquelle contribuent une ou plusieurs défaillances, en prenant en considération les conditions défavorables de fonctionnement et d'environnement applicables. ■

1.3 Mesure de la Sûreté de Fonctionnement

À travers son association aux mesures de Fiabilité, Maintenabilité, Disponibilité et Sécurité (FMDS), la Sûreté de Fonctionnement fait référence à (sans toutefois se réduire à) la notion de « performance » d'une entité. Cette « performance » devant être mesurée et évaluée, des grandeurs et indicateurs sont disponibles. Nous en donnons ici une liste non exhaustive.

1.3.1 Les grandeurs FMDS

1.3.1.1 Fiabilité

La norme NF X 60-500 [44] définit la fiabilité comme « l'aptitude d'une entité (*e.g.* un système ou un composant) à accomplir les fonctions requises dans des conditions données pendant une durée donnée ». La fiabilité de cette entité E est alors donnée par la probabilité F(t) que l'entité accomplisse toujours ses fonctions à l'instant t si elle les accomplissait à l'instant 0.

La fiabilité se rapporte en fait à la continuité du service qu'une entité doit accomplir. À ce propos, la fiabilité peut alors se définir comme la probabilité que E ne tombe pas en panne avant un instant t donné (ou alors que E tombe en panne après l'instant t). Pour illustrer cela, prenons la variable aléatoire T représentant l'instant auquel la défaillance se produit. Nous avons alors :

$$\mathrm{F}(t) = \mathrm{P}\ (E \text{ non défaillant sur l'intervalle } [0,\ t] \text{ sachant que } E \text{ non défaillant à } t{=}0)$$

1.3.1.2 Maintenabilité

Très similaire à la fiabilité de part sa définition, la maintenabilité est, selon la norme NF X 60-500 [44], « l'aptitude d'une entité à être remise en état, par une action de maintenance donnée, d'accomplir des fonctions requises dans les conditions données ». Elle se caractérise par la probabilité que l'entité soit en état à l'instant t sachant qu'elle était défaillante à t=0.

$$\mathrm{M}(t) = \mathrm{P}(E \text{ en état à } t \text{ sachant que } E \text{ défaillant à } t{=}0)$$

Ainsi, la maintenabilité représente l'efficacité de l'action de maintenance sur l'entité. La maintenabilité, c'est la brièveté des pannes (état d'une entité dû à une défaillance) [42].

1.3.1.3 Disponibilité

Selon la norme NF X 60-500 [44], la disponibilité est « l'aptitude d'une entité à être en état d'accomplir une fonction requise dans des conditions données, à un instant donné ou pendant un intervalle de temps donné, en supposant que la fourniture des moyens extérieurs nécessaires de maintenance soit assurée ». Notons que par rapport à la fiabilité, cette notion de disponibilité est une notion instantanée : l'entité peut potentiellement avoir été défaillante puis réparée dans l'intervalle $]0, t[$. Ce qui importe pour la disponibilité est que l'entité remplisse correctement sa fonction au moment où le besoin est exprimé par l'utilisateur.

Mathématiquement, la disponibilité se définit comme la probabilité que l'entité ne soit pas défaillante à l'instant t :

$$\mathrm{D}(t) = \mathrm{P}\ (E \text{ non défaillante à l'instant } t)$$

1.3.1.4 Sécurité

La sécurité est « l'aptitude d'une entité à ne pas causer de dommages dans des conditions données ou à ne pas faire apparaître, dans des conditions données, des évènements critiques ou catastrophiques ». En d'autres termes, la sécurité est l'état dans lequel le risque de dommages corporels ou matériels est limité à un risque acceptable.

Remarquons ici et pour fixer les choses que l'emploi de la notion de « sûreté »est parfois très proche de celui de la notion de « sécurité ». Le sens que nous utiliserons est à rapprocher du terme anglais « safety » (au détriment de « security »). Nous utiliserons donc le terme « sécurité » dans le sens de l'innocuité de l'entité E pour son environnement.

Pour reprendre [42], un apport essentiel de la Sûreté de Fonctionnement est « d'intégrer la sécurité aux notions de fiabilité, de maintenabilité et de disponibilité dans une approche globale ».

1.3.2 Quelques indicateurs

Sont présentés ici quelques indicateurs permettant de mesurer la « performance » (ici, performance est à entendre au sens de la Sûreté de Fonctionnement) de l'entité étudiée. La figure 1.1 a pour vocation de présenter de manière graphique ces différents indicateurs.

MTTF (Mean Time to Failure) : Temps moyen de fonctionnement avant la première défaillance ;

MTTR (Mean Time to Repair) : Temps moyen de remise en service ;

MTBF (Mean Time Between Failure) : Temps moyen entre deux défaillances ;

MUT (Mean Up Time) : Temps moyen des durées de disponibilité (après réparation) ;

MDT (Mean Down Time) : Temps moyen des durées d'indisponibilité (suite aux défaillances).

1.4 Processus de développement d'un système aéronautique et démonstration de sa sécurité

Rappelons ici que le domaine d'application du travail présenté dans ce mémoire est le domaine aéronautique. Nous parlerons donc ici d'évaluation de la Sûreté de Fonctionnement d'un système complexe aéronautique et du processus décrit dans l'ARP (Aerospace Recommended Practice) 4754 [59]. En lien avec cette ARP, l'ARP 4761 [60] fournit un guide de méthodes et de techniques d'évaluation de la sécurité recommandées pour étudier de tels systèmes complexes.

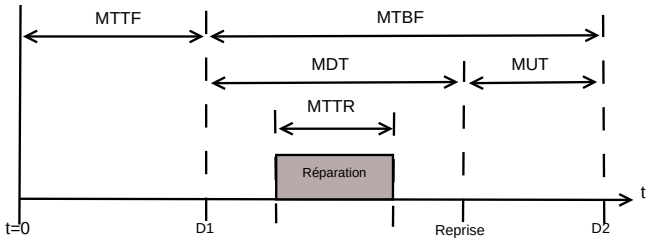


Figure 1.1 – Indicateurs SdF
D_i : Défaillance i, Reprise : Reprise du service

1.4.1 Introduction

Il est courant et fondamental dans le domaine aéronautique de parler de certification. Nous retiendrons que la certification est la démonstration « légale » devant des autorités dites de certification qu'un produit est sûr pour lui même et pour son environnement. Divers standards aéronautiques (*e.g.* ARP 4754, ARP 4761 [59, 60]) visent à guider cette phase de certification en indiquant des pratiques communément admises comme conformes pour assurer le fonctionnement sûr d'un système.

Ainsi, l'application de l'ARP 4761 [60] est recommandée pour démontrer la sécurité d'un système. Le processus défini dans ce guide décrit une méthodologie composée de plusieurs étapes, de la définition des objectifs à tenir par le système jusqu'aux analyses qualitatives et quantitatives justifiant la tenue de ces objectifs. Concernant les objectifs à tenir par le système, ceux-ci sont de deux types : des exigences de sécurité pour lutter contre les potentielles défaillances du système (pendant sa phase d'exploitation) ; des exigences concernant la rigueur à adopter pendant le développement du système pour prévenir d'éventuelles erreurs de conception.

Pour adresser ce type d'exigences, le processus d'analyse de la sécurité d'un système commence par une analyse fonctionnelle des dangers (FHA pour Functional Hazard Assessment) auxquels le système est susceptible d'être soumis. Cette analyse permet de fixer les objectifs de sécurité que le système et ses sous-systèmes se devront d'atteindre. Pour cela, l'approche consiste à décliner, à chaque niveau de détail, les objectifs de sécurité du niveau N au niveau $N-1$. Parallèlement à cette FHA, des exigences sont allouées aux fonctions réalisées par le système pour limiter les erreurs potentiellement introduites lors de la phase de conception de ces fonctions. Ensuite, une évaluation préliminaire de la sécurité du système (PSSA pour Preliminary System Safety Assessment) permet, en fixant des hypothèses sur le système, d'évaluer sa sécurité et de s'assurer que son architecture puisse remplir les objectifs identifiés par la FHA. Enfin et une fois l'architecture figée, une évaluation de la sécurité du système (SSA pour System Safety Assessment) vérifie si les objectifs sont tenus par l'architecture.

La suite de cette section a pour vocation de présenter plus en détails les différentes étapes de ce processus.

1.4.2 Identification des exigences de sécurité

1.4.2.1 Analyse fonctionnelle des dangers

L'analyse fonctionnelle des dangers (*i.e.* la FHA) consiste à identifier les conditions de défaillance affectant la sécurité de l'aéronef et à les classifier en fonction de la gravité de leurs effets.

Effet sur l'aéronef	Sans effet	Faible réduction des capacités ou des marges de sécurité	Réduction significative des capacités ou des marges de sécurité	Réduction importante des capacités ou des marges de sécurité	Perte
Effet sur passagers	Dérangement	Souffrances physiques	Possibles blessures	« Relativement peu » de blessures graves aux passagers	Plusieurs décès ou blessures graves
Effet sur l'équipage	Sans effet	Augmentation de la charge de travail	Dérangement ou augmentation importante de la charge de travail	Souffrances physiques ou augmentation très importante de la charge de travail	Décès ou immobilisation
Classification	Sans effet	Mineur	Majeur	Dangereux	Catastrophique

Tableau 1.1 – Critère de classification de la gravité des conditions de défaillance

Pour des avions civils, l'analyse est traditionnellement conduite à différents niveaux : le niveau « avion » [2] et le niveau « système ».

Pour les différents niveaux d'étude (*i.e.* avion et système) et après avoir identifié leurs fonctions, on s'intéresse à l'identification des conditions de défaillance ainsi qu'à la description de leurs effets sur le système. L'analyse de ces effets permet de classer ces conditions de défaillance conformément à la réglementation en vigueur (tableaux 1.1).

1.4.2.2 Exigences de sécurité dérivées de la FHA

La classification de la gravité d'une défaillance permet de dériver différentes exigences ayant pour objectif de rendre le risque acceptable. La table 1.2 explicite la correspondance entre la classification et les exigences de sécurité qualitatives et quantitatives liées à chacune des conditions de défaillance identifiée. La colonne intitulée « DAL » (Development Assurance Level) concerne les niveaux d'assurance en développement et sera détaillée dans la section 1.4.2.3.

La colonne « Probabilité d'occurrence maximum par heure de vol » fixe les exigences quantitatives découlant de la gravité de la condition de défaillance étudiée. Concernant le côté qualitatif, les exigences de ce type s'expriment le plus souvent par le fait qu'un nombre minimum de défaillances doit précéder l'occurrence d'un évènement redouté. C'est le propos de la troisième colonne intitulée « sécurité intégrée » qui décrit l'idée que parfois, l'architecture doit être naturellement tolérante aux défaillances simples.

2. Nous prendrons comme convention ici d'utiliser le terme « avion » pour désigner le système de plus haut niveau ; le terme « système » sera utilisé pour désigner les systèmes et sous-systèmes du niveau « avion » ; le terme « composant » sera utilisé pour désigner les entités de plus bas niveau.

Gravité des évènements redoutés	DAL	Sécurité intégrée	Probabilité d'occurrence maximum par heure de vol
Catastrophique (CAT)	A	Requis	$P < 10^{-9}$
Dangereux (HAZ)	B	Peut être nécessaire	$P < 10^{-7}$
Majeur (MAJ)	C	Peut être nécessaire	$P < 10^{-5}$
Mineur (MIN)	D	Non	Aucun
Sans effet (NSE)	E	Non	Aucun

Remarque : La colonne « Sécurité intégrée » est l'analogue du concept « fail safe » → une condition de défaillance catastrophique ne doit pas être entraînée par une unique défaillance.

Tableau 1.2 – Objectifs de Sécurité - Source : ARP 4754 [59]

1.4.2.3 Niveaux d'assurance en développement

Dans le milieu aéronautique, les niveaux d'assurance en développement sont alloués aux fonctions ou aux éléments qui constituent une architecture fonctionnelle ou physique du système selon les règles définies dans l'ARP 4754 [59]. Selon cette ARP 4754, le processus d'assurance en développement est « un processus comprenant l'ensemble des actions spécifiques planifiées et systématiques qui garantissent que les erreurs ou omissions en matière d'exigence ou de conception ont été identifiées et corrigées de telle sorte que le système, tel qu'il est réalisé, satisfait aux exigences applicables de certification ».

En pratique, ce niveau d'assurance en développement du système est attribué en fonction de la classification des conditions de défaillance présentée tableau 1.1. Le niveau d'assurance en développement est alors un indicateur de la rigueur à fournir lors du processus de développement du système. Plus le niveau est élevé (A $>$ B $>$ ... $>$ E), plus le développement du système devra être rigoureux.

Exemple : L'allocation d'un niveau d'assurance en développement (d'un DAL) est plus complexe que la simple lecture de la table 1.2. L'ARP 4754 fournit des règles d'allocation en fonction de l'architecture réalisant la fonction considérée. Par exemple, une fonction f dont la défaillance aurait un effet catastrophique sur le système considéré doit, d'après la table, être développée avec un DAL A. En pratique et selon l'ARP 4754 les possibilités pour ce développement sont multiples suivant l'architecture réalisant la fonction. Imaginons que f soit réalisée par trois sous fonctions redondantes f_1, f_2 et f_3. Alors et selon l'ARP 4754 :

- chacune des f_i devra être développée avec un niveau au moins égal à C ;
- si ces fonctions sont indépendantes, on devra avoir au choix :
  - une des fonctions f_i développée au niveau A,
  - deux des fonctions développées au niveau B.

Une solution satisfaisant l'ARP 4754 serait alors, par exemple, de développer f_1 et f_2 au niveau B et de développer f_3 au niveau C.

Suite aux activités menées dans cette section 1.4.2, une architecture préliminaire (à ce stade de haut niveau) est construite.

1.4.3 Évaluation préliminaire de la sécurité du système - PSSA

L'évaluation préliminaire de la sécurité d'un système accompagne la phase de conception du système et a pour objectif de vérifier le plus tôt possible si l'architecture système proposée peut prétendre satisfaire les exigences issues de la FHA.

Ainsi, la PSSA est un processus itératif en cela qu'il commence relativement tôt dans le processus de développement de l'architecture système. Ensuite et dans l'idéal, une nouvelle PSSA doit être réalisée à chaque nouvelle architecture et ceci jusqu'à l'obtention de l'architecture finale. Ainsi, le couplage PSSA - FHA tend à nous assurer que l'ensemble des modes de défaillance a été correctement identifié, les exigences correctement déclinées et démontre que le système parviendra à tenir ses exigences. La PSSA nous fournit alors un bon indicateur concernant le niveau de sécurité de l'architecture système et nous fournit les indications nécessaires à sa modification en vue de satisfaire les exigences nouvellement déclinées de celle de haut niveau. Ces exigences sont obtenues grâce à des analyses de type « top-down ». Classiquement, les ARP 4754 et 4761 [59, 60] préconisent l'utilisation des arbres de défaillance.

Une fois l'architecture préliminaire de notre système figée et les exigences de plus bas niveau déclinées, la conception détaillée du système est en mesure d'être réalisée. La prochaine étape vise à s'assurer que la conception détaillée conçue est conforme aux exigences issues de la FHA, de la PSSA et de la CCA.

1.4.4 Analyse des causes communes - CCA

L'analyse des causes communes (ou CCA pour Common Causes Analysis) a pour objectif d'identifier les défaillances ayant des conséquences sur plusieurs parties du système. En effet, beaucoup d'analyses et d'outils utilisés dans les activités de Sûreté de Fonctionnement supposent l'indépendance des différents évènements (des différentes défaillances) considérés. L'analyse des causes communes permet alors de s'assurer que cette hypothèse d'indépendance est vérifiée et, dans le cas contraire, de modifier l'architecture afin de la vérifier. Pour cela, une CCA est composée de trois analyses principales :

- une analyse de sécurité de zone (ZSA pour Zonal Safety Analysis) effectuée dans chaque zone de l'avion vise à limiter les effets d'une défaillance sur les autres composants ;
- une analyse des risques particuliers (PRA pour Particular Risks Analysis) détermine les risques dus aux éléments externes du système (feu, oiseaux, foudre...) ;
- une analyse des modes communs (CMA pour Common Modes Analysis) vérifie l'indépendance des évènements élémentaires présents dans les analyses de Sûreté de Fonctionnement.

De cette analyse peuvent résulter différentes exigences d'indépendance et / ou de ségrégation.

1.4.5 Évaluation de la sécurité du système - SSA

L'évaluation de la sécurité du système (la SSA) est la phase suivante du processus décrit dans l'ARP 4754 [59]. Si la forme de la SSA est quelque peu similaire à celle de la PSSA, les deux analyses diffèrent dans leurs objets. En effet, là où la PSSA évalue une architecture et permet, grâce à la FHA, de décliner des exigences, la SSA montre que ces exigences qualitatives et quantitatives sont tenues par l'architecture et par la conception détaillée de notre système.

1.5 Démarches et méthodes traditionnelles en SdF

Le processus d'évaluation de la sécurité d'un système aéronautique décrit dans l'ARP 4754 [59] nécessite à ses différentes étapes différents types d'analyses. Pour réaliser ces analyses, diverses approches et méthodes sont classiquement utilisées. Nous présentons ici certaines d'entre elles.

1.5.1 Analyse fonctionnelle

Selon la norme NF X 50-151 [5], une analyse fonctionnelle est « une démarche qui consiste à rechercher, ordonner, caractériser et hiérarchiser les fonctions du produit attendu par l'utilisateur en phase de conception »... Nous retiendrons, pour présenter le pourquoi de son application en Sûreté de Fonctionnement, que l'analyse fonctionnelle « permet d'identifier les fonctions qui réalisent les services pour lesquels un niveau de confiance devra être justifié » [41].

Ainsi, une analyse fonctionnelle est généralement réalisée au commencement d'une étude de Sûreté de Fonctionnement. Elle permet alors :

- de systématiser la recherche des fonctions primaires (fonctions pour lesquelles le produit est réalisé) du système ;
- de systématiser la recherche des fonctions contraintes (imposées par l'environnement extérieur au produit) du système ;
- de représenter les dépendances fonctionnelles entre ses fonctions ;
- de raffiner une fonction d'un niveau N au niveau $N-1$;
- d'identifier et de décrire le périmètre et les limites du système.

Cette identification du système, de ses fonctions et de son environnement est un point de départ satisfaisant pour l'identification des caractéristiques dysfonctionnelles du système : ses évènements redoutés, ses modes de défaillance, ses chemins de propagation de défaillance, ses barrières de sécurité...

1.5.2 Analyse des Modes de Défaillance et de leurs Effets - AMDE

L'Analyse des Modes de Défaillance et de leurs Effets, ou AMDE, est une méthode inductive (car partant des évènements de bas niveau, on en déduit les conséquences sur le système ; par opposition aux méthodes déductives où l'on cherche les évènements élémentaires conduisant à un évènement redouté donné) d'analyse prévisionnelle de la fiabilité permettant de recenser de manière systématique les défaillances potentielles d'un composant d'un système puis d'estimer les conséquences sur le fonctionnement de ce système. Pour cela, l'AMDE s'intéresse à la recherche systématique :

- des modes de défaillance de chacun des composants ou des fonctions du système (*e.g.* perte de la fonction, dégradation de la fonction, réalisation intempestive de la fonction) ;
- des conséquences de chacun de ces modes de défaillance sur l'environnement du composant (la fonction) ainsi que sur le système ;
- des moyens de détection et de mitigation de ces modes de défaillance.

Le tableau 1.3 présente le principe général d'une AMDE. L'exemple considéré est celui d'un système hydraulique composé de deux pompes fonctionnant en parallèle.

Si l'AMDE est à coup sûr un outil intéressant pour évaluer la Sûreté de Fonctionnement d'un système, elle ne permet cependant pas d'obtenir une vision des combinaisons de défaillance

Composant	*Mode de Défaillance*	*Effet local*	*Effet Système*
Pompe	Rupture de la pompe	Plus de fluide en sortie de la pompe	Aucun effet puisque seconde pompe en redondance

Tableau 1.3 – Principe très général d'une AMDE

et de leurs effets sur le système : que ce passe t-il si deux pannes surviennent dans le système ? Pour répondre à ce type de questions, il est nécessaire d'effectuer des analyses complémentaires, par exemple par arbre de défaillance.

1.5.3 Arbre de Défaillance

L'analyse d'un système par arbre de défaillance a initialement été développée par la compagnie Bell Telephone durant les années 1960 afin d'évaluer la sécurité d'un système de lanceur de missile. De nos jours, ce type d'analyse est utilisé dans divers domaines (*e.g.* aéronautique, automobile, ferroviaire, nucléaire...) et est une des techniques recommandées par l'ARP 4761 [60] pour évaluer la Sûreté de Fonctionnement d'un système complexe aéronautique. En deux mots, un arbre de défaillance a pour objectif de représenter, de manière arborescente, les combinaisons d'évènements pouvant conduire le système dans un état redouté.

1.5.3.1 Principe des arbres de défaillance

L'arbre de défaillance, grâce à un formalisme graphique, détermine, représente et quantifie les combinaisons d'évènements qui placent le système dans un état redouté. L'analyse par arbre de défaillance est une analyse déductive : pour un évènement donné, le principe est d'identifier ses causes immédiates provoquant l'occurrence de l'évènement.

En effet, une telle analyse identifie dans un premier temps les causes immédiates, nécessaires et suffisantes à l'occurrence de l'évènement étudié (au commencement, celui-ci est l'évènement redouté de haut niveau ; aux itérations suivantes, il s'agit des évènements identifiés à l'étape précédente). On obtient ainsi une liste de sous-évènements. Ensuite, on classifie chacun de ces évènements : il s'agit soit d'un évènement élémentaire (en général, un tel évènement élémentaire représente une défaillance d'un composant élémentaire du système), soit d'un évènement non développé (évènement que l'on ne souhaite volontairement plus développer), soit d'un évènement intermédiaire (*i.e.* évènement que l'on souhaite encore développer). On réitère le processus (*i.e.* identification des causes immédiates des évènements intermédiaires) jusqu'à n'obtenir que des évènements élémentaires ou non développés. On remonte ainsi de causes en causes jusqu'aux évènements étant à l'origine de l'état redouté du système.

La figure 1.2 montre un exemple d'arbre de défaillance et illustre, de manière très simple, le formalisme attaché à la méthode. Sur la figure, les évènements $E1$, $E2$ et $E3$ sont des évènements élémentaires. Également illustré sur la figure, l'emploi des portes « ET » et « OU » :

- une porte « OU » indique qu'au moins un des évènements doit se produire pour que l'évènement redouté se produise. Ainsi, sur la figure 1.2, l'évènement intermédiaire se produira si $E2$ ou $E3$ se produit ;
- une porte « ET » indique que tous les évènements doivent exister simultanément (leurs occurrences respectives n'ont pas pour autant besoin d'être simultanées) pour que l'évènement redouté se produise. Ainsi, sur la figure 1.2, l'évènement redouté se produira si $E1$ et l'évènement intermédiaire se produisent (et par transitivité, l'évènement redouté se produira si $E1$ et $E2$ se produisent ou si $E1$ et $E3$ se produisent).

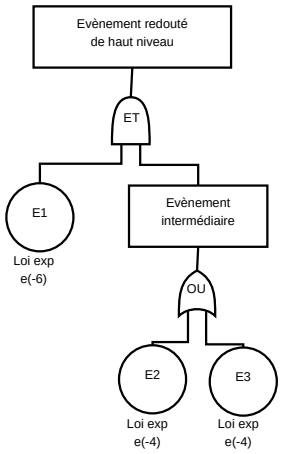


Figure 1.2 – Exemple d'arbre de défaillance

1.5.3.2 Arbres de défaillance : identification des évènements élémentaires

Classiquement, pour identifier les évènements élémentaires (section 1.5.3.1) d'un arbre de défaillance, on construit préalablement une AMDE (section 1.5.2). Cependant souvent et comme recommandé par l'ARP 4761 [60], un résumé d'AMDE (ou FMES pour Failure Modes and Effects Summary) est réalisé notamment dans un soucis de concision des arbres de défaillance.

Selon cette même ARP 4761, une FMES est un résumé des modes de défaillance identifiés dans l'AMDE. Ces modes de défaillance sont alors regroupés en fonction de leurs effets sur le système. Le taux de défaillance résultant est alors la somme des taux (si les lois sont toutes exponentielles) de chacun des modes de défaillance regroupés. En pratique, utiliser un résumé d'AMDE (au lieu de l'AMDE) en entrée des arbres de défaillance permet de limiter le nombre d'évènements élémentaires (section 1.5.3.1) et de rendre ainsi les évaluations qualitatives et quantitatives d'un arbre de défaillance plus performantes.

Effet	Causes	Taux résultant
Perte du système	$C_1.MdD_1$ $C_1.MdD_3$ $C_2.MdD_1$	λ_1
Mode dégradé du système	$C_1.MdD_2$ $C_2.MdD_2$	λ_2
...	...	...

Tableau 1.4 – Résumé d'AMDE
(C_i : Composant i ; MdD_j : Mode de Défaillance j)

1.5.3.3 Exploitation de l'arbre - Aspects qualitatifs

Une fois l'arbre de défaillance construit, il est possible de produire des résultats qualitatifs et quantitatifs relatifs à l'occurrence de l'évènement redouté étudié (*i.e.* l'évènement racine de l'arbre). En ce qui concerne l'aspect qualitatif, il est notamment possible de construire l'ensemble des coupes minimales de l'arbre [54] :

Définition 1.4 : Coupe

Une *coupe* est « une combinaison d'évènements dont l'existence simultanée conduit à l'occurrence de l'évènement redouté ». On parlera donc et par conséquence de coupe « par rapport » à un évènement redouté. ■

Définition 1.5 : Ordre d'une coupe

L'*ordre* d'une coupe est « le nombre d'évènements qui compose la coupe ». ■

Définition 1.6 : Coupe minimale

Une *coupe minimale* est « une coupe ne contenant aucune autre coupe » (précisons ici qu'on ne parle que de combinaison et qu'en conséquence, l'ordre d'occurrence n'a pas d'importance). Une coupe minimale peut alors être considérée comme une combinaison d'évènements nécessaire et suffisante à l'occurrence de l'évènement considéré. ■

Une fois les coupes minimales obtenues, la question de l'exploitation des résultats est naturelle. L'arbre en lui même nous permet de visualiser comment la défaillance d'un composant élémentaire se propage jusqu'à l'événement redouté. Les coupes minimales permettent ensuite de connaître les événements ou les combinaisons d'évènements les plus critiques pour le système en considération. Cependant, se fonder uniquement sur l'ordre des coupes revient à négliger les probabilités d'occurrence des évènements élémentaires en supposant ceux-ci équiprobables. En effet, une coupe d'ordre 1 peut être moins probable qu'une coupe d'ordre 3 (par exemple).

1.5.3.4 Exploitation de l'arbre - Aspects quantitatifs

L'analyse quantitative des arbres de défaillance a pour but d'estimer la probabilité d'occurrence de l'évènement redouté auquel l'arbre se réfère à partir des probabilités d'occurrence de tous les évènements élémentaires présents dans l'arbre.

Pour acquérir les probabilités de ces différents évènements élémentaires, on utilisera des bases de données (essais, retour d'expérience...) ainsi que des jugements d'experts. Connaissant l'ensemble de ces probabilités, il est possible de calculer la probabilité de chaque coupe de l'arbre : la probabilité d'une coupe est le produit des probabilités des évènements qui la compose (ces évènements devant alors être indépendants[3]). En sommant les probabilités de chaque coupe, on majore la probabilité de l'évènement redouté étudié. Cependant, des outils comme l'outil Aralia permettent d'évaluer de manière exacte un arbre de défaillance [56].

1.5.3.5 Avantages et limites

La méthode se voulant analyser la Sûreté de Fonctionnement d'un système à l'aide d'arbres de défaillance a différents avantages et différentes limitations / inconvénients dans son utilisation. Parmi les avantages de la méthode, citons notamment que l'analyse par arbre de défaillance permet :

- une représentation graphique et peut ainsi être utilisée comme un outil de communication visuelle ;
- d'expliciter clairement les relations logiques liant des évènements élémentaires et les évènements redoutés du système ou de ses sous-systèmes ;

3. Remarquons que cette hypothèse d'indépendance justifie le besoin d'analyse de mode commun présenté section 1.4.

- d'obtenir des équations simples (d'un point de vue combinatoire) pour décrire l'occurrence des évènements redoutés ;
- de se focaliser sur un évènement redouté et ne pas prendre en compte les évènements n'intervenant pas dans son occurrence ;
- de pouvoir étudier le comportement du système à différents niveaux d'abstraction ;
- une validation pas à pas en permettant l'étude de sous-arbres indépendants.

Cependant, ces arbres de défaillance présentent également différentes limites d'utilisation. En effet :

- leur taille peut devenir grande dans le cas de systèmes particulièrement complexes ; la lisibilité des arbres peut ainsi devenir difficile ;
- ils permettent difficilement la représentation de défaillances « transitoires » et font abstraction de l'ordre d'apparition des évènements (cependant, l'état de l'art sait modéliser les stratégies de réparation ou de reconfiguration d'un système) ;
- ils ne permettent pas d'une manière générale de modéliser l'évolution d'un système au cours du temps ;
- toute modification de l'architecture système doit être suivie d'une revue et de potentielles modifications sur l'ensemble des arbres ;
- il est nécessaire de réaliser un arbre de défaillance par évènement redouté ; la validation de l'ensemble des arbres peut se révéler coûteuse en temps et en ressources.

L'hypothèse du comportement non évolutif du système peut être en partie levée par l'ajout d'extensions à l'analyse classique par arbres de défaillance (par exemple, l'ajout de porte logique prenant en compte des priorités [63]). On pourra aussi se tourner vers d'autres types d'analyses comme l'analyse par graphes de Markov préconisée par l'ARP 4761 [60] au même titre que l'analyse par arbre de défaillance.

1.5.4 Graphe de Markov

En deux mots, un graphe de Markov est un processus stochastique (*i.e.* aléatoire), sans mémoire (*i.e.* le futur ne dépend que de l'état présent), portant sur un nombre fini d'états reliés entre eux par des probabilités de transition (éventuellement nulle). Un graphe de Markov permet de modéliser le comportement de systèmes évoluant dans le temps.

De manière plus détaillée maintenant, un graphe de Markov représente le comportement d'un système à l'aide d'un automate : les états du système sont représentés par les états de l'automate ; les défaillances ou les évènements fonctionnels (*e.g.* reconfiguration...) modifiant l'état du système sont décrits dans l'automate par des transitions. À l'aide de probabilités attachées à chaque transition et connaissant l'état initial du système, la probabilité d'être dans chaque état peut être calculée à tout moment. Pour plus de détails, le lecteur intéressé pourra consulter [65].

Si l'ensemble des transitions du graphe de Markov suit une loi exponentielle, des calculs exacts peuvent être réalisés de manière relativement aisée. Par exemple, on peut calculer (on trouvera davantage de détails dans [61]) :

- la probabilité d'être dans un état (ou un ensemble d'état) à un instant i donné ;
- le temps moyen de bon fonctionnement avant première défaillance (MTTF) ;
- pour un système réparable, le temps moyen entre deux défaillances (MTBF) ;
- la disponibilité d'un système...

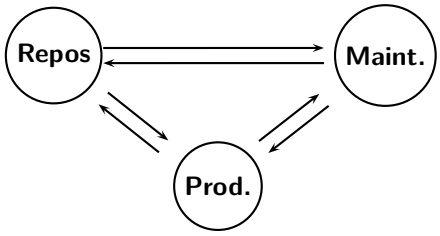


Figure 1.3 – Exemple de graphe de Markov

Permettant de prendre en compte et de modéliser des actions de maintenance (*i.e.* de réparation), les graphes de Markov sont adaptés lorsque le but de l'étude est l'évaluation de la disponibilité du système. Pour les études plutôt orienté sécurité, il est souvent préféré les arbres de défaillance.

Ainsi, les analyses par graphe de Markov présentent plusieurs avantages. Ils autorisent la modélisation d'un certain nombre de comportements ne pouvant pas être décrits par les arbres de défaillance. Par exemple, ils permettent de décrire des reconfigurations ou des réparations du système étudié. Cependant, une des limitations majeures d'une telle approche vient du problème classique d'explosion combinatoire (*i.e.* explosion du nombre d'états du graphe de Markov). En effet, imaginons un système S contenant N composants ayant tous un comportement booléen. Le graphe de Markov modélisant le système S pourra contenir jusqu'à 2^N états. Les outils de calculs actuels ne peuvent alors traiter plus d'une vingtaine de composants (environ 1 million d'états) ce qui limite l'utilisation des graphes de Markov à de petits systèmes.

1.5.5 Réseaux de Petri

Un réseau de Petri (RdP) [64] est un modèle mathématique permettant de représenter le comportement de systèmes dynamiques. Un RdP se représente de manière graphique par un graphe orienté reliant des places (représentées par des cercles) et des transitions (représentées par des traits) par l'intermédiaire d'arcs (représentés par des flèches). Ces arcs relient alors une place à une transition ou une transition à une place (deux places ne peuvent être reliées entre elles, deux transitions ne peuvent être reliées entre elles). Chaque place peut contenir un ou plusieurs jetons qui définissent, à un instant t, l'état (le marquage) du RdP. Si le poids des arcs d'entrée et sortie des transitions est unitaire, il n'est pas indiqué sur les arcs, comme représenté sur la figure 1.4. Dans ce cas, lorsque toutes les places p_i en amont d'une transition T contiennent au moins un jeton, la transition peut être franchie. Dès ce franchissement, on retire alors un jeton des places p_i amont et on ajoute un jeton aux places q_j situées en aval de la transition T[4]. La figure 1.4 présente un RdP avant puis après le franchissement d'une transition.

Les RdP stochastiques sont une extension des RdP où des durées de sensibilisation définies par des distributions géométriques ou exponentielles sont associées aux transitions. Il est alors possible de construire un processus markovien équivalent.

Grâce à une telle représentation, les réseaux de Petri permettent l'obtention d'un modèle du comportement du système considéré. L'exploitation d'un modèle se fait alors soit de façon analytique à partir du processus de Markov équivalent, soit grâce à la simulation de Monte-Carlo et permettent de calculer ([62]) par exemple la disponibilité d'un système, sa fiabilité ou encore certains indicateurs comme le MTTF ou le MTBF (section 1.3.2).

4. Si le poids d'un arc en amont est $w(p_i, t) = k_i, k_i \in N$, on retire k_i de p_i ; si le poids des arcs en aval est $w(t, q_j) = k_j, k_j \in N$, on rajoute k_j jetons à la place q_j).

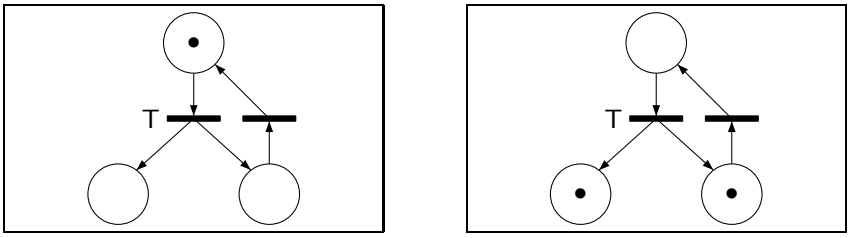


Figure 1.4 – Exemple de Réseau de Petri
Marquage avant puis après le franchissement de la transition T

Remarque : Sans rentrer dans les détails ici, la simulation de Monte-Carlo [62] se veut remplacer le calcul analytique exact par un calcul statistique. Pour cela, la méthode est fondée sur la réalisation d'un grand nombre de simulations d'un modèle stochastique. Le principe est ainsi de simuler un grand nombre de fois le modèle en générant aléatoirement à chaque simulation les évènements se produisant (ces évènements devant alors être dotés de loi de probabilités). Les différentes simulations nous permettent alors d'évaluer la probabilité de se trouver dans tel ou tel état du modèle (on peut par exemple calculer la probabilité d'occurrence d'un évènement redouté). Le résultat est alors une approximation du résultat exact. Plus le nombre de simulations est grand, plus l'approximation est bonne.

1.6 Positionnement des travaux de thèse par rapport à la problématique SdF : vers une modélisation formelle de système

Comme nous le verrons par la suite, les systèmes moteurs étudiés contribuent à la génération et à la distribution d'énergie mécanique au rotor de l'hélicoptère. Pour accomplir ces fonctions, ils sont structurés en différents systèmes (circuit d'air, circuit carburant, circuit d'huile, système de contrôle moteur...) constitués de composants de natures physiques différentes (mécanique, hydromécanique, électronique, logiciel...). On s'intéresse à l'évaluation des exigences de sécurité que ces systèmes doivent satisfaire (FHA, PSSA, SSA du processus décrits dans l'ARP 4754).

Pour l'évaluation de ces exigences, différentes méthodes couramment utilisées pour évaluer la Sûreté de Fonctionnement d'un système complexe aéronautique ont ainsi été présentées. Si les graphes de Markov sont sans doute moins répandus de part la limitation concernant la taille des systèmes pouvant être étudiés, les arbres de défaillance sont aujourd'hui toujours l'approche la plus utilisée pour les industries mettant en œuvre des systèmes critiques complexes (sinon, une identification des pannes simples grâce à l'AMDE suffit). Cependant et de part les limitations déjà évoquées section 1.5.3.5, ces arbres de défaillance présentent des difficultés lors de la représentation de systèmes possédant des défaillances transitoires, lors de la représentation de systèmes sujets à réparation ou reconfiguration ou lors de la représentation de systèmes à l'intérieur desquels l'ordre des évènements influe sur l'état final du système.

Nous présenterons dans le chapitre 2 le langage formel AltaRica qui supporte les analyses de sécurité qualitatives et quantitatives de systèmes critiques complexes.

Chapitre 2

Modélisation formelle de systèmes

Sommaire

Bien que les arbres de défaillance soient aujourd'hui une méthode maitrisée et éprouvée, nous avons vu section 1.5.3.5 qu'ils présentent certaines limites d'utilisation. Citons en ici quelques unes pour rappel :

- les systèmes actuels sont des systèmes complexes hautement reconfigurables et la création des arbres de défaillance devient lourde et couteuse ;
- un arbre de défaillance est centré sur un unique évènement redouté ;
- le formalisme et la taille des arbres peuvent en rendre l'accès difficile aux non-initiés.

Au delà de la volonté de dépasser ces limites, plusieurs besoins majeurs ont été identifiés :

- un besoin de standardisation. [45] définit ce concept comme « la programmation à l'avance de certains aspects d'un travail de manière telle que les standards ainsi prévus aient une certaine stabilité dans le temps et s'appliquent de manière similaire ». On retiendra que, généralement, la standardisation est un procédé qui permet de gagner du temps et de réduire les coûts. Plus rapide, moins cher et souvent plus fiable, ce procédé est aujourd'hui répandu dans de nombreux domaines. Cette notion de standardisation favorise la création de modèles génériques et la réutilisation de ces modèles ;
- un besoin de traçabilité entre les informations implémentées dans les modèles et leurs origines (Quels sont les documents ayant servis à la modélisation ? Quelles sont les hypothèses de modélisation ? ...) ;
- un besoin de posséder des modèles compositionnels. Ainsi, les chemins de propagation de défaillance ne sont plus représentés explicitement (comme sur les arbres de défaillance) mais sont définis à partir de compositions de composants élémentaires ;
- un besoin de cohérence :
  - entre les analyses fonctionnelles et les analyses dysfonctionnelles d'un système, sous-système ou entité,
  - entre différents arbres d'évènements redoutés au sein d'un même système,
  - entre différents arbres d'évènements redoutés sur des systèmes différents de même famille ;
- un besoin d'outils permettant de supporter et d'automatiser l'évaluation de la Sûreté de Fonctionnement d'un système.

Dans l'optique de satisfaire les besoins évoqués, divers travaux tels que [36] ont montré qu'il était possible d'utiliser un modèle formel de propagation de défaillance du système pour évaluer sa sécurité (on parlera alors et notamment de génération automatique d'arbre de défaillance). Parmi les différents langages disponibles (nous en présenterons certains section 2.5 : AADL, Nu-SMV, SCADE, Figaro, HIP-HOPS...), le langage AltaRica a été adopté.

2.1 Généralités autour de la modélisation de système

Comme le présente le chapitre 1, les systèmes complexes sont potentiellement de nature variée. Aussi, ils sont autour de nous omniprésents : systèmes physiques, systèmes informatiques, système sociaux. Afin de pouvoir étudier ces systèmes, une approche adaptée est de se munir d'une représentation du comportement de ces systèmes. Souvent alors, il s'agira d'élaborer, à partir du système à comprendre, un modèle qui reproduit les caractéristiques clefs de son fonctionnement. Il est alors possible de simuler et d'expérimenter, sur le modèle et en dotant celui-ci d'une interface avec l'utilisateur, le fonctionnement du système.

Après avoir traité la notion de système dans le chapitre 1, nous traitons dans cette section plus en détail de la notion de « modèle ».

2.1.1 Notion de modèle

Selon [50], pour échanger, transmettre ou traiter des connaissances, il est nécessaire de pouvoir les représenter sur un support extérieur à notre cerveau sous une forme transmissible et traitable. En guise de représentation, l'apport de l'utilisation de modèles n'est aujourd'hui plus remis en question lors de l'étude de systèmes réels pour en comprendre, décrire et prédire les comportements.

De [24], nous tirons une définition selon laquelle un modèle est un « système de symboles » dont l'extrême souplesse potentielle permet de rendre compte de la plupart des perceptions dont on dispose lorsque nous souhaitons décrire un phénomène (observé ou imaginé) afin de l'interpréter intelligiblement. De cette définition, nous insisterons sur deux principes fondamentaux. Tout d'abord, un modèle est construit dans le but de répondre à une problématique relative au système considéré. Le besoin se devra donc d'être défini au plus tôt dans l'optique de guider et de faciliter la construction du modèle. Ensuite, un modèle étant le reflet de la perception de l'analyste, il n'est qu'une façon de représenter cette perception. Un même système peut donc tout à fait être représenté par deux (ou plus) modèles différents.

... explicitons maintenant cette notion de construction de modèle...

2.1.2 Modélisation

Nous avons donc besoin de modèles pour, dans de nombreux cas, appréhender et comprendre un système donné. Nous appelons très classiquement « modélisation » l'activité de construction de modèle. Commençons par citer quelques définitions issues de la littérature.

- [25] décrit l'action de modéliser comme étant d'abord un processus technique qui permet de représenter, dans un but de connaissance et d'action, un objet ou une situation voire un événement réputé complexe.
- Selon [24], la modélisation est l'opération par laquelle on établit un modèle d'un phénomène, afin d'en proposer une représentation interprétable, reproductible et simulable.
- Enfin, selon [50] et traduit des auteurs de la méthode UML, la modélisation est une activité centrale de tous les processus qui conduisent au déploiement de logiciels de qualité. Nous construisons des modèles pour communiquer la structure et le comportement de nos systèmes. Nous construisons des modèles pour visualiser et contrôler l'architecture du système.

Notons à ce stade qu'il est souvent nécessaire que l'analyste se fasse une première représentation mentale de ce qu'est le système ainsi que de son comportement. Pour donner une définition de ce que nous entendons par « représentation mentale », nous reprenons et adaptons à notre contexte la définition donnée par [35] selon laquelle il s'agit « d'une sorte de modèle intériorisé

que l'analyste construit pour rendre compte du comportement d'une entité (ou d'un système) vis-à-vis d'elle-même et de son environnement ». L'action de modéliser consiste alors à « élaborer, à partir de l'entité ou du système à observer, une représentation qui reproduise artificiellement les caractéristiques de son comportement [...] dans le but de rendre son fonctionnement intelligible par l'analyste ».

Ainsi, l'objectif du modèle est en somme de simplifier le système tout en conservant ses propriétés intéressantes (pour la problématique considérée). Sa construction pouvant se révéler épineuse, une attention particulière doit aussi être accordée à la forme de ce modèle. Ainsi et brièvement, le modèle se devra d'être lisible, clair et de permettre la représentation du comportement du système. En vue d'apporter de la rigueur au sein de ce processus, nous parlerons plus tard et par l'intermédiaire du langage AltaRica, de modèle formel (section 2.2).

2.1.3 Vers l'utilisation du langage AltaRica...

La modélisation d'un système vise donc à comprendre le fonctionnement d'un système en vue de répondre à une ou plusieurs questions. La problématique nous intéressant appartenant au domaine de la Sûreté de Fonctionnement, nous focalisons notre étude des modèles de propagation de défaillance ayant pour but de décrire le comportement d'un système lors de l'occurrence de défaillances.

Si d'autres langages existent, nous nous sommes intéressés au langage AltaRica pour, notamment, des raisons de disponibilité et de maturité d'outils. En effet, les débuts du langage AltaRica datant de la fin des années 1990, de nombreux travaux ont été réalisés afin d'étendre le langage. Citons deux extensions en particulier fondées sur les concepts de flux de données orientés : le langage AltaRica Data-Flow et le langage AltaRica OCAS, supporté par l'outil Cecilia^{TM} OCAS. Très proches l'un de l'autre et fondés sur la notion d'automates de mode, ces langages supportent les études de Sûreté de Fonctionnement en permettant de décrire la logique de dysfonctionnement d'un composant donné, *i.e.* sa logique de changement d'état et les fonctions réalisées dans chacun de ces états.

Le langage AltaRica et ses extensions sont alors supportés par différents outils dont notamment l'outil Cecilia^{TM} OCAS[1] qui supporte l'extension AltaRica OCAS. Développé par Dassault Aviation, cet outil apporte une interface graphique et conviviale au langage, autorise sa simulation interactive et supporte les analyses classiques de Sûreté de Fonctionnement (*e.g.* génération d'arbres de défaillance, séquences d'évènements conduisant le système dans un état redouté). La disponibilité de ces outils n'est pas innocente dans le fait qu'aujourd'hui, différentes entreprises s'intéressent à l'utilisation du langage AltaRica : l'ONERA (Office National D'Études et de Recherches Aérospatiales), Airbus, Thales, Turbomeca... [12, 20, 38, 34].

Nous présentons dans la suite le langage AltaRica OCAS, sa syntaxe et sa sémantique fondée sur la notion d'automate de mode [55].

2.2 Syntaxe du langage AltaRica OCAS

AltaRica est un langage formel développé au LaBRI (Laboratoire Bordelais de Recherche en Informatique). Il résulte d'une double initiative académique et industrielle ayant pour objectif le développement d'un atelier permettant la modélisation du comportement fonctionnel et dysfonctionnel d'un système afin d'en évaluer sa sûreté de fonctionnement. Nous nous intéressons dans cette partie à présenter une variante du langage AltaRica : le langage *AltaRica OCAS*. Sauf

1. Il existe sur le marché différents outils. Citons notamment, en 2010, Cecilia^{TM} OCAS (Dassault Aviation), BPA-FT9 (Dassault System) ou encore SIMFIA (Apsys).

mention contraire, c'est de cette version qu'il s'agira dans la suite de ce mémoire lorsque sera utilisé le terme « AltaRica ».

2.2.1 Exemple préliminaire

« Longue est la route par le précepte, courte et facile par l'exemple » Sénèque

Nous présentons ici l'exemple d'une valve hydraulique qui nous servira de fil conducteur à la présentation de la syntaxe du langage AltaRica. Pour une description simple de son comportement, nous supposons que cette valve a une entrée *entree* et une sortie *sortie* prenant toutes deux leurs valeurs dans l'ensemble {ok, faible, nul}. Cette valve possède trois évènements : *Ouverture*, *Fermeture* et *Blocage*. Le diagramme états-transitions présenté figure 2.1 illustre le fonctionnement d'une telle valve.

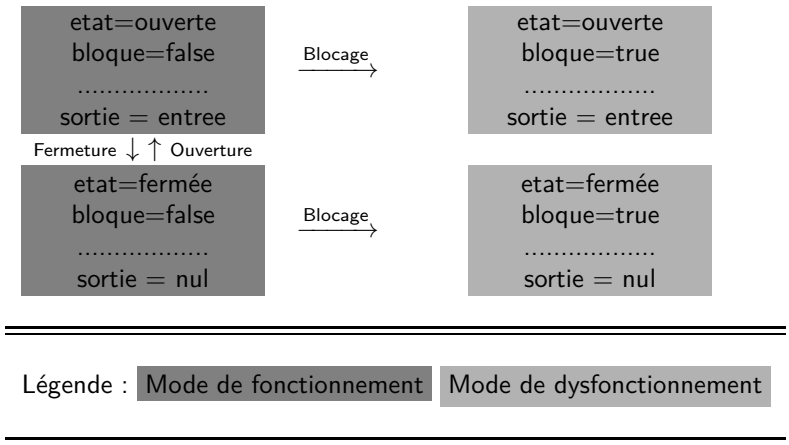


Figure 2.1 – Automate de mode représentant la valve

En AltaRica, la description du comportement d'un système met en œuvre différentes étapes parmi lesquelles : la description des différents éléments composant le système, la description des interactions entre ces composants ou la description des hiérarchies à l'intérieur du système (*i.e.* notion de système / sous-système). Dans cette optique, la construction d'un modèle met en jeu la création de plusieurs unités élémentaires que l'on appellera nœud (node). Chaque nœud AltaRica est alors composé de plusieurs champs permettant de décrire le comportement du nœud :

- une partie statique permettant la déclaration de différents types de variables (section 2.2.2 et 2.2.3 ;
- une partie dynamique permettant la description du comportement du nœud (section 2.2.4 et section 2.2.5).

2.2.2 Variables de flux, Variables d'état

La déclaration d'un nœud AltaRica requiert ainsi la déclaration de variables de flux et de variables d'état. Dans l'optique de donner une vision intuitive des deux concepts, les variables de flux servent à modéliser les échanges d'informations entre le nœud et son milieu environnant (en particulier avec les autres nœuds AltaRica) ; les variables d'état sont utilisées pour mémoriser l'état

courant du nœud (son environnement n'a alors pas accès à la valeur de cet état) et représente ainsi les différents modes de fonctionnement et de dysfonctionnement du composant AltaRica. Les variables de flux sont ainsi visibles et accessibles par l'extérieur du nœud ; les variables d'état demeurent internes au nœud et ne le sont pas.

Dans les deux cas (variables de flux et d'état) et afin de déclarer de telles variables au sein d'un nœud AltaRica, il est nécessaire de déclarer le nom de cette variable ainsi que son type. Deux différences entre les deux types de variables : les variables de flux doivent se voir préciser leurs orientations (*i.e.* si la variable de flux représente une entrée ou une sortie du nœud) ; les variables d'état doivent être initialisées. La plupart du temps, les variables de flux et d'état seront de types booléen ou énuméré (*e.g.* {ok, faible, nul}).

Écriture en BNF <liste* <construction> sep > (resp. <liste+ <construction> sep >) désigne une liste contenant zéro, un ou plusieurs (resp. un ou plusieurs) éléments de type « construction » séparés par « sep ».

```
<champs de flux> ::= flow <liste* <declaration_variable> ; >

<declaration_variable> ::= <symbole> : <domaine> : <sens_variable>

<symbole> ::= chaîne de caractère alphanumérique ou le caractère "_"

<domaine> ::=  { <liste+ <constante> ,> }

<constante> ::= true
            ::= false
            ::= symbole

<sens_variable> ::= in
                ::= out
```

Variables de flux	Variables d'état
flow	**state**
entree : {ok, faible, nul} : in ;	bloque : bool ;
sortie : {ok, faible, nul} : out ;	etat : {ouverte, fermee} ;
	init
	bloque :=false, etat :=ouverte

Figure 2.2 – Définition des variables de flux et d'état en langage AltaRica

2.2.3 Évènements

Au sein d'un nœud AltaRica, les évènements modélisent les phénomènes provoquant les changements d'états. D'un point de vue purement automate, il s'agit uniquement d'une étiquette que l'on place sur une transition (*cf.* figure 2.1 qui contient 3 évènements).

En pratique et la plupart du temps, les évènements modélisés dans le nœud AltaRica représentent des défaillances ou des reconfigurations du composant. Il est possible d'associer à chacun des évènements une loi de probabilité. Ces lois de probabilités seront utiles lors de l'analyse quantitative du modèle AltaRica (un modèle sans loi de probabilité permettra cependant une analyse purement qualitative du système représenté) . Dans le cas de défaillance, cette loi pourra être par exemple une loi exponentielle ou une loi de Weibull. Dans le cas de reconfiguration ou de propagation, on utilisera souvent une loi en *Dirac(0)* (une telle loi signifie que l'évènement sera tiré de manière certaine dès que cela deviendra possible) ; l'évènement sera alors qualifié d'évènement

instantané (en avance sur la section 2.2.4, une transition ayant pour évènement un évènement instantané sera qualifiée de transition instantanée).

Écriture en BNF <liste* <construction> sep > désigne une liste contenant zéro, un ou plusieurs éléments de type « construction » séparés par « sep ». Le champ « <symbole> » est tel que défini précédemment.

```
<champs d'évènement> ::= event <liste* <evenement> , >

<evenement> ::= <symbole>
```

```
event
  Blocage, Ouverture, Fermeture ;
extern
  law <Blocage> = exp(1.0e-8) ;
```

Figure 2.3 – Définition des évènements et de leurs lois de probabilité

2.2.4 Transitions

Au sein d'un nœud AltaRica, les transitions décrivent les changements d'états faisant suite à l'occurrence d'un évènement. Afin de décrire un tel changement d'état et de manière intuitive, il est nécessaire d'indiquer l'évènement qui produira le changement d'état, les conditions sous lesquelles cet évènement peut ce produire et les conséquences de cet évènement. Une transition AltaRica est donc constituée d'une garde, d'un évènement et d'une ou plusieurs affectations d'état :

- La garde d'une transition représente la condition d'occurrence de l'évènement ; c'est une condition booléenne dépendant des variables de flux et des variables d'état du nœud AltaRica ;
- L'évènement est l'étiquette de la transition ;
- Les affectations décrivent les conséquences de l'évènement et définissent les nouvelles valeurs des variables d'état.

Le fonctionnement d'une telle transition est alors simple. Si la garde est satisfaite, l'évènement est qualifié de tirable. Lorsque celui-ci survient, l'état du système est modifié de la façon décrite par les affectations. Sa description, en langage AltaRica, se fait de la façon suivante :

```
garde |- evenement -> assignation ;
```

Écriture en BNF Ici, <liste* <construction> sep > désigne une liste contenant zéro, un ou plusieurs éléments de type « construction » séparés par « sep ». Les champs « <evenement> », « <symbole> » et « <constante> » sont tels que définis précédemment.

```
<champ de transition> ::= trans <liste* <transition> ; >

<transition> ::= <expression> |- <evenement> -> <liste* <affectation> , >

<expression> ::= <expression1> or <expression>
```

```
             ::= <expression1>

<expression1> ::= <expression2> and <expression1>
              ::= <expression2>

<expression2> ::= <expression3> = <expression3>
              ::= <expression3> != <expression3>
              ::= <expression3>

<expression3> ::= not <expression3>
              ::= if <expression> then <expression> else <expression>
              ::= <atom>

<atom> ::= <symbole>
       ::= <constante>

<affectation> ::= <symbole> :=  <constante>
```

Dans l'exemple de la valve hydraulique, les transitions définies sont les suivantes :

```
trans
  etat=ouverte and bloque=false |- Fermeture -> etat :=fermee ;
  etat=fermee and bloque=false |- Ouverture -> etat :=ouverte ;
  bloque=false |- Blocage -> bloque :=true ;
```

Figure 2.4 – Définition des transitions

2.2.5 Assertions

Les assertions servent, pour un nœud AltaRica sans hiérarchie, à définir les valeurs de ses variables de flux de sortie (en présence de sous-nœuds, les assertions permettent aussi de définir les connexions entre ces sous-nœuds). D'un point de vue théorique, une assertion ajoute des contraintes sur les variables de flux de sortie en fonction des valeurs des variables de flux d'entrée et d'état. Ainsi, toutes les configurations du nœud ne seront pas atteignables par le nœud.

D'un point de vue plus pratique désormais, les assertions définissent la valeur des variables de sortie en fonction de celles des variables d'état et de flux d'entrée. Pour cela, une assertion se construit grâce à la classique construction *« case ..., case ..., ..., else ... »* qui est équivalente à une imbrication de *if then else*. À noter que, dans le cas du *case*, le « else » englobe toutes les situations n'ayant pas été auparavant traitées. Si cela a l'avantage de garantir que toute variable de sortie est bien définie à tout moment, cela nécessite de bien identifier les situations englobées dans le « else » (dans toute ces situations, la valeur de la variable de sortie sera identique).

Écriture en BNF <liste* <construction> sep > désigne une liste contenant zéro, un ou plusieurs éléments de type « construction » séparés par « sep ». Le champ « <expression> » est tel que défini précédemment.

```
<champ d'assertion> ::= assert <liste* <expression> ; >
```

assert
sortie = case { etat=ouverte : entree, else nul } ;

Figure 2.5 – Définition des assertions

```
node Valve
 flow
   entree : {ok, faible, nul}:in ;
   sortie : {ok, faible, nul}:out ;
 state
   etat : ouverte, fermee ;
   bloque : bool ;
 init
   etat :=ouverte, bloque :=false ;
 event
   Blocage, Ouverture, Fermeture ;
 trans
   bloque=false |- Blocage -> bloque :=true ;
   etat=ouverte and bloque=false |- Fermeture -> etat :=fermee ;
   etat=fermee and bloque=false |- Ouverture -> etat :=ouverte ;
 assert
   sortie = case { etat=ouverte : entree, else false } ;
edon
```

Figure 2.6 – Code AltaRica de la valve

2.2.6 Un nœud AltaRica

Ayant désormais défini l'ensemble des champs permettant la modélisation du comportement de la valve hydraulique décrite figure 2.1, nous montrons sur la figure 2.6 le nœud AltaRica correspondant.

2.2.7 Deux autres concepts : hiérarchie et synchronisation

2.2.7.1 Concept de hiérarchie

Un nœud AltaRica peut contenir un ou plusieurs sous-nœuds. Pour cela, la démarche de modélisation est constituée de deux étapes : déclarer les sous-nœuds au sein du nœud père, décrire les connexions entre ces sous-nœuds grâce aux assertions du nœud père. La figure 2.7 a pour but d'expliciter cette notion de hiérarchie. Dans cette exemple, le nœud présenté contient deux sous-nœuds nommés A et B. Le nœud A est de type $type\_A$, le nœud B est une instance du nœud décrit dans la figure 2.6, *i.e.* une valve. L'assertion décrit que la sortie du nœud A est connectée à l'entrée du nœud B.

Remarque : Ici, le nœud A est un nœud quelconque de la bibliothèque pouvant être connecté (définition 2.5) au nœud B.

```
node Equipement
  flow
    ...
  sub
    A : type_A ;
    B : Valve ;
    ...
  assert
    A.sortie = B.entree ;
  ...
edon
```

Figure 2.7 – Illustration du concept de hiérarchie au sein d'un nœud AltaRica

2.2.7.2 Synchronisation

Le concept de synchronisation permet à plusieurs évènements de se produire de manière simultanée. Une synchronisation concerne des évènements de nœuds différents mais ayant le même nœud père. Elle est décrite au niveau de ce nœud père par un vecteur de synchronisation défini grâce au mot clé *synch*. Il existe trois types possibles de synchronisation : la synchronisation « forte », la diffusion et la défaillance de cause commune.

- La synchronisation « forte » de deux évènements est possible si et seulement si *tous* les évènements de cette synchronisation sont tirables ; les évènements de la synchronisation « forte » ne sont pas tirables indépendamment les uns des autres.
- La diffusion, plus souple que la synchronisation, est possible si un des évènements (au moins) est tirable ; seuls les changements d'état provoqués par les évènements tirables se produisent ; les évènements de la diffusion ne sont pas tirables indépendamment les uns des autres.
- La défaillance de cause commune (DCC) est similaire à la diffusion à ceci près que les évènements de la DCC peuvent être tirés de manière indépendante.

Le tableau 2.2 reprend les caractéristiques principales de la synchronisation « forte », de la diffusion et de la défaillance de cause commune. La figure 2.8 donne des exemples de vecteur de synchronisation « forte », de diffusion et de défaillance de cause commune.

	Valeur des gardes des évènements	Évènements tirables indépendamment
Synchronisation « forte »	Toutes à $vrai$	Non
Diffusion	Au moins une à $vrai$	Non
DCC	Au moins une à $vrai$	Oui

Tableau 2.2 – Caractéristiques d'une synchronisation « forte », d'une diffusion et d'une défaillance de cause commune

Écriture en BNF <liste+ <construction> sep > désigne une liste contenant un ou plusieurs éléments de type « construction » séparés par « sep ».

```
<champ de synchronisation> ::= synch <liste* <vecteur> ; >
```

```
<vecteur> ::= <vecteur_synchronsation_forte>
              <vecteur_diffusion>
              <vecteur_defaillance_cause_commune>

<vecteur_synchronisation_forte> ::= '<' <liste <composante_de_vecteur> , > '>'

<vecteur_diffusion> ::= '<' <liste <composante_de_vecteur> | > '>'

<vecteur_defaillance_cause_commune> ::= '<' <liste <composante_de_vecteur> ? > '>'

<composante_de_vecteur> ::= <nom_noeud> . <evenement>

<nom_noeud> ::= <symbole>

<evenement> ::= <symbole>
```

Si *noeud_A* et *noeud_B* sont des sous-noeuds d'un noeud *Main*, si *E_1* (respectivement *E_2*) est un évènement du noeud *noeud_A* (respectivement *noeud_B*), alors :

event // Définition d'une synchronisation forte
synchronisation_forte ;
synch
$<$ *nom_synchronisation_forte , noeud_A.E_1 , noeud_B.E_2* $>$;

event // Définition d'une diffusion
diffusion ;
synch
$<$ *diffusion | noeud_A.E_1 | noeud_B.E_2* $>$;

event // Définition d'une défaillance de cause commune
dcc ;
synch
$<$ *dcc ? noeud_A.E_1 ? noeud_B.E_2* $>$;

Figure 2.8 – Définition d'une synchronisation « forte », d'une diffusion et d'une défaillance de cause commune

2.3 Sémantique du langage AltaRica : les automates de mode

La section 2.2 a donc présenté la syntaxe du langage AltaRica. Nous présentons ici sa sémantique fondée sur la notion d'automate de mode [55].

Un automate de mode est un automate à état fini dont on appelle « modes » certaines évaluations de ses variables d'état.

Exemple : Soit $V = [v_1, v_2, v_3]$ en vecteur d'état quelconque où $v_i \in \{0, 1\}, \forall i.\ V$ a 8 « évaluations » : $[0, 0, 0]$, $[0, 0, 1]$, $[0, 1, 0]$, $[1, 0, 0]$, $[0, 1, 1]$, $[1, 1, 0]$, $[1, 0, 1]$, $[1, 1, 1]$.

Parmi ces automates à état fini et pour les initiés, les automates de mode peuvent être considérés comme une généralisation des machines de Mealy [55] puisque les sorties dépendent à

la fois de l'état courant de l'automate et de ses entrées (en opposition aux machines de Moore dont les sorties ne peuvent dépendre que de l'état courant de l'automate). Nous donnons ici la définition formelle de ces automates de mode très fortement inspirée de [55].

2.3.1 Automate de mode

Nous nommerons dans la prochaine définition :

- D un ensemble fini (*i.e.* un domaine) de symboles que nous nommerons constantes ;
- $P(D)$ est l'ensemble des parties de D ;
- V un ensemble fini de symboles que nous nommerons variables ($D \cap V = \emptyset$) ;
- dom une application de V dans $P(D)$ définie telle que :
  - $\forall v \in V, dom(v) \neq \emptyset$,
  - $\forall U \subseteq V, dom(U) = \Pi_{v \in U} dom(v)$, *i.e.* $dom(U)$ est l'ensemble de toutes les possibles valeurs des variables de U.

Définition 2.1 : Automate de mode

Un *automate de mode* est un 9-uplet $(D, dom, S, F^{in}, F^{out}, \Sigma, \delta, \sigma, I)$ tel que :

- D et dom sont définis comme décrits ci-dessus ;
- S, F^{in} et F^{out} des sous-ensembles disjoints de V ; les variables de S, F^{in} et F^{out} sont nommées respectivement variables d'état, variables de flux d'entrée et variables de flux de sortie ;
- Σ est un ensemble fini de symboles appelés évènements ;
- $\delta : dom(S) \times dom(F^{in}) \times dom(\Sigma) \rightarrow dom(S)$ est une application partielle appelée transition ; elle fournit les prochaines valeurs des variables d'état en fonction de leur valeur courante, des valeurs des variables de flux d'entrée et de l'occurrence de l'évènement induisant le changement d'état ;
- $\sigma : dom(S) \times dom(F^{in}) \rightarrow dom(F^{out})$ est une fonction totale appelée assertion ; elle fournit les valeurs des variables de flux de sortie en fonction des valeurs des variables d'état et des valeurs des variables de flux d'entrée ;
- $I \subseteq dom(S)$ définit les valeurs initiales des variables d'état. ■

Remarque : Remarquons ici que l'état initial de l'automate de mode n'est pas forcément unique ($I \subseteq dom(S)$ et pas $I \in dom(S)$). On se restreindra cependant dans la suite à des automates avec variables d'état initialisées de manière unique comme utilisé dans Cecilia^{TM} OCAS.

Exemple : Dans cet exemple, nous modélisons sous forme d'automate de mode la valve hydraulique décrite sur la figure 2.3.1. L'automate de mode décrivant cette valve est le suivant :

- S=$\{etat, bloque\}$;
  - $dom(etat) = \{ouverte, fermee\}$, $dom(bloque) = \{true, false\}$;

- $F^{in} = \{entree\}$, $F^{out} = \{sortie\}$;
  - $dom(entree)$ =$dom(sortie)$=$\{ok, faible, nul\}$;
- $\Sigma = \{Ouverture, Fermeture, Blocage\}$;
- δ et σ sont définis comme sur la figure 2.1 et comme implémenté sur la figure 2.6 ;
- $I =$ (ouverte, false).

Définition 2.2 : Mode & Configuration

On appelle *mode* une évaluation des variables d'état de l'automate ; on appelle *configuration* l'association d'un mode et d'une évaluation des variables de flux d'entrée. ■

Ainsi, ces automates de mode sont composés de modes (ou de configurations) permettant la modélisation des états du système et de transitions représentant les changements entre ces modes. Dans le cas de systèmes industriels réels, le nombre de modes (ou de configurations) d'un tel automate peut rapidement devenir important (*e.g.* un million de modes pour un système de 20 composants dont le fonctionnement peut être considéré comme booléen) si l'on considère un système complexe global. Il n'est donc souvent pas envisageable de décrire grâce à un unique automate de mode un système complet. Pour surmonter cette difficulté de représentation, il est possible de décrire un système comme une hiérarchie de composants élémentaires : chacun des composants du système est modélisé sous la forme d'un automate de mode ; le système est ensuite obtenu en assemblant les différents automates de mode élémentaires. Cet assemblage est réalisé à l'aide d'opérations effectuées sur ces automates de mode : la composition parallèle d'automates, la connexion d'automate et la synchronisation d'automate.

- La *composition parallèle* d'automates consiste à regrouper plusieurs automates s'exécutant séparément en parallèle en un seul et permet, notamment, la modélisation de la notion système / sous-système ;
- La *connexion* d'automate, faisant suite à une opération de composition parallèle, consiste à relier certaines entrées et certaines sorties de l'automate ; la connexion permet la modélisation d'échanges de données ;
- La *synchronisation* d'automate, faisant elle aussi suite à une opération de composition parallèle, permet le déclenchement simultané de plusieurs évènements de l'automate ; ainsi plusieurs transitions peuvent être déclenchées simultanément.

2.3.2 Composition parallèle d'automates de mode

Comme annoncé ci-dessus, il est possible de regrouper plusieurs automates de mode en un seul. L'automate de mode résultant est alors aussi un automate de mode regroupant les définitions de tous les automates de mode regroupés. On dit que les automates appartiennent à un même niveau. La définition formelle de la composition parallèle d'automates est telle que :

Définition 2.3 : Composition parallèle d'automates

Soient A_1, A_2, ..., A_n, n automates de mode tels que $A_i = (D, dom, S_i, F_i^{in}, F_i^{out}, \Sigma_i, \delta_i, \sigma_i, I_i)$. On suppose que les ensembles de variables $V_i = (S_i \cup F_i^{in} \cup F_i^{out})$, $i \in [1, n]$ et Σ_i, $i \in [1, n]$) sont respectivement disjoints deux à deux.

La composition parallèle de n automates de mode A_1, A_2, ..., A_n est un automate de mode $A = (D, dom, S, F^{in}, F^{out}, \Sigma, \delta, \sigma, I)$ tel que :

- $S = \bigcup_{i=1}^{n} S_i$, $F^{in} = \bigcup_{i=1}^{n} F_i^{in}$, $F^{out} = \bigcup_{i=1}^{n} F_i^{out}$, $\Sigma = \prod_{i=1}^{n} \Sigma_i$;
- δ est une fonction de $\prod_i dom(S_i) \times \prod_i dom(F_i^{in}) \times \Sigma_i \rightarrow dom(S)$

$$\delta(S_1, S_2, ..., S_n, F_1^{in}, F_2^{in}, ..., F_n^{in}, e) = (S_1, ..., S_{i-1}, T_i, S_{i+1}, ..., S_n)$$

- σ est une fonction de $\prod_i dom(S_i) \times \prod_i dom(F_i^{in}) \rightarrow dom(F^{out})$

$$\sigma(S_1, S_2, ..., S_n, F_1^{in}, F_2^{in}, ..., F_n^{in}) = (\sigma_1(S_1, F_1^{in}), ..., \sigma_n(S_n, F_n^{in})) = (F_1^{out}, ..., F_n^{out})$$

- $I = \prod_i I_i$. ■

Remarque : D'après la définition de δ, seul l'automate A_i passe dans l'état T_i sous l'effet de l'évènement e ($e \in \Sigma_i$ et dans l'hypothèse où les ensembles Σ_i, $i \in [1, n]$ sont tous disjoints deux à deux).

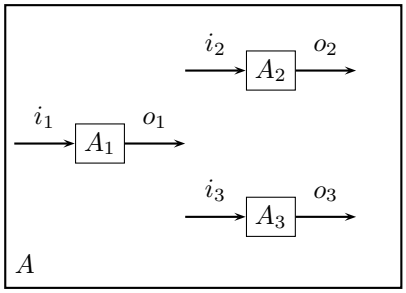


Figure 2.9 – Composition parallèle d'automates de mode

Sur la figure 2.9, l'automate A possède ainsi après l'opération de composition parallèle trois entrées ($F^{in} = \{i_1, i_2, i_3\}$) et trois sorties ($F^{out} = \{o_1, o_2, o_3\}$). La section suivante présente l'opération de connexion d'automate de mode visant à lier certaines de ses sorties et certaines de ses entrées.

2.3.3 Connexion d'automate de mode

2.3.3.1 Préambule

En pratique, un automate de mode permettra la description d'un composant du système. Un système réel étant constitué de plusieurs composants, il est nécessaire de pouvoir relier ces automates entre eux, *i.e.* de pouvoir relier les entrées et les sorties de plusieurs automates afin qu'il puisse échanger des informations. Afin de connecter entre eux deux automates de modes, ceux-ci doivent en premier lieu être regroupés en un unique automate de mode à l'aide d'une composition parallèle comme indiqué en section 2.3.2.

La figure 2.10 présente le principe de l'opération de connexion d'automate de mode. La connexion de A_1, A_2 et A_3 peut être considérée en deux étapes :

- la composition parallèle des automates A_1, A_2 et A_3 $\rightarrow$ on obtient un nouvel automate de mode A (figure 2.9) où $F_A^{in} = \{i_1, i_2, i_3\}$ et $F_A^{out} = \{o_1, o_2, o_3\}$;
- la connexion de l'automate A où on imposera $i_2 = o_1$ et $i_3 = o_1$.

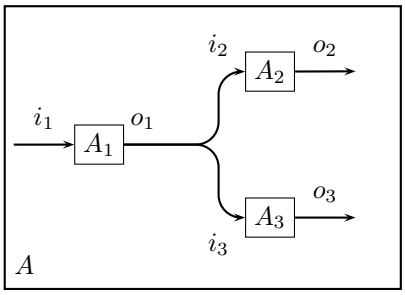


Figure 2.10 – Connexion d'automate de mode

Remarque : Toute opération de connexion d'automate de mode est précédée d'une composition parallèle d'automates de mode. Dans la suite de cette section 2.3.3, nous ne considèrerons qu'un unique automate de mode A au sein duquel on liera certaines de ses entrées à certaines de ses sorties.

2.3.3.2 Définitions

Définition 2.4 : Indépendance entre variables de flux

Pour être valide, l'opération de connexion ne doit introduire aucune boucle. Ainsi, sur la figure 2.10 où i_2 et i_3 dépendent de o_1, cette sortie o_1 ne doit dépendre ni de i_2, ni de i_3.

Notation : Avant d'expliciter cette notion d'indépendance, introduisons quelques notations :

- $\hat{F}^{in} \in dom(F^{in})$ est une évaluation des variables d'entrée d'un automate de mode A ;
- $\hat{F}^{in}[v]$ la valeur de la variable d'entrée v dans l'évaluation $\hat{F}^{in}$ de F^{in} ;
- $\hat{F}^{in}[v \leftarrow c]$ pour $c \in dom(v)$ l'évaluation $\hat{F}^{in}[v]$ où v vaut c et où les autres variables de $\hat{F}^{in}$ restent inchangées.

Remarque : Ici, les exemples sont donnés avec $\hat{F}^{in}$ mais restent valables avec d'autres variables d'un automate de mode A. Ainsi et par exemple, $\hat{S}$ désignera une évaluation des variables d'état d'un automate de mode A...

Exemple : Prenons un automate de mode A tel que $F^{in} = (e_1, e_2, e_3)$ et $dom(e_i) = \{ok, failed\}$ $\forall i \in \{1, 2, 3\}$

- Une évaluation $\hat{F}^{in}$ est par exemple $\hat{F}^{in} = (ok, ok, failed)$.
- L'évaluation $\hat{F}^{in}[e_1]$ est ok. On a aussi $\hat{F}^{in}[e_2] = ok$ et $\hat{F}^{in}[e_3] = failed$

– L'évaluation $\hat{F}^{in}[e_2 \leftarrow failed]$ est $\hat{F}^{in}[e_2 \leftarrow failed] = (ok, failed, failed)$.

En notant $A = (D, dom, S, F^{in}, F^{out}, \Sigma, \delta, \sigma, I)$ un automate de mode, $i \in F^{in}$ et $o \in F^{out}$, o est dite indépendante de i, noté $i \perp 0$, lorsque :

$\forall \hat{S} \in dom(S), \forall \hat{F} \in dom(F^{in}), \forall c \in dom(i), \sigma(\hat{S}, \hat{F})[o] = \sigma(\hat{S}, \hat{F}[i \leftarrow c])[o]$

En termes plus informels, o est dite indépendante de i si la valeur de o ne varie pas quelle que soit la valeur de i et cela pour toutes les évaluations des variables d'état et d'entrée. L'indépendance entre deux variables de flux est cependant, en appliquant cette définition, couteuse à démontrer. Pour s'affranchir de cette difficulté, il est suffisant (mais pas nécessaire) de s'assurer que ni i_2, ni i_3 n'interviennent dans le calcul de o_1 (il n'est par exemple pas impossible de connecter o_2 et i_1). ■

En complément de l'indépendance entre variables, il est également indispensable que le domaine de la variable de sortie soit inclus dans celui des variables d'entrée avec lesquelles elle est connectée.

Définition 2.5 : Connexion d'automate de mode

Donnons maintenant une définition formelle de la connexion d'automate de mode. Nous rappelons qu'une opération de composition parallèle d'automates de mode précède toute opération de connexion d'automate de mode ; le principe de la connexion est alors, à l'intérieur d'un automate de mode, de contraindre des variables d'entrée à être égales à des variables de sortie (afin d'assurer l'absence de définition circulaire dans l'automate résultant de la connexion, on devra s'assurer lors d'une telle opération que les variables reliées sont indépendantes [Définition 2.4]).

Soient un automate de mode $A = (D, dom, S, F^{in}, F^{out}, \Sigma, \delta, \sigma, I)$, $o \in F^{out}$ et f_1, ..., $f_k \in F^{in}$ telles que $\forall i \in [1, k]$, $o \perp f_i$ et $dom(o) \subseteq dom(f_i)$.

Soient $F^{in}_\varsigma = F^{in} - \{f_i\}_{i=1,..,k}$, $\hat{S} \in dom(S)$ et $\hat{F} \in dom(F^{in}_\varsigma)$.

Notons $\hat{F}_{\hat{S},o/f_1,...,o/f_k}[v] = \begin{cases} \hat{F}[v] & \text{si } v \in F^{in}_\varsigma \\ \sigma(\hat{S}, \hat{F}')[o] & \text{sinon} \end{cases}$

où $\hat{F}' \in dom(F^{in})$ est une extension quelconque de $\hat{F}$ ($F^{in}_\varsigma \subset F^{in}$).

L'automate de mode dans lequel la sortie o est connectée aux entrées f_1, ..., f_k est noté $A_{o/f_1,...,o/f_k} = (D, dom, S, F^{in}_\varsigma, F^{out}_\varsigma, \Sigma, \delta', \sigma', I)$ où F^{out}_ς, δ' et σ' sont définis par :

– $F^{out}_\varsigma = F^{out} - \{o\}$;

– $\forall \hat{S} \in dom(S), \forall \hat{F} \in dom(F^{in}_\varsigma), \forall e \in \Sigma$, si $\delta(\hat{S}, \hat{F}_{\hat{S},o/f_1,...,o/f_k}, e)$ est défini, alors

$$\delta'(\hat{S}, \hat{F}, e) = \delta(\hat{S}, \hat{F}_{\hat{S},o/f_1,...,o/f_k}, e) ;$$

– $\forall \hat{S} \in dom(S), \forall \hat{F} \in dom(F^{in}_\varsigma), \sigma'(\hat{S}, \hat{F}) = \sigma(\hat{S}, \hat{F}_{\hat{S},o/f_1,...,o/f_k})$.

En pratique, cela revient à substituer les valeurs des entrées connectées f_1,..., f_k par la valeur de la sortie o pour tous les jeux d'évaluation des états et des entrées. ■

Exemple : Sur la figure 2.10, l'opération de connexion consistera à :

- vérifier l'indépendance des variables à connecter (définition 2.4), *i.e.* vérifier que $i_2 \perp o_1$ et que $i_3 \perp o_1$;
- imposer que les variables d'entrée i_2 et de i_3 de A soient égales à la variable de sortie o_1 de A (définition 2.5), *i.e.* que $i_2 = o_1$ et que $i_3 = o_1$.

2.3.4 Synchronisation d'automate de mode

Remarque : En référence à la section 2.2.7, il s'agit ici de l'opération de synchronisation « forte ». On ne traite donc pas les opérations de type diffusion ou défaillance de cause commune.

Les automates de mode, fondant leurs évolutions sur l'occurrence d'évènement plutôt que sur l'écoulement du temps, sont asynchrones. Ainsi, deux transitions ne peuvent pas, en principe, avoir lieu de manière simultanée. La synchronisation permet de s'affranchir de cette limite. Préalablement à la définition d'une synchronisation, définissons quelques notions préliminaires.

Définition 2.6 : Compatibilité de variables

Soient V un ensemble de variables (*e.g.* variables d'état, d'entrée ou de sortie) et $\hat{V}, \hat{V}_\phi, \hat{V}_\psi \in dom(V)$ trois évaluations de V. $\hat{V}[v]$ désigne la valeur de la variable v dans l'évaluation $\hat{V}$ de V. On dit que $\hat{V}_\phi$ et $\hat{V}_\psi$ sont incompatibles par rapport à $\hat{V}$ si $\exists v \in V$ tel que $\hat{V}[v]$, $\hat{V}_\phi[v]$, $\hat{V}_\psi[v]$ soient tous distincts.

Si $\hat{V}_\phi$ et $\hat{V}_\psi$ ne sont pas incompatibles, alors elles sont dites compatibles par rapport à $\hat{V}$. En d'autres termes, $\hat{V}_\phi$ et $\hat{V}_\psi$ seront dites compatibles par rapport à $\hat{V}$ si, pour chacune des composantes v de V, il existe au moins deux valeurs identiques parmi $\hat{V}[v]$, $\hat{V}_\phi[v]$ et $\hat{V}_\psi[v]$. ■

Exemple : Soit $V = \{v_1, v_2, v_3\}$ et $dom(v_i) = \{a, b, c\}$ $\forall i \in [1, 3]$. Alors par exemple :

- $\hat{V}_\phi = (b, b, c)$ et $\hat{V}_\psi = (c, b, c)$ sont incompatibles par rapport à $\hat{V} = (a, b, c)$;
- $\hat{V}_\phi = (b, b, c)$ et $\hat{V}_\psi = (a, a, c)$ sont compatibles par rapport à $\hat{V} = (a, b, a)$.

Définition 2.7 : Composition de variables

Remarque : On ne confondra pas la notion de composition présentée ici avec celle de « composition parallèle » relative à des automates de mode et présentée section 2.3.2.

Soient V un ensemble de variables (*e.g.* variables d'état, d'entrée ou de sortie) et $\hat{V}$, $\hat{V}_\phi$ et $\hat{V}_\psi \in dom(V)$ trois évaluations de V. La composition de $\hat{V}_\phi$ et $\hat{V}_\psi$ par rapport à $\hat{V}$ est une évaluation de V définie telle que :

$$\forall v \in V,\ (\hat{V}_\phi \circ_{\hat{V}} \hat{V}_\psi)[v] = \begin{cases} \hat{V}_\phi[v] & \text{si } \hat{V}_\phi[v] \neq \hat{V}[v] \\ \hat{V}_\psi[v] & \text{sinon} \end{cases}$$

Composer $\hat{V}_\phi[v]$ et $\hat{V}_\psi[v]$ par rapport à $\hat{V}$ revient donc à créer une quatrième évaluation de V dont les composantes v sont :

- Différentes de $\hat{V}[v]$ si au moins une parmi $\hat{V}_\phi[v]$ et $\hat{V}_\psi[v]$ l'est ;
- Égale à $\hat{V}[v]$ dans le cas ou $\hat{V}[v] = \hat{V}_\phi[v] = \hat{V}_\psi[v]$.

■

Exemple : Reprenons l'exemple précédent avec $\hat{V} = (a, b, a)$, $\hat{V}_\phi = (b, b, c)$ et $\hat{V}_\psi = (a, a, c)$. Alors :

- $\hat{V}[s_1] = a$, $\hat{V}_\phi[s_1] = b$ soit $(\hat{V}_\phi \circ_{\hat{V}} \hat{V}_\psi)[s_1] = \hat{V}_\phi[s_1] = b$;
- $\hat{V}[s_2] = b$, $\hat{V}_\phi[s_2] = b$ soit $(\hat{V}_\phi \circ_{\hat{V}} \hat{V}_\psi)[s_2] = \hat{V}_\psi[s_2] = a$;
- $\hat{V}[s_3] = a$, $\hat{V}_\phi[s_3] = c$ soit $(\hat{V}_\phi \circ_{\hat{V}} \hat{V}_\psi)[s_3] = \hat{V}_\phi[s_3] = c$.

Il en résulte que : $(\hat{V}_\phi \circ_{\hat{V}} \hat{V}_\psi) = (b, a, c)$

Une fois ces deux notions de compatibilité et de composition présentées, la définition formelle de la synchronisation peut être donnée. Auparavant, on précise qu'une telle étape de synchronisation permet l'exécution simultanée de deux évènements d'un même automate de mode. Si l'on souhaite synchroniser deux évènements de deux automates de mode différents, la synchronisation sera précédée d'une composition parallèle de ces automates de mode.

Définition 2.8 : Synchronisation d'automate de mode

Soit un automate de mode $A = (D, dom, S, F^{in}, F^{out}, \Sigma, \delta, \sigma, I)$ et y_1, ..., y_k des vecteurs de synchronisation. On appelle Σ' l'ensemble des évènements présents dans au moins un vecteur de synchronisation. L'automate de mode résultant de la synchronisation de A par y_1, ..., y_k est tel que :

$$A_{|y_1,...,y_k} = (D, dom, S, F^{in}, F^{out}, (\Sigma - \Sigma') \cup \{y_1, ..., y_k\}, \delta', \sigma, I)$$

où δ' est défini de la façon suivante :

- $\forall e \in (\Sigma - \Sigma')$, $\forall \hat{S} \in dom(S)$, $\forall \hat{F} \in dom(F^{in})$, si la transition $\delta(\hat{S}, \hat{F}, e)$ est définie, alors $\delta'(\hat{S}, \hat{F}, e)$ est défini par $\delta'(\hat{S}, \hat{F}, e) = \delta(\hat{S}, \hat{F}, e)$;
- $\forall y_i = (e_1, ..., e_n)$, $\forall \hat{S} \in dom(S)$, $\forall \hat{F} \in dom(F^{in})$, si les $\delta(\hat{S}, \hat{F}, e_1)$, ..., $\delta(\hat{S}, \hat{F}, e_n)$ sont définis et compatibles deux à deux, alors $\delta'(\hat{S}, \hat{F}, y_i)$ est défini par $\delta'(\hat{S}, \hat{F}, y_i) = \delta(\hat{S}, \hat{F}, e_1) \circ_{\hat{S}} ... \circ_{\hat{S}} \delta(\hat{S}, \hat{F}, e_n)$ où $\circ_{\hat{S}}$ représente l'opération de composition de variables (définition 2.7) par rapport à $\hat{S}$.

La première étape consiste donc à insérer dans l'automate synchronisé $A_{|y_1,...,y_k}$ l'ensemble des transitions de A non influencées par un vecteur de synchronisation (du point de vue opposé, on supprime de A les transitions relatives aux évènements synchronisés). On crée ensuite de nouvelles transitions représentant les synchronisations.

■

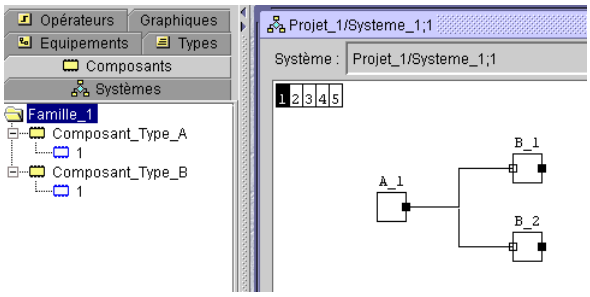


Figure 2.11 – Exemple de bibliothèque et d'instanciation de composants sous Cecilia^{TM} OCAS

2.4 Outils pour la construction et l'analyse de modèles AltaRica

Dans cette section, nous présentons différents outils supportant la construction et / ou l'analyse d'un modèle AltaRica. Seront cités particulièrement deux outils développés par Dassault Aviation : l'outil Cecilia^{TM} OCAS (Outil de Conception et l'Analyse Système) qui supporte la construction et l'analyse qualitative d'un modèle AltaRica ainsi que l'outil Cecilia^{TM} ARBOR[2] de construction d'arbres de défaillance qui nous permettra, en tant que complément de Cecilia^{TM} OCAS, de réaliser des analyses quantitatives à partir d'arbres extraits de modèles AltaRica.

2.4.1 Modélisation d'un système

Avant de pouvoir analyser le modèle AltaRica d'un système, il nous est tout d'abord nécessaire de construire ce modèle... ! D'une façon générale, construire un modèle consiste à traduire le comportement d'un système dans un langage choisi, en l'occurrence et dans notre cas, le langage AltaRica. Nous reviendrons plus en détail sur cette étape de modélisation en langage AltaRica dans un prochain chapitre.

Ici, dans l'optique de construire un modèle AltaRica, nous utiliserons l'outil Cecilia^{TM} OCAS supportant, entre autres, l'édition d'un modèle AltaRica. La philosophie de modélisation de cet outil suit alors la philosophie de modélisation générale du langage AltaRica :

- modélisation de chacun des composants du système ; création d'une bibliothèque de composants AltaRica ;
- instanciation graphique de composants de la librairie (les composants sont sélectionnés dans la bibliothèque puis placés dans une fenêtre graphique par simple « drag & drop ») ;
- connexion des composants : les composants instanciés sont reliés directement dans la fenêtre graphique.

2.4.2 Vérification de la cohérence des modèles AltaRica

Comme aide à la modélisation et à l'écriture de code AltaRica, l'outil Cecilia^{TM} OCAS effectue des tests pour détecter des incohérences syntaxiques ou structurelles dans des modèles AltaRica syntaxiquement correct. Par exemple, il sera vérifié :

2. Aussi de propriété Dassault Aviation, l'outil Cecilia^{TM} ARBOR permet l'édition et le calcul d'arbres de défaillance.

- les mots clés et les opérateurs utilisés dans les transitions et les assertions ;
- que toute variable utilisée ait auparavant bien été déclarée ;
- le fait que tout évènement soit relié à au moins une transition ;
- le fait que toute variable de flux de sortie soit définie par une assertion ;
- l'absence de boucles instantanées (définition circulaire d'une variable de flux de sortie : au même instant logique, la sortie S dépends de l'entrée E qui dépends elle-même, directement ou non, de S) dans le modèle...

Notons que ces tests fournissent des renseignements et nous assurent de la forme syntaxiquement correcte du modèle mais ne s'intéresse pas ou peu à la sémantique, *i.e.* au sens du modèle. Ce problème étant pourtant selon nous un problème capital, il sera traité dans un prochain chapitre.

L'outil Cecilia^{TM} OCAS possède également un simulateur interactif permettant de « jouer » les différents évènements du modèle et d'observer l'évolution de celui-ci. Ainsi, il est possible de comparer si, lors de la simulation d'évènement, le comportement du modèle est conforme au résultat attendu. À noter que cette vérification est rendue relativement aisée par l'outil puisque chaque variable d'état et chaque variable de flux est complètement et totalement observable, *i.e.* à tout instant, il est possible de visionner la valeur d'une variable d'état ou de flux. Grâce à cette possibilité, nous pourrons vérifier après l'occurrence d'un évènement le nouvel état ainsi que la sortie d'un composant donné.

2.4.3 Analyse des modèles AltaRica

Précisons d'ores et déjà que différentes méthodes d'analyses de modèles AltaRica sont possibles : génération d'arbre de défaillance, simulation de Monte Carlo, analyse par chaîne de Markov, model-checking ou encore génération de séquences conduisant le système dans une configuration redoutée. Toutes ne sont cependant pas citées ci-dessous.

2.4.3.1 Arbre de défaillance / Génération de séquences

L'outil Cecilia^{TM} OCAS permet le support des analyses qualitatives et quantitatives (l'outil Cecilia^{TM} ARBOR est alors utilisé en complément) d'un modèle AltaRica. En ce qui concerne les analyses qualitatives, deux philosophies différentes peuvent être utilisées : la génération d'arbres de défaillance ou la recherche de scénarios conduisant le modèle dans une configuration donnée.

Le générateur d'arbres de défaillance produit un arbre décrivant l'ensemble des combinaisons d'évènements conduisant le modèle dans une configuration donnée. L'arbre obtenu l'est dans un format exportable et lisible par Cecilia^{TM} ARBOR. Ainsi, des analyses quantitatives peuvent être réalisées. Cependant, l'algorithme de génération de ces arbres a été prouvé sûr pour un fragment du langage AltaRica. En effet, la génération d'un de ces arbres pour, par exemple, un modèle d'un système reconfigurable n'est pas chose triviale. L'utilisation de transitions instantanées (loi de probabilité de l'évènement de la transition en Dirac(0) - section 2.2.3) ou de variables de flux d'entrée dans les gardes des transitions pose des problèmes lors de la génération de l'arbre. Des arbres incorrects ou des erreurs lors de la génération peuvent être obtenus dans de tels cas.

Un modèle représentant un système réel possédant couramment ce genre d'artifice, Cecilia^{TM} OCAS possède un générateur de séquences qui explore l'espace d'état du modèle à la recherche de séquences (de longueur bornée) conduisant le système dans la configuration étudiée. Ce générateur de séquence fournit donc à l'utilisateur l'ensemble des combinaisons d'évènements qui conduisent le système dans un état donné. La taille maximum de ces combinaisons (*i.e.* l'ordre maximum des séquences recherchées) doit être précisée par l'utilisateur. De plus et contrairement au générateur

d'arbre de défaillance, ce générateur de séquence autorise que l'ordre des évènements ait une influence sur l'état final du système. Suivant le résultat attendu, il est possible d'obtenir les coupes, minimales ou non, ainsi que les séquences, minimales ou non. Contrairement aux séquences, les coupes ne prennent pas en compte l'ordre des évènements.

2.4.3.2 Model-Checking

Sans rentrer dans une explication détaillée de ce qu'est le model-checking, précisons ici qu'il s'agit d'une méthode ayant pour objectif de vérifier formellement un modèle. Elle permet d'évaluer si telle ou telle propriété attendue d'un système est réellement satisfaite par le modèle du système. Couramment, ces propriétés sont exprimées en logique temporelle linéaire (LTL) ou arborescente (CTL) ; le choix dépendant du model checker (outil supportant l'activité de model-checking) utilisé. Une fois la propriété exprimée dans le formalisme adapté, sa vérification est faite automatiquement et de façon exhaustive par le model checker. Dans le cas où la propriété exprimée n'est pas valide, le model checker nous renvoie un contre-exemple (*i.e.* un scénario) violant la propriété.

Concernant AltaRica, il existe différents moyens de faire du model-checking. Par exemple, MecV [66] effectue du model-checking sur un modèle écrit en langage AltaRica originel. Concernant les langages AltaRica Data-Flow et AltaRica OCAS, il est possible d'utiliser le model checker SMV [13]. L'approche se fait alors en deux étapes. Le modèle AltaRica est dans un premier temps traduit en SMV [3]. Ensuite, les propriétés (ici des propriétés LTL) sont également écrites en SMV. Si le modèle AltaRica ne satisfait pas la propriété, un contre-exemple est renvoyé par SMV. Dans [37], un exemple de model-checking a été traité sur un système AltaRica à échelle industrielle.

2.5 Autres langages pour la réalisation d'étude de Sûreté de Fonctionnement

Le présent chapitre a donc été l'occasion de présenter le langage AltaRica et son extension, le langage AltaRica OCAS. Bien que ce langage ce veuille supporter la réalisation d'analyse de sécurité de type arbres de défaillance, il n'est pas le seul à posséder cette prétention. Parmi les différentes alternatives disponibles dans la littérature, nous pouvons distinguer deux types de travaux :

- les langages dédiés à la réalisation d'analyse de SdF ;
- d'autres langages dotés d'extension pour les études de SdF.

Nous présentons dans cette section plusieurs de ces travaux.

2.5.1 Langages étendus pour la Sûreté de Fonctionnement

2.5.1.1 AADL

La conception de systèmes demeurant une tâche complexe, des approches utilisant des langages de description d'architecture ont vu le jour et ont pour but de maitriser cette complexité. En particulier, le langage AADL (Architecture Analysis and Design Language) a été développé pour faciliter la conception et l'analyse de systèmes informatiques complexes critiques temps réel dans les domaines de l'aéronautique, de l'automobile ou encore du spatial. AADL [58] est utilisé pour décrire des architectures matérielles ou logicielles et possède pour cela une notation (graphique et

3. Pour supporter cette activité, un outil de traduction, alta2smv, permet de passer d'un modèle AltaRica à un modèle SMV.

textuelle) standardisée. Il permet de décrire les interfaces entre composants de l'architecture (*e.g.* les entrées et les sorties de chacun des composants), ainsi que le comportement de ces composants et du système modélisé. Enfin, ici, le langage AADL est pensé pour être extensible afin d'autoriser des types d'analyses que le langage « de base » ne permet pas.

Concernant notre problématique, le « AADL Error Model Annex », par exemple, autorise la création, au sein d'une bibliothèque, de modèles d'erreur. Ces modèles permettent la description de caractéristiques classiques de la Sûreté de Fonctionnement comme les modes de défaillance, les potentielles réparations, les propagations de pannes... Ces modèles d'erreur sont alors instanciés et associés aux autres composants déjà présents dans le modèle AADL de l'architecture. Obtenu de la sorte, un modèle AADL permet alors de décrire le comportement dysfonctionnel d'un système.

Les modèles AADL sont ensuite transformés en d'autres modèles permettant des analyses spécifiques. Par exemple, [57] propose de transformer un modèle AADL en un Réseau de Petri (RdP) et d'utiliser ensuite ces RdP pour réaliser les analyses de Sûreté de Fonctionnement. Le principe général de l'approche étant de profiter des outils disponibles sur les RdP tout en masquant leurs complexités. Ainsi, l'utilisateur maitrisant AADL et ayant des connaissances limitées sur les RdP pourra réaliser les analyses. Dans une problématique proche, [26] propose de transformer un modèle AADL en un modèle AltaRica. L'avantage de la transformation est double : le modèle AltaRica est obtenu en parcourant le modèle AADL une unique fois, alors que plusieurs itérations sont nécessaires dans le travail proposé par [57] ; l'utilisation du langage AltaRica permet la génération d'arbre de défaillance (ce que ne permet pas en standard un réseau de Petri).

2.5.1.2 East-ADL2

EAST-ADL2 [21], d'une philosophie issue de celle d'UML [4], est un langage à l'origine développé pour la modélisation et la conception de systèmes électroniques, en particulier dans le domaine automobile. Le langage, structuré en « couches » permet une description hiérarchique du système, du niveau « véhicule » au niveau matériel. Comme AADL, EAST-ADL2 permet de représenter les composants du système ainsi que les interfaces présentes entre ces composants. En tant qu'héritier d'UML, si un modèle EAST-ADL2 représente une large quantité d'informations, son appréhension nécessite de connaître le formalisme associé au langage.

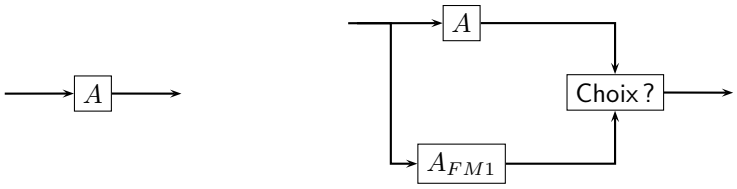
D'un point de vue dysfonctionnel, les modèles écrits en EAST-ADL2 permettent de modéliser les défaillances d'un système, *i.e.* les défaillances d'un composant et les propagations de panne ayant lieu dans le système ensuite. Une particularité de la méthode vient du fait que la construction de ces modèles dysfonctionnels est clairement séparée de celle des modèles fonctionnels nominaux : les modèles dysfonctionnels sont alors plutôt à caractère informatif ; ceux fonctionnels plus voués à la génération automatique de code.

2.5.1.3 Nu-SMV

Le langage Nu-SMV [22], fondé sur les machines à états finis, permet la réalisation de preuve sur des formules de logique temporelle. Il est utilisé pour décrire différents types de systèmes, aussi bien synchrone qu'asynchrone, à différents niveaux de détails. De manière similaire au langage AltaRica, il permet la description de systèmes hiérarchiques et encourage la réutilisation de composants déjà modélisés (*i.e.* encourage la réutilisation de modèle). Également à mettre en analogie avec AltaRica, Nu-SMV ne permet pas la description de données continues et prend en compte des données booléennes ou énumérées.

Dans [18, 19], la description d'un système et de son comportement peut être vue en deux temps. La description du comportement nominal puis la description du comportement dysfonc-

4. Crée à la fin des années 1990, UML est destiné à la modélisation orientée objet de systèmes logiciels. Il a inspiré de nombreux autres langages parmi lesquels SysML (System Modelling Langage) pour l'ingénierie de systèmes ou EAST-ADL puis EAST-ADL2 pour le développement de systèmes électroniques automobiles.



(a) (b)

Figure 2.12 – Principe de modélisation en Nu-SMV
(a) Modèle nominal ; (b) Modèle nominal + dysfonctionnel

tionnel. Pour cela et comme se propose de le montrer la figure 2.12, on ajoute au modèle nominal d'un composant autant de variables que de modes de défaillance (Ici A_{FM1}). Une logique (correspondant à la boîte « Choix ? ») permet ensuite de « choisir » la sortie à appliquer selon qu'un mode de défaillance ait eu lieu ou non. Une chose intéressante à noter est alors que le même cadre de modélisation est utilisé pour le modèle nominal et le modèle dysfonctionnel.

Sur ce type de modèle, les propriétés souhaitant être étudiées sont décrites en logique temporelle LTL ou CTL (LTL : Linear Temporal Logic ; CTL : Computation Tree Logic, *i.e.* logique temporelle respectivement linéaire ou arborescente). Sans décrire de telles logiques ici, les propriétés peuvent être vues comme des expressions booléennes simples auxquelles on adjoint divers opérateurs temporels. Par exemple, en LTL, la formule $Always\ (a\&b)$ sera vraie si $a\&b$ est vraie pour tous les chemins. De telles propriétés sont en pratique utilisées pour vérifier que le modèle satisfait des exigences fournies par une spécification. Des outils (de simulation et de model-checking) sont disponibles pour supporter ces vérifications. Il est également possible de générer l'arbre de défaillance correspondant à un évènement redouté.

2.5.1.4 SCADE

Le langage SCADE est une notation graphique formelle fondée sur le langage formel LUSTRE [31] initialement crée pour spécifier des logiciels réactifs[5] (ou réflexes). C'est donc un langage formel, synchrone à flot de données. Il est supporté par un environnement muni d'un éditeur graphique, d'un moteur de simulation, d'un générateur de code et d'un model-checker permettant de vérifier, sur un modèle écrit en SCADE, des propriétés.

Dans [1] et comme pour Nu-SMV, la description d'un système et de son comportement peut être vu en deux temps : une modélisation nominale et une dysfonctionnelle (le cadre de modélisation est donc là aussi identique pour le comportement nominal et celui dysfonctionnel). Les propriétés sont ici également à exprimer en langage SCADE : on peut ainsi observer directement sur le modèle SCADE si ces propriétés sont satisfaites ou non.

Cette vérification de propriété permet également de générer un ensemble de coupes minimales. Pour cela, il est nécessaire de définir un ou plusieurs observateurs directement sur le modèle sous forme de propriété. Pour mieux comprendre, aidons nous d'un exemple. Considérons un composant A quelconque dont la sortie S_A prend ses valeurs dans $\{ok,\ degrade,\ ko\}$ et imaginons que l'on souhaite obtenir les séquences conduisant à obtenir $S_A = ko$. Dans un tel cas, la première étape consiste à définir un observateur qui fournira la valeur vrai si $S_A \neq ko$ (et donc faux si $S_A = ko$). Ensuite, en demandant au model-checker si la propriété est vraie (si la sortie de l'observateur est vraie), celui ci va nous fournir un contre-exemple violant la propriété (s'il en existe).

5. Un système réactif est un système qui répond constamment aux sollicitations de son environnement en produisant des actions sur celui-ci.

Ce contre-exemple est alors en fait une séquence conduisant le système dans la configuration non souhaitée.

2.5.2 Langages dédiés Sûreté de Fonctionnement

2.5.2.1 Le langage Figaro

Le langage Figaro [16] a été développé par EDF (Électricité De France) dans les années 1990 pour modéliser la propagation des défaillances au sein d'un système. L'article [17] compare les langages AltaRica et Figaro et explicite le fait que, s'il est entendu que les objectifs des deux langages sont similaires, ils différent profondément dans leurs philosophies.

Tout d'abord, Figaro est un langage orienté objet : une base de connaissance décrit des classes de composant ; chacune de ces classes inclut le comportement nominal et dysfonctionnel du composant qu'elle représente ; le système global est ensuite construit par instanciation de composant. À noter que les composants ne sont pas organisés hiérarchiquement, ni encapsulés (il n'y a pas d'emboîtement entre objets) contrairement au langage AltaRica qui, comme nous l'avons vu, est fondé sur la notion de compositionnalité : des objets sont définis puis instanciés pour créer des objets de niveau supérieur (le plus haut niveau représentant alors le modèle global).

Figaro est supporté par l'outil KB3, aussi développé par EDF. KB3 contient un éditeur graphique de modèles ainsi que différents algorithmes permettant d'évaluer quantitativement et qualitativement le modèle réalisé. À ce sujet, notons une différence entre Figaro et AltaRica. La simulation (et donc la sémantique) de modèles Figaro est réalisée en intriquant des caractéristiques qualitatives (gardes des transitions) et quantitatives (loi suivie par la probabilité d'occurrence d'un évènement) des évènements. En AltaRica, la simulation dépend des gardes des transitions mais les relations entre évènements sont définies en terme de priorités. Les lois peuvent être ajoutées ensuite pour par exemple réaliser des simulations stochastiques. En résumé, l'approche Figaro consiste à écrire des modèles quantitatifs à partir desquels on réalise ensuite des analyses qualitatives ; l'approche AltaRica consiste à écrire des modèles qualitatifs pouvant être complétés par des analyses quantitatives.

Enfin, une dernière différence entre AltaRica et Figaro provient de leurs utilisations. La syntaxe de Figaro permet d'écrire des expressions plus proches du langage naturel. Ainsi et en accord avec [17], le langage Figaro est plus adapté à qui veut concevoir des outils de modélisation « clef en main » (une fois la base de connaissance créée, la réalisation de modèle n'imposera l'écriture d'aucune ligne de code Figaro). Le langage AltaRica quant à lui possède une syntaxe plus formelle et moins accessible aux non-initiés. Cependant, une fois cet « obstacle » franchi, le langage AltaRica est plus facilement utilisable pour les ingénieurs étant de manière directe en charge des études de systèmes.

2.5.2.2 HIP-HOPS

HiP-HOPS (Hierarchically Performed Hazard Origin and Propagation Studies) [49] est une méthode se proposant de supporter les analyses de Sûreté de Fonctionnement en autorisant la génération automatique d'arbres de défaillance ou d'AMDE. Pour cela, un modèle hiérarchique du système est construit à partir d'une décomposition de ce système sous forme de schémas blocs successifs. Les modes de défaillance de chaque « bloc » identifiés sont étudiés et introduit dans un tableau décrivant, pour chaque composant, l'effet système de ces modes de défaillance. Un algorithme est ensuite appliqué pour générer l'arbre de défaillance correspondant à un évènement redouté et calculer ses coupes minimales. HIP-HOPS est supporté par un outil réalisant des analyses qualitatives et quantitatives de systèmes.

Le travail présenté dans [14] propose une approche permettant d'insérer la méthode HiP-HOPS dans un processus de développement de systèmes automobiles fondé sur EAST-ADL2.

Pour ce faire, les auteurs proposent d'utiliser différentes transformations de modèles permettant de traduire les informations pertinentes présentes dans les documents de conception (modèle EAST-ADL2) en informations pertinentes pour les analyses de Sûreté de Fonctionnement (en modèle HIP-HOPS).

2.6 Conclusion du chapitre : Pourquoi AltaRica ?

Pour rappel, nous nous intéressons au support et à la réalisation d'analyse de sécurité de type arbre de défaillance. D'après la section précédente, ces analyses peuvent avoir deux philosophies :

- soit la modélisation est réalisée dans une logique d'injection des défaillances au sein d'un modèle de conception (*e.g.* de type Matlab Simulink ou SCADE) déjà existant ;
- soit cette modélisation est réalisée en prenant en compte et en modélisant directement la logique de propagation de défaillance au sein du système considéré.

Concernant la première philosophie tout d'abord, ce type de modélisation est utilisé dans la littérature principalement pour la représentation de systèmes informatiques, électriques ou électroniques là où nous nous intéressons à la représentation de systèmes mécaniques ou hydromécaniques. De plus et pour de tels systèmes, nous n'avions pas au début des travaux et n'avons pas plus aujourd'hui à disposition des modèles de conception.

Nous nous sommes ainsi intéressé pour répondre à notre problématique à l'utilisation de modèles de propagation de défaillance. Le langage AltaRica s'inscrit alors dans la seconde philosophie. Par rapport aux langages Figaro et HiP-HOPS, le langage AltaRica est doté d'une sémantique formelle permettant à la fois d'obtenir du langage AltaRica à partir d'autres formalismes (*e.g.* SysML [23]) et de générer à partir d'AltaRica des modèles du type arbres de défaillance ou analyse de Markov.

Enfin, les travaux et expériences passés [33, 9, 12, 20, 38, 37] ont déjà montré la pertinence de l'utilisation du langage AltaRica pour réaliser l'étude de sécurité d'un système aéronautique. Nous nous inscrivons ainsi directement dans leur continuité et approfondissons l'approche à l'étude de systèmes multi-physiques.

Chapitre 3

Vers une méthodologie unifiée de modélisation AltaRica de systèmes physiques

Sommaire

3.1 Introduction

3.1.1 Généralités & Objectifs

Les études de Sûreté de Fonctionnement se fondent sur la représentation des comportements fonctionnels, des comportements dysfonctionnels et des propagations de panne ayant lieu au sein d'un système (section 1.2.1). Au delà des méthodes classiques (de type arbres de défaillance) présentées section 1.5, divers travaux ont montré le bien fondé d'une approche à base de modèle (section 2.5). Deux approches de modélisation peuvent être différenciées :

- l'ajout d'un modèle dysfonctionnel à un modèle fonctionnel déjà existant ;
- la création d'un modèle dysfonctionnel spécifique.

Nous nous intéresserons ici à la seconde catégorie. D'une manière générale, le but de tels modèles est de faciliter l'appréhension, la compréhension et l'analyse d'un système en autorisant, par exemple, la simulation d'une ou plusieurs défaillances, l'observation des conséquences engendrées par cette défaillance ou encore l'évaluation automatique des combinaisons de défaillance conduisant le système dans un état non souhaité.

Aujourd'hui donc, les modèles se veulent être un support pour les analyses classiques de Sûreté de Fonctionnement et seront en tant que tel utilisés pour démontrer le fonctionnement sûr d'un système (*i.e.* pour la phase de certification du système). La figure 3.1 a pour vocation de reprendre cette philosophie. Les informations présentes à l'intérieur d'un tel modèle doivent donc être correctes. Pour s'assurer de la correction et de la pertinence des informations présentes dans un modèle, nous proposons la mise en place d'un processus rigoureux de modélisation se voulant supporter la construction d'un modèle AltaRica. À ce propos, différentes études traitent de la modélisation en langage AltaRica de systèmes logiciels, électriques ou encore hydrauliques [12, 33] ; peu traitent de l'unification de ces études pour la modélisation de systèmes physiques et multi-physiques (mécanique, hydraulique, hydromécanique...).

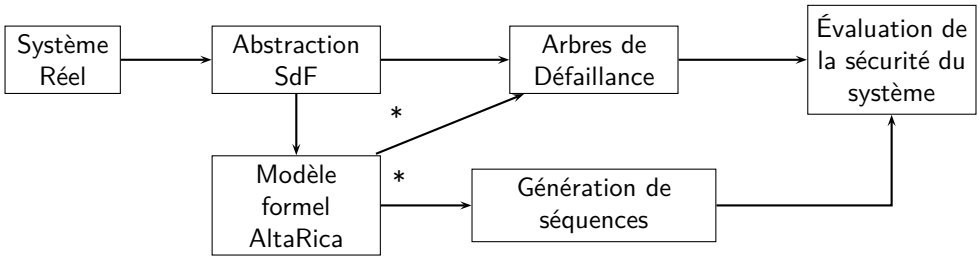


Figure 3.1 – Approche classique Vs Approche à base de modèles
(* : Section 2.4.3)

Le but de l'approche présentée dans cette partie est de proposer une méthodologie systématique supportant la modélisation du comportement d'un système en langage AltaRica. Pour cela, il s'agira d'identifier, de manière systématique, les informations devant être incluses dans le le modèle AltaRica. Nous nous intéresserons à l'identification des objectifs d'un tel modèle, aux évènements qu'il devra propager et aux informations qu'il devra contenir pour permettre cette propagation. Pour cela, l'approche présentée se propose :

- d'identifier l'information à extraire du système considéré ;
- de définir des abstractions permettant de propager cette information.

Au cours de cette section, nous verrons que l'approche proposée est très loin d'une approche *presse-bouton* et que la charge restante à l'analyste en charge de la réalisation du modèle reste conséquente. La section 3.2 présente le processus général de modélisation proposé pour la modélisation de systèmes physiques et multi-physiques. Les concepts présentés seront utilisés dans la section 3.3 présentant le cas d'étude puis dans les sections 3.4 à 3.7 s'intéressant à la modélisation de systèmes physiques. Enfin, la section 3.8 discutera de formalisations possibles pour représenter les informations identifiées aux sections précédentes.

3.1.2 Vision globale du processus de modélisation

Avant de rentrer dans les détails du processus de modélisation, nous donnons pour mieux comprendre la suite une vision globale de ce que sera ce processus. Pour débuter et selon [9], le processus de modélisation comprend 4 étapes majeures :

- analyse préliminaire :
  - définition des objectifs de l'étude (spécification des besoins) ;
  - définition du niveau de modélisation ;
  - définition du périmètre du système étudié et du contenu du modèle, *i.e.* identification des composants présents et de leurs interfaces,
- définition d'abstractions valides et pertinentes à propager dans le modèle ;
- spécification des comportements des composants élémentaires ;
- implémentation des comportements de chaque composant → Obtention d'une bibliothèque de nœuds AltaRica ;
- instanciation et connexion des nœuds.

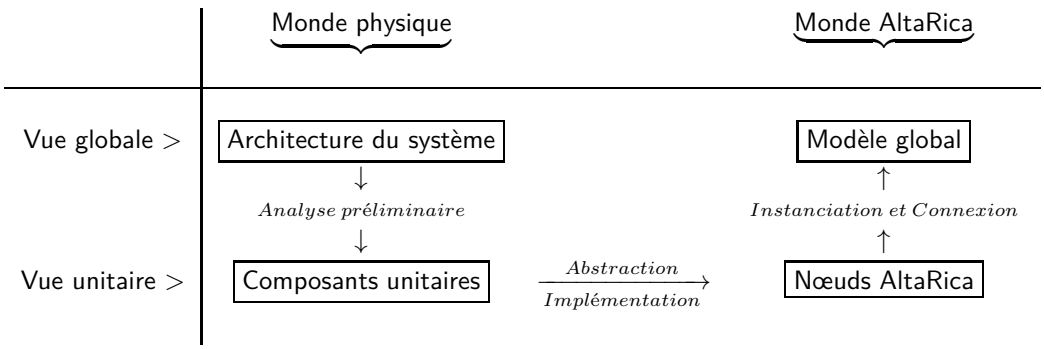


Figure 3.2 – Vision haut niveau du processus de modélisation [9]

Nous expliciterons dans la suite de ce chapitre les différentes étapes du processus de modélisation présenté sur la figure 3.2. Diverses illustrations montreront ensuite son application à des systèmes appartenant à des domaines physiques différents.

3.2 Processus de modélisation

3.2.1 Analyse préliminaire : Composition et objectifs du modèle

3.2.1.1 Caractérisation des objectifs du modèle haut niveau

Commençons par rappeler la définition donnée section 2.1.2 selon laquelle la modélisation est un processus technique qui permet de représenter un objet dans **un but** de connaissance [25]. Cette définition étant prise en compte, il serait sans doute déraisonnable et hasardeux de commencer notre approche sans préciser les objectifs du modèle souhaitant être construit. De manière simple, on exprimera dans un premier temps ces objectifs en précisant que le modèle devra permettre l'étude des évènements redoutés considérés et que pour cela, son périmètre devra couvrir le système d'étude (en prenant en compte l'ensemble des comportements ayant un impact sur l'occurrence des évènements redoutés considérés).

Il faut donc tout d'abord s'intéresser à l'identification de ces objectifs, et se poser la question des évènements redoutés qui devront être évalués (qualitativement et quantitativement) par le modèle du système (on se place dans un premier temps au plus haut niveau). Ici, pas de miracle ! Une liste des évènements redoutés pourra en général être fournie par les réglementations en vigueur dans le domaine considéré (réglementations aéronautiques, ferroviaires, automobiles...). Des évènements redoutés spécifiques pourront également provenir, via des analyses fonctionnelles des dangers (section 1.4), d'exigences clients ou d'exigences internes au concepteur du produit. Parmi l'ensemble de ces exigences, nous en choisirons un sous-ensemble (ou la totalité) que devra évaluer le modèle du système.

Une fois obtenu l'ensemble des évènements redoutés (qualitatifs et quantitatifs) à considérer, nous proposons de nous intéresser à l'identification du périmètre du modèle et de son architecture. Nous supposerons dans la section suivante que le système est suffisamment complexe pour admettre et mériter un découpage en sous-systèmes d'étude.

3.2.1.2 Caractérisation de l'architecture globale du modèle haut niveau

Déterminer l'architecture d'un modèle AltaRica (*i.e.* le choix des composants à modéliser dans le modèle) est crucial dans le sens où cette étape détermine le périmètre et la granularité (*i.e.* le niveau de détail) de ce modèle. Dans cette optique, nous souhaitons commencer par décrire 1) le système haut niveau comme une hiérarchie de sous-systèmes et 2) les dépendances fonctionnelles et physiques entre ces sous-systèmes. Pour cela, la démarche débute par une analyse fonctionnelle (de philosophie typiquement similaire à SADT [40]).

Pour guider le découpage du système en sous-systèmes d'étude, diverses « *bonnes pratiques* » peuvent être fournies à l'analyste. À ce stade, lorsque le système peut être décrit comme une hiérarchie de sous-systèmes et/ou de composants, la structure du modèle reflètera autant que possible la structure du système. *A contrario*, lorsque le système ne présente que peu ou pas de hiérarchie, une option intéressante à ce stade est d'identifier des sous-systèmes d'étude physiquement homogènes, *i.e.* de regrouper différents composants d'un même domaine physique (comme le sera présenté dans de prochaines sections, la propagation de défaillance entre composants d'un même domaine physique peut se faire en propageant un ensemble de grandeurs spécifiques au domaine).

Une fois le système de haut niveau « découpé » en sous-systèmes d'étude, l'analyse fonctionnelle autorise la description des interfaces fonctionnelles et physiques entre ces sous-systèmes. Ces interfaces permettront d'expliciter les chemins de propagation de panne globaux.

3.2.1.3 Déclinaison des objectifs des modèles

Nous supposons ici que l'étape précédente (ou plusieurs itérations de l'étape précédente) a permis l'identification de sous-systèmes d'étude n'admettant pas à leur tour de hiérarchie.

Avant de s'intéresser à la représentation des composants élémentaires [1], il nous est nécessaire de décliner les évènements redoutés du système de haut niveau jusqu'à ces sous-systèmes. Ici, pas de bonne pratique particulière. Il s'agira en général de l'expertise de l'ingénieur permettant d'identifier les sous-causes immédiates d'un évènement redouté de haut niveau. La prochaine étape traite de la caractérisation de l'architecture du modèle du sous-système obtenue.

3.2.1.4 Caractérisation de l'architecture du modèle du sous-système

Pour chaque sous-système identifié dans la section 3.2.1.2, il nous faut caractériser son architecture. Cette section nous ayant permis d'identifier le périmètre d'un sous-système, la section 3.2.1.3 nous ayant permis d'identifier les objectifs (*i.e.* les évènements redoutés) du modèle à concevoir, il nous faut désormais identifier les composants de ce sous-système influençant l'occurrence des évènements redoutés considérés.

Ici encore, peu de bonnes pratiques particulières pour identifier l'ensemble de ces composants. La charge de l'identification pourra souvent revenir à l'ingénieur ou à l'expert connaissant le sous-système. Le principe général de l'identification est cependant simple : pour chacun des composants du système réel, il faut se demander si une des défaillances de ce composant a un impact sur l'occurrence d'un ou plusieurs des évènements redoutés considérés. Pour cela, notons deux voies possibles permettant de supporter cette identification.

- L'utilisation d'une analyse fonctionnelle descendante (type SADT) qui nous permettra de décliner les fonctions du niveau sous-système jusqu'aux composants élémentaires réalisant cette fonction.
- Si disponible, l'utilisation d'une analyse de panne (type AMDE) pour identifier un ensemble de composant ayant une influence sur au moins un des évènements redoutés considérés.

Remarque : Remarquons que l'analyse fonctionnelle peut être également utilisée pour identifier les interfaces entre les différents composants du système. D'une manière générale, il est conseillé de combiner l'utilisation d'une analyse fonctionnelle et d'une analyse de panne pour identifier l'ensemble des composants à modéliser.

Concernant l'architecture du modèle du sous-système, une attention particulière devra également être donnée à l'identification des mécanismes de détection et de tolérance aux défaillances (ces mécanismes devant ensuite être inclus dans le modèle).

À noter cependant la possibilité d'utiliser un des avantages forts du langage AltaRica à savoir permettre la réutilisation des nœuds présents dans la bibliothèque AltaRica. En effet, si des composants du système réels ont des comportements proches ou similaires, il peut être avantageux de modéliser un unique nœud AltaRica qui sera plus tard instancié dans le modèle AltaRica autant de fois que nécessaire (chacune des instanciations du nœud AltaRica se comportant alors de manière autonome).

1. La notion de « élémentaire » est bien entendue totalement subjective et dénote en fait une entité que l'on ne souhaite pas davantage décomposer.

Toujours au niveau du choix des composants à inclure dans le modèle, ce choix peut être raffiné dans le but de rendre explicite certaines fonctions réalisées par un composant. Ainsi, si un composant réel a deux fonctions, il est possible de modéliser chaque fonction par un nœud AltaRica. Au contraire et pour limiter la taille du modèle, il peut être parfois bénéfique de grouper un ensemble de composants réels en un unique nœud AltaRica (par exemple, lorsqu'un ensemble de composants est pris en compte pour le bon fonctionnement du modèle mais n'impacte pas l'occurrence de l'évènement redouté). Dans tous les cas, les choix de modélisation devront être justifiés (majoritairement, grâce à l'analyse fonctionnelle et à l'AMDE) et tracés.

Le but de l'identification des composants à inclure dans le modèle peut se résumer de manière simple :

- réduire au maximum le nombre de nœuds modélisés dans la bibliothèque (en profitant de la capacité de réutilisation de modèle) ;
- permettre une observation fine des comportements et scénarios souhaitant être observés ;
- conserver un comportement du modèle conforme à celui du système réel.

Enfin, pour chaque composant identifié comme étant à modéliser, le choix des ports d'entrée-sortie (E/S) se doit de refléter et de modéliser les dépendances fonctionnelles et physiques entre le composant et son environnement.

3.2.2 Caractérisation du comportement d'un composant unitaire : abstraction

La section 3.2.1 ayant permis d'identifier les composants devant être modélisés, le comportement de chacun d'entre eux doit désormais être abstrait afin de préparer son implémentation en langage AltaRica. Nous fournissons dans cette section diverses étapes permettant d'initier une réflexion sur cette abstraction et d'obtenir une première représentation comprenant les informations devant être implémentées dans le modèle AltaRica. Nous appellerons cette représentation une *spécification*.

Remarque : Par spécification, nous entendons une description du comportement d'un système, d'un sous-système ou d'un composant. Ainsi, une spécification devra décrire ce qui doit être modélisé.

Comme déjà mentionné, le processus proposé ici est loin d'un processus automatique type *presse bouton* et souvent, la réflexion de l'analyste est mise à contribution. Cependant, divers exemples seront donnés dans des sections ultérieures.

Pour un sous-système d'étude considéré et une fois l'architecture et le but du modèle identifiés, nous souhaitons donc à présent 1) caractériser le comportement de chacun des composants identifiés comme étant à modéliser et 2) définir des abstractions permettant de modéliser ce comportement. Il nous est ainsi nécessaire d'identifier, pour chacun des composants :

- les comportements fonctionnels : identifier les évènements ayant trait au fonctionnement nominal du composant ainsi que les modes de fonctionnement nominaux ;
- les comportements dysfonctionnels : identifier les modes de défaillance du composant ainsi que ses modes de dysfonctionnement (i.e. les états que le système peut atteindre après l'occurrence d'une défaillance) ;
- les lois de propagation à l'intérieur du composant : comment réagit le composant à une entrée non nominale ? Comment propage t-il l'information à son tour ?

3.2.2.1 Identification des comportements fonctionnels

Les évènements relatifs au fonctionnement nominal du composant peuvent être de différents types.

- Des changement de modes de fonctionnement (état de fonctionnement nominal du composant) identifiés grâce aux différents documents de conception disponibles (par exemple, la spécification du composant). Il s'agira par exemple de modéliser l'ouverture (souhaitée) d'une valve ou le changement de configuration d'utilisation du système étudié.
- Des changements de phases de fonctionnement du système. Par exemple, le passage d'une phase de fonctionnement normal à une phase de fonctionnement en sous-régime.
- Des changements de conditions d'utilisation du système : le comportement du système peut en effet être modifié suivant l'environnement dans lequel il évolue ou suivant l'utilisation que l'utilisateur en fait.

Remarque : Bien qu'il soit possible de modéliser différents évènements relatifs au fonctionnement du composant, nous ne nous sommes intéressé qu'à la description des modes de fonctionnement dans une phase de fonctionnement donnée et en fixant les conditions d'utilisation.

3.2.2.2 Identification des comportements dysfonctionnels

Les modes de défaillance et les modes de dysfonctionnement peuvent être identifiés à partir d'une analyse fonctionnelle du composant et / ou de l'AMDE. La première, en décrivant les fonctions du composant, fournira les modes de défaillance fonctionnels (par négation des fonctions devant être remplies) ; la seconde fournira pour chaque composant les modes de défaillance organiques. Cependant, bien que plus exhaustive dans la description des modes de défaillance, l'AMDE est excessivement détaillée pour être candidate à un export de l'ensemble de ces modes de défaillance dans le modèle AltaRica. En effet, plusieurs modes de défaillance peuvent avoir une conséquence fonctionnelle identique. Ainsi, le modèle AltaRica sera muni, autant que possible, de modes de défaillance fonctionnels ; chacun d'entre eux pouvant être mis en correspondance avec au moins un mode de défaillance issu de l'AMDE.

3.2.2.3 Caractérisation des propagations

Concernant la modélisation des propagations de défaillance, nous nous devons d'identifier comment le composant réagit à une défaillance de son environnement, i.e. nous devons modéliser la réaction de ce composant lors d'une défaillance ainsi que la façon dont il propage l'information à son tour (propagation directe, mitigation de la défaillance, envoi d'un signal d'alarme si détection...). Propager l'ensemble de ces comportements revient à être capable, pour le nœud AltaRica, de « lire » l'information provenant de son environnement et de « transmettre » l'information à ce même environnement. Pour cela, il nous faut définir des abstractions suffisamment expressives permettant de propager les informations dans le modèle (*i.e.* définir des abstractions pour les variables d'entrées / sorties).

Étude boîte noire : Nous commençons par replacer le composant dans son environnement en étudiant celui-ci comme une boîte noire, *i.e.* on ne s'intéresse pas pour le moment au comportement interne du composant. La figure 3.3 replace le composant étudié dans son environnement :

- le composant et éventuellement sa fonction sont indiqués au centre de la figure ;

- les flux d'entrée E représentent les données que le composant reçoit de son environnement (l'effet de l'environnement sur le composant) ;
- les flux de sortie S représentent les données que le composant transmet à son environnement (l'effet du composant sur l'environnement).

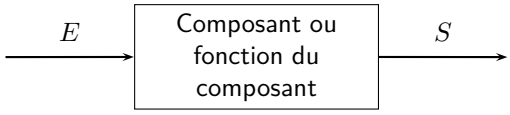


Figure 3.3 – Le composant et son environnement

Exemple : Prenons l'exemple d'un capteur de température. Sa fonction est de mesurer une température, de transformer celle-ci en en valeur numérique et de la transmettre à un calculateur. Son unique flux d'entrée est une température. Son unique flux de sortie est une valeur numérique transmise au calculateur.

Cette analyse boîte noire, proche dans sa philosophie d'une analyse fonctionnelle, a été utilisée afin d'identifier des ports d'entrées / sorties (E/S) de haut niveau (mouvement, puissance, fluide, courant électrique...) qui nous permettrons de propager les évènements identifiés de l'environnement au composant et du composant à son environnement. Pour cela et conformément à notre expérience, les flux identifiés sont de deux types : des flux que nous nommerons « physiques » et qui désigneront un échange d'énergie ou de matière entre l'environnement et le composant ; des flux « informationnels » qui désigneront un échange d'information (une valeur numérique par exemple). Le tableau 3.1 donne la correspondance entre le type de composant (plus généralement le type de système) et les combinaisons flux entrant / flux sortant.

Flux entrant \ Flux sortant	Physique	Informationnel
Physique	Système de transformation physique	Système de mesure (Capteur)
Informationnel	Actionneur	Système de traitement d'information

Tableau 3.1 – Correspondance entre type de système et combinaison des flux [40]

Raffinement des E/S : Pour poursuivre le raisonnement, remarquons que si le concept d'E/S orientées, propre à AltaRica (AltaRica DataFlow et AltaRica OCAS), s'adapte bien aux systèmes logiciels (les composants ont des E/S bien définies), il est plus difficilement applicable aux systèmes physiques (électriques, mécaniques, hydrauliques...) où en général, les dépendances dysfonctionnelles peuvent être difficilement identifiables entre ce qui pourrait être une entrée et une sortie. Or ces E/S doivent nous permettre de propager les évènements identifiés comme étant à propager. Pour cela, nous choisissons de raffiner ces E/S selon le domaine physique considéré. Ce raffinement se fait par l'intermédiaire d'un ensemble de grandeurs physiques dont le domaine de définition (la

plupart du temps énuméré) est fonction des évènements redoutés à observer (*i.e.* des objectifs du modèle) et des évènements à propager.

Remarque : Nous illustrerons dans les cas d'application présents dans ce mémoire (section 3.4 à 3.7) que si ce choix de modélisation nous impose de descendre à bas niveau, il nous fournit aussi :

- la capacité d'expliciter de manière détaillée l'impact des défaillances sur le comportement physique du système ;
- une flexibilité et une capacité de généricité importante dans la prise en compte des évènements devant être modélisés ;
- la possibilité de capitaliser la connaissance (fonctionnelle et dysfonctionnelle) du comportement des composants dans des modèles génériques de bas niveau.

Sens de propagation : Une question importante devant être levée à ce stade est celui du sens de propagation des évènements. En effet le principe de la propagation est d'informer l'environnement de ce qui ce passe à l'intérieur du composant et réciproquement. Souvent en particulier pour la propagation des flux physiques, la propagation se devra d'être bidirectionnelle.

Exemple : Dans un système physique, souvent, « l'entrée » influe sur la sortie et réciproquement. En prenant l'exemple d'un tuyau pouvant se colmater dans un circuit hydraulique, un colmatage entraînera d'une manière générale non seulement une absence de fluide en sortie mais aussi une accumulation de fluide en entrée et donc une augmentation de pression en amont du tuyau. Pour informer l'environnement, il faudra propager l'information de manière bidirectionnelle.

Remarque : Dans la suite, les termes « amont » et « aval » présume un sens « générateur $\rightarrow$ récepteur ».

Sémantique des flux propagés : Les flux propagés peuvent avoir au sein d'un même modèle différentes sémantiques. Au delà des différences entre flux physiques / informationnels et entre flux physiques (flux mécanique, flux électrique, flux hydraulique...), il est possible de propager la valeur (la plage de valeur) d'une grandeur physique et/ou sa qualité (correcte ou erronée). AltaRica étant fondé sur la notion d'automate de mode, il convient ordinairement de faire un compromis entre expressivité et complexité afin d'éviter un temps de calcul trop important (problème d'explosion combinatoire). La plupart du temps, une solution acceptable est de s'assurer que les valeurs manipulées (*i.e.* le domaine de définition des variables manipulées) soient nécessaires et suffisantes à l'observation des évènements redoutés considérés.

En pratique, les flux sont représentés par deux types d'informations : leurs qualités et leurs grandeurs. Par exemple, pour un système informationnel, la valeur du signal transmis est discrétisée en fonction des tests nominaux que le signal subit. Pour un système physique, on parlera de valeur nominale, inférieur à ce que prévoit la spécification ou supérieur à ce que prévoit la spécification (en prenant en compte différents niveaux pour ces infériorités ou supériorités). La qualité permettra, par exemple pour un fluide, d'indiquer la présence de pollutions dans le fluide où d'indiquer des oscillations (plus ou moins importante) du débit de fluide.

// Transitions - Événement transitoire	// Transitions - Événement inverse
$ST = ok \mid - Evenement\_transitoire \rightarrow ST := transitoire;$	$ST = transitoire \mid - Evenement\_inverse \rightarrow ST := ok;$

Dynamique du composant : Une fois les évènements et les ports d'E/S AltaRica identifiés, on s'intéresse à la dynamique du composant : on s'intéresse aux transitions et aux assertions du nœud AltaRica. En ce qui concerne les transitions, il faut s'intéresser 1) aux conditions sous lesquelles les évènements peuvent avoir lieu et 2) au potentiel caractère transitoire des évènements. Le premier point définit les gardes des transitions AltaRica (par exemple, un évènement peut n'être tirable que dans un certain état) et permettront, entre autres choses, de modéliser des pannes en cascade. Le second point identifie si l'évènement est permanent ou transitoire. Dans le cas d'évènements transitoires, on ajoute au modèle un évènement « inverse ».

Enfin, pour spécifier les assertions AltaRica, on réalise différentes tables où les valeurs des variables de sortie sont définies en fonctions de celles des variables d'entrée et de l'état interne du nœud AltaRica. Ces tables sont écrites en se fondant sur les effets locaux des défaillances (dont certains sont décrits dans l'AMDE) ainsi que sur sa propre compréhension du système.

3.2.2.4 Obtention d'une spécification de modèle AltaRica

Ci-dessous, un bilan des activités menées au long de cette section 3.2.2. Celles-ci nous permettrons d'obtenir une « spécification » du modèle AltaRica à réaliser.

1. Identification des objectifs du modèle, des évènements redoutés à observer sur le système étudié
2. Analyse fonctionnelle du système à étudier
   (a) Décomposition du système global en sous-systèmes
   (b) Identification des interfaces entre ces sous-systèmes
3. Pour chacun des sous-systèmes identifiés
   (a) Déclinaison des évènements redoutés aux différents sous-systèmes
   (b) Identification des composants ayant un impact sur l'occurrence des évènements redoutés
   (c) Identification des interfaces entre ces composants
4. Pour chacun des composants identifiés
   (a) Réalisation d'une analyse fonctionnelle
      - Identification de ports d'E/S de haut niveau
      - Identification des comportements fonctionnels
   (b) Identification des comportements dysfonctionnels
   (c) Identification de ports d'E/S raffinés
   (d) Identification des domaines de définition des différentes variables
   (e) Identification des lois de propagation (de la fonction de transfert du composant) et de la dynamique relative aux comportements fonctionnels et dysfonctionnels du composant

On suppose donc ici que ces différentes étapes permettent l'obtention, pour chacun des composants, d'une représentation (pouvant être de graphique, tabulaire...) que l'on nomme « spécification » et qui comprend l'ensemble des informations que doit contenir le nœud AltaRica

représentant le composant. Une proposition de forme pour cette spécification est donnée en section 3.8.7.

Une fois cette spécification disponible, le comportement du composant est implémenté au sein d'une bibliothèque de nœuds AltaRica. On suppose donc ici que nous possédons une bibliothèque de nœuds AltaRica unitaire.

3.2.3 Modélisation d'un système en AltaRica

3.2.3.1 Instanciation et Connexion

Les composants ayant donc été abstraits puis implémentés au sein d'une bibliothèque de nœuds, il est à présent possible de construire le modèle AltaRica des sous-systèmes étudiés puis du système global (l'assemblage des sous-systèmes).

Pour cela et comme présenté dans la section 2.4.1, les nœuds de la bibliothèque seront instanciés dans le modèle à réaliser (*i.e.* on insère dans le modèle autant de nœuds unitaires que nécessaire) puis reliés entre eux conformément à l'architecture du système. Ces opérations seront en pratique réalisées à l'aide d'un outil supportant le langage AltaRica. Par exemple Cecilia^{TM} OCAS (Dassault Aviation), BPA-FT9 (Dassault System) ou encore SIMFIA (Apsys).

Cependant et en accord avec la formule bien connue « *le tout est plus que la somme de ses parties* », un modèle ne saurait bien souvent se réduire à une simple interconnexion de nœuds unitaires. Ainsi, des stratégies de reconfigurations, de mitigations de défaillance... pourront être modélisées grâce au modèle global. D'une manière générale, un comportement touchant deux nœuds A et B devra être modélisé au sein d'un parent commun à A et B. Pour cela, l'utilisation de synchronisation (Cf. section 2.3.4) pourra être pertinente.

3.2.3.2 Observation des évènements redoutés

Il nous faut également pouvoir observer les évènements redoutés considérés sur le modèle. Cette observation passe, dans le modèle AltaRica, par la création de nœuds appelés « observateurs ». Ces nœuds sont similaires à des nœuds AltaRica classiques. Cependant, s'ils possèdent des variables d'E/S, ils ne possèdent en général pas d'évènements et donc pas de variables d'état. Typiquement, un tel observateur possèdera N variables d'entrée ($N > 0$) et une unique variable de sortie booléenne représentant l'occurrence, ou non, d'un évènement redouté. Son unique assertion permettra de modéliser l'occurrence de l'évènement redouté : la variable de sortie sera vraie si les N variables d'entrées prennent une certaine combinaison de valeurs.

3.2.3.3 Bilan

Ci-dessous, un bilan des activités à réaliser pour la modélisation d'un système (pouvant être un sous-système ou le système global) en langage AltaRica. On rappelle qu'à ce stade, on suppose disponible les informations récoltées grâce aux sections 3.2.1 et 3.2.2.

1. Construction de la bibliothèque de nœuds. Pour chacun des nœuds :
   (a) Modélisation des variables d'E/S et des variables d'états
   (b) Modélisation des évènements
   (c) Modélisation des transitions et des assertions
2. Construction de l'architecture du modèle
   (a) Insertion des nœuds dans le modèle

   (b) Connexion entre ces nœuds
   (c) Modélisation des synchronisations
3. Modélisation des évènements redoutés

3.3 Modélisation des bibliothèques : cas d'étude et analyse préliminaire

Remarque : *Dans un souci de clarté et de lisibilité, le cas d'étude proposé est inspiré de la réalité mais n'en reprend pas toute la complexité, ni même tout le réalisme. Ainsi, certains aspects et phénomènes pourront être présent dans un but d'illustration de la démarche sans qu'ils n'aient forcément lieu dans le système physique réel.*

Dans cette section, nous présentons le cas d'étude sur lequel nous appliquerons la méthodologie proposée à la section 3.2. Nous identifierons en particulier les évènements redoutés à étudier sur ce cas d'étude.

3.3.1 Cas d'étude

3.3.1.1 Description

En phase de vol, un turbomoteur (moteur d'hélicoptère) est potentiellement soumis à une phase de survitesse (i.e. le moteur tourne à une vitesse supérieure à la normale). La raison peut avoir diverses origines. Par exemple, la cause peut être :

- mécanique : en cas de rupture d'arbre de transmission ;
- hydromécanique : si la quantité de carburant envoyée dans la chambre de combustion est supérieure à la consigne ;
- logicielle : si cette même consigne est calculée de manière erronée ;
- opérationnelle : en cas d'action pilote inappropriée.

Quelle qu'en soit l'origine, l'effet de cette survitesse sur le moteur doit être limité. Pour cela, un turbomoteur possède différentes protections possibles, du blindage (protection mécanique) à la coupure électronique du débit carburant dont le but est de détecter cette survitesse puis d'éteindre le moteur « proprement », *i.e.* sans autre effet.

Les différents sous-systèmes composant le *système turbomoteur*, utilisé comme fil rouge par la suite et représenté sur la figure 3.4, sont les suivants :

- La chaine de transmission, que l'on considérera comme une succession d'engrenages et d'arbres de transmission, est constituée principalement d'engrenages (assemblage de 2 roues dentées) et de roulements. Le rôle principal d'un tel système est de transmettre de la puissance mécanique d'un point d'entrée à un point de sortie (ici de la sortie de la turbine jusqu'au rotor de l'hélicoptère). À noter également, ce système comporte différents capteurs permettant de mesurer la vitesse de rotation de l'ensemble et de transmettre cette vitesse au calculateur.
- Le circuit carburant, que l'on considérera comme un circuit hydromécanique, est composé de tuyaux, de pompes, de valves, de filtre, *etc.* Ce système assure, entre autres, les fonctions de dosage du carburant et d'alimentation de la chambre de combustion. Brièvement, il

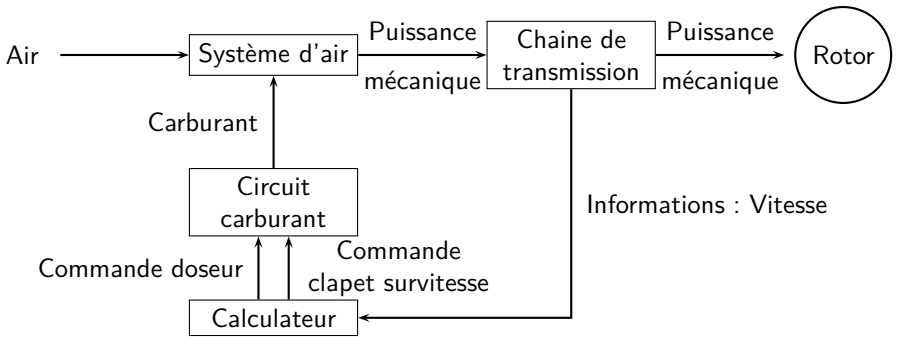


Figure 3.4 – Vision schématique simplifiée du cas d'étude
(Le circuit de lubrification n'est pas représenté)

.

s'agira d'injecter la juste quantité (consigne élaborée par le calculateur) de carburant dans la chambre de combustion afin que celle-ci produise les gaz entraînant la turbine à la vitesse voulue. Dans ce circuit carburant, se situe un clapet d'arrêt permettant de forcer le retour du carburant dans le réservoir, l'empêchant ainsi d'être injecté dans la chambre de combustion du moteur. Lorsque l'ouverture de ce clapet est commandée (par exemple, dans un cas de survitesse détectée), le moteur s'arrête.

- Le calculateur traite les différentes mesures provenant du moteur afin 1) d'élaborer la consigne de débit carburant à injecter dans le moteur et 2) de surveiller le fonctionnement du moteur et en particulier de couper l'arrivée du carburant si une survitesse est détectée. Dans un tel cas, le calculateur commande l'ouverture du clapet d'arrêt situé dans le circuit carburant.
- Le système d'air, composé de compresseurs, d'une chambre de combustion et de turbines, permet, à partir d'air et de carburant, de produire et de transmettre de la puissance mécanique à la chaîne de transmission.

Dans un soucis de généricité, nous retiendrons principalement que seront étudiés :

- un système mécanique en rotation constitué d'engrenages et de roulements ;
- un système hydromécanique constitué de tuyaux, pompes, valves, filtres, *etc.* ;
- un système logiciel ;
- un système d'air composé de compresseurs, d'une chambre de combustion et de turbines.

3.3.1.2 Évènements redoutés

Nous nous intéresserons ici à l'occurrence de deux évènements redoutés :

- l'occurrence d'une survitesse du moteur, *i.e.* , le moteur tourne plus vite que la normale ;
- l'Arrêt en Vol Non Commandé (AEVNC) du moteur, *i.e.* le moteur s'arrête de manière non souhaitée par le pilote.

Ces deux évènements serviront de support dans la suite de ce chapitre et nous permettront d'illustrer la méthodologie de modélisation proposée.

	Survitesse	AEVNC
Sous-système mécanique	Survitesse de la transmission	Perte de la transmission
Sous-système hydromécanique	Trop de carburant est injecté dans la chambre de combustion	Trop peu de carburant est injecté dans la chambre de combustion
Sous-système logiciel	La quantité de carburant à injecter calculée est trop importante	La quantité de carburant à injecter calculée est trop faible ou commande intempestive du clapet d'arrêt
Système d'air principal	Survitesse de l'ensemble	Perte de la génération de mouvement

Tableau 3.2 – Déclinaison des évènements redoutés système aux sous-systèmes

3.3.2 Analyse préliminaire

3.3.2.1 Décomposition du système global en sous-systèmes d'étude

En accord avec la figure 3.4, avec la section 3.2.1.2 et avec la description du système ci-dessous, nous décomposons le cas d'étude en différents sous-systèmes physiquement homogènes : un système mécanique, un système hydromécanique, un système d'air ainsi qu'un système logiciel.

Concernant les interfaces entre ces différents sous-systèmes, celles-ci ont d'ores et déjà été mentionnées section 3.3.1.1 et sont représentées sur la figure 3.4.

3.3.2.2 Déclinaison des évènements redoutés système aux sous-systèmes

Il s'agit ici de décliner aux sous-systèmes identifiés les évènements redoutés système explicités dans la partie 3.3.1.2. Pour rappel, il s'agit de 1) l'occurrence d'une survitesse moteur et 2) l'Arrêt en Vol Non Commandé. La table 3.2 a pour vocation de présenter la déclinaison de ces évènements redoutés aux sous-systèmes d'étude considérés.

Il nous faut maintenant nous intéresser à la modélisation de chacun des composants identifiés comme étant à modéliser. Pour cela, nous particularisons l'étude en fonction du domaine physique considéré. Pourquoi ? Deux arguments étayeront ici notre choix :

- les grandeurs physiques manipulées sont différentes d'un domaine à l'autre ;
- Dans le cas réel, plus complexe que le cas d'étude simple présenté ici, tous les évènements redoutés considérés n'ont pas leurs occurrences influencées par l'ensemble des sous-systèmes d'études (schématiquement, cela signifie que le tableau 3.2 possède des cases vides). Ainsi, les évènements redoutés à observer seront de granularités (*i.e.* niveaux de détail) différentes.

3.4 Modélisation d'un sous-système mécanique

Nous étudions dans cette section la caractérisation du modèle du sous-système mécanique « chaine de transmission » présent sur la figure 3.4.

3.4.1 Architecture du sous-système mécanique

Cette section fait référence à celle 3.2.1.4 (Caractérisation de l'architecture du modèle du sous-système) et a dans ce sens pour vocation de caractériser l'architecture permettant de modéliser la propagation de défaillance à l'intérieur du sous-système mécanique. Dans cette optique, nous identifions les composants dont au moins un mode de défaillance a un impact sur au moins un des évènements redoutés considérés (puis, nous ne modéliserons dans le modèle du composant que les modes de défaillance ayant un impact sur au moins un des évènements redoutés considérés).

Remarque : L'objectif à terme est ici :

- d'inclure dans le modèle les informations utiles à la représentation des propagations de défaillance dans le système ;
- de justifier la non-inclusion des autres informations.

Ici et sans toutefois imposer la méthode à utiliser, une tâche pertinente consiste à réaliser une analyse fonctionnelle du sous-système au niveau organique, *i.e.* de décomposer le sous-système en composants élémentaires. Cette étape permet de décrire les liens matériels et fonctionnels entre les composants du système et donc de décrire la logique d'interconnexion entre les différents éléments mis en jeux. Une telle approche nous permet d'obtenir un schéma de la forme de celui représenté sur la figure 3.5. Les entrées provenant du circuit de lubrification ne sont pas représentées (l'ensemble baigne en pratique dans un « brouillard d'huile »).

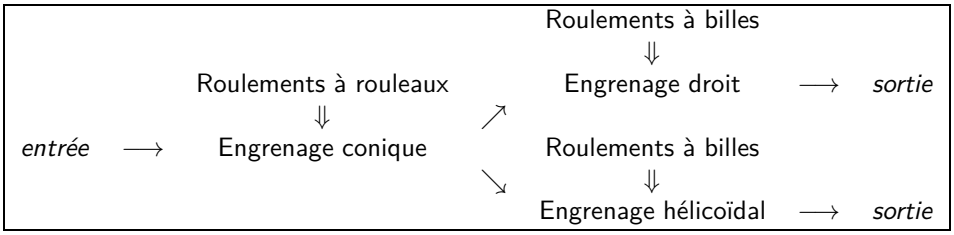


Figure 3.5 – Schéma de principe simplifié du sous-système mécanique
(⇓ : Guidage ; ⟶ : Entraînement mécanique)

Ainsi, nous déterminons l'architecture du modèle, les interfaces entre les différents composants et identifions un ensemble de composants à modéliser : engrenages, roulements (billes, rouleaux...) et arbres de transmission (représentés par les flèches simples sur la figure 3.5). Typiquement et pour illustrer le principe d'instanciation du langage AltaRica, il ne sera modélisé qu'un unique nœud pour représenter à la fois les roulements à billes et ceux à rouleaux. La même philosophie peut s'appliquer aux engrenages où l'on ne modélisera qu'un seul nœud pour modéliser les 3 types présents sur la figure 3.5. D'une manière générale, n'utiliser qu'un unique modèle pour la représentation de plusieurs composants n'est possible que lorsque les composants ont des fonctions et des interfaces avec leurs environnement similaires.

Dans la suite de cette section, nous illustrerons la création d'un nœud AltaRica en nous appuyant sur l'exemple d'un engrenage, qu'il soit droit, conique ou hélicoïdal. La démarche intellectuelle à réaliser reste *a priori* similaire pour tout autre composant.

3.4.2 Une première identification des entrées / sorties

3.4.2.1 De manière informelle...

Dans un premier temps, le but est ici d'identifier un ensemble de ports d'entrées / sorties de haut niveau. Pour cela, l'ensemble des interactions que le composant (*i.e.* l'engrenage) peut avoir avec son environnement est caractérisé. Pour cela, le schéma de principe figure 3.5 ainsi que les connaissances de l'ingénieur ou expert sont utilisées et conduisent au second schéma centré cette fois-ci sur le composant et présenté figure 3.6.

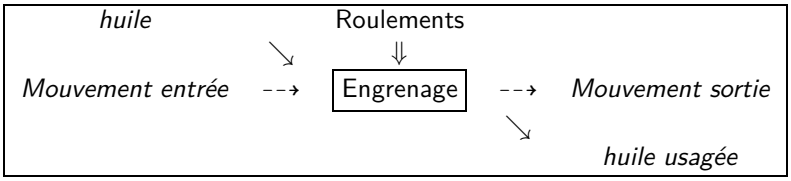


Figure 3.6 – Identification des E/S d'un engrenage
(⇓ : Guidage ; ⟶ : huile ; ⇢ : Entraînement mécanique)

Les interactions identifiées nous permettent de déterminer un premier ensemble d'entrées et de sorties dont l'objectif sera de permettre la modélisation des propagations de défaillance. Pour notre engrenage et conformément à la figure 3.6, nous identifions :

- trois entrées :
  - l'entrée « *Mouvement entrée* » par laquelle arrive le mouvement et la puissance,
  - l'entrée « *huile* » nécessaire au bon fonctionnement de l'engrenage,
  - l'entrée « *Roulements* » correspondant à la fonction de guidage de l'engrenage assurée par le roulement ;
- deux sorties :
  - la sortie « *Mouvement sortie* » par laquelle est transmise le mouvement et la puissance,
  - la sortie « *huile usagée* » modélisant le flux d'huile après qu'il ait lubrifié l'engrenage.

3.4.2.2 ... puis en langage AltaRica

Ainsi, il est possible, avant même d'aborder l'abstraction du comportement de l'engrenage, de définir un premier ensemble de flux en décrivant uniquement leurs orientations.

```
node engrenage
 flow
   Mouvement_entree : ... :in ;
   huile : ... : in ;
   guidage : ... : in ;

   Mouvement_sortie : ... : out ;
   huile_usagée : ... : out ;
 ...
edon
```

3.4.3 Caractérisation du comportement interne de l'engrenage

3.4.3.1 De manière informelle...

Cette section est à mettre en relation avec celle 3.2.2 (Caractérisation du comportement d'un composant unitaire : abstraction). Le but ici est d'identifier deux types d'informations :

- les évènements qui seront à intégrer dans le modèle, *i.e.* les évènements relatifs au fonctionnement nominal du composant ainsi qu'à son dysfonctionnement ;
- les modes de fonctionnement et de dysfonctionnement ainsi que les transitions entre ces modes.

Comme explicité dans la section 3.2.2, les évènements (relatifs au fonctionnement et au dysfonctionnement) du composant « engrenage » peuvent être obtenus à partir de l'analyse fonctionnelle et de l'AMDE. Ici, on admettra que ceux retenus sont :

- la perte de transmission par rupture de l'engrenage (*e.g.* perte de plusieurs dents de l'engrenage) ;
- la production de particules métalliques dans l'huile due à une usure de l'engrenage.

Il n'y a ainsi pas d'évènement relatif au comportement nominal de l'engrenage. Une fois ces évènements identifiés, les modes de fonctionnement et de dysfonctionnement sont en mesure d'être déterminés. Ici et en relation avec les deux évènements retenus, nous avons :

- un unique mode de fonctionnement nominal, lorsque la transmission de mouvement est correcte et qu'aucune particule métallique n'est libérée dans l'huile ;
- trois modes de dysfonctionnement : la perte de la transmission de mouvement sans émission de particule mécanique, l'émission de particules métalliques tout en conservant la transmission de mouvement, la combinaison des deux.

En combinant la connaissance des évènements et des modes de fonctionnement / dysfonctionnement, il est possible d'écrire les différentes transitions entre ces modes (par exemple, on passe du fonctionnement nominal à la perte de transmission par la rupture de l'engrenage). Une représentation possible à ce stade est un diagramme état - transition classique (figure 3.7) où les états sont représentés par des rectangles.

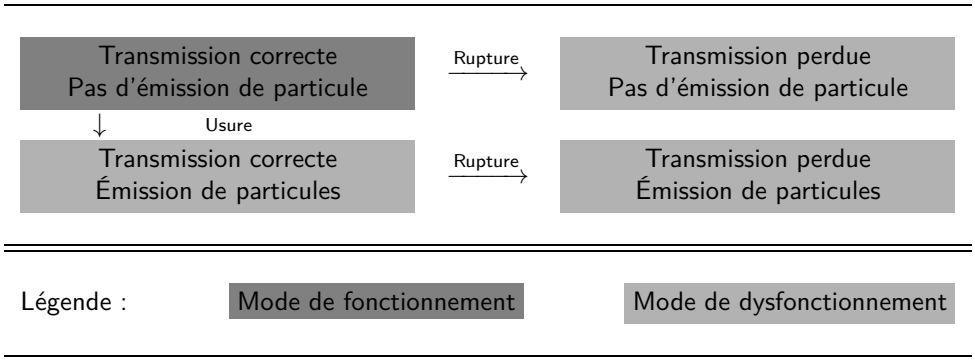


Figure 3.7 – Diagramme état / transition de l'engrenage

3.4.3.2 ... puis en langage AltaRica

En langage AltaRica, la figure 3.7 se traduit par la création de deux variables d'état *ST_transmission* et *ST_particles* représentant respectivement l'état de la transmission de mouvement et l'état de l'émission de particules métalliques dans l'huile. Concernant le niveau de détail de ces variables, nous les prendrons à valeurs respectives dans $\{ok, ko\}$ (ok correspondant à une transmission correcte ; ko à une transmission incorrecte) et dans $\{no, yes\}$ (no correspondant à l'absence de particule émise, yes à l'émission de particules). Notons que initialement, la transmission est correcte et qu'aucune particule n'est encore émise.

À ce stade, il est également possible de créer les évènements « $Rupture$ » et « $Usure$ » qui correspondent respectivement à la rupture et à l'usure (entraînant l'émission de particules) de l'engrenage. Une fois les états et les évènements déclarés, on implémente les différentes transitions du nœud (représentées par des arcs orientés sur la figure 3.7).

```
node engrenage
 ...
 state
   ST_transmission : {ok, ko} ;
   ST_particles : {no, yes} ;
 init
   ST_transmission := ok ;
   ST_particles := no ;
 event
   Rupture, Usure ;
 trans
   ST_transmission = ok |- Rupture -> ST_transmission:=ko ;
   ST_particles = no |- Usure -> ST_particles = yes ;
 ...
edon
```

3.4.4 Identification des évènements externes à propager

Le comportement de l'engrenage ne dépend cependant pas uniquement des évènements ayant lieu en son sein. Le composant doit propager à l'environnement non seulement ses évènements internes ($Rupture$ et $Usure$) mais aussi les évènements d'autres composants (appartenant à son système ou à des systèmes en interaction avec le sien) qui parviennent jusqu'à lui.

Exemple : Sur la figure 3.4, l'engrenage se situe par exemple à l'intérieur du système « Chaîne de transmission ». Au delà de ses évènements internes, il devra aussi propager des scénarios tels que :

- une survitesse due au système d'air : si le système d'air entre en survitesse, l'engrenage verra une survitesse en entrée et propagera une survitesse en sortie ;
- une rupture du circuit de lubrification : si la lubrification de l'engrenage n'est plus assurée, celui-ci cassera et ne sera plus à même d'assurer sa fonction ;
- ...

Il est donc nécessaire, pour modéliser la propagation des défaillances, d'identifier et de considérer les autres évènements pouvant survenir dans le système. Cette propagation peut alors se résumer en deux questions :

- comment propager les défaillances ?
- comment modéliser la réaction du composant vis-à-vis des évènements extérieur ?

L'approche proposée est de réaliser un inventaire des possibles évènements (et notamment des modes de défaillance) pouvant avoir lieu au sein des autres composants du système et au sein des systèmes en interaction avec celui considéré. De cet inventaire, on extrait des modes de défaillance génériques. Ici et pour rester simple dans l'appréhension du problème, on ne considérera que deux modes de défaillance génériques concernant la transmission de mouvement : la rupture d'un élément et la survitesse de la chaîne de transmission. Il faudra donc modéliser le comportement de l'engrenage si une rupture ou une survitesse a lieu en amont ou en aval de celui-ci. Également à propager dans la modélisation de notre engrenage, une perte de guidage ainsi qu'une perte de lubrification (plus d'huile en entrée).

3.4.5 Caractérisation de l'abstraction des variables de flux

3.4.5.1 Identification des E/S raffinées

Ici, l'objet de l'approche est de raffiner les entrées et les sorties de haut niveau identifiées section 3.4.2 dans le but de propager à l'environnement les évènements internes et externes au composant. Le travail décrit ici se propose de propager ces évènements (fonctionnels et dysfonctionnels) via un ensemble d'abstractions de grandeurs physiques.

Pour comprendre les prochaines lignes, on indique brièvement ci-dessous les éléments que le modèle de « l'engrenage » devra propager :

- la rupture de l'engrenage ;
- l'usure de l'engrenage (*i.e.* une émission de particules dans l'huile) ;
- une rupture en amont de l'engrenage ;
- une rupture en aval de l'engrenage ;
- une survitesse en amont de l'engrenage ;
- une survitesse en aval de l'engrenage ;
- une perte du guidage de l'engrenage (*i.e.* une perte du roulement associé à l'engrenage) ;
- une perte de la lubrification de l'engrenage (*i.e.* plus d'huile en entrée de l'engrenage).

Pour identifier des abstractions permettant de propager cet ensemble d'évènements, nous utilisons l'analyse de panne (*i.e.* l'AMDE) qui décrit les modes de défaillance d'un composant mécanique (typiquement, une rupture) comme ayant des conséquences sur le *couple mécanique* transmis et la *vitesse de rotation* transmise. Ainsi, une défaillance peut avoir des conséquences sur le couple, la vitesse de rotation ou encore les deux simultanément. Ces deux informations raffinent l'entrée « *Mouvement entrée* » et la sortie « *Mouvement sortie* » présente sur la figure 3.6 et nous permettent de propager la rupture de l'engrenage à son environnement. Une rapide réflexion assure également que transmettre le couple et la vitesse nous permettra de propager les ruptures et les survitesses à l'aval du composant.

Cependant, dans un système de transmission mécanique, une défaillance de l'engrenage n'a pas uniquement des conséquences sur les composants situés en aval mais aussi sur les composants situés en amont.

Exemple : Une rupture de l'engrenage entraînera non seulement un arrêt de l'entraînement du rotor (si l'hélicoptère possède un unique moteur) mais conduira également à une perte du couple résistant sur la turbine... ce qui conduira à une survitesse de celle ci... ce qui entraînera l'activation de la protection survitesse.

Par conséquent, il nous faut propager le couple de variable {couple, vitesse} de façon bidirectionnelle, *i.e.* en amont et en aval.

Plus « simple » à présent est l'étude de la propagation des défaillances ayant trait à la lubrification et au guidage (*i.e.* au roulement) de l'engrenage. Au sujet de la lubrification, une propagation monodirectionnelle (de l'amont vers l'aval) du couple de variable {présence d'huile, présence de particules dans l'huile} suffit pour propager à la fois l'usure de l'engrenage et sa perte de lubrification. Au sujet du guidage, ce qui nous intéresse est de savoir si l'engrenage est guidé en rotation ou non, *i.e.* si la fonction de guidage est assurée correctement. Nous propageons ainsi, de façon monodirectionnelle, l'effet de la réalisation correcte (ou incorrecte) de la fonction de guidage.

L'ensemble de ses informations nous permet ainsi de définir un ensemble raffiné d'entrées et de sorties nous permettant de propager les évènements du composant et de son environnement. La figure 3.8 a pour vocation de présenter de manière graphique cet ensemble.

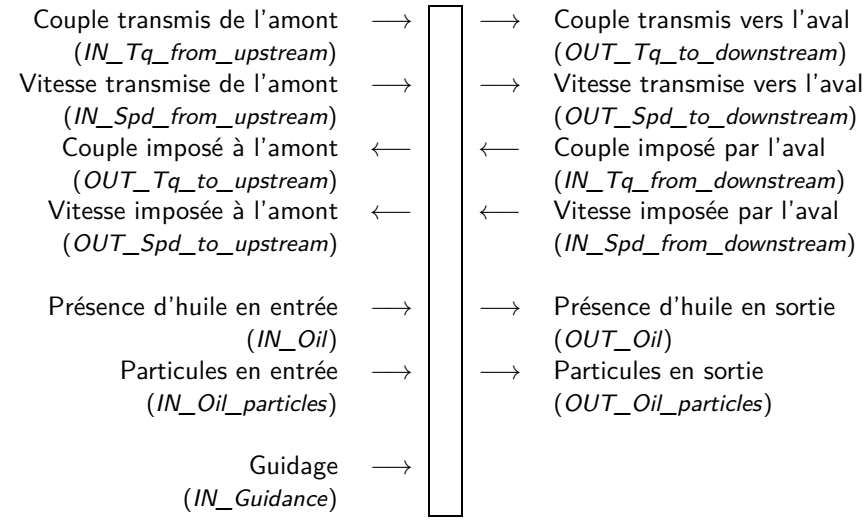


Figure 3.8 – Identification des E/S raffinées d'un engrenage

3.4.5.2 Identification de la granularité des variables de flux

Une fois l'ensemble de variables allant nous permettre de propager nos évènements fonctionnels et dysfonctionnels déterminé, il est nécessaire de nous intéresser à la détermination du niveau de granularité de ces variables. Par « niveau de granularité », nous rappelons ici que nous entendons « niveau de détail » ou alors « domaine de définition » ; nous rappelons également que ces variables seront discrètes.

Remarque : À aucun moment nous ne prétendrons fournir LA solution optimale permettant de répondre à nos objectifs. La solution proposée ici nous semble cependant satisfaisante et pertinente pour propager les évènements fonctionnels et dysfonctionnels identifiés et observer les évènements redoutés considérés. La solution proposée est alors et aussi à mettre en relation avec le cas d'étude proposé.

Variable « couple » et « vitesse »

Nous nous intéressons pour commencer à la détermination du niveau de granularité des variables représentant le « couple » et la « vitesse ». Pour déterminer ce niveau de granularité, nous prenons le parti d'inclure les informations nécessaires et suffisantes à la propagation des évènements (fonctionnels et dysfonctionnels) considérées et à l'observation des évènements redoutés à étudier.

Ici et en adéquation avec le tableau 3.2, nous souhaitons observer une perte de la génération de mouvement et une survitesse de la transmission. Nous souhaitons également pouvoir propager leurs effets à l'environnement. Nous choisissons comme granularité :

- le domaine de définition {ok, nul} pour les variables « couple » ;
- le domaine de définition {ok, nulle, survitesse} pour les variables « vitesse ».

Exemple : Si on se contente ici et pour l'exemple de considérer la transmission de mouvement entre l'engrenage et ses composants en aval :

- une perte de la génération de mouvement sera caractérisée par un couple transmis égal à « nul » et une vitesse transmise égale à « nulle » ;
- une survitesse sera caractérisée par un couple égal à « ok » et une vitesse égale à « survitesse ».

Variables relatives à la lubrification

Ici, nous nous intéressons à la détermination du niveau de granularité des variables relatives à la lubrification de notre engrenage, *i.e.* à la variable modélisant la présence d'huile ainsi qu'à la variable représentant la présence de particules métalliques dans l'huile. Pour les comportements de notre système, ce qui nous intéresse est de savoir : s'il y a de l'huile ou non ; si des particules métalliques sont présentes ou non dans l'huile. En particulier, l'observation de modes dégradés ne nous intéresse pas ici. Pour cette raison, nous choisissons comme granularité :

- le domaine de définition {yes, no} pour les variables d'E/S « présence d'huile » ;
- le domaine de définition {yes, no} pour les variables d'E/S « particules ».

Variable « guidage »

De manière relativement similaire aux variables modélisant la lubrification de l'engrenage, nous sommes intéressés en ce qui concerne la fonction de guidage à la bonne réalisation de cette fonction. Pour fonctionner correctement, l'engrenage doit être correctement guidé en rotation. Par conséquent, nous choisissons comme granularité pour les variables « guidage » le domaine de définition {ok, ko}.

Modélisation AltaRica

L'ensemble de flux peut à présent être raffiné et être muni, pour chacun d'entre eux, de leurs domaines de définition respectifs et de leurs orientations.

```
node engrenage
 flow
   IN_Tq_from_upstream : {ok, nul} : in ; \\ Couple transmis de l'amont
   IN_Spd_from_upstream : {ok, nul} : in ; \\ Vitesse transmise de l'amont

   OUT_Tq_to_downstream : {ok, nul} : out ; \\ Couple transmis vers l'aval
   OUT_Spd_to_downstream : {ok, nul} : out ; \\ Vitesse transmise vers l'aval

   OUT_Tq_to_upstream : {ok, nul} : out ; \\ Couple imposé à l'amont
   OUT_Spd_to_upstream : {ok, nul} : out ; \\ Vitesse imposée à l'amont

   IN_Tq_from_downstream : {ok, nul} : in ; \\ Couple imposé par l'aval
   IN_Spd_from_downstream : {ok, nul} : in ; \\ Vitesse imposée par l'aval

   IN_Oil :{yes, no} : in ; \\ Présence d'huile en entrée
   OUT_Oil : {yes, no} : out ; \\ Présence d'huile en sortie

   IN_Oil_particles : {yes, no} : in ; \\ Particules en entrée
   OUT_Oil_particles : {yes, no} : out ; \\ Particules en sortie

   IN_Guidance : {ok, ko} : in ; \\ Guidage
 ...
edon
```

3.4.6 Caractérisation des propagations des effets des défaillances

Pour aller au delà de la modélisation d'un simple composant, il est possible de définir des modes de défaillance génériques en termes d'entrées et de sorties (ou, d'un autre point de vue, de définir les effets de chaque mode de défaillance sur les E/S propagées dans le modèle). Il s'agira alors de propager dans le modèle des informations telles que « une rupture entraîne un couple transmis nul et une vitesse transmise nulle ». Nous fournissons ici divers exemples se voulant décrire comment les abstractions définies précédemment sont utilisées pour propager les effets des évènements identifiés.

Ainsi et pour commencer, le tableau 3.3 se veut modéliser une rupture de l'engrenage.

Mode nominal		**Rupture de l'engrenage**	
Sortie	*Valeur*	*Sortie*	*Valeur*
Couple transmis vers l'aval	Couple transmis de l'amont	Couple transmis vers l'aval	nul
Vitesse transmise vers l'aval	Vitesse transmise de l'amont	Vitesse transmise vers l'aval	Vitesse imposée par l'aval
Couple imposé à l'amont	Couple imposé par l'aval	Couple imposé à l'amont	nul
Vitesse imposée à l'amont	Vitesse imposée par l'aval	Vitesse imposée à l'amont	Vitesse transmise de l'amont

Tableau 3.3 – Modélisation d'un mode de défaillance du domaine mécanique : la rupture

Remarque : Un exemple de l'influence des évènements à propager sur la granularité des variables. Imaginons ici que nous voulions propager un blocage de l'engrenage. Ce blocage aurait pour effet de transmettre un couple très important en amont et en aval ainsi qu'une vitesse nulle. Ainsi pour prendre en compte un tel mode de défaillance, nous prendrions {ok, nul, important} comme domaine de définition pour la variable « couple » et le modéliserions de la façon suivante :

Blocage de l'engrenage	
Sortie	*Valeur*
Couple transmis vers l'aval	important
Vitesse transmise vers l'aval	nulle
Couple imposé à l'amont	important
Vitesse imposée à l'amont	nulle

De manière analogue, nous définissons l'effet de l'usure sur la variable « particules ». À noter que l'on ne considère pas ici que l'engrenage puisse conduire à l'arrêt de la lubrification d'huile.

Mode nominal		**Usure de l'engrenage**	
Sortie	*Valeur*	*Sortie*	*Valeur*
Présence d'huile en sortie	Présence d'huile en entrée	Présence d'huile en sortie	Présence d'huile en entrée
Particules en sortie	Particules en entrée	Particules en sortie	yes

En langage AltaRica...

Le travail présenté ci-dessus permet alors, avec la prise en compte du fonctionnement nominal (ici, le fonctionnement nominal se résume à « Mouvement sortie » = « Mouvement entrée », *i.e.* le composant est passif), de définir la valeur des sorties dans chaque mode de fonctionnement / dysfonctionnement. Les assertions AltaRica peuvent être implémentées. Les références aux différentes entrées et sorties sont illustrées sur la figure 3.8.

```
node engrenage
 ...
 assert
   OUT_Tq_to_downstream = case { ST_Transmission = ko : nul,
                                 IN_Guidance = ko : nul,
                                 else IN_Tq_from_upstream } ;

   OUT_Spd_to_downstream = case {ST_Transmission = ko : IN_Spd_from_downstream,
                                 IN_Guidance = ko : IN_Spd_from_downstream,
                                 else IN_Spd_from_upstream } ;

   OUT_Tq_to_upstream = case { ST_Transmission = ko : nul,
                               IN_Guidance = ko : nul,
                               else IN_Tq_from_downstream } ;

   OUT_Spd_to_upstream = case { ST_Transmission = ko : IN_Spd_from_downstream,
                                IN_Guidance = ko : IN_Spd_from_downstream,
                                else IN_Spd_from_downstream } ;
```

```
   OUT_Oil : = IN_Oil ;

   OUT_Oil_particles = case { ST_particles=yes : yes, else IN_Oil_particles} ;
edon
```

3.4.7 Bilan sur la modélisation d'un sous-système de transmission mécanique

Remarque : D'une manière générale, on rappelle ici que doivent être modélisées les grandeurs permettant de propager les évènements fonctionnels et dysfonctionnels. Les résultats présentés ici n'ont pas pour objectif d'être exhaustifs mais se veulent davantage être un fondement pour de futures réflexions. Le travail relaté ici et les résultats présentés sont donc à mettre en relation avec le cas d'étude considéré (système + évènements redoutés considérés) mais ne peuvent être appliqués directement.

On rappelle ici les principaux résultats traitant de la modélisation d'une chaîne d'engrenage.

- Types de composants à modéliser :
  - Engrenage
  - Roulement (à billes, à rouleaux)
  - Arbre de transmission
  - Vis
  - ...
- Types d'évènements à propager
  - Rupture de l'élément mécanique
  - Blocage de l'élément mécanique
  - Perte de la lubrification de l'élément mécanique
  - Perte du guidage et/ou du maintien en position (perte d'un roulement, perte d'une vis...) de l'élément mécanique
  - Survitesse de l'élément mécanique
  - ...
- Abstraction proposée
  - Couple
    - Granularité : {ok, nul} (ou {ok, nul, important})
    - Orientation : Bidirectionnelle
  - Vitesse
    - Granularité : {ok, nulle, survitesse}
    - Orientation : Bidirectionnelle
  - Huile
    - Granularité : {yes, no}
    - Orientation : Monodirectionnelle
  - Particules dans l'huile
    - Granularité : {yes, no}
    - Orientation : Monodirectionnelle

3.5 Modélisation d'un sous-système hydromécanique

Nous présentons ici l'application de la méthodologie proposée au sous-système hydromécanique de notre cas d'étude (figure 3.4). Rappelons ici que ce sous-système assure les fonctions de dosage et de distribution du carburant à la chambre de combustion du moteur.

3.5.1 Architecture du sous-système hydromécanique

En premier lieu, nous souhaitons identifier l'architecture du modèle allant permettre la modélisation des propagations de défaillance à l'intérieur du sous-système étudié. Sans reprendre tout le principe évoqué dans l'exemple précédent (section 3.4.1), nous identifions une architecture composée principalement de clapets, de pompes, de tuyaux ou encore de filtres. La figure 3.9 explicite le schéma de principe d'une telle architecture.

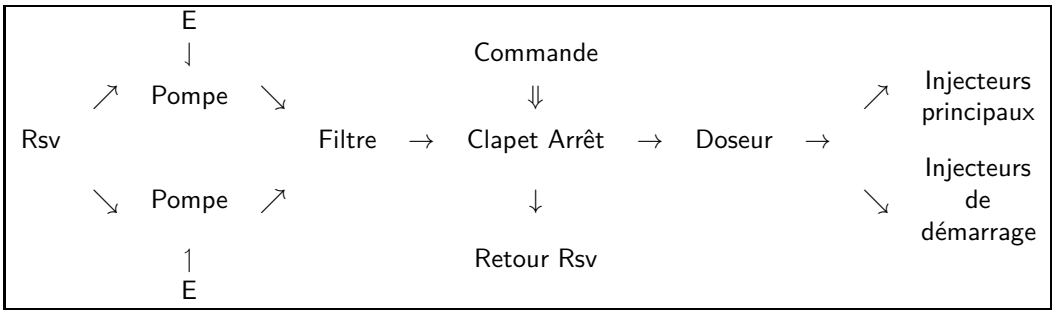


Figure 3.9 – Schéma de principe simplifié du sous-système hydromécanique
(Rsv : Réservoir ; E : Entraînement pompe ; $\rightarrow$: tuyaux ; $\Downarrow$: Commande calculateur ; $\downarrow$: Puissance électrique)

Nous identifions ainsi ce qui sera l'architecture de notre modèle : les composants à modéliser et les interfaces entre ceux-ci. Ici, le principe d'instanciation du langage AltaRica conduira à ne modéliser qu'un nœud « Pompe » au sein de la bibliothèque pour modéliser les deux pompes présentes dans le modèle et qu'un nœud « Injecteur » pour modéliser les deux types d'injecteurs. Pour chacun des nœuds, deux instances seront présentes dans le modèle.

Dans la suite de cette section, nous illustrerons le principe de modélisation en nous appuyant sur l'exemple du clapet d'arrêt. Promptement et pour bien comprendre le fonctionnement de ce mécanisme, le clapet d'arrêt peut être vu comme un robinet à une entrée et deux sorties correspondant à deux positions possibles : une, que l'on nommera « position 1 » et qui conduira le fluide de l'entrée à la première sortie ; la seconde, « position 2 » qui conduira le fluide de l'entrée à la seconde sortie. La mise dans l'une ou l'autre des positions est commandée par un ordre extérieur.

Remarque : Pour des raisons de confidentialité, le mécanisme du clapet est ici simplifié par rapport à son utilisation réelle.

3.5.2 Une première identification des entrées / sorties

Dans un premier temps, nous souhaitons identifier un premier ensemble d'entrées et de sorties de haut niveau. Pour cela, le schéma de principe du clapet étudié (figure 3.10) fournit une aide pour caractériser les interfaces que le composant peut avoir avec son environnement.

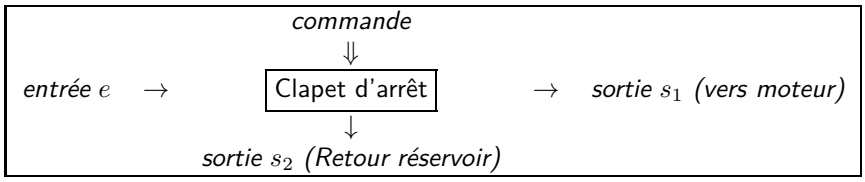


Figure 3.10 – Schéma de principe du clapet d'arrêt
($\rightarrow$: fluide ; $\Downarrow$: Commande calculateur)

Un premier ensemble d'entrées et de sorties est alors identifié dans l'optique de modéliser les propagations ayant lieu au sein du composant :

- deux entrées :
  - l'entrée « e » par laquelle arrive le fluide (ici du carburant),
  - l'entrée « *commande* » commandant la position du clapet (position 1 ou position 2) ;
- deux sorties :
  - la sortie « s_1 » par laquelle est transmis le fluide lorsque le clapet est en position 1,
  - la sortie « s_2 » par laquelle est transmis le fluide lorsque le clapet est en position 2.

3.5.3 Caractérisation du comportement interne du clapet d'arrêt

Le but est ici d'identifier les évènements fonctionnels et dysfonctionnels à intégrer dans le modèle, les modes de fonctionnement et de dysfonctionnement ainsi que les transitions entre ces modes. À partir de l'AMDE et en accord avec la section 3.2.2, les évènements identifiés sont :

- la mise du clapet en position 1 ou en position 2 - On nommera CP1 et CP2 (pour respectivement Commande en Position 1 et 2) ces deux évènements ;
- le blocage du clapet dans l'une ou l'autre des positions - On nommera BPOS (pour Blocage en POSition) cet évènement ;
- le blocage du clapet en position intermédiaire (dans cette position, le fluide est bloqué et ne sort ni par s_1, ni par s_2) - On nommera cet évènement CLG pour CLoGing (colmatage) puisque cela peut être vu de manière évidente comme similaire à un colmatage du circuit.

Une fois ces évènements identifiés, les modes de fonctionnement et de dysfonctionnement sont en mesure d'être déterminés. Ici et en relation avec les évènements retenus, nous avons :

- deux modes de fonctionnement, lorsque le clapet est en position 1 ou lorsque le clapet est en position 2 (de manière volontaire) ;
- trois modes de dysfonctionnement, lorsque le clapet est bloqué en position 1, lorsque le clapet est bloqué en position 2 ou lorsque le clapet est bloqué en position intermédiaire.

En combinant la connaissance des évènements et des modes de fonctionnement / dysfonctionnement, il est possible d'écrire les différentes transitions entre ces modes. Une représentation possible est un diagramme état - transition tel que celui présenté sur la figure 3.11.

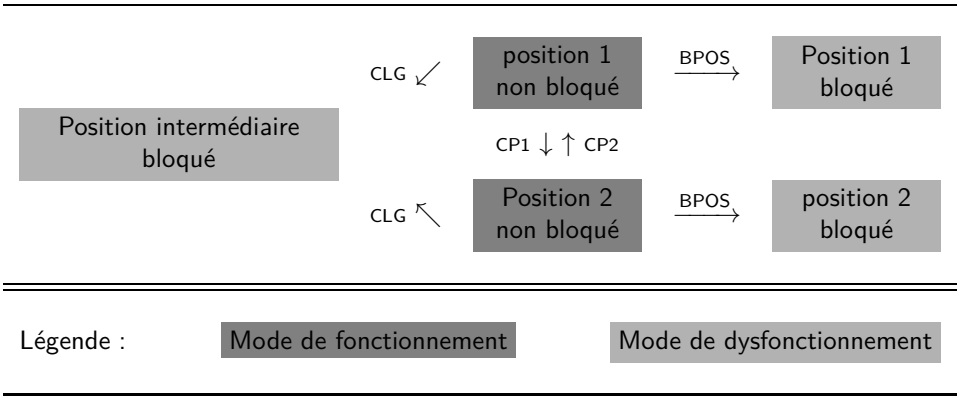


Figure 3.11 – Diagramme état / transition représentant les transitions entre modes de fonctionnement et de dysfonctionnement du clapet d'arrêt

La figure 3.11 se traduit, en langage AltaRica, par la création de deux variables d'état : $ST\_position$ représentant la position du clapet et prenant ses valeurs dans {*pos1*, *pos2*, *intermediate*} et $ST\_blocage$ que nous considérerons booléen et qui décrit si le clapet est bloqué ou non ($ST\_blocage = false$ lorsque le clapet n'est pas bloqué, $ST\_blocage = true$ lorsque le clapet est bloqué). Initialement le clapet est en position 1 et n'est pas bloqué. Ainsi, les évènements et les transitions du nœud AltaRica peuvent être implémentés.

3.5.4 Identification des évènements externes à propager

Pour rappel, la motivation première du travail exposé ici est de propager, grâce au modèle du composant, à la fois les effets de ses propres évènements et les évènements de son environnement sur lesquels il agit ou non. Dans cette optique, nous recherchons les modes de défaillance des autres composants du système et en tentons d'en tirer un ensemble de modes de défaillance typique du domaine. Dans le système hydromécanique, il s'agira ici de caractériser comment le composant réagit à une rupture du circuit (la totalité du fluide sort du circuit), à un colmatage du circuit (ici et par exemple, le blocage du clapet en position intermédiaire peut être vu comme analogue à un colmatage) ou à une dérivation du fluide s'écoulant dans le circuit (une partie du fluide sort du circuit). Il faudra considérer et modéliser les effets de ces évènements qu'ils aient lieu en aval ou en amont du composant étudié.

3.5.5 Caractérisation de l'abstraction utilisée

Avant de poursuivre, on rappelle les différents évènements que le composant se devra de décrire et de propager :

- la mise en position 1 ou position 2 du clapet ;
- le blocage du clapet en position 1 ou en position 2 ;
- un blocage du clapet en position intermédiaire ;
- une rupture du circuit en amont ou en aval du composant ;
- un colmatage (ou assimilé) du circuit en amont ou en aval du composant ;
- une dérivation du fluide s'écoulant dans le circuit (la défaillance provoquant cette dérivation ayant alors lieu en amont ou en aval du composant).

Exemple : Ces effets seront dans la suite propagés via les E/S qui seront identifiées. Par exemple à l'heure actuelle et en reprenant les notations de la figure 3.10 :

- la mise en position 1 se traduit par $(s_1, s_2) = (e, 0)$;
- la mise en position 2 se traduit par $(s_1, s_2) = (0, e)$;
- un blocage du clapet en position intermédiaire se traduit par $s_1 \neq e$ et $s_2 \neq e$;
- une rupture (ou un colmatage total) du circuit en amont se traduit par $e = 0$.

3.5.5.1 Identification des E/S raffinées

Pour raffiner les entrées et sorties de haut niveau précédemment définies, nous nous intéressons à la description des effets sur le sous-système des différents évènements à propager. L'utilisation d'une analyse de panne (AMDE par exemple) peut être pertinente à ce stade. Ainsi, l'utilisation d'une telle analyse ainsi que la connaissance du système permet de constater que les conséquences d'un mode de défaillance d'un composant hydromécanique (comme le clapet représenté figure 3.10) peuvent être exprimées en terme de conséquences sur le débit et la pression du fluide s'écoulant à travers le composant :

- si le clapet est en position 1, le fluide s'écoule de l'entrée $entree$ à la sortie *sortie 1* ;
- si le clapet est en position 2, le fluide s'écoule de l'entrée $entree$ à la sortie *sortie 2* ;
- si le clapet est en position intermédiaire, le débit de fluide envoyé vers les sorties *sortie 1* et *sortie 2* est nul - la pression augmente en entrée du clapet - le débit de fluide passant à travers le clapet est nul ;
- en cas de rupture d'un composant (le circuit est ouvert), le débit de fluide envoyé en aval est nul ; constat identique pour la pression ;
- non représenté sur la figure 3.9, le système réel contient des mécanismes de sécurité chargés de mitiger par exemple une surpression due à un colmatage en aval de ces mécanismes. Ces colmatages doivent ainsi être propagés en amont des composants à leurs origines ;
- ...

Ainsi, une solution pertinente dans notre cas satisfaisante pour propager les évènements devant l'être est de propager, en amont et en aval (*i.e.* de manière bidirectionnelle) le couple de variable $\{$debit, pression$\}$. En effet, nous verrons par la suite que les différents évènements peuvent être exprimés en terme de conséquences sur le débit et / ou la pression du fluide s'écoulant à travers le composant. Ce couple de variable raffinera les entrées et sorties *entree*, *sortie 1* et *sortie 2* et nous permettra de propager les différents évènements devant l'être dans le modèle. De manière similaire au domaine mécanique, une défaillance n'a pas uniquement des conséquences sur les composants situés en aval mais aussi sur les composants en amont de celui étudié.

Exemple : Le blocage du clapet en position intermédiaire (ou de manière générique toute obturation ou colmatage dans le circuit) entraînera non seulement une perte du débit et de la pression en aval du clapet mais conduira également à une augmentation de la pression en amont de celui-ci.

Dans le cas réel, d'autres évènements sont à propager comme par exemple la présence de particules dans le fluide (ce qui pourrait colmater les injecteurs), l'augmentation de la température

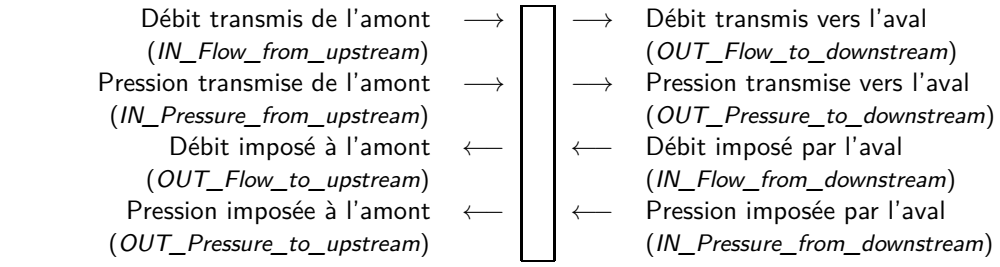


Figure 3.12 – Identification des E/S raffinées d'un composant hydromécanique

du fluide qui augmente les probabilités que certaines défaillances surviennent. Pour propager de telles défaillances, on ajoutera lorsque cela sera estimé nécessaire des variables d'entrées / sorties.

Un point méthodologique intéressant cependant : la modélisation de facteurs influençant le taux d'occurrence d'évènement. Pour modéliser ce type de comportement, on propagera les facteurs en question via des variables d'entrées / sorties et on créera différents évènements. Le principe est illustré ci-dessous.

```
node Composant
 flow
   ...
   IN_Facteur : {f1, f2} : in ;
   OUT_Facteur : {f1, f2} : out ;
 event
   // Entre E1 et E2, seul le taux change
   E1, E2 ;
 trans
   //Condition et Assignation sont identiques dans les deux transitions
   Condition & IN_Facteur=f1 |- E1 -> Assignation ;
   Condition & IN_Facteur=f2 |- E2 -> Assignation ;
 assert
   ...
   // On ne change rien, on ne fait que propager
   OUT_Facteur = IN_Facteur ;
edon
```

3.5.5.2 Identification de la granularité

Nous nous intéressons désormais à la détermination du niveau de granularité des variables définies précédemment. Comme pour le domaine mécanique, nous prenons le parti d'inclure les informations nécessaires et suffisantes à la propagation des évènements (fonctionnels et dysfonctionnels) considérées et à l'observation des évènements redoutés à étudier (tableau 3.2). Nous rappelons ici que les évènements redoutés observés sont, sur le modèle du sous-système hydromécanique, l'injection d'un trop peu ou d'un surplus de carburant dans la chambre de combustion. En combinant cela avec les évènements à propager, nous choisissons comme granularité :

- {nominal, faible, nul, important} pour les variables représentant le flux de fluide ;
- {nominale, nulle, importante} pour les variables représentant la pression de fluide.

3.5.6 Caractérisation des propagations

Comme pour le domaine de la transmission mécanique, il est possible à ce stade de définir les effets de chaque mode de défaillance sur les variables d'entrées / sorties définies. Nous fournissons divers exemples décrivant comment les abstractions définies précédemment sont utilisées pour propager les évènements identifiés (tableau 3.5).

Mode nominal		**Obturation clapet**	
Sortie	*Valeur*	*Sortie*	*Valeur*
Débit transmis vers l'aval	Débit transmis de l'amont	Débit transmis vers l'aval	nul
Pression transmise vers l'aval	Pression transmise de l'amont	Pression transmise vers l'aval	nulle
Débit imposé à l'amont	Débit imposé par l'aval	Débit imposé à l'amont	nul
Pression imposée à l'amont	Pression imposée par l'aval	Pression imposée à l'amont	importante
Rupture		**Colmatage partiel**	
Sortie	*Valeur*	*Sortie*	*Valeur*
Débit transmis vers l'aval	nul	Débit transmis vers l'aval	faible
Pression transmise vers l'aval	nulle	Pression transmise vers l'aval	Pression transmise de l'amont
Débit imposé à l'amont	Débit transmis de l'amont	Débit imposé à l'amont	min(Débit imposé par l'aval, faible)
Pression imposée à l'amont	nulle	Pression imposée à l'amont	Pression imposée par l'aval

Tableau 3.5 – Modélisation de mode de défaillance du domaine hydromécanique

3.5.7 Code AltaRica du clapet

Avant de conclure sur la modélisation d'un composant hydromécanique, on présente ici le code AltaRica correspondant au clapet étudié au long de cette section 3.5. Il est rappelé que, pour des raisons de confidentialité, le comportement du clapet est simplifié par rapport au comportement réel.

Remarque : Le code AltaRica présenté ci-dessous est *une* implémentation possible permettant de représenter le comportement du clapet. Cependant cette implémentation n'est pas unique. Par exemple et pour des raisons de lisibilité et de compréhension, les assertions sont codées de telle sorte à rendre non-évaluable le « else ». Par exemple, les deux écritures ci-dessous sont de sens équivalent. Dans la première écriture et contrairement à la seconde, le « else » ne sera jamais évalué.

- case { ST_pos=2 or ST_pos=intermediate : nul, ST_pos=1 : IN_Flow_from_upstream, else nominal } ;
- case { ST_pos=1 : IN_Flow_from_upstream, else nul } ;

```
node Composant
 flow
   IN_Flow_from_upstream : {nominal, faible, nul, important} : in ;
   IN_Pressure_from_upstream : {nominale, nulle, importante} : in ;
   OUT_Flow_to_upstream : {nominal, faible, nul, important} : out ;
   OUT_Pressure_to_upstream : {nominale, nulle, importante} : out ;

   OUT1_Flow_to_downstream : {nominal, faible, nul, important} : out ;
   OUT1_Pressure_to_downstream : {nominale, nulle, importante} : out ;
   IN1_Flow_from_downstream : {nominal, faible, nul, important} : in ;
   IN1_Pressure_from_downstream : {nominale, nulle, importante} : in ;

   OUT2_Flow_to_downstream : {nominal, faible, nul, important} : out ;
   OUT2_Pressure_to_downstream : {nominale, nulle, importante} : out ;
   IN2_Flow_from_downstream : {nominal, faible, nul, important} : in ;
   IN2_Pressure_from_downstream : {nominale, nulle, importante} : in ;

   IN_Comm : P1, P2 ;

 state
   ST_pos : {pos1, pos2, intermediate} ;
   ST_bloc : bool ;

 init
   ST_pos := pos1 ;
   ST_bloc := false ;

 event
   CP1, CP2, BPOS, CLG ;

 trans
   ST_bloc=false and ST_pos=pos1 and IN_Comm = P2  |- CP2 -> ST_pos:=pos2 ;
   ST_bloc=false and ST_pos=pos2 and IN_Comm = P1 |- CP1 -> ST_pos:=pos1 ;
   ST_bloc=false |- BPOS -> ST_bloc:=true ;
   ST_bloc=false |- CLG -> ST_bloc:=true, ST_pos:=intermediate ;

 assert
   OUT1_Flow_to_downstream = case { ST_pos=2 or ST_pos=intermediate : nul,
                                    ST_pos=1 : IN_Flow_from_upstream,
                                    else nominal } ; // "else" n'est jamais évalué

   OUT1_Pressure_to_downstream = case { ST_pos=2 or ST_pos=intermediate : nulle,
                                    ST_pos=1 : IN_Pressure_from_upstream,
                                    else nominale } ; // "else" jamais évalué

   OUT2_Flow_to_downstream = case { ST_pos=1 or ST_pos=intermediate : nul,
                                    ST_pos=2 : IN_Flow_from_upstream,
                                    else nominal } ; // "else" jamais évalué

   OUT2_Pressure_to_downstream = case { ST_pos=1 or ST_pos=intermediate : nulle,
                                    ST_pos=2 : IN_Pressure_from_upstream,
                                    else nominale } ; // "else" jamais évalué

   OUT_Flow_to_upstream = case { ST_pos=1 : IN1_Flow_from_downstream,
                                     ST_pos=2 : IN2_Flow_from_downstream,
                                     ST_pos=intermediate : nul,
```

```
                                    else nominal } ; // "else" jamais évalué

   OUT_Pressure_to_upstream = case { ST_pos=1 : IN1_Pressure_from_downstream,
                                    ST_pos=2 : IN2_Pressure_from_downstream,
                                    ST_pos=intermediate : maximum,
                                    else nominal } ; // "else" jamais évalué
edon
```

3.5.8 Bilan sur la modélisation d'un sous-système hydromécanique

Remarque : On rappelle ici que doivent être modélisées les grandeurs permettant de propager les évènements fonctionnels et dysfonctionnels. Le travail relaté ici et les résultats présentés sont donc à mettre en relation avec le cas d'étude considéré (système + évènements redoutés considérés)

On présente donc ici les principaux résultats traitant de la modélisation d'un système hydromécanique.

- Composants type à modéliser
  - Clapet
  - Pompe
  - Filtre
  - Tuyau
  - ...
- Évènements type à propager
  - Fuites plus ou moins importantes
  - Colmatage plus ou moins importants
  - ...
- Abstraction modélisée
  - Débit
    - Granularité : {nominal, faible, nul, important}
    - Orientation : Bidirectionnelle
  - Pression
    - Granularité : {nominale, nulle, importante}
    - Orientation : Bidirectionnelle
  - ...

Remarquons que les évènements à propager sont ici très génériques. En effet, les termes de fuite et de colmatage seront vus dans leurs plus large sens. Ainsi, une déviation d'un fluide pourra être vu, par le chemin d'écoulement nominal, comme une fuite ; un blocage de clapet pourra être vu comme un colmatage.

3.6 Modélisation du système d'air principal

Nous présentons l'application du processus de modélisation pour la représentation des propagations de défaillance à l'intérieur du circuit d'air principal du moteur. D'une manière plus détaillée que la simple génération de mouvement à partir de l'air ambiant, ce système a pour fonction de transformer en puissance mécanique cet air ambiant en l'accélérant, en le pressurisant, en le chauffant et en l'utilisant à travers différentes turbines.

Le principe général de la modélisation réalisée ici étant très similaire aux sous-systèmes de transmission mécanique et hydromécanique, nous ne nous focalisons que sur les détails particuliers à l'implémentation de la bibliothèque correspondante.

3.6.1 Caractérisation de l'architecture

Toujours pas de méthode miracle ici pour caractériser l'architecture du circuit d'air principal du turbomoteur. On commence par lister, à partir de l'analyse de pannes (typiquement, une AMDE), l'ensemble des composants du système ayant une influence sur l'occurrence des évènements redoutés. Pour rappel, chaque composant ayant un mode de défaillance pouvant provoquer l'occurrence d'un des évènements redoutés étudiés devra être implémenté en langage AltaRica et être dans le modèle du sous-système (et donc les composants n'ayant aucune influence sur l'occurrence de ces évènements redoutés pourront ne pas être modélisés).

Une fois que la liste des composants devant être modélisés est établie, une tâche pertinente est la réalisation d'une analyse fonctionnelle du système d'air principal. Tout comme pour les autres systèmes, une telle analyse nous permet d'identifier les liens matériels et fonctionnels entre les composants qui seront modélisés. Notons que si, dans les autres sous-systèmes, le flux de propagation était relativement intuitif et facile à identifier, cela est loin d'être le cas ici... En effet, si le flux principal peut être qualifié d'aisément identifiable à haut niveau, le nombre d'interactions laisse l'imagination se révolter à bas niveau. Une explication à cela vient du fait que dans le système étudié ici, les composants principaux doivent être décomposés en sous-composants de plus bas niveau.

Exemple : Pour obtenir le même niveau de détails que dans les AMDE, une turbine ne peut pas se contenter d'être modélisée que par l'intermédiaire d'un unique composant AltaRica. Il faut la décomposer tout d'abord en deux parties : la partie tournante et la partie statique. Influençant le comportement de la turbine et donc également à modéliser, différents éléments mécaniques (*e.g.* écrous) sont utilisés pour le maintien en position de la turbine. Bien que ces éléments ne soient pas physiquement dans le flux d'air, leurs bons fonctionnements respectifs sont nécessaires à celui de la turbine. Nous adopterons dans un tel cas une modélisation fonctionnelle plutôt que matérielle.

La structure du modèle obtenu ici est donc beaucoup plus fonctionnelle que matérielle. Ce fait se retrouvera également dans la caractérisation de l'abstraction qui comme nous le verrons se focalise sur la qualité des grandeurs propagées plutôt que sur leurs valeurs à proprement parler.

3.6.2 Caractérisation de l'abstraction

Le but est ici de définir une abstraction définissant le flux d'air propagé entre les différents composants du système d'air. Pour cela, un triplet permettant la propagation des défaillances des composants est le triplet {vitesse d'air, pression d'air, température d'air}. Pour justifier un tel triplet, rappelons que la fonction du circuit est de transformer l'air ambiant en l'accélérant, en

le mettant sous pression et en augmentant sa température. Les évènements redoutés surviennent lorsqu'une ou plusieurs de ces transformations sont réalisées de manière incorrecte.

Au niveau de la granularité de ces trois variables, nous avons choisi pour des raisons de simplicité de propager la qualité des différentes transformations plutôt que la valeur de température, vitesse ou pression. On souhaite donc observer différentes baisses ou augmentations de puissance. On choisit d'inclure dans notre granularité une valeur par observation souhaitée. Nous aurons donc, pour chacune des variables, une valeur représentant le mode nominal, une valeur par plage de diminution de puissance souhaitant être observée, une valeur par plage d'augmentation de puissance souhaitant être observée.

Exemple : Dans le cas, simple, où on souhaite observer uniquement l'arrêt en vol et l'occurrence d'une survitesse, nous aurons pour chacune des variables une granularité du type {nominal, excessif, perte} où la valeur « excessif » correspondra à une survitesse et où la valeur « perte » correspondra à l'arrêt en vol.

Aussi d'après les analyses de panne, la présence de corps solides étrangers au moteur peut entraîner des défaillances, par exemple au niveau des compresseurs ou des turbines. Il nous faut donc propager cette information dans le modèle. Pour propager la présence ou l'absence de tels corps, nous choisissons de propager dans le modèle une unique variable de type booléen.

3.6.3 Bilan sur la modélisation du circuit d'air principal du moteur

Remarque : On rappelle ici que doivent être modélisées les grandeurs permettant de propager les évènements fonctionnels et dysfonctionnels. Le travail relaté ici et les résultats présentés sont donc à mettre en relation avec le cas d'étude considéré (système + évènements redoutés considérés).

On présente donc ici les principaux résultats traitant de la modélisation du circuit d'air principal du moteur.

- Composants type à modéliser
  - Pales (de compresseurs, de turbines...)
  - Arbres de transmission
  - Éléments de carter
  - Chambre de combustion
  - ...
- Évènements type à propager
  - Rupture, Explosion, Crique
  - Érosion
  - ...
- Abstraction modélisée
  - Température
    - Granularité : {nominal, excessive, nul}
    - Orientation : Monodirectionnelle

- Pression
  - Granularité : {nominal, excessive, nul}
  - Orientation : Monodirectionnelle
- Vitesse
  - Granularité : {nominal, excessive, nul}
  - Orientation : Monodirectionnelle
- Corps étranger
  - Granularité : Booléenne
  - Orientation : Monodirectionnelle

Remarque : Au niveau du sens de propagation des flux, il peut être surprenant en comparaison avec la modélisation des systèmes mécaniques (section 3.4) et hydromécaniques (section 3.5) de n'utiliser qu'une propagation monodirectionnelle. Cela s'explique par le fait que dans le cas d'étude considéré, il a été montré par des calculs numériques hors du périmètre des travaux présentés ici que le flux d'air était monodirectionnel et qu'il ne pouvait ainsi pas « revenir en arrière » dans le moteur. Notons que si le choix du cas d'étude avait été différent, la propagation des flux aurait été potentiellement bidirectionnelle.

3.7 Modélisation d'un sous-système logiciel [20]

Les résultats présentés dans cette section sont fondés sur le retour d'expérience tiré de différentes références comme [20, 34]. De manière logique, puisque la méthodologie présentée dans ce mémoire en est inspirée, on retrouve dans ces références les principales étapes utilisées pour la modélisation de systèmes physiques (mécanique, hydromécanique) présentés section 3.4 à 3.6.

3.7.1 Caractérisation de l'architecture

Comme dans les cas présentés auparavant, la construction du modèle AltaRica est fondée sur des spécifications fonctionnelles et sur des analyses de pannes.

- À partir des spécifications fonctionnelles, nous extrayons la liste des composants (des fonctions) à modéliser, les dépendances fonctionnelles (*i.e.* quel composant utilise quels résultats de quels autres composants ?), les modes de fonctionnement de ces composants ainsi que les dispositifs utilisés pour détecter et contrôler les modes de défaillance prévus.
- À partir des analyses de pannes, nous identifions les modes de défaillance, leurs effets sur l'environnement du composant ainsi que les modes de dysfonctionnement du composant.

À partir de ces documents, l'architecture du modèle est donc identifiée. Pour chacun des composants considérés, nous identifions ses interfaces avec son environnement et en déduisons ses ports d'entrées / sorties. Notons sur ce point une différence avec les domaines physiques précédemment étudiés. En effet, ici l'étude est la plupart du temps effectuée au niveau fonctionnel et l'information propagée est ainsi souvent limitée à la qualité du signal transmis d'un composant à l'autre (d'une fonction à l'autre). Aussi, la propagation n'a souvent qu'à être monodirectionnelle dans le sens où la défaillance d'une fonction f n'influe que les fonctions situées en aval de f (n'influe que les fonctions utilisant spécifiquement le résultat de f).

3.7.2 Caractérisation des comportements

L'identification des évènements à propager utilise également la spécification fonctionnelle du composant ainsi que son analyse de panne.

- Dans la spécification fonctionnelle, nous identifions les modes d'utilisation de la fonction (si celle-ci peut être utilisée dans plusieurs configurations par exemple) et les transitions entre ces modes.
- Dans les analyses de pannes, nous identifions les modes de défaillance et les modes de dysfonctionnement de la fonction.

Concernant ces analyses de pannes et dans l'optique d'obtenir un certain niveau d'abstraction dans notre modèle, différents modes de défaillance génériques peuvent être définis. De [20], nous tirons la liste non exhaustive présentée dans le tableau 3.6.

Défaillances	Effets des défaillances
Défaillances externes traduites par la qualité des entrées	
• Une entrée est absente, le logiciel n'est pas protégé	• Les résultats des calculs dépendant de cette entrée peuvent être incorrects
• Une entrée est incorrecte, le logiciel n'est pas protégé	• Les résultats des calculs dépendant de cette entrée peuvent être incorrects
Défaillances induites par une erreur de mode	
• Le mode d'utilisation de la fonction n'est pas cohérent avec la phase opérationnelle effective	Les résultats des calculs dépendant du mode peuvent être incorrects
Défaillance de la fonction	
• La fonction est perdue, aucun calcul n'est effectué	• Aucune sortie n'est produite
• La partie « calcul de données » est erronée	• Les sorties sont produites mais sont incorrectes quelle que soit la qualité des valeurs en entrée
Défaillance de la détection	
• La détection est intempestive	• Quelle que soit la situation, une erreur est signalée
• La détection est inefficace ou perdue	• Quelle que soit la situation, aucune erreur n'est signalée

Tableau 3.6 – Défaillances d'une architecture logicielle et leurs effets [20]

Ensuite, l'utilisation conjointe de la spécification fonctionnelle (celle-ci nous fournit la qualité des données observées par les dispositifs de détection de défaillance) et de l'analyse de panne (celle-ci nous fournit la qualité des données après défaillance), il nous est possible d'identifier une abstraction nous permettant de modéliser les propagations de panne au sein d'un système logiciel. En se référant au tableau 3.6, les valeurs abstraites d'une données sont par exemple {correcte, incorrecte, absente}.

Une fois cette abstraction déterminée, [20] construit des tables de décisions pour décrire les calculs en tenant compte des modes de fonctionnement et de dysfonctionnement. Ces tables sont construites pour chaque flux de sortie du nœud AltaRica et sont de la forme « si le composant est dans le mode de (dys)fonctionnement X et si les valeurs des flux d'entrées satisfont la contraintes C, alors la valeur du flux de sortie est Y ». De telles règles peuvent facilement être implémentées sous la forme d'assertions AltaRica.

Une fois la bibliothèque de fonctions logicielles implémentée, on construit le modèle du système logiciel étudié par instanciation et interconnexion des nœuds de cette bibliothèque. Les références [20] et [34] fournissent des exemples d'applications d'une telle méthodologie.

3.7.3 Bilan sur la modélisation d'un sous-système logiciel

Cette section a donc eu pour objectif de décrire le retour d'expérience concernant la modélisation AltaRica d'un système logiciel tels que présentés dans [20] et [34]. Parmi les points majeurs à retenir :

- la modélisation d'un système logiciel s'appuie sur des spécifications fonctionnelles des composants (des fonctions) à modéliser ainsi que sur leurs analyses de pannes ;
- des modes de défaillance génériques peuvent être définis : perte d'une fonction, réalisation erronée d'une fonction, réalisation intempestive d'une fonction... ;
- la propagation des évènements est monodirectionnelle ;
- l'abstraction est relative à la qualité des données transmises - Par exemple, on prendra {correcte, incorrecte, absente}. Si des tests de détection d'erreurs sont présents, on propagera également l'information de détection, ou de non détection. On prendra dans ce cas et par exemple {correcte, erronée non détectée, erronée détectée, absente}.

3.8 Formalisation des données extraites

L'approche proposée dans ce chapitre permet donc de « préparer » la modélisation AltaRica de systèmes physiques. L'objectif est alors de faciliter le support des analyses de Sûreté de Fonctionnement effectué sur le système. Or, dans le domaine industriel, il est relativement courant de devoir gérer plusieurs configurations d'un même système. Par exemple :

- si un équipement du système est fourni par différents fournisseurs ;
- si des modifications ont eu lieu depuis la mise en service initiale du système, celui-ci aura plusieurs versions en service.

Dans les deux exemples cités ici, nous avons à chaque fois plusieurs configurations d'un même système et donc plusieurs analyses de Sûreté de Fonctionnement à faire vivre et à maintenir en parallèle. Les résultats présentés au cours de ce chapitre restant à un stade informel (on parlera de « spécification informelle » de nœud AltaRica), une attitude naturelle conduit à s'intéresser à la formalisation des données récoltées à l'aide de notre méthodologie. Nous pourrons alors parler de « spécification formelle » ou à défaut de « spécification semi-formelle » de nœud AltaRica. Dans la suite, nous nous permettrons d'utiliser uniquement le terme « spécification ».

Remarque : Nous parlons ici de spécification de nœud AltaRica. Nous pourrions potentiellement étendre la visée de la démarche et parler de « spécification de modèle de propagations de pannes ». Le langage AltaRica ne sera que l'outil permettant d'implémenter les informations recueillies grâce à l'approche proposée et formalisées dans la spécification évoquée ici.

Plusieurs questions peuvent alors se poser. Comment présenter cette spécification ? Comment la formaliser ? Que doit elle contenir ? Dans l'optique de répondre à de telles questions, nous nous intéresserons à la présentation de différents formalismes que nous croyons propices à la représentation de spécifications. Avant cela, nous nous permettons de fournir quelques arguments justifiant le besoin de cette formalisation.

3.8.1 Pourquoi formaliser

On précise ici un ensemble, sans doute non exhaustif, de raisons, de justifications ou d'avantages nous poussant à nous intéresser à la formalisation des données extraites aux sections précédentes.

- améliorer la traçabilité entre le système réel et le modèle ;
- faciliter la communication au sein d'un projet grâce à un formalisme commun aux différents acteurs ;
- faciliter la mise à jour des études de Sûreté de Fonctionnement lors, par exemple, de l'intégration au sein du système de nouveaux équipements ;
- faciliter la création de nouvelles études de Sûreté de Fonctionnement pour de nouvelles configurations du système ;
- gérer simultanément plusieurs configurations de systèmes ;
- décrire de manière identique les dysfonctionnements d'un même composant réalisant les mêmes fonctions dans le même environnement ;
- standardiser les termes utilisés dans les analyses de Sûreté de Fonctionnement (*e.g.* codification des modes de défaillance) :
  - → assurer la cohérence à l'intérieur d'une analyse de Sûreté de Fonctionnement et entre des analyses différentes,
  - → éviter la multiplication des données identiques (*i.e.* éviter d'avoir à plusieurs endroits la même information) ;
- réduire le nombre d'interprétation possible des informations présentes dans une analyse de sécurité ;
- faciliter la revue des modèles réalisés par les experts et / ou ingénieurs ;
- capitaliser le travail effectué...

... pour reprendre ce dernier point, nous pensons que cette spécification est plus qu'une simple aide à la modélisation, qu'une simple « préparation » du modèle AltaRica. En effet, elle sera un moyen performant de « capitaliser » les informations identifiées et présentées auparavant. Elle devra pour cela décrire de manière aussi formelle que possible le comportement du composant en incluant la description de ses interfaces avec son environnement et la description des différentes variables utilisées. Par exemple, cette spécification devra contenir une description détaillée de l'ensemble des variables utilisées en incluant leurs justifications (pourquoi avoir choisi ces variables, pourquoi avoir choisi ce domaine de définition) et leurs significations (à quoi correspond telle valeur de telle variable ?).

Avant de s'intéresser plus amplement à la forme de cette nécessaire (nous en sommes désormais convaincus) formalisation, nous donnons ici quelques contraintes qui guideront (qui devront guider) sa mise en place.

3.8.2 Des contraintes sur cette formalisation

Premièrement, si on ne demande pas à la spécification de représenter obligatoirement la totalité des comportements du système étudié, ceux représentés devront l'être de manière correcte. Second point, à mettre en relation avec le premier, la forme de cette spécification se devra d'être compréhensible et comprise des experts et des ingénieurs. En effet, cette spécification (et les informations contenues à l'intérieur) servira de fondement au modèle AltaRica devant être construit. Une information incorrecte à l'intérieur de cette spécification aura donc des chances

non négligeables de conduire à un modèle AltaRica incorrect. Ainsi, cette spécification devra être validée avant son utilisation par les experts et / ou les ingénieurs.

Remarque : Au sujet de cette validation et en prenant de l'avance sur le prochain chapitre, nous sommes d'accord que valider cette spécification par une « simple » relecture peut sembler étrange... Cependant, nous croyons que posséder une représentation du comportement du composant étudié est un pré-requis dans la construction d'un nœud AltaRica. De plus, une telle spécification pourra, comme nous l'observerons au chapitre 4, être utile pour la validation des nœuds modélisés. Enfin, la spécification et l'implémentation du modèle peuvent être réalisés par des personnes différentes. On introduit ainsi deux analyses indépendantes, ce qui augmente les chances de détecter des erreurs potentielles.

Toujours dans l'optique de conserver cette spécification compréhensible par les experts et ingénieurs, on se limitera souvent à la représentation du comportement d'un composant élémentaire (comme l'engrenage ou le clapet étudiés dans les sections précédentes). En effet, représenter un trop grand nombre de comportement pourra souvent devenir problématique non seulement au niveau de la validation de cette spécification (celle-ci peut potentiellement devenir illisible) mais aussi au niveau de sa construction. Nous prendrons donc comme hypothèse qu'une spécification représentera les comportements d'un nœud élémentaire AltaRica. On aura donc autant de spécifications que de nœuds présents dans la bibliothèque (*i.e.* une spécification du comportement du clapet, une spécification du comportement de l'engrenage...).

Enfin et en prenant de l'avance sur le prochain chapitre, cette spécification devra permettre la génération automatique de scénarios de tests (que nous simulerons ensuite sur le modèle AltaRica). Pour cela, le formalisme choisi devra permettre cette génération. Sans davantage de détails pour l'instant et sans imposer de méthode particulière, nous présentons dans la suite de cette section différents formalismes utilisés dans la littérature pour mettre en forme et formaliser (ou au moins semi-formaliser) les informations à inclure dans le modèle.

3.8.3 AMDE formelle

La proposition serait ici de formaliser les informations présentes dans les actuelles analyses de pannes (*i.e.* dans les AMDE). Plutôt que de décrire les effets en langage naturel, on introduirait, dès la construction de cette analyse de panne, la notion d'état du système. On pourrait également introduire dans l'analyse de panne, d'une façon similaire à [49], les effets d'une déviation d'entrée sur les états et les sorties des composants. [49] parle alors de IF-FMEA pour « Interface Focused » FMEA, *i.e.* AMDE orientée sur les interfaces que le composant possède avec son environnement. Le tout peut alors être présenté sous une forme tabulaire comme le montre le tableau 3.7.

Mode de défaillance sur la sortie	Description	Déviation entrée	Mode de défaillance composant	Taux
$OUT = Nul$	...	$IN = Nulle$	-	-
	...	-	E_1	$1.5e^{-6}$
	...	-	E_2	$2e^{-6}$
	...	$IN = low$	E_3	$1e^{-5}$
$OUT = low$	...	$IN = low$	-	-
	...	-	E_3	$1.5e^{-6}$
...	...	...	...	...

Tableau 3.7 – Proposition du principe d'une AMDE formelle

On représente en fait sur ce tableau 3.7 les modes de défaillance sur les sorties des composants. Un avantage de cette méthode est la mise à disposition de scénarios de tests quasi-immédiat. En effet, pour un composant donné, le tableau obtenu nous fournit un ensemble de scénario munis de déviations d'entrées et / ou de modes de défaillance internes au composant. De ce tableau également, nous connaissons la sortie attendue du modèle. Il sera donc possible de comparer la sortie effective du modèle avec l'effet décrit dans la première colonne du tableau.

Cependant, il est difficile à partir de ce formalisme d'appréhender l'historique d'un état du système, de tracer les scénarios de pannes possibles (*i.e.* le chemin ayant conduis à telle ou telle valeur de sortie). De plus et en l'état, ce formalisme est plutôt utilisé pour la représentation du comportement dysfonctionnel du composant. Pour aller plus loin, différents travaux présentés dans la littérature se proposent de représenter le comportement (fonctionnel et dysfonctionnel) d'un système à l'aide de diagrammes états-transitions. On trouvera alors des références utilisant UML, Statecharts, ou la méthode SCR.

3.8.4 Méthode SCR

La méthode SCR (pour Software Cost Reduction [32]) est à la base une méthode se voulant permettre de spécifier formellement des exigences sur un système à l'aide d'une représentation tabulaire. Introduit à l'origine pour décrire les comportements d'un logiciel de manière précise et non ambiguë, la méthode s'est peu à peu étendue à une gamme plus large de système. Pour aider à l'application pratique de la méthode, celle-ci est supportée par une suite d'outils incluant par exemple un éditeur (pour écrire formellement l'ensemble des exigences), un outil de vérification de la cohérence de cette spécification (pour trouver des « oublis », des cas manquants dans la spécification), un simulateur interactif, un vérificateur permettant de tester si la spécification possède une propriété donnée.

Le principe de la méthode SCR a des similarités avec le principe du langage AltaRica. Brièvement et pour ne présenter que très simplement la chose, un système (ou un logiciel) est représenté à l'aide de différentes variables. Nous en considérerons trois ici : des variables d'entrée, des variables de sorties et des variables d'état. La spécification, présentée sous forme tabulaire, consiste tout d'abord à décrire la logique de changement d'état, pour chaque variable d'état. Le principe est illustré sur le tableau 3.8.

Remarque : Un remarque au sujet des évènements modélisés. Ils le sont par l'intermédiaire d'un changement de valeur des variables d'entrée et sont formalisés grâce à une écriture du type « @T (Changement de l'entrée) ». Dans les exemples vus dans la littérature, les défaillances propres au système ne sont pas modélisées.

L'exemple est celui d'un système d'alarme observant le niveau d'eau d'une cuve. Le système reçoit en entrée un signal IN lui indiquant le niveau d'eau (ok, bas ou $haut$). Le logiciel traite ce signal et si besoin, renvoie en sortie un signal d'alarme OUT (OUT sera vrai si le niveau est haut ou bas). Le système possède également une entrée $switch$ lui indiquant s'il est allumé ou éteint ($switch = ON$ ou OFF).

Valeur de départ	Évènement	Valeur d'arrivée
ok	@T($IN <$ Seuil bas)	bas
ok	@T($IN >$ Seuil haut)	$haut$
bas	@T($IN >$ Seuil bas)	ok
$haut$	@T($IN <$ Seuil haut)	$haut$

Tableau 3.8 – Méthode SCR : Table de transition pour une variable d'état ST

Une variable d'état peut donc être vue comme étant une abstraction d'une variable d'entrée. Une fois la logique de changement d'état définie, on définit les valeurs des variables de sortie (représentant ici une activation d'alarme). Le tableau 3.9 illustre la démarche. Sur ce tableau, l'alarme sera activée lorsque le niveau sera $haut$ ou bas et que le système est en fonctionnement ($switch$=ON).

Nom sortie	**Variable d'état**	
OUT	ST	
État	**Évènement**	
$haut$, bas	True when $switch = ON$	@T(Switch = OFF)
ok	False	True
OUT	ON	OFF

Tableau 3.9 – Méthode SCR : Définition de la valeur d'une variable de sortie

De cette méthode SCR, il est bien entendu hors de propos de reprendre la totalité de la démarche. La méthode SCR est un processus à part entière et de nombreuses références traitent du sujet [32, 11]. Cependant, la notation tabulaire semble intéressante et semble pouvoir permettre la description des comportements d'un composant.

En y regardant de plus près, cela n'a cependant rien d'étonnant puisque derrière cette notation tabulaire, se cache en fait des machines à état finis (*i.e.* , des diagrammes états-transitions). Les tables décrites lors de l'application de la méthode SCR peuvent donc potentiellement être considérées comme une « traduction » tabulaire des informations décrites par un diagramme état-transition. Pour faire l'analogie avec le langage AltaRica, les tables de transitions (tableau 3.8) sont analogues aux transitions AltaRica, la définition des variables de sortie (tableau 3.9) sont l'analogie des assertions AltaRica.

3.8.5 Diagrammes états-transitions

Au delà de la méthode SCR (qui propose une description tabulaire du comportement d'un système), il est possible de décrire graphiquement le comportement d'un système à l'aide de diagrammes états-transitions. On trouvera différents travaux utilisant les machines à état finis, les Statecharts ou encore les diagrammes d'états du formalisme UML. Sans rentrer dans les détails et spécificités de chacun des formalismes, nous retiendrons ici que de tels formalismes permettent de décrire la logique de changement d'état à l'intérieur d'un composant. Elle nécessite donc de définir ces différents états du système puis de s'intéresser aux mécanismes (*i.e.* aux évènements fonctionnels ou dysfonctionnels) provoquant un changement d'état.

Pour présenter différents types de diagrammes états-transitions, nous prenons l'exemple simple d'un robinet pouvant d'ouvrir et se fermer (pour l'instant on suppose que celui-ci n'a pas de mode de défaillance). La figure 3.13 fournit alors un diagramme états-transitions simple représentant le fonctionnement du robinet.

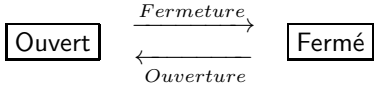


Figure 3.13 – Exemple de diagramme états-transitions

Remarque : Ces diagrammes permettent ainsi une représentation du comportement interne d'un système. Pour que cette représentation reste simple et lisible, on se limitera à représenter le comportement d'un composant élémentaire.
Remarquons également que les étiquettes des transitions permettent la représentation du comportement interne (changement d'état suite à un évènement interne du composant) mais aussi des changements d'états suite à un changement des valeurs d'entrées (figure 3.15).

3.8.6 Diagrammes états-transitions étendus pour les automates de mode

Pour aller plus loin, ces diagrammes états-transitions sont en mesures de représenter les valeurs des variables de sortie du composant. Pour appréhender cela, nous pouvons nous rapprocher de la notion d'automate de mode (Cf. section 2.3) qui rappelons-le se propose d'adjoindre la notion de flux aux diagrammes états-transitions classiques. Sans prétendre fournir un automate de mode complet, nous nous proposons de compléter les diagrammes états-transitions en fournissant dans chaque état les équations définissant les valeurs des variables de sortie. Ainsi, on combine la vue graphique des diagrammes états-transitions avec les tables de décision permettant de définir les sorties en mode de fonctionnement (esprit semblable à la méthode SCR) où en mode de dysfonctionnement (esprit semblable à l'AMDE formelle). La figure 3.13 devient alors la figure 3.14.

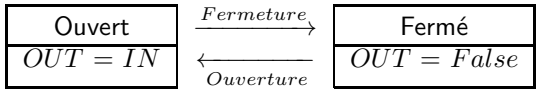


Figure 3.14 – Exemple de diagramme états-transitions avec définition des variables de sorties

Pour plus de détails dans la représentation graphique et lorsque les équations définissant les sorties sont complexes, il peut être souhaité de représenter le diagramme des configurations (figure 3.15). Si on gagne en lisibilité dans les écritures des équations, on augmente le nombre de « boîtes » dans le diagramme.

Remarque : Pour appréhender correctement les termes, nous appellerons indifféremment « mode » ou « état » les états du composant modélisé (*i.e.* les boîtes du diagramme états-transitions). Nous appellerons « configuration » l'association d'un état et d'une évaluation des variables d'entrée.

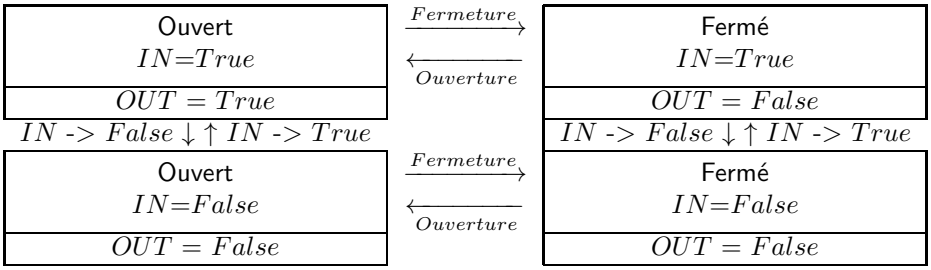


Figure 3.15 – Exemple de diagramme des configurations

3.8.7 Proposition de spécification

Nous proposons ici une solution nous semblant pertinente pour formaliser les différentes informations obtenues grâce à la méthodologie de modélisation. Tout d'abord, nous formalisons et documentons les variables de flux (les entrées et les sorties), les variables d'état et leurs types respectifs. Notons que la formalisation de ces types n'est souvent pas spécifique au composant et sera commun à l'ensemble des composant du modèle.

Nom	Type	Granularité	Description
Couple	Énuméré	{ok, nul}	Décrit le couple transmis
Vitesse	Énuméré	{ok, nulle, survitesse}	Décrit la vitesse transmise
...	...	...	...

Tableau 3.10 – Spécification : Liste des types

Nom	Type	Orientation	Description
IN_Tq_from_upstream	Couple	Entrée	Couple reçu des composants en amont
...	...	...	...
OUT_Tq_to_downstream	Couple	Sortie	Décrit le couple transmis en aval du composant
OUT_Spd_to_upstream	Vitesse	Sortie	Décrit la vitesse imposée aux composants amonts par l'ensemble {composant + composants avals}
...	...	...	...

Tableau 3.11 – Spécification : Liste des variables d'entrées / sorties

Nom	Type	Init.	Description
ST_transmission	{ok, ko}	ok	Représente l'état de la transmission
ST_particles	{no, yes}	no	Représente si le composant émet, ou non, des particules
...	...	...	...

Tableau 3.12 – Spécification : Liste des variables d'état

Une fois ces différentes informations capitalisées, on s'intéresse aux transitions entre états ainsi qu'aux équations définissant les variables de sorties. Pour cela, nous choisissons ici une représentation graphique et laissons le choix à l'analyste de choisir entre :

- un diagramme états-transitions « simple » tel que celui montré sur la figure 3.14 ;
- un diagramme représentant les changements de configurations (figure 3.15) ;
- un diagramme hybride ne s'intéressant aux configurations que lorsque cela est nécessaire.

Si une recommandation devait être fournie quant au choix de la représentation, celle-ci serait guidé par un soucis de lisibilité. Ainsi, lorsque le comportement du composant ne dépend pas de ses entrées, on choisira de préférence une représentation par diagramme états-transitions « simple ». Lorsque le comportement du composant est influencé par des modifications de ses entrées, on essaiera autant que possible de se limiter à ne représenter les configurations que

lorsque cela apporte une plus-value. Deux exemples de diagrammes état-transitions sont fournis sur les figures 3.7 et 3.11.

Pour chacun des éléments de la bibliothèque de nœud AltaRica, on rassemble l'ensemble de ces informations dans un document de capitalisation reprenant :

- les tableaux 3.10, 3.11 et 3.12
- une représentation sous forme de diagramme états-transitions étendus (section 3.8.6) représentant le comportement interne du nœud considéré ainsi que les tables de décision (les équations logiques) permettant de définir ses variables de sortie.

3.9 Conclusion

Nous avons donc à ce stade identifié un ensemble d'abstractions qu'il faudra intégrer au modèle AltaRica pour répondre à ses objectifs (*i.e.* aux évènements redoutés qu'il devra observer) et capitaliser ces informations sous forme tabulaire et / ou graphique.

... l'étape suivante est alors l'implémentation du modèle AltaRica. Comme déjà suggéré au cours de ce mémoire, celle-ci se déroulera en deux temps : la modélisation d'une bibliothèque de composant puis la création du modèle global par instanciation des composants de cette bibliothèque. La méthodologie proposé dans ce chapitre se veut proposer un processus rigoureux pour modéliser un composant AltaRica. L'objectif poursuivi est alors de limiter, de prévenir les erreurs dans le modèle AltaRica. Cependant, prévenir ne peut suffire et il nous faut nous assurer que le modèle ne contient effectivement pas d'erreur. Pour cela, nous nous sommes donc intéressé à une phase de validation du modèle AltaRica qui nous permettra d'éliminer, si erreur il y a, les erreurs restantes.

Chapitre 4

Processus pour la validation de modèle AltaRica

Sommaire

4.1 Avant propos

4.1.1 Introduction

L'approche proposée au chapitre 3 propose ainsi une méthode fondée sur l'Ingénierie Dirigée par les Modèles (IDM) et plus particulièrement sur l'utilisation de modèles de propagation de pannes pour supporter les analyses classiques de Sûreté de Fonctionnement (*e.g.* arbres de défaillance, AMDE - section 1.5). Cette approche permet, entre autres choses, une meilleure traçabilité entre la structure du modèle et l'architecture du système qu'il décrit grâce notamment à la mise en place d'une capitalisation et d'une documentation des informations implémentées dans le modèle (section 3.8). Ainsi, un de nos soucis majeurs est de fournir les justifications des données incluses dans le modèle, de tracer les hypothèses de modélisation et de capitaliser les abstractions définies. L'objectif de l'ensemble de ces tâches et du chapitre 3 est alors de prévenir et de minimiser le nombre d'erreurs implémentées dans le modèle.

Pour aller plus loin, il est important de réaliser que si avoir un processus rigoureux d'obtention du modèle est une étape, le risque lié à l'introduction d'erreurs demeure présent tout au long du cycle de développement du modèle. En particulier, il est nécessaire et indispensable de considérer l'éventualité selon laquelle certaines erreurs puissent être potentiellement introduites lors de l'implémentation du modèle. Un comportement naturel est alors de vouloir éliminer ces erreurs et de vouloir les éliminer au plus tôt pour limiter le coût de leurs détections, de leurs éliminations et de leurs corrections. En pratique, nous souhaitons démontrer que le comportement du modèle est conforme au comportement du système qu'il décrit. Pour cela, diverses activités peuvent être envisagées comme par exemple :

- des activités de revues du modèle (relecture par des ingénieurs et / ou des experts) ;
- des activités de preuve ayant pour objectif de démontrer formellement que le modèle satisfait une ou plusieurs propriétés ;
- des activités de simulation de tests effectuées sur le modèle.

Concernant le premier point et par exemple, le projet MISSA (More Integrated System Safety Assessment - [53]) s'est penché sur la définition de « checklists ». Ces checklists posent des questions qui imposent à l'analyste de s'intéresser à un certain nombre d'aspects relatifs au modèle considéré : ses limites, ses conditions d'utilisations... Des « plans d'argumentation » sont ensuite décrits et comportent par exemple la justification de l'approche de modélisation, la justification des hypothèses de modélisation ou la justification des résultats obtenus. Ces argumentations permettent d'augmenter la confiance dans le modèle et sont sans aucun doute un premier pas vers une validation rigoureuse d'un modèle. Cependant, si une telle validation semble être un point clef pour espérer insérer une telle approche dans un processus industriel, les checklists évoquées ne s'intéressent que peu au comportement à proprement parler du modèle et restent à un stade informel.

Pour aller au delà de simples revues du modèle, des activités de preuve et / ou de test peuvent être menées sur le modèle. Divers travaux se sont intéressés à la première activité et par exemple

à l'utilisation du model-checking [38] pour montrer qu'un modèle satisfait une propriété donnée. Nous nous intéressons dans ce mémoire davantage à des activités de test sur le modèle. L'objectif est alors de « tester » le comportement du modèle dans le but de détecter des fautes potentiellement introduites lors de la construction du modèle (lors de la réalisation de la spécification - chapitre 3 - ou lors de l'implémentation), *i.e.* de détecter des écarts entre le comportement attendu du modèle et les résultats des tests.

Avant de nous intéresser davantage à ce processus de validation par test de conformité d'un modèle AltaRica par rapport à sa spécification, nous pensons utile d'expliciter les différences entre preuve et test et les raisons pour lesquels nous préférons, pour notre problématique, le test.

4.1.2 Pourquoi faire du test ?

Si la notion de test sera davantage explicitée dans la suite, nous donnons ici, des définitions intuitives des termes *preuve* et *test*.

- Une preuve est une démonstration formelle qu'un modèle satisfait une propriété. Ainsi, on peut s'assurer qu'une propriété est vraie pour *toutes* les exécutions possibles du modèle.
- Un test consiste à tester *un* chemin d'exécution du modèle. On fournit au modèle des entrées et on observe la sortie obtenue. On compare ensuite la sortie obtenue avec celle attendue.

Pour aller plus loin à ce stade, il est nécessaire de connaître, dans le processus de développement du modèle, quelles activités et quels documents peuvent être formels et lesquels ne peuvent pas l'être. Pour cela, la figure 4.1 inspirée de [29] présente une vision concise de ce processus de développement d'un modèle. On distinguera dans cette figure les documents (représentés par des boîtes) et les activités (représentées par des flèches).

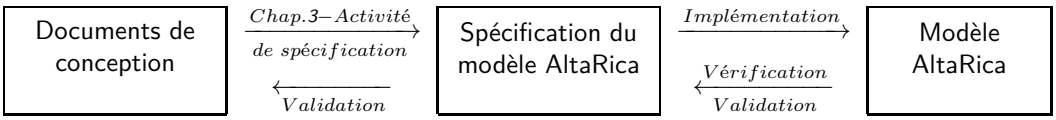


Figure 4.1 – Processus de développement d'un modèle

Remarque : L'activité de « vérification » consiste à s'assurer que le modèle construit l'est de manière correcte. On souhaite démontrer que les exigences spécifiées sont satisfaites par le modèle. L'activité de validation vise quant à elle à s'assurer de la pertinence du modèle par rapport au système réel qu'il représente (a-t-on construit le bon modèle ?).

Sur la figure 4.1, on s'intéresse à ce qui peut être formel et à ce qui ne peut pas l'être. L'argumentation est pour cela reprise de [29]. On s'intéresse dans un premier temps aux documents : les documents de conception, la spécification du modèle AltaRica et le modèle AltaRica.

- Les documents de conceptions sont par nature informels. Il s'agira par exemple de spécifications de conception écrites en langage naturel, de schémas techniques ou de connaissances experts.
- La spécification du modèle AltaRica est d'après notre expérience rarement totalement formelle. En accord avec la section 3.8, elle peut être semi-formelle.
- Le modèle peut lui aussi être formel si on suppose que le langage utilisé l'est. Dans le cas du langage AltaRica, le modèle est donc formel.

Pour traiter le cas des activités, on supposera qu'une activité est formelle si ses documents d'entrées et de sorties sont formels. Ainsi :

- les activités de spécification (construire la spécification du modèle AltaRica à partir des documents de conception) et de validation de la spécification du modèle ne peuvent donc pas être des activités formelles ;
- l'activité d'implémentation peut potentiellement être formelle si on s'intéresse à une génération automatique du modèle à partir de la spécification de ce modèle. Dans notre cas, l'implémentation est restée manuelle et l'activité ne peut donc être qualifiée de formelle ;
- les activités de vérification / validation du modèle AltaRica par rapport à sa spécification ne peuvent être formelle que si la spécification du modèle est formelle.

L'activité de preuve ne pouvant être utilisée que pour comparer une implémentation à une spécification formelle, une telle activité ne peut être employée pour comparer directement la spécification du modèle AltaRica avec les documents de conception. L'utilisation de preuves pourrait cependant être étudiée et utilisée pour l'activité de vérification du modèle par rapport à sa spécification, après formalisation de celle-ci. La preuve garantira alors le lien entre la spécification et l'implémentation du modèle AltaRica. Néanmoins, le lien entre la spécification informelle (ou semi-formelle) et la spécification formelle restera encore à être garanti. Ne voulant pas seulement déplacer le problème, nous conservons notre volonté de garantir le lien entre la spécification du modèle AltaRica et son implémentation...

Par essence, l'analyse de sécurité amène les analystes à produire les séquences d'évènements (en nombre borné) conduisant à l'occurrence d'un évènement redouté. En vue de la revue et de la validation des modèles construits, ces séquences doivent être lisibles et compréhensibles par des spécialistes métiers (non spécialistes Sûreté de Fonctionnement) et par les ingénieurs Sûreté de Fonctionnement en charge de la réalisation du modèle AltaRica. De plus, étant donné le déterminisme des modèles AltaRica construits, ces séquences constituent naturellement des scénarios de test pouvant être simulés sur le modèle AltaRica. Ainsi dans ce contexte, le test permet, avec un surcoût réduit, à la fois d'estimer la conformité du modèle AltaRica par rapport à sa spécification mais aussi d'aborder la validation de cette spécification via la simulation des séquences et l'examen de leurs effets.

Par contre, l'examen d'un nombre limité de séquences ne peut à lui seul garantir dans tous les cas que l'implémentation du modèle AltaRica est conforme à sa spécification et au comportement du système qu'il décrit. Des mesures de couvertures restent à définir dans ce contexte afin de pouvoir accorder une confiance justifiable dans les activités de test réalisées sur le modèle AltaRica.

Nous avons donc deux problèmes de validation à résoudre : celui de la spécification par rapport aux documents de conception, celui du modèle par rapport à sa spécification. Pour le premier point, le chapitre 3 adresse le problème et propose un processus de modélisation. En particulier, la section 3.8 propose diverses formalisations de cette spécification pour un composant élémentaire. Des activités de revues pourront et devront être complémentaires à ce processus de modélisation pour améliorer la confiance de l'utilisateur. Dans ce chapitre, nous traiterons majoritairement du second problème de validation, à savoir celui de l'implémentation du modèle. Nous fonderons pour cela notre approche sur des activités de test.

4.2 Processus de validation du modèle AltaRica

Nous rentrons ici directement dans le vif du sujet en proposant les fondements d'un processus de validation d'un modèle AltaRica. Nous focaliserons ensuite sur un processus plus adapté à la validation d'un composant élémentaire (d'un composant d'une bibliothèque AltaRica) et développerons ce processus « unitaire » dans la suite de ce chapitre.

4.2.1 Différents niveaux de validation

Pour augmenter l'efficacité de l'activité de validation, nous souhaitons détecter au plus tôt les potentielles erreurs d'implémentation dans le modèle. Dans cette optique, nous découpons l'approche de validation en trois phases distinctes :

- une phase de **validation unitaire** qui consistera à valider le comportement de chacun des composants de la ou des bibliothèques ;
- une phase de **validation de l'intégration** qui focalisera son objectif sur la validation des interfaces que chaque composant possède avec son environnement (*i.e.* avec les autres composants du modèle) ;
- une phase de **validation système** qui s'intéressera, comme son nom l'indique, au comportement du modèle représentant le système.

Ainsi, la validation unitaire aura pour but de démontrer que chaque composant effectue la fonction prévue et seulement cette fonction. On pourra distinguer dans cette validation la validation de la logique interne du composant (valider les états et les transitions entre états) et la validation des calculs réalisés par le composant (est ce que les effets des évènements sont correctement implémentées ?). La validation de l'intégration aura pour objectif de démontrer que les composants sont correctement interfacés les uns avec les autres. La validation système validera le comportement du modèle complet représentant le système considéré. Une vue du processus de validation est résumée sur la figure 4.2.

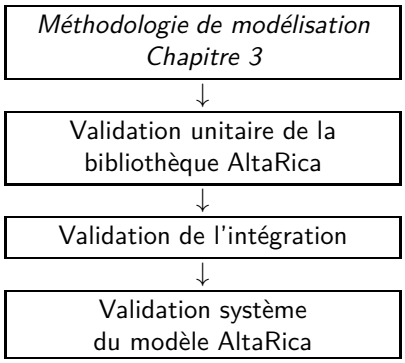


Figure 4.2 – Processus de validation haut niveau

4.2.2 Proposition de processus général

Nous décrivons dans un premier temps un processus général de validation que nous croyons applicable aux différents niveaux de validation décrits auparavant (unitaire, intégrée et système). Ce processus est présenté sur la figure 4.3 qui montre comment les différentes étapes se succèdent les unes aux autres. Brièvement, la stratégie générale de l'approche consiste à :

- se doter d'un jeu de tests (*i.e.* d'un jeu de scénarios à simuler sur le modèle) ;
- simuler ce jeu de tests sur le modèle AltaRica et de comparer les résultats obtenus par rapport aux résultats attendus ;
- étudier le niveau de revue des modèles atteint lors de la simulation du jeu de tests (détecter les portions de modèle éprouvées par les tests et celles ne l'étant pas) ;

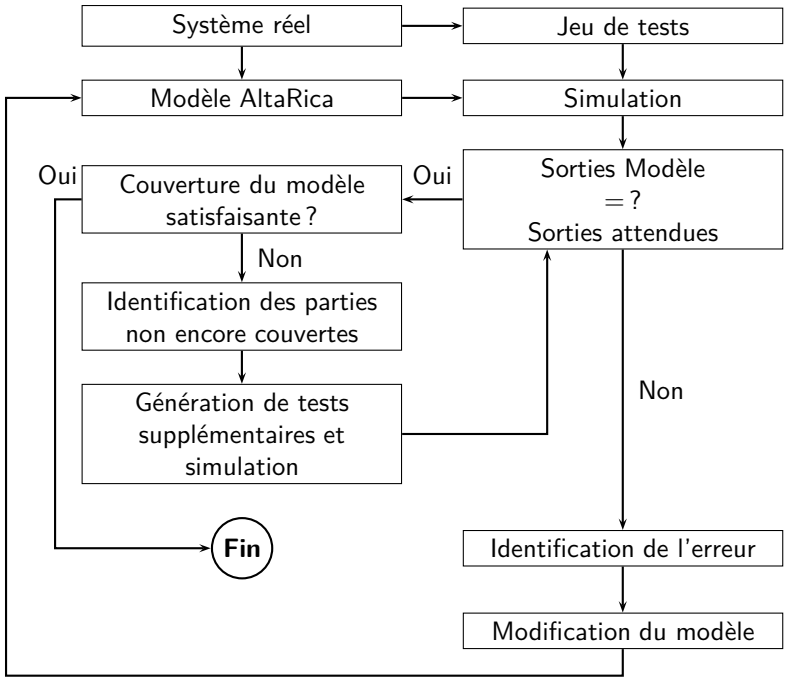


Figure 4.3 – Processus général de validation d'un modèle AltaRica

– générer des tests supplémentaires pour activer les parties non encore éprouvées par le jeu de tests.

Remarquons que ce processus reste, de manière volontaire, d'assez haut niveau. En particulier ici, il n'est fait nullement mention de l'obtention du jeu de tests. Dans cette optique et pour rebondir sur la section 3.8, nous développerons particulièrement l'étape de validation unitaire (*i.e.* la validation d'un composant élémentaire de la bibliothèque AltaRica). Pour cela et comme déjà explicité dans cette même section 3.8, nous nous appuierons sur une représentation formelle (à défaut semi-formelle) du comportement du composant étudié. Nous appellerons « spécification » une telle représentation. Ainsi, la validation d'un modèle AltaRica sera transformée en un problème classique de vérification entre une spécification et son implémentation.

4.2.3 Un processus adapté à la validation unitaire

La seule différence entre les figures 4.3 et 4.4 vient alors de l'étape consistant à obtenir un jeu de tests : pour la validation unitaire, il pourra être généré grâce à la spécification obtenue suite à la phase de modélisation (Chapitre 3).

Notons également à ce stade que ne sera pas étudiée dans ce mémoire la génération automatique de modèle AltaRica à partir de la spécification obtenue. D'une manière générale, il existera un risque en cas de définition de la spécification et du modèle AltaRica par une unique personne ou une unique équipe. Ainsi, si cette spécification est sans aucun doute une ressource fondamentale dans la construction d'un modèle AltaRica, nous ne souhaitons pas à l'heure actuelle générer à partir de la spécification à la fois le modèle et les jeux de tests. Pour aller plus loin, il serait même préférable que la spécification et le modèle AltaRica soient réalisés par des personnes ou équipes différentes.

Une fois ce jeu de tests obtenu, nous nous intéresserons à la simulation des tests sur le modèle AltaRica et à la couverture effective de ce modèle. Nous nous intéresserons à identifier et mesurer quelles portions du modèle ont été effectivement testées par le jeu de scénarios. On

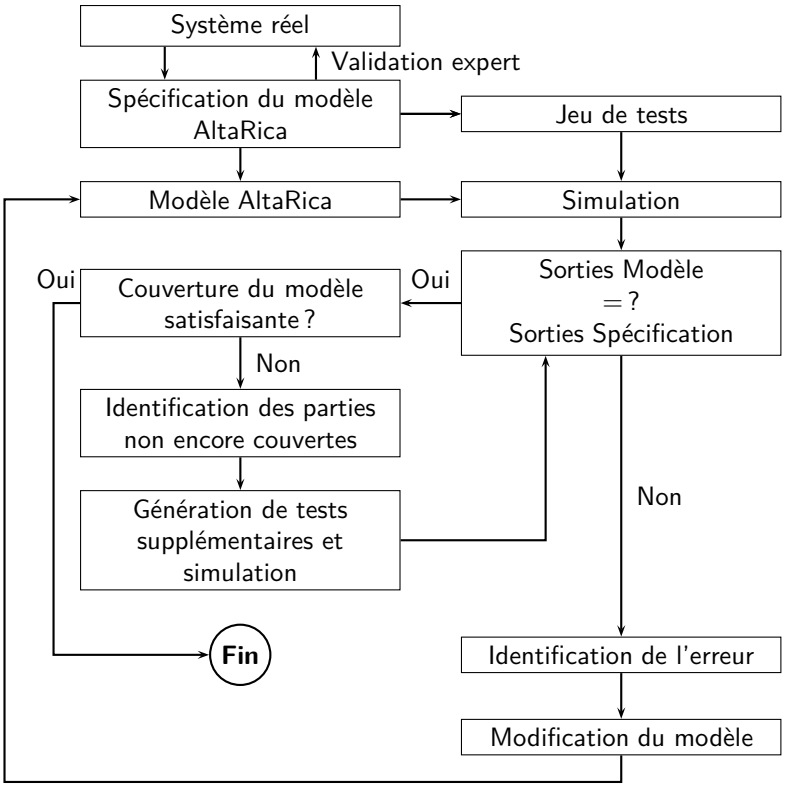


Figure 4.4 – Processus de validation unitaire d'un modèle AltaRica

parlera de couverture de modèles. Nous définirons des critères de couverture d'automates de mode et donc des critères de couverture de modèles AltaRica. Ces notions seront définies dans la prochaine section.

Remarque : Précisons, étant donné qu'un modèle AltaRica est un automate de mode, la correspondance entre les termes « validation de modèle AltaRica » et « validation d'automate de mode ». Nous utiliserons au cours de cette section les deux appellations selon le contexte.

4.3 Vocabulaire du test

4.3.1 C'est quoi le test ?

Les activités comme celles décrites au chapitre 3 ont pour objectif d'assurer la fidélité des modèles réalisés (vis-à-vis des systèmes qu'ils décrivent) et d'améliorer la confiance de l'utilisateur lors de l'emploi de ces modèles. Cependant, ces activités sont réalisées en amont de la phase d'implémentation et ne peuvent, à elles seules, garantir l'absence d'erreurs dans le modèle une fois cette phase effectuée.

L'activité de test, décrite en détails dans [8], à un dessein similaire (*i.e.* assurer la fidélité des modèles, améliorer la confiance de leurs utilisateurs) mais est quant à elle réalisée à la suite de l'implémentation. Cette activité, fondée sur la simulation d'un modèle, s'intéresse alors à détecter les différences entre des résultats prédits (des résultats attendus) et les résultats réellement fournis

par le modèle considéré lors de la simulation. Cette activité est donc une activité indispensable et complémentaire de celle présentée au chapitre 3.

Selon [43], le but de l'activité de test n'est cependant pas de démontrer l'absence d'erreur dans le programme ou de démontrer que le programme satisfait certaines propriétés attendues. Ainsi une définition préférée serait plutôt celle définissant le test comme une activité visant à détecter des erreurs dans un logiciel (ou dans notre cas, dans un modèle). Ainsi, « tester » un modèle peut être aperçu comme une « mise à l'épreuve » de ce modèle dans le but de détecter le plus grand nombre d'erreurs possibles. Remarquons alors que deux principaux types d'erreurs peuvent être présents dans un modèle : lorsque le modèle ne fait pas (ou fait de manière erronée) ce qu'il est supposé faire ; lorsque le modèle fait ce qu'il n'est pas censé faire. L'activité de test aura pour vocation de détecter ces erreurs pour permettre leurs corrections.

4.3.2 Test et jeu de tests

De façon relativement intuitive, un test est un couple (entrée, sortie) où l'entrée est un ensemble d'évènements à simuler sur le modèle et où la sortie est le résultat attendu. Ainsi, un test peut être décrit simplement par une phrase du type « si on joue un évènement donné sur le modèle, alors la sortie devra être égale à une valeur v ».

De manière formelle, nous définirons un test comme un 4-uplet $t = (M, v, \Phi, o)$ où :

- M est le modèle étudié, *i.e.* le modèle sur lequel on va effectuer les tests ;
- $v = (e_i)_{i \geq 0}$ un vecteur d'évènements ;
- Φ un observateur du modèle, *i.e.* une sortie du modèle ;
- $o \in dom(\Phi)$ la valeur attendue de Φ.

De manière spécifique aux automates de modes, on peut définir un test comme un 5-uplet $t_A = (A, v, s, \Phi, o)$ où :

- $A = (D, dom, S, F^{in}, F^{out}, \Sigma, \delta, \sigma, I)$ est un automate de mode (Cf. section 2.3) ;
- $v = (e_i)_{i \geq 0}$ un vecteur d'évènements où $\forall i, e_i \in \Sigma$;
- s une évaluation des variables de S ($s \in dom(S)$) représentant l'état attendu de A à la suite de v ;
- Φ un observateur du modèle : $\Phi \in F^{out}$;
- $o \in dom(\Phi)$ la valeur attendue de Φ à la suite de v.

On se donne également la possibilité de définir des tests spécifiant l'impossibilité de réaliser un vecteur d'évènements. Un tel test sera défini de façon analogue à la définition précédente en fixant $s = \emptyset$, $\Phi = \emptyset$ et $o = \emptyset$. Un tel test sera ainsi de la forme $t_A = (A, v, \emptyset, \emptyset, \emptyset)$.

Remarque : La définition de test proposée ne précise les valeurs de sortie attendues qu'après la séquence d'évènements. C'est un choix. Dans d'autres contextes (protocoles de télécommunication, systèmes temps réel), on peut aussi préciser les valeurs attendues en sortie après chaque action de la séquence.

Remarque : Dans les deux définitions ci-dessus, on suppose que les modèles testés sont déterministes. Un même scénario de test conduira donc toujours à la même valeur des variables de sortie.

On définira ensuite, pour ces automates de modes un ensemble de tests (un jeu de tests) $T = (t_A^i)_{i \geq 1}$ où chaque t_A^i est un 5-uplet $(A, v_i, s_i, \Phi_i, o_i)$.

4.3.3 Test boîte noire - Test boîte blanche

4.3.3.1 Test boîte noire

L'approche de test de type « boîte noire » (aussi nommé test fonctionnel) est une des deux principales approches de conception des tests appliqués aux logiciels pour vérifier la conformité d'un logiciel par rapport à sa spécification. On cherche ainsi à détecter les différents cas où le comportement du modèle est différent de celui attendu, *i.e.* dans quelles circonstances le comportement du modèle est différent de celui prédit par sa spécification ?

En suivant cette approche, les différents cas de test sont construits à partir de l'examen de la spécification, sans se référer à la structure du code testé. La spécification permet de déterminer à la fois les cas importants à considérer et les valeurs attendues dans les cas sélectionnés.

Une préoccupation importante est alors celle de la complétude des cas de test considérés par rapport à la spécification. Dans l'idéal, un test exhaustif de toutes les entrées du modèle permettrait de s'assurer du bon comportement de celui-ci, quelles que soient les circonstances. Bien souvent cependant et à cause de la complexité des modèles à éprouver, une telle exhaustivité sera impossible à satisfaire. Pour répondre à cette question, on pourra établir des critères de couverture de la spécification utilisée pour construire les tests.

La notion peut être facilement transposée aux tests de modèle si la spécification du modèle est connue. L'approche consiste alors à considérer le modèle comme une boîte noire dont on ne considère pas la structure interne : on soumet des entrées choisies à partir de la spécification du modèle, on regarde la sortie (on ne sait pas comment celle-ci est obtenue) et la spécification du modèle est à nouveau utilisée pour décider de l'acceptabilité de la sortie obtenue.

4.3.3.2 Test boîte blanche

L'autre approche de conception des tests est le test « boîte blanche » ou test structurel. Dans ce cas, on suppose la structure interne du modèle connue et la sélection des jeux de tests tient compte de la logique implémentée dans le modèle.

Cette approche permet, entre autre, d'affiner la définition des tests fonctionnels en leur permettant d'explorer des parties relativement distinctes du programme/modèle évalué. La complétude des parties du programme/modèle exercées par les tests peut alors être mesurée à l'aide de critère de couverture structurelle de l'implémentation.

Dans notre cas, le but sera, par exemple, de passer par tous les chemins du modèle et permettra par exemple, d'exécuter des scénarios improbables voire non souhaités (scénarios auxquels potentiellement le programmeur ou l'utilisateur n'avait pas pensé). On parlera alors de critère de couverture du modèle (Cf. section 4.3.4).

4.3.3.3 Exemple

En pratique, il est important de combiner les deux types d'approche de conception des tests comme le montre l'exemple ci-dessous. Imaginons donc que l'on souhaite implémenter un programme calculant la somme de deux entiers modulo 10000 et que nous l'implémentions de la façon, fausse, suivante :

```
void Addition_entier_modulo_10000 (x : int, y : int) {
if (x=500 && y=600) then
{
printf(x-y) ; // Erreur 1
}
else
{
printf(x+y) ; //Erreur 2
}}
```

Ici, si l'erreur 1 sera difficilement identifiée par le test boîte noire (extraire de la spécification le cas de test $x = 500$ et $y = 600$ est relativement peu probable), l'erreur 2 sera en théorie identifiée facilement (étant donné le calcul modulo 10000, il semble naturel de tester le programme pour des valeurs telles que $x + y > 10000$).

Par contre, en travaillant sur la structure du programme, le test boîte blanche identifiera deux tests permettant de passer dans les deux branches du « if - then - else ». Ainsi, il faudra un test avec $x = 500$ et $y = 600$ et l'erreur 1 sera identifié facilement. Il faudra également un test avec $x \neq 500$ ou $y \neq 600$ (*i.e.* $(x = 500 \;\&\&\; y = 600) = faux$) et l'erreur 2 ne sera pas naturellement détectée (le test boîte blanche n'imposera pas d'ordre de grandeur pour les valeurs de x et de y).

Les deux stratégies sont donc plus complémentaires que concurrentes. Le test boîte blanche (approche structurelle) identifiera plus facilement les défauts de programmation. Le test boîte noire (approche fonctionnelle) sera plus adapté pour la détection d'erreurs de spécification ou d'omission par rapport à la spécification.

4.3.4 Couverture de modèle, Critère de couverture

4.3.4.1 Couverture de modèle

Intuitivement, la notion de couverture de modèle a pour objectif de regarder quelle quantité, quelle portion de modèle a été effectivement testée par un jeu de tests. Elle s'inspire de la couverture de code utilisée en génie logiciel. Mesurer la couverture d'un jeu de tests, c'est évaluer l'efficacité de ce jeu de tests sur le modèle. En effet, il est possible (mais souvent inutile) d'avoir un grand nombre de tests qui activent la même portion du modèle alors que peu de tests peuvent éprouver de manière efficace une plus grande partie du modèle.

Imaginons que tous les tests fournissent un résultat correct. Il serait tentant de penser que le modèle est bon et représente bien le comportement souhaité. Mais est-on certain de l'efficacité de notre jeu de tests ? Est-on certain d'avoir tout testé ? S'il reste des portions non testées, leurs comportements respectifs sont-ils bons ? Sont-elles même utiles dans le modèle ? C'est ce genre de questions pour lesquelles la couverture de modèle a vocation à trouver des réponses.

Pour mesurer objectivement cette couverture de modèle, des indicateurs ont été définis pour mettre l'accent sur les portions de modèle testées et celles ne l'étant pas.

4.3.4.2 Mesure de la couverture

La mesure de la couverture d'un programme détermine comment le programme est exercé par le jeu de tests défini et permet d'établir la traçabilité entre le programme (entre les portions

du programme) et les différents tests. En effet, on peut associer à un programme impératif[1] un graphe de contrôle construit de la manière suivante :

- les noeuds du graphe sont :
  - soit les séquences maximales d'instruction sans branchement (les séquences maximales sans instruction « à saut », *i.e.* de type « if - then - else »),
  - soit des prédicats (ou des conditions) testées dans des instructions conditionnelles ;
- les arcs correspondent au transfert de contrôle entre nœud ([30]).

Les critères de couverture du graphe de contrôle permettent alors d'identifier quels nœuds ou quels arcs sont atteints lors de l'exécution d'un test.

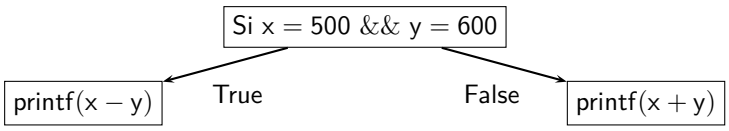


Figure 4.5 – Exemple de diagramme flot de contrôle

Des critères de couverture des nœuds ou des arcs peuvent alors être précisés pour tenir compte d'une manière plus fine des conditions impactant le contrôle.

Définition 4.1 : Décision / Condition

On appelle *décision* une expression booléenne quelconque.

On appelle *condition* une sous-expression atomique d'une décision. Une condition ne peut donc pas être davantage décomposée. ■

Exemple : Dans l'exemple ci-dessus, « $x = 500$ && $x = 600$ » est une décision composée de deux conditions « x=500 » et « y=600 ».

Nous présentons ici les critères de couverture structurels utilisés dans l'aéronautique (la couverture d'automates de mode et de modèles AltaRica sera présentée plus en aval de ce mémoire).

Couverture des instructions : Le plus simple des critères de couverture. Il permet de savoir si pour chaque instruction du programme, il existe un test exécutant cette instruction. Souvent, ce critère est considéré comme trop faible.

Couverture des décisions : Ce critère permet une prise en compte des instructions du type « if - then - else » en permettant d'assurer que l'on passe effectivement dans chacune des branches de l'instruction (*i.e.* que l'on passe dans le « then » et dans le « else »).

Couverture des conditions : Ce critère permet aussi une prise en compte des instructions du type « if - then - else » en permettant d'assurer que toutes les conditions sont évaluées à $vrai$ et à $faux$.

Couverture des conditions / décisions : Combinaison des deux précédents critères.

1. Les opérations réalisées par un programme impératif sont décrites en termes de séquences d'instructions exécutées par l'ordinateur

Couverture MC/DC : (Modified Condition / Decision Coverage). Ce critère de couverture étend le critère de couverture des conditions / décisions en introduisant l'exigence selon laquelle chaque condition doit indépendamment affecter la valeur de la décision. En d'autres termes, pour chaque condition, il faut définir deux tests couvrant les décisions et où les valeurs des autres conditions demeurent constantes.

Couverture des conditions multiples : il s'agit ici de tester l'ensemble des combinaisons possibles des valeurs des conditions.

Couverture des chemins : Ce critère indique si chaque chemin d'exécution a été effectivement exécuté par le jeu de tests. C'est le plus complet des critères mais couvrir 100% des chemins est souvent inaccessible (par exemple, un code contenant n instruction de type « if - then - else » contient potentiellement 2^n chemins d'exécution différents).

Exemple : Pour comprendre la différence entre ces critères, prenons l'exemple fictif issu de [48] d'une instruction de la forme : `if ((a || b) && c) { ... }`. En supposant les trois variables a, b et c booléennes, il y a 8 possibilités de tests pour cette instruction. On les notera (T représentera la valeur $vrai$, F représentera la valeur $faux$) [TTT T], [TTF F], [TFT T], [FTT T], [TFF F], [FTF F], [FFT F] et [FFF F]. Alors :

- la couverture des instructions est satisfaite avec un unique test évaluant `(a || b) && c)` à $vrai$ (par exemple [TTT T]) ;
- la couverture des décisions est satisfaite avec deux tests évaluant `(a || b) && c)` à $vrai$ et à $faux$ (par exemple [TTT T] et [TTF F]) ;
- la couverture des conditions est satisfaite avec deux tests qui diffèrent sur toutes les conditions, par exemple [TTF F] et [FFT F]. On remarquera ici que couvrir les conditions n'implique pas automatiquement la couverture des décisions ;
- la couverture des conditions / décisions est satisfaite par deux tests qui diffèrent sur toutes les conditions mais qui couvrent la décision (par exemple [TTT T] et [FFF F]) ;
- la couverture MC/DC est satisfaite avec quatre tests. Pour chaque condition, il est nécessaire de définir deux tests qui ne diffèrent que par rapport à cette condition et qui couvrent la décision. Par exemple, pour montrer l'indépendance de la condition a, on peut choisir les deux tests [TFT T] et [FFT F]. Les tests peuvent servir pour montrer l'indépendance de plusieurs variables. Pour b on choisira ainsi les deux tests [FFT F] et [FTT T], pour c on choisira [TFT T] et [TFF F]. Ainsi, le jeu de tests [TFT T], [FFT F], [FTT T] et [TFF F] permet d'obtenir une couverture MC/DC de 100% ;
- pour la couverture des conditions multiples, les huit tests sont ici nécessaires ;
- pour la couverture des chemins, les huit tests sont également nécessaires.

Remarque : La relation d'inclusion est habituellement utilisée pour comparer deux critères de couverture. Soient trois critères de couverture C_1, C_2 et C_3.
On dit que C_1 inclut C_2 (C_2 est inclus dans C_1) si tous les scénarios de test qui satisfont C_1 satisfont également C_2. Par exemple et comme l'illustre la figure 4.6, le critère de couverture des décisions inclut le critère de couverture des instructions. La relation d'inclusion est transitive (si C_1 inclut C_2 et que C_2 inclut C_3 alors C_1 inclut C_3).

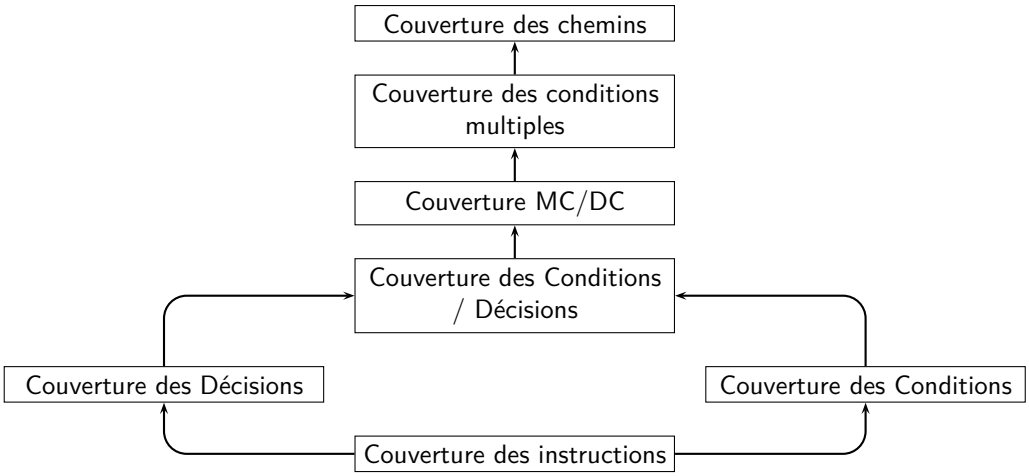


Figure 4.6 – Hiérarchie des critères de couverture
(Légende : $\rightarrow$: est inclus dans)

Pour la certification des logiciels les plus critiques, le critère MC/DC est apparu comme un raisonnable compromis entre un critère trop faible (imposant trop peu de tests) et un critère visant à couvrir l'ensemble des chemins d'exécution (2^n tests si n variables booléennes).

La couverture de code et la définition de critères permettent donc de se doter d'une métrique d'évaluation de la qualité de l'activité de test réalisée sur un programme ou sur un modèle. En fournissant des résultats numériques sur la quantité de code éprouvée par le jeu de tests, elle améliorera et justifiera la confiance de l'utilisateur lors de l'exploitation du programme ou du modèle.

Et par rapport à notre problématique, ça donne quoi ?

Avant d'attaquer le vif du sujet et connaissant désormais la signification de quelques termes, explicitons un peu plus en détails l'objet de notre approche. Le processus proposé figure 4.4 pour la validation d'un modèle AltaRica unitaire est d'une certaine manière hybride aux deux approches « test boîte noire » et « test boîte blanche ». Pour rester simple et appréhender facilement la problématique, le processus se décompose en deux phases principales : un travail sur la spécification (on suppose une spécification formelle disponible pour la validation unitaire), un travail directement sur le modèle AltaRica (concernant tous les niveaux de validation).

Concernant la spécification, celle-ci va nous permettre classiquement de générer un jeu de tests. Cette génération sera guidée par la structure de cette spécification si la forme de celle-ci nous y autorise (si par exemple, celle-ci est sous forme de diagrammes états-transitions). Lors de la simulation des tests sur le modèle AltaRica, une différence entre le résultat obtenu et celui théorique devra avoir pour effet d'envisager une erreur non seulement sur le modèle mais aussi potentiellement de mettre en défaut la spécification. Simuler les tests sur le modèle AltaRica sera donc également un moyen de tester la spécification.

Une fois le jeu de tests établi (par exemple, grâce à la spécification), on les simule sur le modèle et on compare le résultat obtenu avec celui attendu. Cela s'apparente à première vue comme du test *boîte noire*. Cependant, la structure interne du modèle AltaRica étant connue, nous proposerons également d'étudier la couverture réalisée par les tests effectués. Pour cela, nous proposerons des critères de couverture applicables à des automates de mode et à des modèles

AltaRica. Cette *mesure de la couverture* nous permettra d'identifier les parties du modèle non couvertes. Il sera alors possible si souhaité de générer des tests supplémentaires pour atteindre une couverture satisfaisante du modèle.

Ainsi, en simulant des tests issus de la spécification sur le modèle AltaRica, une erreur rencontrée permettra de réfléchir à la fois sur le modèle et sur la spécification. En assurant une certaine couverture de ces deux documents (spécification et implémentation), il sera alors possible de traquer les informations manquantes dans le modèle et celles en surplus.

4.4 Sélection de cas de test pour un modèle AltaRica

Le premier objectif de la démarche est donc de se doter d'un jeu de tests. Pour cela, nous avons évoqué que, pour la validation unitaire, l'utilisation d'une spécification formelle ou semi-formelle pourrait nous permettre cette génération... Le principe étant, nous le rappelons, d'obtenir un jeu de tests dont on étudiera par la suite la couverture du modèle AltaRica (quelle portion du modèle couvre ce jeu de tests ?).

Pour commencer, nous présenterons dans la section 4.4.1 la génération d'un jeu de tests pour valider un composant AltaRica élémentaire (validation unitaire). Pour cela, nous nous appuierons principalement sur la section 3.8. Cependant, la validation unitaire n'est pas en mesure (et il serait injuste de le lui reprocher) d'assurer une complète absence d'erreurs dans le modèle. En particulier, elle ne permet pas d'exhiber des erreurs liées à la présence de plusieurs composants dans le modèle (par exemple, des erreurs dans la modélisation d'un processus de reconfiguration ou dans une synchronisation). Pour ces raisons, nous introduirons dans les sections 4.4.2 et 4.4.3 la génération de cas de test pour la validation de l'intégration et la validation système.

4.4.1 Validation unitaire : Génération d'un jeu de tests pour le modèle AltaRica

4.4.1.1 Génération de tests à partir d'une spécification par AMDE formelle

Nous avons vu section 3.8 que la spécification du modèle AltaRica pouvait potentiellement être écrite en formalisant les informations de l'AMDE (pour rappel : Analyse des Modes de Défaillance et de leurs Effets). On rappelle ici le principe du tableau 3.7.

Mode de défaillance sur la sortie	Description	Déviation entrée	Mode de défaillance composant	Taux
$OUT = Nul$	...	$IN = Nulle$	-	-
	...	-	E_1	$1.5e^{-6}$
...	...	...	...	...

Chaque ligne de cette AMDE nous donne donc un scénario de test à appliquer sur le modèle. Les composantes de ce scénario sont soit des modes de défaillance, soit des déviations des entrées du composant, soit éventuellement une combinaison des deux. Pour chacun de ces scénarios, la valeur attendue de la sortie est indiquée dans la première colonne du tableau. Par contre et à cause du formalisme choisi, un tel tableau ne nous fournit aucune indication sur l'état attendu du modèle.

4.4.1.2 Génération de tests à partir d'une machine à état finis

Ici, nous faisons référence à différents types de spécifications présentés en section 3.8 :

- aux spécifications utilisant des machines à états finis : diagramme états-transitions, diagramme d'états UML, Statechart... ;
- aux spécifications décrites sous forme tabulaire et utilisant une philosophie similaire à SCR (Software Cost Reduction [32]).

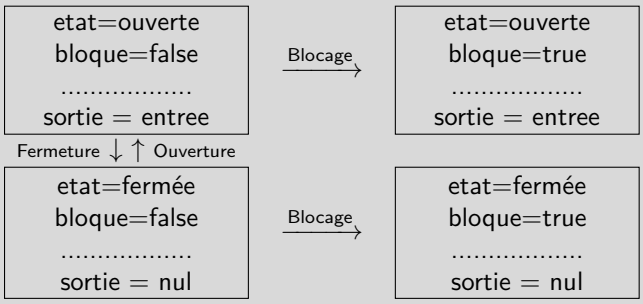
Le traitement des deux points ci-dessus dans la même section se justifie avec le fait, déjà évoqué en section 3.8.4 que le formalisme sous-jacent à l'écriture tabulaire de SCR est un formalisme utilisant des machines à état finis.

Avec de telles formes de spécifications, la génération de tests se fera avec deux objectifs distincts : valider la logique interne décrite dans la spécification (*i.e.* les changements d'état) et valider les calculs décrits dans la spécification (*i.e.* comment sont définies les sorties en fonction de l'état et des entrées). Le principe de génération est alors de :

- générer des scénarios permettant de visiter chacun des états de la spécification ;
- générer, dans chacun de ces états, des scénarios pour tester les équations définissant les valeurs des sorties ;
- mettre en place des critères de couverture sur la spécification pour assurer et garantir la mise à l'épreuve des différentes transitions entre états et des différentes équations de sorties.

On pourra trouver dans les références [47, 15, 46] divers travaux traitant de la génération de tests à partir de diagrammes états-transitions (sous forme graphique ou tabulaire) et de la mise en place de critères de couverture sur ce type de spécification. Nous développerons pour notre part plus en détails et à partir de la section 4.5 de tels mécanismes appliqués aux modèles AltaRica et aux automates de mode.

Exemple : Pour illustrer le principe général de la génération de tests à partir d'une spécification, nous reprenons l'exemple de la valve dont la spécification de comportement est indiquée figure 2.1 (page 34). Nous la rappelons ici :



Ici et sans passer en revue les différents critères de couverture pouvant être définis sur une telle spécification, on souhaitera tester les différents états et dans chacun des états, tester les différentes évaluations de la variable de flux « sortie ». Une génération de tests possible ici est (A désigne l'automate de mode considéré ici, s_i désigne une évaluation du couple $(etat, bloque)$, Φ désigne la variable de flux « sortie ») :

- $t_A^1 = (A,\ v_1,\ s_1,\ \Phi,\ o_1)$ avec $v_1 = \emptyset$, $s_1 = (ouverte, false)$ et $o_1 = entree$;
- $t_A^2 = (A,\ v_2,\ s_2,\ \Phi,\ o_2)$ avec $v_2 = \{Blocage\}$, $s_2 = (ouverte, true)$ et $o_2 = entree$;
- $t_A^3 = (A,\ v_3,\ s_3,\ \Phi,\ o_3)$ avec $v_3 = \{Fermeture\}$, $s_1 = (fermée, false)$ et $o_3 = nul$;

- $t_A^4 = (A,\ v_4,\ s_4,\ \Phi,\ o_4)$ avec $v_4 = \{Fermeture, Ouverture\}$, $s_4 = (ouverte, false)$ et $o_4 = entree$;
- $t_A^5 = (A,\ v_5,\ s_5,\ \Phi,\ o_5)$ avec $v_5 = \{Fermeture, Blocage\}$, $s_5 = (fermée, true)$ et $o_5 = nul$;

Remarquons qu'ici, on ne teste pas pour conserver l'exemple simple l'influence de changements d'évaluation de la variable de flux « entree ». Les cinq tests précédents nous permettront ainsi de tester que les comportements spécifiés sont réalisés par le modèle AltaRica de manière correcte. Au delà de ces tests, on en génère des supplémentaires qui nous assurerons de l'absence de comportements non spécifiés dans le modèle. L'idée principale est de traduire le fait qu'un évènement donné n'est pas tirable dans un état donné. Ainsi, les tests suivants sont générés :

- $t_A^6 = (A,\ v_6,\ \emptyset,\ \emptyset,\ \emptyset)$ avec $v_6 = \{Ouverture\}$;
- $t_A^7 = (A,\ v_7,\ \emptyset,\ \emptyset,\ \emptyset)$ avec $v_7 = \{Blocage, Ouverture\}$;
- $t_A^8 = (A,\ v_8,\ \emptyset,\ \emptyset,\ \emptyset)$ avec $v_8 = \{Blocage, Fermeture\}$;
- $t_A^9 = (A,\ v_9,\ \emptyset,\ \emptyset,\ \emptyset)$ avec $v_9 = \{Blocage, Blocage\}$;
- $t_A^10 = (A,\ v_{10},\ \emptyset,\ \emptyset,\ \emptyset)$ avec $v_{10} = \{Fermeture, Fermeture\}$;
- $t_A^{11} = (A,\ v_{11},\ \emptyset,\ \emptyset,\ \emptyset)$ avec $v_{11} = \{Fermeture, Blocage, Fermeture\}$;
- $t_A^{12} = (A,\ v_{12},\ \emptyset,\ \emptyset,\ \emptyset)$ avec $v_{12} = \{Fermeture, Blocage, Ouverture\}$;
- $t_A^{13} = (A,\ v_{13},\ \emptyset,\ \emptyset,\ \emptyset)$ avec $v_{13} = \{Fermeture, Blocage, Blocage\}$;

4.4.1.3 Validation unitaire : autres voies pour l'obtention d'un jeu de tests

Que faire lorsque nous ne possédons aucune spécification du comportement de notre modèle ? Est-il possible de tout de même envisager la génération d'un jeu de tests ? ...

En fait, et cela servira de conclusion à la section, l'important est ici on le rappelle de se doter d'un jeu de tests. Le moyen d'obtention de ce jeu de tests est même relativement secondaire par rapport à la problématique de ce mémoire...

En cas de construction d'une spécification, il peut être pertinent de l'utiliser pour obtenir ce jeu de tests. Si aucune spécification n'est disponible, il faut chercher ce jeu de tests ailleurs... Deux possibilités peuvent alors être mentionnées ici.

- Une première solution peut être de définir « à la main » un ensemble de scénarios (un ensemble de tests) de référence dont on connaîtra les issues théoriques par jugement d'experts. Ainsi, un plan de tests est défini, simulé sur le modèle AltaRica et les sorties de celui-ci sont observées. La référence n'est alors plus la spécification mais le jugement d'expert qui aura défini la sortie attendue.
- Une seconde solution est de regarder le taux de couverture atteint par les scénarios conduisant à l'occurrence des évènements redoutés. Ces scénarios étant générés pendant la phase d'analyse du modèle, leur génération ne constitue pas, en l'absence de spécification du modèle AltaRica, une tâche supplémentaire à la charge de l'ingénieur Sûreté de Fonctionnement.

Ces méthodes, bien que selon nous moins « propres » qu'une génération de tests à partir de spécifications formelles, auront cependant comme objectifs d'améliorer la confiance de l'utilisateur dans son modèle. En effet, bien que ne permettant pas l'étude d'une couverture a priori sur la

spécification, elle ne remettent pas en causes l'utilisation et l'application de critère de couverture sur le modèle AltaRica obtenu.

4.4.2 Génération de cas de test pour la validation de l'intégration

La validation de l'intégration est la phase de validation dans laquelle les composants unitaires sont connectés (définition 4.5) et testés en tant qu'ensemble de composants. Brièvement, on rappelle que connecter différents composants entre eux peut être vu comme un ajout d'un certain nombre de contraintes dont il faudra tenir compte. Ainsi, après intégration, les possibilités d'évolutions des composants sont réduites (par rapport à leurs évolutions possibles lorsqu'ils sont considérés seuls) et avec elles le nombre de comportements possibles du composant. Sans développer ici un processus pour cette phase de validation, nous nous permettons cependant quelques mots à son sujet.

Nous prendrons l'exemple de deux composants (deux automates de mode) A ($F_A^{in} = \{i_A\}$, $F_A^{out} = \{o_A\}$) et B ($F_B^{in} = \{i_B\}$, $F_B^{out} = \{o_B\}$) connectés de telle sorte que la sortie o_A de A soit reliée à l'entrée i_B de B. On nomme C (noté $C = A \leftrightarrow B$) l'automate de mode résultat de la connexion de A et B : $F_C^{in} = \{i_A\}$, $F_C^{out} = \{o_B\}$. On suppose la validation unitaire de A et B réalisée et on souhaite tester le lien entre les deux composants.

Idée N°1 : Processus de validation unitaire sur $A \leftrightarrow B$

Cette première approche est fondée sur le fait que l'automate obtenu par composition de deux automates de mode est un automate de mode (section 2.3.2). Ainsi, pour valider l'intégration des composants A et B, on validera unitairement l'automate $A \leftrightarrow B$.

Cette approche, bien qu'ayant l'avantage de ne pas introduire de notions supplémentaires, sera souvent difficilement applicable dans un cas réel. En particulier, l'automate $A \leftrightarrow B$ devient vite de taille importante et posséder une spécification d'un tel automate n'est pas chose aisée. La génération de tests pourra souvent être compliquée à obtenir.

Idée N°2 : Faire du test fonctionnel

Le principe général proposé ici sera « d'oublier » le comportement interne au profit d'une approche boîte noire (*i.e.* d'une approche fonctionnelle). Ce qui nous intéresse n'est alors pas d'activer toutes les portions du modèle mais seulement les différentes interfaces que le composant possède avec son environnement (on suppose que l'activation des parties internes a été obtenue grâce à la validation unitaire).

Pour cela, nous souhaitons étudier la génération de scénarios de test pour l'automate $A \leftrightarrow B$ à partir des tests générés sur A et de ceux générés sur B (lors de la phase de validation unitaire de A et B). Pour rappel et en commençant par considérer le composant A, un test est composé :

- d'un vecteur d'évènement $v = (e_i)_{i \geq 0}$ où chaque e_i est soit un évènement de A, soit une déviation de la valeur d'entrée i_A ;
- s l'évaluation attendue des variables d'état de A ;
- Φ_A la variable de sortie de A ;
- ω_A la valeur attendue de Φ_A.

En ne s'intéressant pas au comportement interne de A, on obtient, pour la variable de sortie Φ_A, un ensemble de couples {scénario, valeur attendue} où les scénarios sont ceux obtenus de la phase de génération de tests pour la validation unitaire (section 4.4.1). v_{A_i} désigne le ième test unitaire de l'automate de mode A.

Scénarios de test	Valeur attendue de la variable de flux de sortie
v_{A_1}	ω_1
v_{A_2}	ω_2
v_{A_3}	ω_2
v_{A_4}	ω_3
v_{A_5}	ω_1

À partir de cet ensemble $(v_{A_i})_i$, on en définit un nouveau v de la façon suivante :

- si v_{A_i} contient au moins un changement de valeur d'entrée, il est ajouté à v ;
- si v_{A_i} ne contient que des évènements internes à A :
  - si la sortie ω_j obtenue après v_{A_i} a déjà été obtenue par un scénario déjà présent dans v, v_{A_i} ne fera pas partie de l'ensemble v,
  - sinon v_{A_i} est ajouté à v.

On réalise cette opération pour l'ensemble des composants du modèle (en particulier dans notre exemple pour A et pour B).

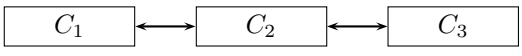
Scénarios de test sur A	Valeur attendue de la variable de sortie de A
v_{A_1}	ω_1
v_{A_2}	ω_2
v_{A_4}	ω_3

Scénarios de test sur B	Valeur attendue de la variable de sortie de B
v_{B_1}	ω_1
...	...
...	...

Une fois donc le travail réalisé sur chacun des composants, on cherche à dériver de ces scénarios des scénarios de test sur l'assemblage de composants (dans notre exemple, sur $A \leftrightarrow B$). Ces scénarios seront en fait obtenus de manière itérative en remplaçant, dans les scénarios incluant des changements de valeurs d'entrée, ces changements par les scénarios des composants auxquels ils sont connectés... Un exemple paraît utile...

Exemple : Imaginons par exemple que le scénario de test du composant B soit $v_{B_1} = \{ e, i_B \rightarrow \omega_2 \}$ où $i_B \rightarrow \omega_2$ désigne le fait que l'entrée de B prenne la valeur ω_2. D'après le tableau ci-dessus, cette valeur ω_2 est obtenue après le scénario v_{A_2} du composant A. Le scénario v_{B_1} devient donc $\{e\} \times v_{A_2}$ où $\times$ est utilisé pour signifier que v_{A_2} sera simulé à la suite de $\{e\}$. Le scénario « $\{e\} \times v_{A_2}$ » est un scénario de test qui active une interface présente entre A et B.

Remarque : En pratique, cette « validation de l'intégration » pourra potentiellement être envisagée pour valider l'intégration d'un composant au sein de son environnement amont et aval. Par exemple sur la figure ci-dessous, considérer les composants C_1 et C_3 permettra de valider l'intégration de C_2 dans le système.



4.4.3 Génération de cas de test pour la validation système

D'une manière générale, expliquer la raison d'être de cette étape de validation du modèle du système est chose plus complexe que pour les précédentes. En effet, l'objectif n'est pas réellement de tester le comportement du modèle dans les détails puisque cela a déjà été en théorie effectué par la phase de validation de l'intégration. L'objectif est plutôt de s'assurer que les propriétés tenues par le système réel sont elles aussi tenues par le modèle. Ces propriétés peuvent être :

- un ou plusieurs scénarios de test dont on connaît l'issue sur le système réel (par exemple, la présence d'une barrière de sécurité limitant l'effet de certaines défaillances) ;
- une propriété générale que l'on sait vérifiée par le système réel (si le réservoir carburant est vide, alors il n'y aura pas de carburant à la sortie des injecteurs).

Si la propriété est exprimée sous forme de scénario de test, la vérification sera fondée sur la simulation de ce scénario sur le modèle. La sortie effectivement obtenue sera comparée à celle attendue (souvent, cette valeur sera issue du retour d'expérience ou du jugement d'expert). Dans le second cas, on parle par exemple de propriétés exprimées en logique temporelle et qui pourront être vérifiées sur le modèle par model-checking [38].

Il existe une large littérature à ce sujet : le scénario « système » est encodé par un automate, un observateur ou une formule de logique temporelle. Les model-checkers sont ensuite utilisés pour obtenir des chemins qui rendent vrai l'observateur « objectif de test ». Sous Cecilia^{TM} OCAS, l'outil de génération de séquences rempli cette tâche. Il ne reste plus alors qu'à mesurer les chemins du modèle couverts par les cas de test.

Des cas de test pour les différents niveaux de validation étant obtenus, il nous faut à présent évaluer leurs qualités. Pour cela, deux possibilités peuvent être étudiées.

- Critères de couverture par rapport à la spécification de référence du modèle AltaRica. Cette couverture varie en fonction du niveau de validation souhaité et en fonction de la forme de la spécification présente en entrée. Ainsi, par exemple, une AMDE doit être couverte à la fois pour la phase de validation unitaire (pour valider les effets locaux des défaillances) et pour la phase de validation de l'intégration ou système (pour tester et valider les effets globaux des défaillances).
- Critères de couverture par rapport au code AltaRica exercé. Ici, nous définissons des critères de couverture directement applicable sur un modèle AltaRica, quelque soit le niveau de celui-ci. Ainsi, les critères sont les mêmes pour les différentes phases de validation. La couverture totale du modèle AltaRica devra être effective après la réalisation de l'ensemble des cas de test.

Dans la suite de ce chapitre, nous détaillons la définition et la mise en place de critère de couverture permettant de mesurer quelle portion de modèle AltaRica est testée par les différents cas de test.

4.5 Critère de couverture de modèles AltaRica par des tests

4.5.1 Forme Normale Disjonctive (FND)

La section 4.3.4.2 ayant présentée différents critères de couverture usuellement évalués pour tester des logiciels critiques aéronautiques, nous nous inspirons dans la suite de ces critères pour évaluer la couverture (par un jeu de scénarios de test) d'un modèle AltaRica.

Pour cela et comme présenté dans [52], la satisfaction des critères de couverture d'une expression booléenne E peut être garantie par la mise sous Forme Normale Disjonctive (FND) de E et par la mise en place de différentes réécritures de E que nous appellerons "dépliages".

Nous présentons ainsi dans cette section la mise sous Forme Normale Disjonctive d'une expression booléenne (section 4.5.1.1) ainsi que les dépliages présentés dans [52] permettant de couvrir (au sens des différents critères de couverture) cette expression booléenne (section 4.5.1.2).

4.5.1.1 Définition

Avant de nous intéresser à la définition de critères de couverture sur un modèle AltaRica, nous décrivons donc ici une méthode de décomposition d'une expression booléenne quelconque. L'objectif d'une telle approche est alors d'exhiber différentes possibilités d'activer (ou de désactiver) cette expression, *i.e.* différentes possibilités de rendre vraie (ou fausse) cette expression. Le principe de cette méthode consiste en la réécriture de l'expression booléenne sous Forme Normale Disjonctive (FND).

Définition 4.2 : Conjonctions élémentaires

On appelle conjonction élémentaire une expression booléenne de la forme $c_1 \wedge c_2 \wedge ... \wedge c_n$, $n \geq 0$ où chaque c_i est une condition (définition 4.1). ■

Trivialement, une condition est une conjonction élémentaire.

Définition 4.3 : Forme Normale Disjonctive

Une expression booléenne est sous forme normale disjonctive si et seulement si elle est de la forme $C_1 \vee ... \vee C_p$, $p \geq 0$ ou chaque C_i est une conjonction élémentaire. ■

En pratique, nous commençons par supposer que chaque conjonction élémentaire contient l'ensemble des conditions possibles. Puisque nous n'avons dans les expressions booléennes considérées que des opérateurs « $\wedge$ » (and), « $\vee$ » (or) et « $\neg$ » (la négation), la Forme Normale Disjonctive d'une expression est obtenue en :

- appliquant les règles d'associativité du « and » et du « or » ;
- appliquant les règles de *De Morgan* :
  - « $\neg (c_1 \wedge c_2)$ » devient « $\neg c_1 \vee \neg c_2$ »,
  - « $\neg (c_1 \vee c_2)$ » devient « $\neg c_1 \wedge \neg c_2$ » ;
- supprimant les doubles négations : « $\neg (\neg c)$ » devient « c ».

4.5.1.2 Dépliage des disjonctions

Une fois l'expression booléenne mise sous Forme Normale Disjonctive, nous souhaitons exhiber des disjonctions obtenues des activations possibles de l'expression booléenne considérée (*i.e.*

les chemins permettant de rendre vraie cette expression). Ce traitement est alors à réaliser en fonction de l'activation souhaitée de l'expression booléenne, *i.e.* en fonction du critère de couverture choisi.

Ici et inspiré de [52], nous présentons plusieurs possibilités de dépliage des disjonctions sans toutefois en privilégier une particulière. Le principe général, illustré figure 4.7, est de déplier une expression écrite sous FND en N cas distincts selon le critère souhaitant être couvert (N dépendant alors du critère).

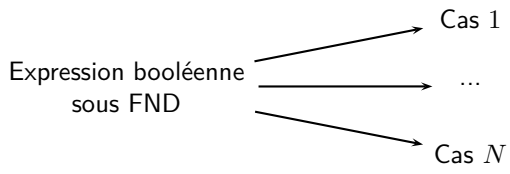


Figure 4.7 – Dépliage d'une expression booléenne sous FND

L'opération de dépliage fera alors apparaître des cas n'étant pas des conjonctions élémentaires. Dans ce cas et tant que les cas identifiés lors d'un nouveau dépliage ne sont pas des conjonctions élémentaires, on renouvèle l'opération de dépliage. Un exemple sera donné page 123. Pour l'instant, nous nous contentons de mettre en concurrence le dépliage avec le critère qu'il permet de couvrir et traiterons l'exemple du dépliage de l'expression « $C_1 \vee C_2$ » où C_1 et C_2 sont des conjonctions élémentaires.

Remarque : Le dépliage d'une condition c conduira à la considération de deux cas : $c = vrai$ et $c = faux$.

Dépliage N° 1 : Dépliage pour critère de couverture sur les décisions

Ce premier dépliage n'en est en fait pas un ! En effet, on ne choisit pas quelle conjonction vérifier pour activer $C_1 \vee C_2$ (ce qui nous intéresse est simplement de l'activer et de ne pas l'activer). Ainsi la disjonction $C_1 \vee C_2$ conduira à traiter deux cas :

- $C_1 \vee C_2 = vrai$;
- $(\neg\, C_1 \wedge \neg\, C_2) = vrai$.

Remarquons que ce dépliage ne dépends pas du nombre de conjonctions élémentaires présentes dans l'expression (sous Forme Normale Disjonctive). Ainsi et quelque soit l'expression E à déplier, le dépliage se fera en deux cas : un assurant que E a été évaluée à $vrai$, un assurant que l'expression a été évaluée à $faux$.

Dépliage N° 2 : Dépliage pour critère de couverture sur les conditions

Ici, ce qui nous intéresse est de vérifier que chacune des conditions de la décision a été évaluée à $vrai$ et à $faux$. Couvrir la décision n'a pas d'importance ici. Ainsi pour la décision $C_1 \vee C_2$, le dépliage se fera en quatre cas :

- $C_1 = vrai$;
- $C_1 = faux$;
- $C_2 = vrai$;
- $C_2 = faux$.

Ici donc, ce critère n'implique pas la couverture du critère de couverture sur les décisions. En particulier, on pourra couvrir les conditions en assurant une décision évaluée à $vrai$. Dans ce cas là, le dépliage peut se faire en deux cas : « $C_1 = vrai \wedge C_2 = faux$ » et « $C_1 = faux \wedge C_2 = vrai$ ». D'une manière générale et si p désigne le nombre de conditions de la décision, le nombre de cas du dépliage sera majoré par « $2 \times p$ ». Dans la suite, nous n'utiliserons pas ce critère au profit du suivant.

Dépliage N° 3 : Dépliage pour critère de couverture sur les conditions et les décisions

Nous regroupons ici les deux critères précédents et souhaitons couvrir à la fois les décisions et les conditions. Pour cela et pour la disjonction $C_1 \vee C_2$, le dépliage se fera en trois cas :

- $C_1 = vrai$;
- $C_2 = vrai$;
- $(\neg\, C_1 \wedge \neg\, C_2) = vrai$.

Les deux premiers cas assurent que les conditions et la décision ont été évaluées à $vrai$. Le dernier évalue les conditions et la décision à $faux$.

Dépliage N° 4 : Dépliage pour critère de couverture MC/DC

Pour satisfaire le critère de couverture MC/DC, on cherche à s'assurer que chaque condition influe de manière indépendante sur la valeur de la décision. Ainsi, le dépliage de la disjonction $C_1 \vee C_2$ conduira à traiter trois cas :

- $(C_1 \wedge \neg\, C_2) = vrai$;
- $(\neg\, C_1 \wedge C_2) = vrai$;
- $(\neg\, C_1 \wedge \neg\, C_2) = vrai$.

Dépliage N° 5 : Dépliage pour critère de couverture des conditions multiples

Ce dépliage consiste à envisager toutes les combinaisons possibles des conditions de l'expression considérée. Ainsi, le dépliage de la disjonction $C_1 \vee C_2$ se fera en quatre cas :

- $(C_1 \wedge C_2) = vrai$;
- $(C_1 \wedge \neg\, C_2) = vrai$;
- $(\neg\, C_1 \wedge C_2) = vrai$;
- $(\neg\, C_1 \wedge \neg\, C_2) = vrai$.

Ainsi, nous obtenons un dépliage d'une expression qui est fonction du critère de test souhaité. En activant l'expression suivant les différents cas présentés, on est assuré de couvrir les critères correspondants. Remarquons que si les cas permettent d'assurer la satisfaction du critère, il ne constituent pas une condition nécessaire (*i.e.* le critère peut être satisfait avec un nombre inférieur de cas à celui obtenu ici).

Exemple : Si l'expression est plus complexe, le dépliage sera obtenu par dépliage successif de cette expression. Par exemple, en dépliant l'expression « $a \vee (b \wedge c)$ » (a, b et c désignant des conditions) de façon à satisfaire le critère MC/DC (dépliage N°4), l'opération consistera en plusieurs étapes successives :

1. $a \vee (b \wedge c)$ est dans un premier temps considéré comme égale à $C_1 \vee C_2$ où $C_1 = a$ et $C_2 = b \wedge c$. Le premier dépliage fourni 3 cas :
   - $a \wedge \neg (b \wedge c) = a \wedge (\neg b \vee \neg c)$,
   - $\neg a \wedge b \wedge c \rightarrow$ Déjà une conjonction élémentaire,
   - $\neg a \wedge \neg (b \wedge c) = \neg a \wedge (\neg b \vee \neg c)$;
2. On déplie ensuite $\neg b \vee \neg c$ dans $a \wedge (\neg b \vee \neg c)$ suivant le dépliage N°4. On obtient trois conjonctions élémentaires :
   - $a \wedge \neg b \wedge c$,
   - $a \wedge b \wedge \neg c$,
   - $a \wedge b \wedge c$;
3. On déplie enfin $\neg b \vee \neg c$ dans $\neg a \wedge (\neg b \vee \neg c)$ suivant le dépliage N°4. On obtient trois conjonctions élémentaires :
   - $\neg a \wedge \neg b \wedge c$,
   - $\neg a \wedge b \wedge \neg c$,
   - $\neg a \wedge b \wedge c \rightarrow$ Déjà obtenu - Doublon ;
4. En éliminant les doublons, on a 6 cas : « $a \wedge \neg b \wedge c$ », « $a \wedge b \wedge \neg c$ », « $a \wedge b \wedge c$ », « $\neg a \wedge \neg b \wedge c$ », « $\neg a \wedge b \wedge \neg c$ », « $\neg a \wedge b \wedge c$ ».

Nous avons déjà évoqué en section 4.2 la démarche générale de validation d'un modèle AltaRica que nous souhaitions mettre en place : obtenir un jeu de tests ; simuler ces tests sur le modèle et étudier leurs couvertures sur ce modèle (*i.e.* quelles sont les portions du modèle testées par ce jeu de tests ?). La figure 4.3 reprend les principales étapes de ce processus.

On suppose dans les sections 4.5.2 et 4.5.3 que nous possédons un jeu de tests (un ensemble de scénarios de test) à simuler sur le modèle. Chacun de ces scénarios est sous la forme d'un 5-uplet $t = (A, v, s, \Phi, o)$ tel que celui défini en section 4.3.2. Souhaitant connaître quelle partie du modèle est éprouvée par le jeu de tests, nous définissons différents critères de couverture applicables aux automates de mode et aux modèles AltaRica. Selon nous, ces critères permettront aux ingénieurs en charge des tests sur le modèle de justifier le nombre de tests réalisés et la pertinence de chacun d'entre eux (en traçant la correspondance entre le test et la portion du modèle couverte). Nous proposerons également une implémentation possible de ces critères de couverture directement au sein du modèle afin de permettre une évaluation directe de la couverture effective du modèle par le jeu de tests.

4.5.2 Critères de couverture des assertions d'un modèle AltaRica

4.5.2.1 Introduction et idée directrice

Rappelons dans un premier temps qu'un modèle AltaRica est un automate de mode, *i.e.* un 9-uplet $(D, dom, S, F^{in}, F^{out}, \Sigma, \delta, \sigma, I)$ où σ est appelée fonction d'assertion et fournit les valeurs des variables de flux de sorties en fonction des valeurs des variables d'état et des variables

de flux d'entrée. En termes pratiques, couvrir les assertions nous permettra de valider les différentes évaluations des variables de flux de sortie dans chacune des configurations de l'automate de mode (définition 2.2).

En langage AltaRica, la fonction d'assertion s'écrit à l'aide de fonctions « if - then - else » imbriquées. On trouvera souvent les assertions écrites sous formes de « case ». Le tableau ci-dessous présente la correspondance entre les deux écritures. Les p_i sont des expressions booléennes sur les variables d'états et les variables de flux d'entrées. On les appellera « prédicats » dans la suite. Les ω_i sont les valeurs prises par la variable de flux de sortie considérée ($\omega_i \in dom(F^{out})$).

case { $p_1 : \omega_1$, $p_2 : \omega_2$, ... $p_{n-1} : \omega_{n-1}$ else ω_n } ;	if p_1 then ω_1 else if p_2 then ω_2 ... if p_{n-1} then ω_{n-1} else ω_n end if ... end if end if

Toute assertion peut donc s'écrire sous une forme « if - then - else » classique. Les critères définis en section 4.3.4 peuvent donc être adaptés à ces assertions.

4.5.2.2 Critère de couverture simple des prédicats d'une assertion

Ce critère se veut permettre une prise en compte des différentes branches de l'assertion, *i.e.* passer au moins une fois par chaque « cas » de l'assertion. Puisque les prédicats p_i sont sous Forme Normale Disjonctive, l'opération consiste à utiliser le dépliage N°1 pour le critère de couverture sur les décisions (section 4.5.1.2).

Critère de couverture simple des prédicats d'une assertion : pour chaque prédicat p_i d'une assertion, il existe au moins deux tests conduisant le modèle dans deux configurations satisfaisant respectivement $p_i = vrai$ et $p_i = faux$. En d'autres termes, il faut que :

- pour chaque prédicat p_i il existe au moins un test amenant le modèle dans une configuration vérifiant « $p_i = vrai \wedge p_j = faux$ » $\forall j < i$;
- il existe au moins un test amenant le modèle dans une configuration vérifiant $\forall i$, $p_i = faux$.

Ainsi le premier point permet de s'assurer que tous les $n-1$ premiers cas de l'assertion sont testés par au moins un test. Le second point assure que le jeu de tests éprouve le « else ». Ce critère est le pendant du critère de couverture des décisions défini en section 4.3.4.

4.5.2.3 Critère de couverture des conditions / décisions des prédicats

En supplément du critère précédent, ce critère vise à assurer que toutes les conditions d'un prédicat p_i sont évaluées au moins une fois à $vrai$ et à $faux$. Ce critère est le pendant du critère de couverture des conditions / décisions défini en section 4.3.4.

Nous rappelons ici que chaque prédicat p_i de l'assertion peut être mis sous Forme Normale Disjonctive, *i.e.* sous la forme $C_{i1} \vee C_{i2} \vee ... \vee C_{ip}$ où C_{ij} est une conjonction élémentaire. Nous utilisons ici le dépliage N°3 pour le critère de couverture sur les conditions et les décisions défini en section 4.5.1.2.

Couverture des conditions / décisions des prédicats : soit p_i un prédicat écrit sous Forme Normale Disjonctive, *i.e.* sous la forme $C_{i1} \vee C_{i2} \vee ... \vee C_{ip}$ où C_{ij} est une conjonction élémentaire. Le critère de couverture des conditions / décisions des prédicats sera satisfait si :

- pour chaque prédicat p_i de l'assertion AltaRica, on applique dans une configuration où $p_i = vrai$ et $p_j = faux$ $\forall j < i$ le dépliage pour le critère de couverture sur les conditions et les décisions décrit en section 4.5.1.2 ;
- il existe au moins un test amenant le modèle dans une configuration vérifiant $\forall i$, $p_i = faux$.

Exemple : Soit l'assertion suivante : « case { $c_1 \vee c_2 : v_1$, $c_3 : v_2$, else v_3 } ». On a ici deux prédicats p_1 et p_2 : $p_1 = (c_1 \vee c_2)$; $p_2 = c_3$. Pour satisfaire le critère de couverture des conditions / décisions des prédicats, on procède de la manière suivante :

- On déplie le prédicat p_1 $(c_1 \vee c_2)$ suivant le dépliage pour le critère de couverture sur les conditions et les décisions :
  - $c_1 = vrai$,
  - $c_2 = vrai$;
- p_2 étant ici une condition, on souhaite uniquement l'évaluer à $vrai$ et à $faux$. Cela conduit au deux cas suivant :
  - $c_1 = faux \wedge c_2 = faux \wedge c_3 = vrai$,
  - $c_1 = faux \wedge c_2 = faux \wedge c_3 = faux$.
- le dernier test permet alors d'évaluer le « else » de l'assertion.

Ici, les tests [c_1 c_2 c_3] suivants satisfont le critère : [1 1 0], [0 0 1] et [0 0 0].

4.5.2.4 Couverture des conditions modifiées des prédicats

Ce critère étend le critère de couverture précédent en introduisant l'exigence selon laquelle chaque condition d'un prédicat p_i doit affecter indépendamment des autres conditions la valeur de ce prédicat. Pour chaque condition d'un prédicat, il faut ainsi définir deux tests couvrant le prédicat (un test fait prendre à p_i la valeur $vrai$; un autre lui fait prendre la valeur $faux$) en fixant les valeurs des autres conditions de p_i. Ce critère est le pendant du critère MC/DC défini en section 4.3.4.

Nous rappelons ici que chaque prédicat p_i de l'assertion peut être mis sous Forme Normale Disjonctive, *i.e.* sous la forme $C_{i1} \vee C_{i2} \vee ... \vee C_{ip}$ où C_{ij} est une conjonction élémentaire. Nous utilisons ici le dépliage N°4 pour critère de couverture MC/DC défini en section 4.5.1.2.

Couverture des conditions modifiées des prédicats : soit p_i un prédicat écrit sous Forme Normale Disjonctive, *i.e.* sous la forme $C_{i1} \vee C_{i2} \vee ... \vee C_{ip}$ où C_{ij} est une conjonction élémentaire. Le critère de couverture des conditions modifiées des prédicats sera satisfait si :

- pour chaque prédicat p_i de l'assertion AltaRica, on applique dans une configuration où $p_i = vrai$ et $p_j = faux$ $\forall j < i$ le dépliage pour le critère de couverture MC/DC décrit en section 4.5.1.2 ;
- il existe au moins un test amenant le modèle dans une configuration vérifiant $\forall i$, $p_i = faux$.

Exemple : Soit l'assertion suivante : « case { $c_1 \vee c_2 : v_1$, $c_3 : v_2$, else v_3 } ». Comme dans l'exemple précédent, on a ici deux prédicats p_1 et p_2 : $p_1 = (c_1 \vee c_2)$; $p_2 = c_3$. Pour satisfaire le critère de couverture des conditions modifiées des prédicats, on procède de la manière suivante :

- On déplie le prédicat p_1 $(c_1 \vee c_2)$ suivant le dépliage pour le critère de couverture des conditions modifiées des prédicats :
  - $c_1 = vrai \wedge c_2 = faux$,
  - $c_1 = faux \wedge c_2 = vrai$;
- p_2 étant ici une condition, on souhaite uniquement l'évaluer à $vrai$ et à $faux$. Cela conduit au deux cas suivant :
  - $c_1 = faux \wedge c_2 = faux \wedge c_3 = vrai$,
  - $c_1 = faux \wedge c_2 = faux \wedge c_3 = faux$.
- le dernier test permet alors d'évaluer le « else » de l'assertion.

Ici, les tests [c_1 c_2 c_3] suivants satisfont le critère : [1 0 0], [0 1 0], [0 0 1] et [0 0 0].

4.5.2.5 Critère de couverture des conditions multiples des prédicats

De manière identique à la section 4.3.4, ce critère vise, pour chaque prédicat p_i à tester l'ensemble des combinaisons possibles des valeurs des conditions. Pour cela, le critère de couverture des conditions multiples des prédicats sera satisfait si :

- pour chaque prédicat p_i de l'assertion AltaRica, on applique dans une configuration où $p_i = vrai$ et $p_j = faux$ $\forall j < i$ le dépliage N°5 pour critère de couverture des conditions multiples défini en section 4.5.1.2 ;
- il existe au moins un test amenant le modèle dans une configuration vérifiant $\forall i$, $p_i = faux$.

Exemple : Soit l'assertion suivante : « case { $c_1 \vee c_2 : v_1$, $c_3 : v_2$, else v_3 } ». Comme dans les exemples précédents, on a ici deux prédicats p_1 et p_2 : $p_1 = (c_1 \vee c_2)$; $p_2 = c_3$. Pour satisfaire le critère de couverture des conditions multiples des prédicats, on procède de la manière suivante :

- On déplie le prédicat p_1 $(c_1 \vee c_2)$ suivant le dépliage pour le critère de couverture des conditions multiples des prédicats :
  - $c_1 = vrai \wedge c_2 = vrai$,
  - $c_1 = vrai \wedge c_2 = faux$,
  - $c_1 = faux \wedge c_2 = vrai$;
- p_2 étant ici une condition, on souhaite uniquement l'évaluer à $vrai$ et à $faux$. Cela conduit au deux cas suivant :
  - $c_1 = faux \wedge c_2 = faux \wedge c_3 = vrai$,
  - $c_1 = faux \wedge c_2 = faux \wedge c_3 = faux$.
- le dernier test permet alors d'évaluer le « else » de l'assertion.

Ici, les tests [c_1 c_2 c_3] suivants satisfont le critère : [1 1 0], [1 0 0], [0 1 0], [0 0 1] et [0 0 0].

4.5.3 Critères de couverture des transitions d'un modèle AltaRica

4.5.3.1 Introduction et idée directrice

Pour rappel, un automate de mode se présente sous la forme d'un 9-uplet (D, dom, S, F^{in}, F^{out}, Σ, δ, σ, I) avec notamment $\delta : dom(S) \times dom(F^{in}) \rightarrow dom(S)$ la fonction de transition qui fournit les prochaines valeurs des variables d'état en fonction de leurs valeurs courantes, des valeurs des variables de flux d'entrée et de l'occurrence de l'évènement induisant le changement d'état. Pour plus de détail concernant un tel automate de mode, on pourra se référer à la section 2.3. En termes pratiques, tester une transition reviendra à tester qu'un modèle AltaRica admet un certain comportement présent dans la spécification de référence. On pourra également tester que le modèle AltaRica n'admet pas de comportements non spécifiés.

En langage AltaRica, nous avons vu en section 2.2.4 qu'une telle transition s'écrivait : $garde(s, f_{in}) \mid\!- e \rightarrow s'$ avec $garde(s, f_{in})$ une condition booléenne sur les évaluations courantes s des variables d'état et f_{in} des variables de flux d'entrée du nœud AltaRica, e un évènement et s' la nouvelle évaluation des variables d'état du nœud. La sémantique d'une telle écriture est similaire à une écriture de la forme « if - then - else » :

If $garde(s, f_{in})$ and e **then** s'

On remarque alors que plutôt qu'une forme en « if - then - else », une transition AltaRica s'écrit sous une forme en « if - then ». Pour tester le franchissement d'une transition, nous souhaitons tester les conditions permettant d'évaluer à vrai l'expression « $garde(s, f_{in})$ and e » (nous ignorons ainsi les conditions conduisant à une garde non satisfaite). Ainsi pour tester le franchissement d'une transition, il nous faut :

- positionner le modèle AltaRica dans une configuration vérifiant l'expression $garde(s, f_{in})$;
- simuler l'évènement e de la transition.

Au delà de tester le simple franchissement d'une transition, nous souhaitons également vouloir tester les différentes façons de rendre vraie l'expression « $garde(s, f_{in})$ ». Pour cela, nous proposons d'écrire cette expression sous forme normale disjonctive (section 4.5.1.1) et d'appliquer les dépliages proposés section 4.5.1.2. Nous définissons ainsi dans la suite différents critères de couverture sur les transitions d'un modèle AltaRica. Ces critères nous permettrons de décomposer le domaine d'activation de la garde de la transition (*i.e.* le domaine de valeur où l'expression $garde(s, f_{in})$ est évaluée à la valeur $vrai$) en sous-domaines, fonctions du critère de couverture souhaité.

Remarque : Pour tester qu'un scénario ne peut se réaliser sur le modèle, la tâche revient à l'heure actuelle à l'analyste. Ainsi, dans une certaine configuration du modèle AltaRica, un évènement **ne devra pas** pouvoir être exécuté sur le modèle AltaRica.

Dans la suite et pour la définition des critères de couverture, nous supposerons que « $garde$ (s, f_{in}) » est écrit sous Forme Normale Disjonctive.

4.5.3.2 Critère de couverture simple d'une transition

Franchir au moins une fois l'ensemble des transitions du modèle AltaRica semble être l'exigence minimum que l'on peut espérer de la phase de test du modèle. Dans l'optique de s'assurer que l'ensemble de tests répond à cet objectif, nous définissons un premier critère de couverture pour les transitions d'un modèle AltaRica.

Critère de couverture simple sur les transitions : Pour toute transition du modèle AltaRica de la forme $garde(s, f_{in}) \mid\!- e \rightarrow s'$, il existe au moins un scénario de test $v = (e_1, ..., e_{p-1}, e, e_{p+1}...)$ tel que le scénario $(e_1, ..., e_{p-1})$ conduise dans une configuration du modèle satisfaisant $garde(s, f_{in})$.

En d'autres termes, pour toutes les transitions du modèle, il existe au moins un test qui permet son franchissement. Pratiquement la mise en œuvre d'un tel critère consiste, lorsque « $garde(s, f_{in})$ » est sous Forme Normale Disjonctive, à utiliser le dépliage N°1 pour le critère de couverture sur les décisions (section 4.5.1.2).

Ce critère, bien que facile à mettre en œuvre, est cependant trop simple pour éprouver de manière suffisante le modèle. En effet, il suffira de satisfaire la garde d'une manière quelconque pour satisfaire le critère. Par exemple, si la garde est « $c_1 \vee c_2$ », franchir la transition avec le couple (c_1, c_2)=$(1, 0)$ suffira à satisfaire le critère. Nous n'aurons pas besoin de tester le franchissement avec c_2=1... Pour faire l'analogie avec les critères présentés en section 4.3.4, ce critère est le pendant du critère de couverture d'instruction.

4.5.3.3 Critère de couverture des conjonctions élémentaires d'une transition

On rappelle dès ici qu'on appelle conjonction élémentaire une expression booléenne de la forme $c_1 \wedge c_2 \wedge ... \wedge c_n$, $n \geq 0$ où chaque c_i est une condition (expression booléenne ne pouvant être décomposée sous forme de conjonction ou de disjonction).

Nous rappelons ici que la garde « $garde(s, f_{in})$ » est écrite sous Forme Normale Disjonctive (FND), *i.e.* sous une forme similaire à $C_1 \vee C_2 \vee ... \vee C_p$ où chaque C_j est une conjonction élémentaire. Pour définir le critère de couverture, nous utiliserons le dépliage N°3 de la FND pour le critère de couverture sur les conditions et les décisions (section 4.5.1.2) et décomposerons donc « $garde(s, f_{in})$ » en p cas différents : $C_i = vrai \ \forall i \in [1, p]$.

Critère de couverture des conjonctions élémentaires des transitions : pour toute transition du modèle AltaRica de la forme $garde(s, f_{in}) \mid\!- e \rightarrow s'$ où $garde(s, f_{in})$ est de la forme $C_1 \vee C_2 \vee ... \vee C_p$, il existe pour chaque conjonction élémentaire C_i un scénario $v_i = (e_{i1}, ..., e_{i(p-1)}, e, e_{i(p+1)}...)$ tel que le scénario $(e_{i1}, ..., e_{i(p-1)})$ conduise le modèle dans une configuration vérifiant C_i=$vrai$.

4.5.3.4 Critère de couverture des conjonctions élémentaires indépendantes d'une transition

Si le critère précédent permet de s'assurer que lors des différents franchissements d'une transition celle-ci est franchie en évaluant au moins une fois chaque conjonction élémentaire à la valeur $vrai$, il n'assure pas encore que la transition est franchie grâce à chaque conjonction élémentaire. Par exemple, pour la transition « $c_1 \vee c_2 \mid\!- e \rightarrow$ », faire deux tests (c_1, c_2) = (1,0) et (c_1, c_2) = (1,1) permet de satisfaire le critère précédent et l'influence de c_2 sur le franchissement de la transition n'est pas incontestable... Pour cela nous définissons un nouveau critère de couverture sur les transitions inspiré du critère MC/DC.

On suppose toujours ici que la garde de la transition est mise sous Forme Normale Disjonctive, *i.e.* de la forme $C_1 \vee C_2 \vee ... \vee C_p$ où C_p est une conjonction élémentaire. Nous utilisons le dépliage N°4 de la FND pour le critère de couverture MC/DC (section 4.5.1.2) qui décompose « $C_1 \vee C_2 \vee ... \vee C_p$ » en p cas : $\forall i \in [1, p]$, C_i=$vrai$ and $(C_j)_{j \neq i} = faux$.

Critère de couverture des conjonctions élémentaires indépendantes des transitions : pour toute transition du modèle AltaRica de la forme $garde(s, f_{in}) \mid\!- e \rightarrow s'$ où $garde(s, f_{in})$ est de la forme $C_1 \vee C_2 \vee ... \vee C_p$, il existe pour chaque conjonction élémentaire C_i un scénario $v_i = (e_{i1}, ..., e_{i(p-1)}, e, e_{i(p+1)}...)$ tel que le scénario $(e_{i1}, ..., e_{i(p-1)})$ conduise dans une configuration du modèle satisfaisant $C_i = vrai$ et $(C_j)_{j \neq i} = faux$.

Ainsi ce critère de couverture pour les transitions d'un modèle AltaRica vise à assurer que, lors d'une disjonction dans la garde, la transition est indépendamment franchie grâce à chacun des termes de la disjonction.

4.5.3.5 Critère de couverture des conjonctions élémentaires multiples d'une transition

Il s'agit ici de tester l'ensemble des combinaisons possibles des valeurs des conjonctions élémentaires de la transition.

Critère de couverture des conjonctions élémentaires multiples des transitions : pour toute transition du modèle AltaRica de la forme $garde(s, f_{in}) \mid\!- \; e \rightarrow s'$ où $garde(s, f_{in})$ est de la forme $C_1 \vee C_2 \vee ... \vee C_p$, il existe pour chaque combinaison de conjonctions élémentaires vérifiant $garde(s, f_{in}) = vrai$ un scénario $v_i = (e_{i1}, ..., e_{i(p-1)}, e, e_{i(p+1)}...)$ tel que le scénario $(e_{i1}, ..., e_{i(p-1)})$ conduise dans une configuration du modèle satisfaisant la configuration en question.

Exemple : Soit la transition « $c_1 \vee c_2 \mid\!- \; e \rightarrow s$ » où c_1 et c_2 sont deux conditions. Ce critère sera vérifié avec un jeu de tests composé de trois scénarios :

- un scénario $v_1 = (e_{11}, ..., e_{1(p-1)}, e, e_{1(p+1)}...)$ tel que $(e_{11}, ..., e_{1(p-1)})$ conduise à $c_1 = vrai$ et $c_2 = vrai$;
- un scénario $v_2 = (e_{21}, ..., e_{2(p-1)}, e, e_{2(p+1)}...)$ tel que $(e_{21}, ..., e_{2(p-1)})$ conduise à $c_1 = faux$ et $c_2 = vrai$;
- un scénario $v_3 = (e_{31}, ..., e_{2(p-1)}, e, e_{3(p+1)}...)$ tel que $(e_{31}, ..., e_{3(p-1)})$ conduise à $c_1 = vrai$ et $c_2 = faux$.

4.6 Implémentation des critères de couverture

La section précédente ayant permis la définition et la mise en place de critères de couverture sur les transitions et les assertions d'un modèle AltaRica, nous souhaitons à présent mettre en place un moyen pratique de mesurer leurs satisfactions directement sur le modèle AltaRica. Pour cela, la méthode proposée est une méthode ayant pour objectif l'insertion de « drapeaux » directement dans le modèle AltaRica. Chaque « drapeau » correspondra à une partie du modèle que nous souhaitons éprouver. Nous verrons que ces drapeaux se formalisent à l'aide de variables d'état booléennes qui seront initialisées à la valeur $faux$. L'implémentation d'un critère consistera à la création d'un certain nombre de ces drapeaux (le nombre en question dépendant du critère) permettant ainsi une traçabilité entre le scénario de test et la partie de modèle éprouvé. En effet, nous verrons que l'activation d'une partie du modèle entraînera la mise à la valeur $vrai$ des drapeaux correspondants. Le taux de couverture du jeu de tests sur le modèle pourra être calculé classiquement à l'aide d'une formule du type :

$$\text{Taux de couverture} = \frac{Nombre\ de\ drapeaux\ à\ la\ valeur\ vrai}{Nombre\ total\ de\ drapeaux}$$

4.6.1 Implémentation des critères de couverture d'une transition

4.6.1.1 Idée générale

Notre objectif est ici de savoir, selon le critère choisi, si une transition donnée a été franchie et de quelle manière s'est fait ce franchissement.

L'idée générale est de créer des « drapeaux » sous forme de variables d'état $flag\_t\_i$ dans le modèle AltaRica. Ces drapeaux seront initialisés à la valeur $faux$ et changeront de valeur dans les assignations des transitions (*i.e.* dans la partie droite de la transition). Ainsi, une transition quelconque « $garde(s, f_{in}) \mid\!- e \rightarrow s'$ » deviendra « $garde(s, f_{in}) \mid\!- e \rightarrow s', flag\_t\_i := true$ ». Observer la valeur de $flag\_t\_i$ nous permettra de savoir si la transition associée a été éprouvée ou non par un scénario de test. Le nombre de drapeaux dépend alors du nombre de transitions initialement implémentées et du critère choisi (*i.e.* de l'objectif des tests à réaliser).

La figure 4.8 se veut reprendre le principe général de cette phase d'insertion des drapeaux. Une fois l'ensemble des drapeaux créé, le jeu de tests est alors simulé sur le modèle. À la fin de chaque test, la valeur des drapeaux est observée et on trace la correspondance entre le test et les drapeaux qu'il active (*i.e.* les transitions qu'il franchit). Ainsi, il y aura autant d'observations que de tests.

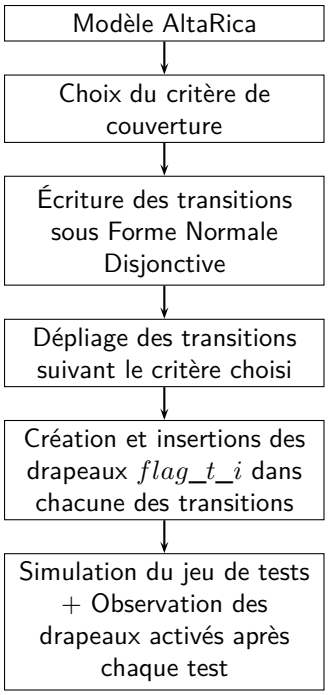


Figure 4.8 – Observation de la satisfaction des critères de couverture sur les transitions

Dans ce qui suit, nous présentons deux applications se voulant servir d'exemples à l'approche. Le principe restera similaire lors de l'application d'autres critères de couverture.

4.6.1.2 Application 1 : Critère de couverture simple d'une transition

Si le critère choisi est le critère de couverture simple (section 4.5.3.2), notre objectif est de franchir au moins une fois chaque transition codée dans le modèle. Pour rappel, le dépliage effectué est en fait une absence de dépliage puisque l'on souhaite uniquement tester le franchissement de la transition sans s'intéresser à la façon dont doit se faire ce franchissement. Nous définissons ainsi un drapeau $flag\_t\_i$ par transition implémentée par l'analyste dans le modèle.

Exemple : La transition « $ST = ok\ or\ ST = degradé\ |-\ perte \rightarrow ST := perdu$ » deviendra « $ST = ok\ or\ ST = degradé\ |-\ perte \rightarrow ST := perdu, flag\_t\_1 := true$ ».

La valeur de $flag\_t\_1$ deviendra $vrai$ lorsque l'on simulera un test franchissant la transition donnée ici en exemple.

4.6.1.3 Application 2 : Critère de couverture des conjonctions élémentaires d'une transition

Si le critère choisi est le critère de couverture des conjonctions élémentaires d'une transition (section 4.5.3.3), on modifie la forme de la transition de sorte à ce que celle-ci soit sous Forme Normale Disjonctive (section 4.5.1), *i.e.* sous la forme $C_1 \vee C_2 \vee ... \vee C_n$ où C_i est une conjonction élémentaire.

L'étape suivante consiste en un dépliage de la transition en accord avec le dépliage correspondant à ce critère. Ainsi, on crée autant de transitions qu'il y a de conjonctions élémentaires dans la garde de la transition. Pour chacune des transitions nouvellement obtenues, on définit un drapeau $flag\_t\_i$.

Exemple : Par exemple, la transition « $ST = ok\ or\ ST = degradé\ |-\ perte \rightarrow ST := perdu$ » sera tout d'abord dépliée en deux transitions :

- $ST = ok\ |-\ perte \rightarrow ST := perdu$;
- $ST = degradé\ |-\ perte \rightarrow ST := perdu$.

On crée ensuite, pour chacune des transitions, un drapeau initialisé à $faux$: $flag\_t\_1$ et $flag\_t\_2$:

- $ST = ok\ |-\ perte \rightarrow ST := perdu, flag\_t\_1 := true$;
- $ST = degradé\ |-\ perte \rightarrow ST := perdu, flag\_t\_2 := true$.

4.6.2 Implémentation des critères de couverture d'une assertion

4.6.2.1 Idée générale

Notre objectif est à présent de tester, selon le critère choisi, si une assertion donnée a été suffisamment éprouvée, de savoir quel prédicat a été testé, quel prédicat ne l'a pas été... Cela nous permettra, comme pour les transitions, d'assurer une traçabilité entre le test et la portion du modèle couverte par celui-ci.

Dans la suite, on se donne une assertion définissant une variable de flux de sortie OUT de la forme suivante où les p_i désignent des prédicats et ou les ω_i appartiennent au domaine de définition de OUT :

OUT = case { $p_1 : \omega_1$,
$p_2 : \omega_2$,
...
$p_{n-1} : \omega_{n-1}$
else ω_n } ;

L'idée générale, similaire à celle utilisée pour la mesure de la couverture des transitions, est de déplier les différents prédicats de l'assertion puis de créer des « drapeaux » sous forme de variables d'état. Ainsi, on créera n variables d'état booléennes $flag\_OUT\_suff_i$ où OUT sera le nom de la variable de sortie concernée par l'assertion et $suff_i$ prendra différentes formes permettant de différencier les drapeaux (*e.g.* $suff_i = i$ ou $suff_i = c_j\_t$). On initialisera chacun de ces drapeaux à la valeur $faux$. Des transitions instantannées (l'évènement de la transition se produit dès que la garde est satisfaite) se chargeront de leur faire prendre la valeur $vrai$. Le principe est illustré par les trois applications ci-dessous (E désigne un évènement instantané - section 2.2.3).

4.6.2.2 Application 1 : Critère de de couverture simple des prédicats d'une assertion

Un drapeau et une transition sont créés pour chaque prédicat p_i de l'assertion. Ce critère peut être satisfait avec $n+1$ tests ou n est le nombre de prédicats de l'assertion (*i.e.* un test pour chaque prédicat et un test pour le « else » où il sera nécessaire d'évaluer l'ensemble des prédicats à $faux$). Par exemple pour l'assertion ci-dessus, l'implémentation du critère de couverture simple des prédicats d'une assertion est telle que :

state
$flag\_OUT\_1$: bool ;
...
$flag\_OUT\_n$: bool ;
init
$flag\_OUT\_1$:=false, ..., $flag\_OUT\_n$:=false ;
trans
$flag\_OUT\_1$=false and p_1=true |- E -> $flag\_OUT\_1$:=true ;
...
$flag\_OUT\_n - 1$=false and (p_1=false and ... and p_{n-2}=false) and p_{n-1}=true |- E -> $flag\_OUT\_n$:=true ;

$flag\_OUT\_n$=false and (p_1=false and ... and p_{n-1}=false) |- E -> $flag\_OUT\_n$:=true ;

Ainsi, à chaque fois qu'un « cas » de l'assertion est activé (lorsque p_i est $vrai$ et que p_j est $faux$ pour $j < i$), une transition est franchie et un drapeau $flag\_OUT\_i$ est évalué à $vrai$. Il le restera tout au long de la simulation du scénario de test et pourra être observé à la fin de celle-ci. Comme pour les transitions, il sera nécessaire d'effectuer une observation par scénario de test effectué.

4.6.2.3 Application 2 : Critère de couverture des conditions / décisions des prédicats

Ici, l'objectif est, tout en satisfaisant le critère précédent, de s'assurer que les conditions présentes dans les prédicats sont évaluées à $vrai$ et à $faux$. Illustrons le principe grâce à une assertion simple : « case { $c_1 \vee c_2 : v_1$, else v_2 } ; ».

---

state
$flag\_OUT\_1\_c1\_t$: bool ;
$flag\_OUT\_1\_c2\_t$: bool ;
$flag\_OUT\_2$: bool ;
init
$flag\_OUT\_1\_c1\_t$:=false, $flag\_OUT\_1\_c2\_t$:=false, $flag\_OUT\_2$:=false ;
trans
$flag\_OUT\_1\_c1\_t$=false and c_1=true |- E -> $flag\_OUT\_1\_c1\_t$:=true ;
$flag\_OUT\_1\_c2\_t$=false and c_2=true |- E -> $flag\_OUT\_1\_c2\_t$:=true ;
$flag\_OUT\_2$=false and (c_1=false and c_2=false) |- E -> $flag\_OUT\_2$:=true ;

4.6.2.4 Application 3 : Critère de couverture des conditions modifiées des prédicats

Pour ce critère, nous déplierons chacun des prédicats en accord avec le dépliage N°4 pour critère de couverture MC/DC défini en section 4.5.1.2.

Pour l'assertion : « case $\{ c_1 \vee c_2 : v_1, c_3 : v_2$, else $v_3 \}$ », une implémentation du critère de couverture des conditions modifiées des prédicats est la suivante :

---

state
$flag\_OUT\_1$: bool ;
$flag\_OUT\_2$: bool ;
$flag\_OUT\_3$: bool ;
$flag\_OUT\_4$: bool ;
init
$flag\_OUT\_1$:=false, $flag\_OUT\_2$:=false, $flag\_OUT\_3$:=false, $flag\_OUT\_4$:=false
trans
$flag\_OUT\_1$=false and c_1=true and c_2=false |- E -> $flag\_OUT\_1$:=true ;
$flag\_OUT\_2$=false and c_1=false and c_2=true |- E -> $flag\_OUT\_2$:=true ;
$flag\_OUT\_3$=false and c_1=false and c_2=false and c_3=true |- E -> $flag\_OUT\_3$:=true ;
$flag\_OUT\_4$=false and c_1=false and c_2=false and c_3=false |- E -> $flag\_OUT\_4$:=true ;

4.7 Retours sur l'approche de validation unitaire proposée

4.7.1 Comment assurer la traçabilité ?

Le souci d'assurer une traçabilité entre les tests effectués sur le modèle AltaRica et les portions du modèle effectivement activées a déjà été évoqué dans ce chapitre. Cela est, nous le pensons, un point capital dans le processus permettant un suivi cohérent des tests. Un tableau tel que celui présenté sur la figure 4.2 se veut reprendre les informations nécessaires à un suivi pertinent de la phase consistant à simuler les tests. Ainsi et grâce à un tableau, nous serons en mesure d'identifier quel drapeau est activé par quel test, de connaître les drapeaux activés par plusieurs tests et ceux non encore activés. En particulier, il sera possible de justifier une non-activation d'un ou plusieurs drapeaux.

Remarque : Au sujet de la non activation souhaitée de certains drapeaux, nous faisons en particulier référence aux assertions et à leurs clauses « else ». En effet, les chemins conduisant à passer dans le « else » sont moins explicite que les chemins activés par les autres cas de l'assertion. Ainsi et pour garder la maîtrise de l'assertion, il est parfois préféré par l'analyste de rendre non évaluable cette clause *i.e.* de ne volontairement pas permettre que cette branche de l'instruction soit évaluée.
Par exemple, au lieu de « OUT = case$\{c_1$ = true : v_1, else $v_2\}$ » on écrira « OUT = case$\{c_1$ = true : v_1, c_1 = false : v_2, else $v_x\}$ » où $v_x \in dom(OUT)$. Dans ce deuxième cas, la clause « else » ne sera jamais évaluée.

Modèle AltaRica	Drapeaux	Description	N° Test	Commentaires
...	...	...	...	...

Tableau 4.2 – Table de traçabilité entre le drapeau activé et le test qui l'active

La table présentée sera ainsi complétée tout au long des différentes simulations. À la suite de chacun des tests, l'analyste en charge de la simulation vérifiera la valeur de chacun des drapeaux implémentés et complétera si besoin cette table de traçabilité. À tout moment et en particulier lorsque tous les tests auront été simulés, il sera aisé de rendre compte, de manière quasi instantanée, des drapeaux ayant été activés et ceux ne l'ayant pas été. Pour ces derniers, on pourra au choix :

- simuler davantage de tests provenant d'une source annexe (jugements experts, scénarios issus du retour d'expérience...) ;
- générer un scénario activant un drapeau particulier (à partir de l'outil de génération de séquences lié au langage AltaRica) ;
- justifier que ce drapeau n'est pas couvert par la phase de test.

4.7.2 Retours sur l'implémentation proposée

L'implémentation des critères de couverture sur un modèle AltaRica, proposée en section 4.6, est fondée sur l'insertion de drapeaux directement au sein du modèle AltaRica (*i.e.* du code AltaRica). L'approche est donc « intrusive » et son influence sur le modèle est multiple :

- ajout de nouvelles transitions ;
- ajout de variables d'état supplémentaires ;
- ajout de transitions supplémentaires.

Bien qu'en théorie et en se rappelant le principe de ces drapeaux, la dynamique implémentée du composant ne soit pas modifiée (les changements de valeur des variables d'état déjà implémentées restent inchangés ; les assertions restent inchangées), le risque lié à l'introduction de ces drapeaux ne peut être totalement considéré comme nul.

Sans proposer pour autant une « validation de la validation » (ce qui nous entraînerait facilement dans une boucle pouvant potentiellement ne pas avoir de fin), nous recommandons la plus grande prudence lors de l'implémentation de ces drapeaux. Pour aller plus loin et lors de futurs travaux, on envisagera une implémentation automatisé de ces drapeaux selon le critère de la section 4.5 choisi. Brièvement, la première étape consistera à la mise de l'expression booléenne sous Forme Normale Disjonctive (section 4.5.1). Une fois cette écriture obtenue, le dépliage de

l'expression peut être systématisé en appliquant de manière itérative les dépliages définis en section 4.5.1.2. Le dépliage pertinent étant obtenu, la création des drapeaux, leurs initialisations et l'écriture des transitions sont triviales.

4.7.3 Retours sur la phase de simulation

D'une manière générale et pour les travaux futurs, l'automatisation de la phase de simulation des scénarios de test présente plusieurs axes de travail.

Le premier besoin se situe au niveau de la simulation à proprement parler. Pour l'instant, les cas d'études restent « scolaires » et le processus reste manuel. En particulier, la génération de tests se fait à partir de la spécification du modèle AltaRica (*e.g.* d'un diagramme états-transitions) et la simulation est réalisée par l'analyste sur le modèle AltaRica. Une amélioration utile serait d'écrire, de manière automatique, ce jeu de tests sous forme de séquences directement lisibles par le simulateur AltaRica (par exemple, par l'outil Cecilia^{TM} OCAS). Ainsi, l'analyste n'aurait plus besoin de simuler « à la main » chacun des évènements composant les différents scénarios de test.

Le second besoin se situe dans l'observation des drapeaux activés lors de la simulation. En effet, après chacune des simulations, il est nécessaire de regarder quels drapeaux ont été activés par le scénario et de remplir la table de traçabilité (section 4.7.1). Avant de lancer une nouvelle simulation (*i.e.* avant de simuler un nouveau scénario de test), le modèle est réinitialisé et avec lui, l'ensemble des drapeaux. L'opération est identique tout au long du jeu de tests et reste manuelle. Il n'est donc pas possible, en l'état, de visionner de manière totalement automatique l'ensemble des drapeaux ayant été activés par le jeu de tests. Ce besoin, combiné au précédent permettrait une automatisation complète de la phase de simulation. Les différentes phases d'une telle approche seraient :

- de transformer le jeu de tests en un jeu de séquences $\{seq_1, seq_2, ..., seq_n\}$ lisible par l'outil supportant la simulation de modèle AltaRica ;
- à partir de ces séquences, d'obtenir une unique séquence ξ : $seq_1 \times reset \times seq_2 \times reset \times ... \times seq_n$, où « $reset$ » est une ré-initialisation du modèle (cet évènement devrait donc être disponible) et où $seq_1 \times seq_2$ signifie que seq_2 est simulée à la suite de seq_1 ;
- de simuler cette séquence ξ sur le modèle ;
- d'observer l'ensemble des drapeaux activés par cette séquence.

Cela permettrait d'obtenir rapidement la couverture d'un jeu de tests sur le modèle. En contrepartie, l'observation du modèle par l'analyste à la suite de chacun des scénarios n'est plus possible. Le processus décrit ci-dessus n'est alors pas applicable en l'état sur les outils disponibles à l'heure actuelle. Cela ne représente que des possibilités qui devront être davantage étudiées dans de futurs travaux.

Chapitre 5

Cas d'étude et retour sur l'approche

Sommaire

« Plutôt que de répéter sans cesse à l'enfant que le feu brûle, consentons à le laisser un peu se brûler : l'expérience instruit plus sûrement que le conseil. »

André Gide - Les Faux Monnayeurs

Les chapitres 3 et 4 ont présenté respectivement les processus de modélisation et de validation d'un modèle AltaRica permettant à l'analyste de s'assurer que le modèle en question représente bien et de manière valide ce qu'il est censé décrire. Au delà des aspects théoriques présentés dans ces chapitres, le travail réalisé au cours de cette thèse comportait également un aspect pratique non négligeable. L'objectif du présent chapitre est alors de présenter cet aspect pratique sur notre cas d'étude : un moteur d'hélicoptère.

Nous étudierons dans la section 5.1 l'application de la méthodologie proposée chapitre 3 et la construction des bibliothèques de composants pour différents sous-systèmes du moteur d'hélicoptère. Nous donnerons suite à cela un retour d'expérience sur les différentes analyses effectuées. Concernant la validation des modèles AltaRica réalisés, nous présenterons dans la section 5.2 les résultats issus de la comparaison entre les chemins de propagation présents dans les arbres de défaillance et ceux fournis par les modèles. Nous présenterons les analogies entre les deux types de résultats ainsi que leurs différences. Ensuite et même si celui-ci n'en est qu'à ses prémisses, nous discuterons de la mise en place du processus de validation présenté au chapitre 4.

Nous reviendrons finalement dans la section 5.3 sur quelques avantages liées à l'utilisation de l'approche « Model Based Safety Analysis ». Nous fournirons en section 5.4 ce qui nous semble être des « bonnes pratiques » à avoir lors de l'activité de modélisation. Enfin nous présenterons en section 5.5 les difficultés rencontrées et discuterons de possibilités pour les résoudre.

5.1 Bilan de l'activité de modélisation

Nous avons présenté en section 3.3 le cas d'étude ayant illustré la construction des bibliothèques de composants en langage AltaRica. Nous nous proposons de faire ici un bilan de l'activité de modélisation. Nous présenterons quelques chiffres relatifs à l'utilisation des bibliothèques réalisées et discuterons du « périmètre » couvert par les modèles ayant été réalisés.

Remarque : Pour limiter toute confusion dans la suite, nous parlerons de « composant » pour désigner un composant physique et nous parlerons de « nœud » pour désigner un modèle AltaRica d'un composant ou d'une fonction.

5.1.1 Bibliothèques AltaRica pour la modélisation du moteur d'hélicoptère

Pour rappel, le chapitre 3 s'est intéressée à la mise en place de différentes bibliothèques et nous a permis de modéliser divers sous-systèmes (tableau 5.1) d'un turbomoteur d'hélicoptère tels que :

- le circuit d'air principal (modélisé en section 3.6 - nous l'appellerons *S_Air*) du turbomoteur qui crée de la puissance à partir d'une entrée d'air et de carburant ;
- le circuit d'alimentation en carburant (modélisé en section 3.5 - nous l'appellerons *S_Fuel*) qui conditionne, dose et qui distribue la juste quantité de carburant à la chambre de combustion du moteur (*i.e.* là où le carburant et l'air se mélangent) ;
- différents systèmes mécaniques (section 3.4) principalement composés d'engrenages et ayant pour fonction de transmettre un mouvement de rotation d'un point d'entrée à un point de sortie :
  - un système (*S_Grbx*) transmet la puissance du circuit d'air principal aux accessoires du moteur (*e.g.* pompes, démarreur...) et réduit la vitesse de rotation,
  - un système (*S_Red*) transmet la puissance du circuit d'air principal au rotor de l'hélicoptère ;
- le circuit de lubrification (*S_Oil*) du turbomoteur qui conduit l'huile de son réservoir jusqu'à différents emplacements du turbomoteur (*e.g.* jusqu'aux différents roulements des engrenages) ;
- différentes logiques relatives à la surveillance et au contrôle du turbomoteur (section 3.7 - le système modélisant ces logiques sera nommé *S_Moni*) ;

Nous avons développés cinq bibliothèques nous permettant de modéliser ces sous-systèmes :

- une bibliothèque *Lib_Meca* nous permet de modéliser les systèmes de transmission mécaniques composés principalement d'engrenages, d'arbres de transmission et de roulements ;
- une bibliothèque *Lib_Fuel* nous permet de modéliser le circuit carburant d'un turbomoteur ;
- une bibliothèque *Lib_Oil* nous permet de modéliser le circuit de lubrification d'un turbomoteur ;
- une bibliothèque *Lib_Air* nous permet de modéliser le circuit d'air principal d'un turbomoteur ;
- une bibliothèque *Lib_Moni* nous permet de modéliser des fonctions utilisées pour la surveillance du turbomoteur.

Le tableau 5.1 reprends quelques caractéristiques des modèles réalisés. Cependant, les chiffres présentés sont donnés à titre indicatifs mais ne représente pas la modélisation de l'ensemble d'un turbomoteur (voir section 5.1.1.3)

Sous-système	S_Grbx	S_Red	S_Air	S_Oil	S_Fuel	S_Moni
Bibliothèque	Lib_Meca		Lib_Air	Lib_Oil	Lib_Fuel	Lib_Moni
Nombre d'évènements redoutés étudiés	2		3	3	4	1
Nombre de nœuds dans la bibliothèque	11		122	25	26	41
Nombre d'instances dans le modèle	37	18	124	34	31	160

Tableau 5.1 – Caractéristiques des modèles réalisés

Ces cinq bibliothèques contiennent 238 nœuds (quelques nœuds transverses aux bibliothèques ont été ajoutés, *e.g.* portes logiques pour certains aspects fonctionnels de la modélisation). À partir de l'ensemble de cette bibliothèque, les modèles des sous-systèmes ont été construits. Ensuite et par connexion de ces modèles de sous-système, un modèle « global » a été réalisé.

Le modèle « global » est constitué de 586 instances de nœud (586 instances des 238 nœuds des bibliothèques). Ces 586 instances introduisent 642 évènements terminaux dans le modèle.

Nous développerons dans la suite de cette section une analyse détaillée des informations présentes dans le tableau 5.1.

5.1.1.1 Généricité des nœuds au sein d'un modèle

La philosophie visant à la réutilisation des modèles de la bibliothèque étant sans aucun doute fondamentale et directrice lors de la modélisation d'un système en langage AltaRica, il est légitime d'observer le degré de réutilisation effectif présent dans les modèles. D'une façon simple, ce degré peut être formalisé comme le ratio du nombre de nœuds dans la bibliothèque sur le nombre d'instances dans le modèle.

$$D_R = \frac{Nombre\ de\ noeuds\ dans\ la\ bibliothèque}{Nombre\ d'instances\ dans\ le\ modèle\ du\ sous-système}$$

Ainsi, une valeur de D_R proche de l'unité indiquera un degré de réutilisation faible du modèle. Au contraire, plus D_R sera petit et proche de 0, plus le taux de réutilisation des nœuds sera important.

Sous-système	S_Grbx	S_Red	S_Air	S_Oil	S_Fuel	S_Moni
D_R	0.3	0.6	0.98	0.73	0.84	0.25

Tableau 5.2 – Degré de réutilisation des nœuds dans les modèles réalisés

On voit donc que, si le degré de réutilisation des nœuds décrivant des composants informatiques est bon (un nœud est en moyenne instancié quatre fois), celui des nœuds utilisés pour décrire le circuit d'air est quasiment nul. Cela est après analyse justifiable par plusieurs raisons.

Justifions déjà le grand nombre de composants dans le modèle. Le circuit d'air est intrinsèquement plus important en taille. Celui-ci représente en effet quasiment à lui seul toute la mécanique présente dans le moteur : entrée d'air, plusieurs compresseurs, une chambre de combustion, plusieurs turbines, des arbres de transmission auxquels s'ajoutent tous les composants nécessaires à la mise et au maintien en position de ces différentes pièces. Une seconde justification vient du fait que si pour les systèmes S_Grbx, S_Red, S_Oil ou S_Fuel, il est possible de construire un modèle très proche de l'architecture matérielle du système réel, nous sommes dans S_Air contraint de raisonner « fonctionnel » à de nombreuses reprises. Ainsi, là où dans les autres systèmes nous avions souvent un composant physique représenté par un nœud AltaRica, ceci n'est pas souvent applicable dans le modèle du circuit d'air où plusieurs nœuds AltaRica représentent un composant physique.

Concernant le faible degré de réutilisation des composants au sein de S_Air, cela s'explique par l'étendue spatiale d'un tel système. En effet, les conditions d'utilisation d'un même composant peuvent être multiples selon sa situation dans le système : les vitesses de rotation évoluent, la pression d'air évolue, la gamme de température varie d'une température ambiante à plus de 1200°C à la sortie de la chambre de combustion... Il en résulte que souvent pour un même composant, ses modes de défaillance ou les probabilités d'occurrence de ses modes de défaillance ne soient pas identiques selon son utilisation. Dans un tel cas et en langage AltaRica, il est nécessaire de modéliser différents nœuds : un pour chaque cas d'utilisation du composant. La figure 5.1 présente une vue simple du système S_Air.

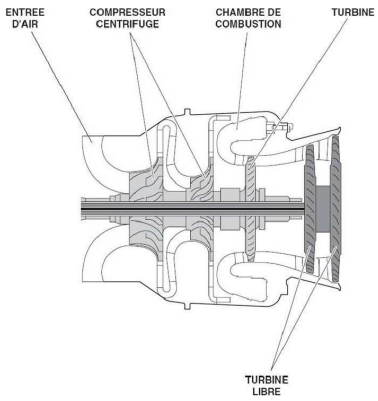


Figure 5.1 – Vue en coupe du circuit d'air principal d'un turbomoteur

5.1.1.2 Pourquoi Lib_Oil $\neq$ Lib_Fuel ?

Une question légitime à ce stade serait au sujet de l'utilisation de bibliothèques différentes pour modéliser un circuit carburant et un circuit d'huile. En effet, ces deux types de systèmes étant composés des mêmes types de composants (pompes, filtres, valves, injecteurs...), il aurait été sans doute dans la logique de l'approche d'utiliser une bibliothèque commune pour la modélisation de ces deux sous-systèmes. En pratique, cela n'est pourtant pas le cas... Pourquoi ?

Pour répondre à cette question et justifier notre choix, il est possible de se rappeler la section 3.2.1 et en particulier l'étape d'analyse préliminaire d'un système nécessaire à la réalisation d'un modèle. Il avait été noté l'importance de l'identification des besoins de l'analyste (des buts du modèle) en amont de la modélisation pour définir ce qu'on souhaitait étudier sur le modèle. Nous avons ensuite pris le parti d'inclure autant que possible uniquement les informations nécessaires et suffisantes pour répondre aux besoins de l'analyste, *i.e.* ne pas inclure d'informations inutiles dans le modèle qui alourdiraient celui-ci.

Or, comme le montre le tableau 5.1, le nombre d'évènements redoutés étudiés sur le circuit carburant et celui de lubrification n'est pas le même (les ensembles d'objectifs des deux modèles ne sont pas les mêmes). Les flux et leurs granularités ne seront donc pas identiques dans les deux modèles. Par exemple, si nous souhaitons observer les oscillations de débit dans le circuit carburant, cela ne nous importe pas dans le circuit de lubrification. Plutôt que d'alourdir le modèle du circuit de lubrification avec des informations qui lui seraient inutiles, nous avons mis en place deux bibliothèques AltaRica distinctes pour modéliser le circuit carburant et le circuit d'huile.

5.1.1.3 Couverture « spatiale » du système d'étude

Une fois les bibliothèques permettant la modélisation du moteur d'hélicoptère présentée, il est légitime de vouloir clarifier quelle portion d'un turbomoteur a réellement été modélisée lors des travaux décrits dans ce mémoire. Au delà des systèmes décrits ci-dessus et par manque de temps, deux manquent à l'appel dans le modèle AltaRica réalisé :

- le circuit d'air secondaire a pour fonction d'assurer le prélèvement d'air primaire (*i.e.* air à la sortie des compresseurs) pour assurer l'équilibre thermique des pièces (répartir les températures des pièces et éviter un échauffement excessif), l'équilibre des forces (diminuer les efforts sur certaines pièces) et l'étanchéité par pressurisation de certains joints ;
- le système de régulation a pour fonction de maintenir automatiquement le moteur dans des limites déterminées en calculant la juste quantité de carburant à délivrer dans la chambre

de combustion en fonction de la valeur de signaux de référence (pression, température, vitesse...).

Concernant la modélisation du système de régulation, nous ne pouvons que fournir des résultats issus de [33] qui s'est intéressé à la modélisation des erreurs logicielles dans le système de régulation d'un turbomoteur. Le périmètre de cette étude était composé de l'ensemble des composants fournissant des données au calculateur (capteurs moteurs, commandes hélicoptère...) ainsi que des différentes fonctions assurées par le calculateur (acquisition des données, pilotage des sorties, surveillance, gestion des communications à l'intérieur du calculateur et avec l'hélicoptère...). Le modèle réalisé contenait environ 60 nœuds dans la bibliothèques, 200 instances dans le modèle et 360 évènements. Ces évènements concernaient principalement les fonctions logicielles et décrivaient des erreurs de conception hypothétiques. Bien que cela ne soit pas réellement le cas d'application traité dans les analyses de sécurité actuelles (on y traite plutôt les défaillances aléatoires plutôt que celles systématiques dues à la conception), cela nous fournit une indication pertinente sur la potentielle taille du modèle d'un système de régulation. À lui seul donc, il se pourrait que le modèle du système de régulation soit équivalent au tiers du modèle présenté en terme d'architecture (200 instances là où nous en avons 586) et à sa moitié (au moins) en terme de comportements (360 évènements là où nous en avons 642).

Concernant le système d'air secondaire, il s'agit clairement d'une perspective intéressante pour la suite de ces travaux. Sans pour autant trop s'avancer sur la taille ou la complexité d'un potentiel futur modèle, certaines caractéristiques d'un tel système nous permettent cependant de réfléchir. En effet, le principe et la fonction du circuit d'air secondaire est de prélever de l'air sur le circuit d'air primaire (celui-ci est modélisé) et d'envoyer cet air à différents endroits du moteur. Pour ce faire, l'air emprunte des chemins à travers les composants mécaniques du moteur (une partie est modélisée) et arrive jusqu'aux composants à refroidir (turbines, compresseurs... eux aussi déjà modélisés). Sans prétendre que l'ensemble des nœuds nécessaires à la modélisation du circuit d'air secondaire sont déjà présents dans les bibliothèques et dans notre modèle (les joints par exemple ne le sont pas pour la plupart), nous pensons que la complexité de modélisation d'un tel système résidera plus probablement dans les nombreuses interactions qu'il implique et qu'il crée entre les autres sous-systèmes déjà modélisés que dans la modélisation des composants de ce système.

5.1.2 Retour sur l'analyse des modèles

5.1.2.1 Couverture de l'étude des évènements redoutés

Dans un premiers temps, précisons qu'actuellement les évènements redoutés de haut niveau étudiés sur le cas d'application présenté sont au nombre de 16 [1]. Ceux-ci sont d'origines diverses et proviennent aussi bien du règlement moteur [27], d'exigences clients ou encore d'exigences internes.

Parmi ces 16 évènements redoutés, nous nous sommes donné 3 évènements redoutés de haut niveau à étudier : l'arrêt en vol du moteur, la non disponibilité de la puissance maximale ainsi que l'occurrence d'un feu sur le moteur. En déclinant ces évènements redoutés aux sous-systèmes, le tableau 5.1 montre que les ensembles d'objectifs de chacun des sous-modèles sont différents : tous les systèmes ne sont pas concernés par l'ensemble des évènements redoutés ; tous les évènements redoutés ne se manifestent pas de façon similaire sur les systèmes.

1. Sans donner l'ensemble de ces évènements redoutés, nous pouvons citer l'arrêt en vol du moteur, l'impossibilité d'arrêter le moteur si cela est souhaité, l'indisponibilité de la puissance maximale pouvant être fournie par le moteur, l'occurrence d'un feu (contrôlé et non contrôlé) sur le moteur, l'émission d'air toxique en cabine, une rupture du système d'attache entre le moteur et l'hélicoptère...

5.1.2.2 Commentaires sur les durées d'analyse

Sur une configuration bureautique (Intel Celeron 3.06 GHz, 1 Go de RAM), les temps de calculs pour ce modèle et en considérant les 3 évènements redoutés considérés (section 5.1.2.1) sont fournis dans le tableau 5.3.

	Étude d'un sous-système	Étude du système global
Ordre 1	Quelques secondes	De 1 à 5 minutes
Ordre 2	1 ou 2 minutes	$\simeq$ 15 minutes
Ordre 3	$\simeq$ 15 minutes	17% des coupes en 15H...

Tableau 5.3 – Temps de calcul

D'après [33], le temps de calcul est proportionnel au nombre d'évènements présents dans le modèle et exponentiel par rapport à l'ordre des séquences demandées. En augmentant la taille du modèle, on peut donc supposer que le temps de calcul va augmenter de manière importante mais linéaire tant que nous restons à des analyses d'ordres 1, 2 ou 3.

5.2 Bilan de l'activité de validation des modèles

Nous présentons dans cette section les activités ayant été réalisées pour valider les modèles AltaRica construits. Ces activités sont alors fondées sur des activités de revue du modèle (section 5.2.1) et sur la comparaison entre les résultats fournis par le modèle et ceux des analyses classiques de sécurité de type AMDE et arbres de défaillance (section 5.2.2). Finalement, nous montrons en section plan : Chapitre 5 : Bilan Validation : vers le processus un exemple d'application du processus de validation présenté au chapitre **??**.

5.2.1 Revues des modèles

Actuellement, une première approche visant à valider le comportement du modèle est de pratiquer une revue du modèle pour détecter d'éventuelles erreurs. Une telle activité de revue consistera la plupart du temps à simuler un certains nombre de scénarios sur le modèle et observer que le résultat obtenu est bien celui attendu. Si l'ingénieur en charge de la revue est familier avec le langage AltaRica et avec l'outil, il est également possible de faire une revue du code implémentant le comportement observé lors de la simulation (revue boîte noire / revue boîte blanche - section 4.3.3). Ces revues seront en pratique effectuées par :

- des ingénieurs ou des experts Sûreté de Fonctionnement ;
- des ingénieurs ou des experts des différents métiers traités dans les modèles (informatique, mécanique, hydromécanique...).

Les deux types de revues sont alors complémentaires. En effet, là ou un ingénieur Sûreté de Fonctionnement sera à même de confirmer l'approche employée (les objectifs du modèle, la logique d'implémentation, le niveau d'abstraction...), l'ingénieur métier pourra quant à lui assurer que les phénomènes physiques décrits le sont de manière correcte. Ensemble, ils pourront attester de la bonne représentation des scénarios de pannes observés.

5.2.2 Comparaison des résultats entre modèle AltaRica et arbres de défaillance

Une fois la revue du modèle effectuée et avant de s'intéresser à l'application du processus de validation défini au chapitre 4, nous nous sommes intéressé à la comparaison entre les résultats fournis par le modèle et ceux fournis par les arbres de défaillance qui nous le rappelons sont existants dans le contexte de recherche dans lequel se déroule les travaux rapportés ici.

Le travail étant conséquent et les retombées théoriques limitées, nous ne présentons que le principe général de l'approche et les leçons qui en sont tirées.

5.2.2.1 Processus expérimental

Notre objectif est de s'assurer que les résultats fournis par les modèles sont similaires à ceux fournis par les arbres de défaillance actuels. Pour cela, nous utilisons les outils d'analyse supportant le langage AltaRica. Sont donc candidats l'outil de génération d'arbres de défaillance et l'outil de génération des séquences conduisant à l'occurrence d'un évènement redouté (section 2.4.3.1). Nos modèles comportant des reconfigurations et des boucles de modélisation, nous utiliserons le générateur de séquences. Ces différentes séquences sont représentés sous forme d'arbres de défaillance de type « râteau » : un évènement de haut niveau, une porte « OU » ayant en entrée l'ensemble des séquences. Une fois ces séquences obtenues et mises sous forme d'arbres de défaillance, l'outil Cecilia^{TM} Arbor nous permet d'importer cet arbre afin d'en réaliser l'étude quantitative.

Le principe de la comparaison est donc d'une part, de comparer les séquences données par le modèle AltaRica avec les coupes minimales fournis par les arbres de défaillance. Une fois ces séquences comparées, nous comparons la probabilité d'occurrence de l'évènement redouté considéré à l'aide de l'outil Cecilia^{TM} Arbor. Si une différence est détectée, deux cas de figure se présentent :

- si la différence est une erreur du modèle, il faudra modifier le modèle ;
- si la différence est due à une erreur présente dans les arbres de défaillance, celle-ci devra être tracée et justifiée.

5.2.2.2 Comparaison des résultats qualitatifs

L'analyse qualitative d'un modèle AltaRica s'intéresse à l'identification des chemins de propagation de défaillances du système représenté par le modèle. Ces chemins correspondent aux séquences obtenues à partir du modèle AltaRica (grâce à l'outil Cecilia^{TM} OCAS). Le principe du travail décrit ici est d'observer si pour toutes les séquences minimales issues du modèle AltaRica, il existe une coupe correspondante issue des arbres de défaillance. Par contre, l'arbre de défaillance et le modèle AltaRica portant sur des périmètres de systèmes différents, il n'est pas pertinent de vérifier que l'ensemble des coupes de l'arbre sont incluses dans les séquences générées à partir du modèle AltaRica.

Sans autre outil que Microsoft Excel pour supporter cette activité, ce travail a été réalisé pour l'occurrence d'un évènement redouté (l'arrêt en vol) et sur les modèles suivants :

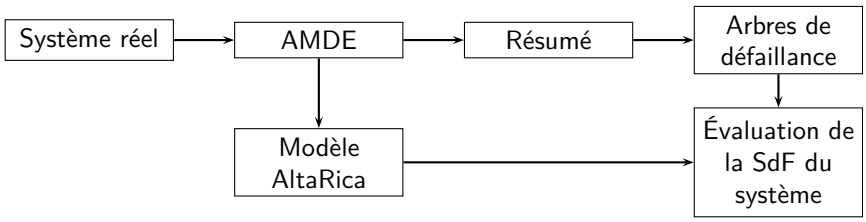
- S_Grbx et S_Red pour les séquences d'ordres 1, 2 et 3 ;
- S_Oil, S_Fuel pour les séquences d'ordres 1 et 2 ;
- S_Air pour les séquences d'ordre 1 ;

Par comparaison à l'ordre 1, nous entendons en fait la validation des effets des défaillances simples. Cela revient en fait à comparer les séquences d'ordre 1 issues du modèle avec les défaillances présentes dans l'AMDE. Concernant les ordres supérieurs, cela nous permettra de valider

les combinaisons de défaillances décrites dans les modèles (par exemple de valider les défaillances dormantes et les phénomènes qui les activent).

Des différences ont été identifiées entre les résultats des deux approches. Par exemple, l'arbre de défaillance est parfois plus concis que le modèle AltaRica. Une des raisons est la différence existante au niveau même des modèles. Par exemple dans un arbre de défaillance, les ruptures des différents pignons de S_Grbx sont regroupées dans un unique évènement élémentaire. Dans le modèle AltaRica, on conserve un évènement par composant « Pignon ». Ainsi, là où l'arbre de défaillance nous fournit une unique coupe « Rupture des pignons », le modèle AltaRica nous en fournit n où n est le nombre de pignons (« Rupture des pignons »= « Rupture pignon 1 » or ... or « Rupture pignon n »).

Remarque : Comme le présente la section 1.5.3.2, un résumé d'AMDE précède en pratique la construction d'un arbre de défaillance. Dans la méthodologie proposée au chapitre 3 et pour rester au plus bas niveau, nous avons choisi de construire le modèle AltaRica directement à partir de l'AMDE *i.e.* sans passer par un résumé. Cela nous permet ainsi de conserver une structure du modèle AltaRica proche de l'architecture réelle du système à modéliser. En contrepartie, les séquences obtenues grâce au modèle AltaRica sont parfois de plus bas niveau que les coupes des arbres de défaillance. Cela explique alors la différence mentionnée ci-dessus.



Une autre raison est la différence de périmètres entre les deux approches. En effet, on ne pourra comparer les résultats entre modèle et arbres qu'en s'assurant que les périmètres des deux modélisations soient identiques. Cela pose une difficulté importante, surtout en début de modélisation, puisque dans les arbres, les interactions entre différents sous-systèmes rentrent rapidement en jeu. Cependant, au fur et à mesure de l'intégration des différents sous-systèmes, nous sommes parvenus à retrouver dans le modèle AltaRica une importante quantité des chemins de propagation de défaillance présents dans les arbres de défaillance.

5.2.2.3 Comparaison des résultats quantitatifs

Une fois les résultats qualitatifs retrouvés, la comparaison des résultats quantitatifs est immédiate grâce à l'outil Cecilia^{TM} Arbor. Insistons cependant sur deux « bonnes pratiques » à adopter lors de la phase de modélisation pour que l'obtention des résultats quantitatifs soit possible.

- Comme indiqué en section 2.2.3, les lois de probabilité doivent être indiquées dans les nœuds AltaRica pour pouvoir réaliser des analyses quantitatives sur les arbres extraits du modèle AltaRica.
- Concernant les pannes dormantes, il ne faut pas oublier d'inclure les temps de latence dans les nœuds AltaRica. En effet là aussi, ces temps de latence seront exportés sous Cecilia^{TM} Arbor.

En suivant ces recommandations, on parvient à retrouver dans Cecilia^{TM} Arbor les probabilités des évènements élémentaires présents dans les arbres et la probabilité de l'évènement redouté de haut niveau. Par contre et étant donné que l'arbre est sous-forme de « râteau », on perd la représentation et la probabilité des évènements intermédiaires présents dans les arbres de défaillance.

Remarque : L'arbre représentant les séquences générées par le modèle peut être vu d'une certaine manière comme une « mise à plat » de l'arbre de défaillance disponible dans les analyses de sécurité actuelles.

5.2.3 Vers l'utilisation du processus de validation proposé

Nous avons proposé au chapitre 4 un processus visant à valider le comportement du modèle AltaRica et plus particulièrement des nœuds d'une bibliothèque. Nous rappelons qu'il est composé de trois étapes principales : la validation unitaire (*i.e.* la validation de la bibliothèque de nœud AltaRica), la validation de l'intégration et la validation globale (figure 4.2).

Ce processus concerne surtout la validation unitaire. Nous avons alors vu que potentiellement, la validation d'un nœud peut nécessiter l'introduction d'un nombre important de modifications (section 4.6) dans le code AltaRica implémentant le nœud. N'ayant alors pas d'outil supportant l'écriture automatique de ces modifications (section 4.7.3), celui-ci n'a pas été au cours de ces travaux déployé à une échelle importante. Ainsi, seul le principe général de l'approche a été étudié en l'appliquant à un nœud AltaRica unitaire. L'intégration des nœuds entre eux ainsi que le modèle global n'ont pas été validé de la sorte.

Nous montrons ici l'application du processus au composant *Valve* dont le code est présenté figure 2.6. Une spécification typique du comportement d'une telle valve est présentée figure 2.1. De ce schéma, on tire sans grande difficulté et en suivant le principe de l'exemple page 116 un ensemble de tests T composé de plusieurs scénarios de test :

- des scénarios permettant de s'assurer que les comportements spécifiés sont correctement implémentés : {$\emptyset$}, {Blocage}, {Fermeture}, {Fermeture, Ouverture}, {Fermeture, Blocage} ;
- des scénarios permettant de s'assurer que les comportements non spécifiés ne sont pas implémentés : {Ouverture}, {Blocage, Ouverture}, {Blocage, Fermeture}, {Blocage, Blocage}, {Fermeture, Fermeture}, {Fermeture, Blocage, Fermeture}, {Fermeture, Blocage, Ouverture} et {Fermeture, Blocage, Blocage}.

Pour rappel, les transitions implémentant ce comportement sont les suivantes :

```
trans
  bloque=false |- Blocage -> bloque :=true ;
  etat=ouverte and bloque=false |- Fermeture -> etat :=fermee ;
  etat=fermee and bloque=false |- Ouverture -> etat :=ouverte ;
```

Imaginons pour présenter le principe de l'approche que l'analyste en charge de l'implémentation fasse une erreur et implémente de la façon suivante (la fermeture et l'ouverture peuvent alors avoir lieu même après blocage de la valve) :

```
trans
  bloque=false |- Blocage -> bloque :=true ;
  etat=ouverte |- Fermeture -> etat :=fermee ; % Erreur
  etat=fermee |- Ouverture -> etat :=ouverte ; % Erreur
```

Les gardes des transitions étant de simples conditions (en particulier elles sont donc déjà sous forme normale disjonctive), les critères définis en section 4.5.3.4 sont équivalents à franchir la transition en évaluant la condition à la valeur $vrai$. En insérant les drapeaux (tous initialisés à la valeur « $false$ ») conformément au chapitre 4, les transitions deviennent :

```
trans
  bloque=false |- Bloquage -> bloque :=true, flag_t_1:=true ;
  etat=ouverte |- Fermeture -> etat :=fermee, flag_t_2:=true ;
  etat=fermee  |- Ouverture -> etat :=ouverte, flag_t_3:=true ;
```

La simulation des scénarios de test de T permet d'activer l'ensemble des drapeaux définis (l'état initial est etat=ouverte et bloque=false) :

- {Blocage} active le drapeau 1 ;
- {Fermeture} active le drapeau 2 ;
- {Fermeture, Ouverture} active les drapeaux 2 puis 3 ;
- {Fermeture, Blocage} active les drapeaux 2 puis 1.

En poursuivant la simulation de scénarios de test, on obtient :

- les scénarios {Ouverture}, {Blocage, Blocage}, {Fermeture, Fermeture}, {Fermeture, Blocage, Fermeture} et {Fermeture, Blocage, Blocage} sont impossibles à réaliser conformément au comportement spécifié ;
- les scénarios {Blocage, Ouverture}, {Blocage, Fermeture} et {Fermeture, Blocage, Ouverture} peuvent par contre être simulés sur le modèle AltaRica alors qu'il s'agit de comportements non spécifiés.

Deux solutions sont alors possibles : soit la spécification du nœud est fausse ou incomplète et ne fournit pas un ensemble satisfaisant de scénarios de test, soit le nœud est implémenté de manière incorrecte. Dans l'un ou l'autre des cas, le résultat impose à l'analyste d'approfondir la question. Ici, l'analyste se rendra compte que les scénarios sont erronés et pourra corriger l'implémentation en conséquence.

Le principe est identique concernant la validation des assertions. En imposant une réflexion sur à la fois la spécification et le nœud AltaRica, le processus permet de détecter de potentielles erreurs dans ces deux documents. Ainsi on pourra détecter des comportements admis par le nœud mais non spécifiés. On pourra également détecter des comportements spécifiés mais non permis par le nœud (*i.e.* scénarios non réalisables sur le nœud).

5.3 Accent sur quelques avantages de l'approche « Model Based Safety Analysis » et de la méthodologie de modélisation proposée

La méthodologie présentée au chapitre 3 s'appuie sur l'approche « Model Based Safety Analysis ». En tant que telle, différents de ces avantages peuvent être explicités ici :

- Garantir la cohérence de différents arbres de défaillance sur un même système.
- Garantir la cohérence des analyses de sécurité entre différents systèmes.
- Assurer une meilleure traçabilité entre les analyses fonctionnelles et dysfonctionnelles d'un système.
- Capitaliser la connaissance fonctionnelle et dysfonctionnelle des composants au sein de modèles génériques de bas niveau.
- Mieux expliciter l'effet physique d'une défaillance sur le comportement du système.
- Outiller l'élaboration d'un modèle avec une bibliothèque de nœuds génériques.
- Avoir une représentation fonctionnelle du comportement du système considéré.
- Avoir une représentation graphique moins abstraite, pour un non-connaisseur, que les arbres de défaillance.

5.3.1 Capitalisation des connaissances

Un avantage de l'approche à base de modèles est de permettre une capitalisation de la connaissance des composants dans des nœuds AltaRica génériques. En effet et puisque la sémantique formelle d'un nœud AltaRica est fondée sur la notion d'automate de mode (section 2.3), on modélise au sein d'un élément de la bibliothèque diverses informations concernant le comportement fonctionnel et dysfonctionnel du ou des composants qu'il représente telle que :

- les données influençant son comportement (ses entrées) ;
- ses évènements fonctionnels et dysfonctionnels (ses reconfigurations, ses modes de défaillance...) ;
- l'influence de ces évènements sur son état (sa logique interne) ;
- ses fonctions de transfert et son effet sur l'environnement (ses sorties et leurs modes de calcul).

Ainsi, plus qu'un simple modèle des propagations de défaillance dans un système, le modèle AltaRica inclut diverses informations concernant le comportement de ses composants. Il inclut et capitalise le savoir de l'analyste et des experts concernant les aspects fonctionnels et dysfonctionnels des composants.

Remarque : Attention. Cette capitalisation doit être accompagnée d'une phase de documentation rigoureuse des modèles réalisés. En effet, notre expérience nous a montré que la description d'un composant peut rarement se targuer d'être totalement générique : souvent doit être pris en compte le domaine et le cas d'utilisation du ou des composants modélisés. Par exemple et selon les cas d'utilisation d'un composant, les modes de défaillance ou leurs taux d'occurrence peuvent varier. Dans un tel cas et même si la référence est identique dans le système réel, le nœud AltaRica utilisé devra être différent.

5.3.2 Traçabilité et cohérence des analyses

Comme annoncé en début de section, la méthodologie proposée dans ce mémoire et plus généralement l'approche « Model Based Safety Analysis » permet d'améliorer la traçabilité et la cohérence entre :

- les modèles métiers et les modèles de Sûreté de Fonctionnement ;

- l'analyse de l'occurrence de plusieurs évènements redoutés sur un même système ;
- les analyses de sécurité réalisées sur des systèmes différents.

Concernant le premier point, les modèles réalisés ont été choisis de bas niveau afin d'être en mesure de retrouver les résultats issus des analyses actuelles de sécurité. Descendant dans une telle analyse au niveau du composant, les modèles AltaRica réalisés sont donc très proches des architectures réelles des systèmes modélisés (systèmes mécaniques, hydromécaniques, informatiques...).

Concernant les deux autres points et en partie grâce à la présence de bibliothèques de bas niveau, la cohérence est améliorée entre les analyses de sécurité d'un même système et entre les analyses de sécurité de différents systèmes. En effet actuellement, il est courant de trouver un même mode de défaillance nommé de façon différente entre deux analyses de sécurité d'un même système. Par exemple si on analyse deux systèmes différents A et B, la perte de la fonction d'une vis pourra être intitulée « rupture » sur le système A et « cassure » sur le système B (cela n'est qu'un exemple !). En utilisant un unique nœud de la bibliothèque pour la modélisation de la vis au sein des systèmes A et B, le même terme sera utilisé dans les deux analyses.

La cohérence des analyses est également améliorée au niveau de la description des effets des modes de défaillance des composants. En effet et comme présenté au chapitre 3, la méthodologie de modélisation propose la définition de modes de défaillance génériques. Un analyste, lorsqu'il sera confronté à la représentation d'un mode de défaillance dans un nouveau modèle, pourra aller puiser dans les documentations existantes des modèles AltaRica pour coder son modèle AltaRica. Grâce à une cohérence et une telle traçabilité, la prise en charge d'un modèle par un analyste ne l'ayant pas réalisé sera aussi largement améliorée. Celui-ci, lors de la lecture du modèle, retrouvera en effet des termes connus et une logique déjà validée indépendamment du nœud.

5.4 Quelques bonnes pratiques recommandées lors de la modélisation

Cette section a pour vocation d'exhiber certaines bonnes pratiques qui faciliteront selon nous certains aspects de la modélisation. Si quelques unes de ses bonnes pratiques ont déjà été évoquées au chapitre 3, cette section les regroupent, les développent et les complètent.

5.4.1 Analyse préliminaire - Identification des besoins

Comme nous avons vu dans le chapitre 2, un modèle est réalisé avec l'objectif de répondre à une ou plusieurs questions que l'on se pose sur un système. Il n'a donc pas pour vocation de reprendre l'ensemble des caractéristiques du système qu'il représente [40]. L'analyse en charge de la réalisation d'un modèle cherchera ainsi à construire un modèle pertinent vis-à-vis de la question à laquelle il doit répondre... L'analyse en charge de la réalisation d'un modèle cherchera ainsi à construire un modèle mettant en évidence les caractéristiques du système qui l'intéressent.

Exemple : Il n'est pas nécessaire de modéliser le déplacement des électrons dans le silicium à l'aide de la théorie des gaz de Fermi pour faire comprendre le fonctionnement de son système informatique à un utilisateur. [40]

Il est donc indispensable pour l'analyste d'identifier les objectifs de la modélisation. Identifier *l'ensemble* des objectifs de la modélisation permettra leurs prises en compte dès le commencement

et facilitera l'obtention d'un modèle générique répondant à la question posée. Plus le besoin est identifié en amont de l'implémentation, plus il est facile à prendre en compte. Il sera ainsi plus couteux de répondre à une question posée tardivement qu'à une question émise au début de la modélisation.

5.4.2 Analyse préliminaire - Décomposition d'un système

Pour un système donné, les possibilités de décomposition sont multiples : décomposition fonctionnelle, décomposition organique, décomposition hiérarchique... La façon de décomposer un système n'est donc pas une caractéristique intrinsèque de celui ci. L'analyste en charge de cette décomposition se trouve confronté à de nombreux choix. Comment regrouper ? Comment découper ? Comment choisir le périmètre des sous-systèmes ? ...

Pour aider l'analyste, des analyses fonctionnelles et / ou organiques peuvent être utiles mais doivent être guidées par des critères. Nous en recommandons deux.

- Le premier critère est de choisir une décomposition fonctionnelle ou organique qui limite le nombre et la complexité des interactions entre les sous-systèmes.
- Le second critère incite à choisir une décomposition faisant apparaître des sous-systèmes physiquement homogènes (*i.e.* ne faisant apparaitre autant que possible qu'un unique métier en leur sein).

Le premier critère facilitera l'assemblage des sous-systèmes entre eux. En limitant dès l'analyse préliminaire le nombre et la complexité des interfaces entre les modèles des sous-systèmes, leur assemblage sera facilité à la fin de modélisation. Il faudra cependant penser à inclure dans les interfaces les informations nécessaires à la propagation des comportements d'un sous-système à un autre.

Le second critère permettra de propager à l'intérieur d'un sous-système des défaillances de même nature physique. C'est sur ce second critère que s'appuie très fortement la méthodologie de modélisation présentée au chapitre 3. Des modes de défaillance génériques ont ainsi été identifiés et définis pour chaque domaine physique étudié ce qui facilite leurs implémentations dans le modèle.

Exemple : La décomposition de notre cas d'étude présenté figure 3.4 reprends cette philosophie d'identification de sous-systèmes physiquement homogènes limitant le nombre d'interfaces entre sous-systèmes.

5.4.3 Mise en place d'une organisation

Ce besoin peut être vu comme similaire à celui ayant sans doute conduit à la notion de « gestion de configuration ». De manière similaire au domaine logiciel, il est entre autres choses nécessaire de gérer :

- les différentes versions des bibliothèques et des nœuds des bibliothèques ;
- les différentes versions des modèles de systèmes réalisés ;
- les différentes versions des documentations réalisées et le nœud auquel chacune se rattache.

En pratique, les versions de modèles et de bibliothèque s'incrémentant rapidement, il sera nécessaire de bien tracer les modifications d'un modèle à un autre. En cas de besoin, il sera aisé de retrouver une version antérieure et d'identifier les différences entre deux versions. Ainsi, cette

mise en place de traçabilité de l'historique d'un modèle ou d'un document permettra à un ou plusieurs analystes de pouvoir retrouver de manière plus aisée l'information pertinente et utile à la compréhension du fonctionnement du modèle.

Remarque : Notons ici qu'en 2010 et à notre connaissance, une telle gestion des modèles n'est pas encore supportée par un outil. Il serait pertinent dans le futur d'imaginer et de développer un outil permettant de lier un modèle AltaRica et l'ensemble des informations qui lui sont liées : ses nœuds, ses hypothèses de modélisation, sa documentation, ses analyses, ses résultats...

5.4.4 Codification des informations

Toujours dans une optique de traçabilité et de cohérence, une bonne pratique de modélisation consiste à standardiser les termes utilisés dans les bibliothèques et dans les modèles. Pour cela et selon nous, un moyen adéquat est de doter d'une codification des informations implémentées dans le modèle. En pratique, cela passe par la mise en place de divers moyens parmi lesquels et par exemple :

- l'utilisation d'une codification des nœuds présents dans les modèles : par exemple et pour éviter qu'une pompe soit nommée « pompe »ou « pump », on recommandera l'utilisation du code « PMP » ;
- l'utilisation de préfixes pour la déclaration des variables de flux et d'état : le préfixe « IN_ » désignera une variable de flux d'entrée, le préfixe « OUT_ » désignera une variable de flux de sortie, le préfixe « LOC_ » désignera une variable de flux local, le préfixe « ST_ » désignera une variable d'état ;
- l'utilisation de termes spécifiques pour désigner les flux des nœuds, *e.g.* on utilisera « SPD » pour désigner la vitesse ;
- l'utilisation d'une codification des évènements implémentés dans les modèles, *e.g.* on utilisera le terme « DRD » pour désigner une dérive d'un équipement (DRD pour DRift Down).

La liste ci-dessus n'est pas exhaustive. Le principe est de standardiser au maximum les termes utilisés lors de la modélisation en se fondant sur un référentiel du domaine lorsqu'il existe ou à défaut sur les choix de l'entreprise.. Plutôt que les termes précis, ce qui est donc à retenir serait plutôt la démarche et son objet. Sans vouloir ôter toute liberté à l'analyste en charge de la modélisation, l'objectif est de renforcer et de faciliter encore la cohérence entre les différentes analyses de sécurité réalisée. Ainsi, il sera aisé d'appréhender un modèle une fois familiarisé avec une codification commune à l'ensemble des modèles.

5.5 Retours sur quelques difficultés rencontrées

Bien que l'expressivité du langage AltaRica se soit révélée suffisante pour modéliser notre cas d'étude, nous nous sommes heurtés au cours de ces travaux à quelques difficultés qu'il nous a fallu gérer. Nous présentons ici ces difficultés ainsi que des pistes pour les surmonter.

5.5.1 Traitement des boucles instantanées de modélisation

La modélisation d'un système tel que celui présenté sur la figure 5.2 peut poser des problèmes d'indétermination si le comportement de C_1 dépends au même instant logique t de celui de C_2.On parle de boucles instantanées de modélisation (section 2.4.2). Les outils d'analyse vont

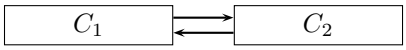


Figure 5.2 – Exemple de système bouclé

alors potentiellement avoir des difficultés pour évaluer l'état et les sorties d'un tel système : il faut éviter ce type de cas.

Le principe est en fait d'éviter qu'à un moment donnée t une variable soit définie par elle même. Pour cela, il est possible de « casser » les boucles instantanées présentes dans le modèle en introduisant par exemple des délais dans la propagation de certaines données du modèle. Notons qu'il existe à l'intérieur de Cecilia™ OCAS un outil permettant la détection des boucles instantanées[2] de modélisation.

5.5.2 Multiplicité des données transmises

Une autre difficulté liée au grand nombre de données à propager dans le modèle se situe au niveau de la définition des assertions des nœuds AltaRica, *i.e.* à la définition des variables de sorties des nœuds. En effet, si un nœud C a n entrées booléennes, la table de vérité à construire pour définir chaque variable de sortie de C comporte 2^n cases à remplir. Ce nombre augmente avec l'augmentation de la granularité des données transmises. En prenant l'exemple d'un engrenage du système de transmission mécanique dans notre cas d'étude (section 3.4), celui-ci à 8 entrées et 5 sorties. Parmi les 8 entrées, 7 ont une granularité de 3 et une est booléenne. Ainsi pour chaque variable de sortie, la table de vérité contient $3^7 \times 2 = 4374$! Même si des cas se regroupent et qu'ainsi, plusieurs cas se remplissent en une unique équation, le travail à réaliser est potentiellement conséquent.

5.5.3 Difficultés d'abstraction des données

Une première difficulté provient de la notion même de modèle. La plupart des systèmes auxquels nous nous sommes intéressés sont des systèmes continus. Le langage AltaRica étant un langage discret, le comportement des systèmes doit être abstrait et discrétisé en un nombre fini de domaines (nominal, dégradé, perdu, erroné...). Pour de tels systèmes, un écart par rapport au comportement réel est donc introduit à la création du modèle.

Ensuite et pour prendre en compte les différents domaines à l'intérieur desquels le système évolue, le chapitre 3 (section 3.4 à 3.7) a montré que le nombre de données à considérer est grand. Une seconde difficulté se trouve au niveau de l'identification de ces données. Même si l'idée est sans aucun doute séduisante, il n'existe pas selon nous de moyen miracle permettant l'identification systématique des données nous permettant de décrire la propagation d'une défaillance d'un domaine physique à un autre.

Toujours au sujet de l'identification de ces données, nous avons vu, au chapitre 3, que cette identification est fondée sur la représentation et sur l'abstraction de phénomènes physiques. En plus de faciliter la généricité dans la modélisation des modes de défaillance, cette philosophie permet de faciliter les revues du modèle par les spécialistes métiers en leurs permettant de retrouver les relations physiques avec lesquelles ils sont familiers. En contrepartie, cette philosophie conduit à une difficulté lors de l'abstraction si l'analyste en charge de la création du modèle n'a pas une bonne connaissance de ces relations physiques. Pour faire face à ce « problème », le chapitre 3 adresse cependant un nombre de questions conséquent permettant de guider l'identification de données devant en théorie permettre la propagation des évènements identifiés. Par exemple, il faudra s'intéresser aux évènements redoutés à observer, aux conditions d'utilisation du système, au niveau de modélisation souhaité...

2. Une boucle instantanée est une boucle dans la définition des variables du modèle où au même instant logique, une variable est définie plus d'une fois.

Conclusion

Synthèse des travaux

Le domaine des travaux explicités dans ce mémoire est celui de l'évaluation de la sécurité d'un système complexe aéronautique. La problématique abordée s'intéresse à l'utilisation d'une modélisation formelle des propagations de défaillance dans un système multi-physique pour en évaluer sa sécurité. Le changement culturel induit par une telle approche est important : représenter les chemins de propagation de défaillance non plus de manière explicite à l'aide d'arbres de défaillance mais de manière implicite grâce à une approche compositionnelle (modulaire) où les chemins globaux émergent de l'interconnexion entre différentes entités élémentaires. Le changement culturel est important et l'investissement lourd en temps et en ressources : il est par conséquent hors de propos, en particulier dans un contexte industriel, de changer de façon de travailler (de raisonner) sans une réflexion préalable sur les modalités à mettre en oeuvre pour gérer un tel changement.

Les travaux décrits ici se veulent être une partie des réflexions déjà existantes concernant l'introduction de modèles formels de propagation de défaillance dans le processus d'évaluation de la sécurité d'un système. Un notion fondamentale est alors celle de la confiance à accorder dans les modèles construits. Usuellement, cette confiance se gagne en se dotant d'un processus rigoureux de construction du modèle et/ou en pratiquant des activités de revues et de test sur les modèles réalisés pour s'assurer qu'ils sont une abstraction valide des systèmes qu'ils représentent. Les travaux décrits dans ce mémoire adressent les deux problématiques.

Synthèse sur l'activité de réalisation de modèles

Concernant la construction d'un modèle formel des propagations de défaillance au sein d'un système, le langage AltaRica a été utilisé et la méthodologie proposée dans [34] étendue pour s'intéresser à la construction d'un modèle AltaRica multi-systèmes et multi-physiques (système contenant différents sous-systèmes de natures physiques hétérogènes). Les principales étapes proposées dans cette méthodologie sont ainsi décrites ci-dessous.

- La décomposition du système global en différents sous-systèmes physiquement homogènes. Ainsi, les hétérogénéités présentent dans le système sont limitées dans les sous-systèmes et gérées au niveau des interfaces entre ces sous-systèmes : on parlera de sous-systèmes fortement cohérents (nombreuses interactions entre les composants d'un sous-système) et faiblement couplés (peu d'interactions entre les composants de deux sous-systèmes différents).
- L'abstraction du comportement fonctionnel et dysfonctionnel de chaque composant de chaque sous-système. De nouvelles abstractions sont définies pour la construction de modèles de systèmes de transmission mécanique et de systèmes hydromécaniques.
- De manière préliminaire à l'implémentation en langage AltaRica de cette abstraction, la construction d'une représentation préliminaire du comportement de chaque composant.

Cette représentation est nommée « spécification » et est un support facilitant la compréhension des informations qui seront codées. Cette spécification pourra également être un efficace support de discussion lors de revues du modèle par quiconque ne connaît pas le langage AltaRica.

Concernant l'application pratique de la méthodologie proposée, des bibliothèques ont été construites en langage AltaRica et permettent la modélisation des propagations de défaillance au sein de et entre les différents sous-systèmes d'un turbomoteur d'hélicoptère. Cinq bibliothèques de noeuds AltaRica ont été construites et permettent de modéliser la majeure partie d'un turbomoteur d'hélicoptère (circuit carburant, circuit de transmission mécanique, circuit de lubrification, circuit d'air principal et système de surveillance moteur). Trois évènements redoutés (sur les seize que comporte une analyse actuelle de sécurité) ont été évalués sur ces modèles AltaRica et les résultats obtenus comparés avec ceux des analyses de sécurité actuelles (de type arbre de défaillance). À périmètre modèle - arbres équivalent, les résultats qualitatifs (coupes minimales) et quantitatifs (probabilités d'occurrence d'un évènement redouté) ont été retrouvés.

Synthèse sur l'activité de validation de modèles

Si la comparaison entre les résultats fournis par le modèle AltaRica et ceux issus des analyses de sécurité actuelle est sans aucun doute pertinente pour s'assurer qu'il est possible de retrouver les résultats des arbres de défaillance grâce aux modèles AltaRica, elle ne suffit pas pour s'assurer que l'implémentation du modèle soit valide. Il faut en effet s'assurer que le comportement interne du modèle décrive correctement le système que le modèle est censé décrire. Dans cette optique, nous avons proposé un processus de validation de modèles AltaRica fondé sur la simulation d'un jeu de tests et sur l'observation des parties du modèle éprouvées par ce jeu de tests. Des critères de couverture, inspirés du génie logiciel, ont été définis et une implémentation de ces critères est proposé de façon à observer leur satisfaction directement sur le modèle AltaRica considéré.

En pratique, l'accent est mis sur la validation d'un composant AltaRica unitaire. La philosophie est de modifier le code AltaRica du composant à valider en y insérant des variables d'état booléennes supplémentaires appelées « drapeaux » et représentant pour chacune d'entre elle une portion du code AltaRica implémentant le comportement du composant. Ces variables d'état sont initialisées à la valeur fausse et deviendront vraies au fur à mesure que les scénarios de test simulés éprouveront des portions du modèle. En visionnant, à la fin de la simulation, la valeur des drapeaux, nous serons en mesure de connaître les parties éprouvées du modèle et celles ne l'étant pas.

Discussion / perspectives

La démarche décrite dans ce mémoire permet ainsi à la fois de couvrir la mise en place d'un processus rigoureux de modélisation en langage AltaRica et d'assister les activités de test sur le modèle pour évaluer l'efficacité de celles-ci. Les perspectives de ces travaux sont nombreuses. Certaines sont décrites ci-dessous.

1. La première perspective et selon nous une des plus importantes concerne l'outillage du processus de validation proposé au chapitre 4. En effet et aujourd'hui, le processus décrit est réalisé de manière manuelle. Une utilisation de l'approche à l'échelle industrielle est ainsi à l'heure actuelle hors de propos. Différents outils supportant l'approche peuvent et selon nous doivent être mis en place.

- Un outil permettant la prise en compte automatique des différents critères de couverture définis. Grâce à un tel outil, plutôt que d'implémenter de manière manuelle les différents drapeaux proposés en section 4.6, ceux-ci seraient implémentés automatiquement dans le modèle AltaRica. Une modification du critère sélectionné entraînerait alors une modification des drapeaux implémentés.
- Un outil permettant de simuler de manière automatique les différents scénarios de test issus d'une spécification de référence. Si Cecilia^{TM} OCAS possède déjà un outil capable de simuler un ensemble de séquences sur le modèle, un tel outil devrait dans l'idéal être capable de comparer le résultat attendu et celui obtenu, identifier les différents drapeaux activés durant la simulation ou indiquer à l'utilisateur si un scénario ne devant pas être exécutable ne l'est effectivement pas.
- la liste n'est sans doute pas exhaustive...

2. En se situant à bas niveau et en s'intéressant à l'abstraction des phénomènes physiques qu'engendrent les défaillances, la création des bibliothèques AltaRica fournit une importante base de données capitalisant les connaissances fonctionnelles et dysfonctionnelles sur les composants des systèmes. Plus qu'un simple support pour les analyses de sécurité, le modèle AltaRica peut alors être envisagé comme un support pour quiconque veut aborder de manière graphique et interactive les aspects dysfonctionnels d'un système. Dans ce sens, [51] utilise un modèle AltaRica dans l'intention d'entraîner un pilote au fonctionnement du système carburant d'un avion. Le pilote peut alors simuler des actions pouvant être exécutées en vol et visionner les conséquences de celles-ci sur le modèle.

3. Une autre perspective liée à l'utilisation d'une modélisation à bas niveau se situe dans la possibilité d'envisager de faire de l'aide à la détection de panne. Imaginons un modèle AltaRica bas niveau comportant différents voyants présents sur un cockpit d'hélicoptère. Grâce à un tel modèle, il est possible d'imaginer une génération des scénarios conduisant à l'allumage de plusieurs de ces voyants. Les scénarios obtenus pourraient alors potentiellement être un départ intéressant pour identifier la cause de ces allumages...

4. Une autre problématique qui devra sans aucun doute être traitée dans de futurs travaux est celle de l'intégration entre elles de différentes analyses de sécurité dont certaines d'entre-elles seraient potentiellement formalisées sous forme de modèles AltaRica. Comment intégrer l'évaluation de la sécurité du moteur dans celle de l'aéronef ? Comment intégrer les analyses de sécurité des équipements du moteur (réalisés par des fournisseurs extérieur) dans celle du moteur ?

   Différents cas de figure peuvent être observés si l'on s'intéresse à l'intégration de l'analyse de sécurité d'un système A au sein de l'analyse de sécurité d'un système B (on suppose connu ici l'intégration d'un arbre de défaillance au sein d'un autre arbre de défaillance).

   - Analyse de sécurité de A sous forme de modèle AltaRica / Analyse de sécurité de B sous forme de modèle AltaRica : il faut s'assurer que les deux modèles puissent être connectés entre eux (que les interfaces de A et de B permettent cette connexion).
   - Analyse de sécurité de A sous forme de modèle AltaRica / Analyse de sécurité de B sous forme d'arbre de défaillance : ici, une piste à explorer serait de générer des arbres de défaillance à partir du modèle de A et d'intégrer ces arbres dans ceux de B. Si la génération d'arbre de défaillance est impossible à partir du modèle AltaRica de A (modèle bouclé), une autre piste à explorer serait d'évaluer l'occurrence d'évènement de A puis d'intégrer ces évènements dans les arbres de B sous forme d'évènement terminaux. Les hypothèses à prendre en compte seraient à étudier.
   - Analyse de sécurité de A sous forme d'arbre de défaillance / Analyse de sécurité de B sous forme de modèle AltaRica : perspectives à étudier...

Index

Bibliographie

[1] Parosh Aziz Abdulla et Johann Deneux : Designing safe, reliable, systems using Scade. *In Proc. ISoLA 2004*, 2004.

[2] Romain Adeline, Janette Cardoso, Pierre Darfeuil et Christel Seguin : Toward a methodology for the AltaRica modelling of multi-physical systems. *In Proceedings of the European Safety & Reliability Conference 2010 (ESREL 2010)*, Rhodes, Grèce, 2010.

[3] Romain Adeline, Janette Cardoso, Pierre Darfeuil et Christel Seguin : Toward a validation process for model based safety analysis. *In Proceedings - Embedded Real Time Software and Systems 2010 (ERTSS 2010)*, Toulouse, France, 2010.

[4] Romain Adeline, Janette Cardoso, Pierre Darfeuil et Christel Seguin : Vers une méthodologie de modélisation AltaRica pour des systèmes physiques. *In Actes du congrès Lambda Mu 17*, La Rochelle, France, 2010.

[5] AFNOR : NF X 50-151 : Management de la Valeur - Expression fonctionnelle du besoin et cahier des charges fonctionnel.

[6] AFNOR : NF X 60-319 : Terminologie de la Maintenance.

[7] Paul Allard : Éléments pour une problématique de l'histoire du risque. Du risque accepté au risque maîtrisé. Représentations et gestion du risque d'inondation en Camargue. XVIIIe-XIXe siècles. 2005. http ://ruralia.revues.org/document152.html.

[8] Boris Beizer : *Software testing techniques*. Van Nostrand Reinhold Co., New York, USA, 1990.

[9] Romain Bernard : *Analyses de sûreté de fonctionnement multi-systèmes*. Thèse de doctorat, Université Bordeaux 1, 2009.

[10] Ludwig Von Bertalanffy : *Théorie générale des systèmes*. Dunod, 1993.

[11] Ramesh Bharadwaj et Constance Heitmeyer : Hardware/software co-design and co-validation using the SCR method. *In Proceedings of the IEEE International High Level Design Validation and Test Workshop (HLDVT)*, pages 6–37, 1999.

[12] Pierre Bieber, Christian Bougnol, Charles Castel, Jean-Pierre Heckmann, Christophe Kehren, Sylvain Metge et Christel Seguin : Safety assessment with AltaRica - lessons learnt based on two aircraft system studies. *In 18th IFIP World Computer Congress, Topical Day on New Methods for Avionics Certification*, page 26, 2004.

[13] Pierre Bieber, Charles Castel et Christel Seguin : Combination of fault tree analysis and model checking for safety assessment of complex system. *In Proc. 4th European Dependable Computing Conference, volume 2485 of LNCS*. Springer-Verlag, 2002.

[14] Matthias Biehl, Chen DeJiu et Martin Törngren : Integrating safety analysis into the model-based development toolchain of automotive embedded systems. *In SIGPLAN Not.*, volume 45, pages 125–132, New York, NY, USA, 2010. ACM.

[15] Mark R. Blackburn, Robert D. Busser et Joseph S. Fontaine : Automatic generation of test vectors for SCR-style specifications. *In Compass'97 : Twelfth Annual Conference on Computer Assurance*, pages 54–67, Gaithersburg, Maryland, 1997. National Institute of Standards and Technology.

[16] Marc Bouissou, Henri Bouhadana, Marc Bannelier et Nathalie Villatte : Knowledge modeling and reliability processing : Presentation of the Figaro language and associated tools. *In Proceeding of Safecomp'91*, 1991.

[17] Marc Bouissou et Christel Seguin : Comparaison des langages de modélisation AltaRica et Figaro. *In Actes du congrès Lambda Mu 15*, 2006.

[18] M. Bozzano, A. Villafiorita, O. Åkerlund, P. Bieber, C. Bougnol, E. Böde, M. Bretschneider, A. Cavallo, C. Castel, M. Cifaldi, A. Cimatti, A. Griffault, C. Kehren, B. Lawrence, A. Lüdtke, S. Metge, C. Papadopoulos, R. Passarello, T. Peikenkamp, P. Persson, C. Seguin, L. Trotta, L. Valacca et G. Zacco : ESACS : an integrated methodology for design and safety analysis of complex systems. *In Proc. ESREL'2003*. Balkema publisher, 2003.

[19] Marco Bozzano et Adolfo Villafiorita : Improving system reliability via model checking : The FSAP/NuSMV-SA safety analysis platform, 2003.

[20] Charles Castel et Christel Seguin : Modèles formels pour l'évaluation de la sûreté de fonctionnement des architectures logicielles d'avionique modulaire intégrée. *In Actes du Congrès AFADL'01*, 2001.

[21] Dejiu Chen, Rolf Johansson, Henrik Lönn, Yiannis Papadopoulos, Anders Sandberg, Fredrik Törner et Martin Törngren : Modelling support for design of safety-critical automotive embedded systems. *In SAFECOMP '08 : Proceedings of the 27th international conference on Computer Safety, Reliability, and Security*, pages 72–85, Berlin, Heidelberg, 2008. Springer-Verlag.

[22] A. Cimatti, E. Clarke, F. Giunchiglia et M. Roveri : NUSMV : a new symbolic model checker. *In International Journal on Software Tools for Technology Transfer*, volume 2, page 2000, 2000.

[23] Pierre David : *Contribution à l'analyse de sûreté de fonctionnement des systèmes complexes en phase de conception : application à l'évaluation des missions d'un réseau de capteurs de présence humaine*. Thèse de doctorat, Université Orléans, 2009.

[24] Serge Diebolt : Le petit lexique des termes de la complexité, 1998. www.mcxapc.org.

[25] Gérard Donnadieu, Daniel Durand, Danièle Neel, Emmanuel Nunez et Lionel Saint-Paul : L'approche systémique : De quoi s'agit-il ? Rapport technique, Groupe de travail AFSET - Diffusion de la pensée systémique, 2003.

[26] Xavier Dumas, Claire Pagetti, Laurent Sagaspe, Pierre Bieber et Philippe Dhaussy : Vers la génération de modèles de sûreté de fonctionnement. *In CAL*, volume RNTI-L-2 de *Revue des Nouvelles Technologies de l'Information*, pages 157–172. Yamine Aït Ameur, 2008.

[27] European Aviation Safety Agency : CS-E : Certification Specifications for Engines, 2007.

[28] European Aviation Safety Agency : CS-25 : Certification Specifications for Large Aeroplanes, 2010.

[29] Marie Claude GAUDEL : Advantages and limits of formal approaches for ultra-high dependability. *In IWSSD '91 : Proceedings of the 6th international workshop on Software specification and design*, pages 237–241, Los Alamitos, CA, USA, 1991. IEEE Computer Society Press.

[30] Marie-Claude GAUDEL, Bruno MARRE, Gilles BERNOT et Françoise SCHLIENGER : *Précis de génie logiciel*. 1996.

[31] N. HALBWACHS, P. CASPI, P. RAYMOND et D. PILAUD : The synchronous dataflow programming language Lustre. *In Proceedings of the IEEE*, pages 1305–1320, 1991.

[32] Constance HEITMEYER, James KIRBY, Bruce LABAW et Ramesh BHARADWAJ : SCR* : A Toolset for Specifying and Analyzing Software Requirements. *In Lecture notes in computer science*, pages 526–531. Springer-Verlag, 1998.

[33] Sophie HUMBERT : *Déclinaison d'exigences de sécurité, du niveau système jusqu'au niveau logiciel, assistée par des modèles formels*. Thèse de doctorat, Université Bordeaux 1, 2008.

[34] Sophie HUMBERT, Jean-Marc BOSC, Charles CASTEL, Pierre DARFEUIL, Yves DUTUIT et Christel SEGUIN : Méthodologie de modélisation AltaRica pour la sûreté de fonctionnement d'un système de propulsion hélicoptère incluant une partie logicielle. *In Actes du congrès Lambda Mu 15*, 2006.

[35] Guillaume HUTZLER : *Du Jardin des Hasards aux Jardins de Données : une approche artistique et multi-agent des interfaces homme / systèmes complexes*. Thèse de doctorat, Université Pierre et Maris Curie - Paris 6, 2000.

[36] Anjali JOSHI et Mike WHALEN : Model Based Safety Analysis : Final Report, NASA. Rapport technique, 2005.

[37] Christophe KEHREN, Christel SEGUIN, Pierre BIEBER et Charles CASTEL : Analyse des exigences de sûreté d'un système électrique par model-checking. *In Actes du congrès Lambda Mu 14*, Bourges, France, 2004.

[38] Christophe KEHREN, Christel SEGUIN, Pierre BIEBER, Charles CASTEL, Christian BOUGNOL, Jean pierre HECKMANN et Sylvain METGE : Advanced multi-system simulation capabilities with AltaRica. *In Proceedings of Safety Critial System Conference*, 2004.

[39] Jean-Claude LUGAN : *Lexique de systémique et de prospective*. Conseil économique et social Midi-Pyrénées - Section Prospective, 2006.

[40] Jean-Pierre MEINADIER : *Ingénierie et intégration des systèmes*. Hermes, 1998.

[41] Sylvain METGE : Etes-vous sûr de la sûreté de fonctionnement de votre système ? pas si sûr... ! CENA, NT96-623, 1996.

[42] Yves MORTUREUX : la sûreté de fonctionnement : Démarche pour maitriser les risques. *In Techniques de l'ingénieur*, volume AG 4670.

[43] Glenford J. MYERS et Corey SANDLER : *The Art of Software Testing*. John Wiley & Sons, 2004.

[44] NF X 60-500 : Terminologie relative à la fiabilité - maintenabilité - disponibilité. AFNOR.

[45] Jean NIZET et François PICHAULT : *Introduction à la théorie des configurations. Du "one best way" à la diversité organisationnelle*. De Boeck Université, 2001.

[46] Jeff OFFUTT et Aynur ABDURAZIK : Generating Tests from UML Specifications. *In UML'99 - The Unified Modeling Language. Beyond the Standard. Second International Conference*, volume 1723, pages 416–429. France, Robert and Rumpe, Bernhard, 1999.

[47] Jeff OFFUTT, Shaoying LIU, Aynur ABDURAZIK et Paul AMMANN : Generating test data from state-based specifications. *In The Journal of Software Testing, Verification and Reliability*, volume 13, pages 25–53, 2003.

[48] Bruno PAGANO, Benjamin CANOU, Emmanuel CHAILLOUX, Jean-Louis COLAÇO et Philippe WANG : Couverture de code Caml pour la réalisation d'outils de développement certifiés. *In Proceedings Journées Francophones des Langages Applicatifs (JFLA)*, 2007.

[49] Yannis PAPADOPOULOS, John MCDERMID, Ralph SASSE et Guenter HEINER : Analysis and synthesis of the behaviour of complex programmable electronic systems in conditions of failure. *In Reliability Engineering & System Safety*, volume 71, pages 229 – 247, 2001.

[50] Gilbert PAQUETTE : *Modélisation des connaissances et des compétences. Un langage graphique pour concevoir et apprendre*. Presse Universitaire du Québec, 2002.

[51] Benoit PERROT, Tatiana PROSVIRNOVA, Antoine RAUZY et Jean-Philippe Sahut D'IZARN : Expérience de couplage de modèles AltaRica avec des interfaces métiers. *In Actes du congrès Lambda Mu 17*, la Rochelle, France, 2010.

[52] Fabien PEUREUX : *Génération de tests aux limites à partir de spécifications B en programmation logique avec contraintes ensemblistes*. Thèse de doctorat, Université Franche Comté, 2002.

[53] PROJET MISSA : More Integrated System Safety Assessment. http ://www.missa-fp7.eu/.

[54] Antoine RAUZY : Mathematical foundation of minimal cutsets language. *In Reliability, IEEE Transactions on*, volume 50, pages 389–396, 2001.

[55] Antoine RAUZY : Mode automata and their compilation into fault trees. *In Reliability Engineering and System Safety*, volume 78, pages 1–12, 2002.

[56] Antoine RAUZY et Yves DUTUIT : Exact and truncated computations of prime implicants of coherent and non-coherent fault trees within Aralia. *In Reliability Engineering & System Safety*, volume 58, pages 127 – 144, 1997. ESREL'95.

[57] Ana-Elena RUGINA, Karama KANOUN et Mohamed KAÂNICHE : Modélisation de la sûreté de fonctionnement de systèmes à partir du langage AADL. *In Actes du congrès Lambda Mu 15*, 2006.

[58] SAE : The SAE Architecture Analysis & Design Language (AADL) Standard. http ://www.AADL.info/AADLinfosite/LinkedDocuments/AADLLanguageSummary.pdf.

[59] SAE : ARP 4754 : Certification Considerations for Highly-Integrated Or Complex Aircraft Systems, 1996.

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

[61] Jean-Pierre SIGNORET : Analyse des risques des systèmes dynamiques : approche markovienne. *In Techniques de l'ingénieur*, volume SE 4071.

[62] Jean-Pierre SIGNORET : Analyse des risques des systèmes dynamiques : Réseaux de Petri. *In Techniques de l'ingénieur*, volume SE 4072.

[63] Dr. Michael STAMATELATOS et Nasa HEADQUARTERS : Fault tree handbook with aerospace applications version 1.1, 2002.

[64] Robert VALETTE : Les Réseaux de Petri, 2002. http ://www.laas.fr/ robert/enseignement.d/cour01.pdf.

[65] Alain VILLEMEUR : *Sûreté de fonctionnement des systèmes industriels*. Eyrolles, Paris, 1988.

[66] Aymeric VINCENT : *Conception et réalisation d'un vérificateur de modèles AltaRica*. Thèse de doctorat, Université Bordeaux 1, 2003.

Résumé

Pour certifier un système aéronautique, des études de Sûreté de Fonctionnement (SdF) visent à démontrer qu'il répond à des exigences de sécurité de haut niveau. La complexité des systèmes étudiés ayant évolué, les exigences à démontrer devenant toujours plus nombreuses, les analyses actuelles (e.g. arbre de défaillance) peuvent aujourd'hui présenter des limites d'utilisation.

Pour aller à l'encontre de ces limites, l'Ingénierie Dirigée par les Modèles s'est développée et s'intéresse aux études de SdF. L'objectif est alors de 1) modéliser dans un langage adapté (le langage AltaRica a ici été utilisé) les comportements fonctionnels et dysfonctionnels d'un système et de ses composants en présence de défaillances, 2) s'assurer que le modèle est une abstraction valide du système réel et 3) vérifier la tenue des exigences du système par le modèle.

Les travaux effectués dans cette thèse se sont intéressés aux deux premiers points. Une méthodologie a été proposée pour spécifier l'abstraction du comportement de composants de systèmes multi physiques. Des bibliothèques AltaRica ont été réalisées pour modéliser des sous-systèmes d'un turbomoteur d'hélicoptère. Les résultats obtenus via le modèle ont été comparés avec ceux des analyses existantes de SdF. Pour les projets futurs où celles-ci ne seraient plus disponibles, un processus de validation a été proposé pour caractériser le degré de revue atteint lors de la simulation d'un jeu de tests sur le modèle. Inspiré du « génie logiciel », ce processus s'étend sur trois niveaux de validation (unitaire ; intégration des composants ; modèle complet) et propose des critères de couvertures applicables et mesurables sur un modèle AltaRica.

Abstract

To certify an aeronautical system, safety studies aim to demonstrate several requirements on this system. Because of the increasing complexity of systems and of the growing number of requirements, classical studies (such as Fault Tree Analysis) have shown limits.

To overcome these limits, Model-Driven Engineering, initially developed for software systems, deals nowadays with the achievement of safety analyses. The development process of a model can be divided into three main parts : 1) Modelling activities : system components and their behaviours are described in an adapted formal language (in the thesis, AltaRica has been used) ; 2) Validation of the model : ensuring the model is a valid abstraction of the real system ; 3) Verification of the model : checking if the system requirements are satisfied by the model.

The two first points have been addressed during this thesis. A methodology (for different kinds of physical systems) has been proposed to specify AltaRica libraries. Some sub-systems of a turboshaft engine have been modelled. The results given by the models were compared with the results extracted from classical safety analyses. For future projects, where these analyses won't be available anymore, a validation process aims at giving a coverage measure of a test set simulated on the model. Inspired from several works about software testing, this validation process is divided in three levels (unit testing to validate the AltaRica library, integration testing to validate the I/O of components, system testing to validate the global model) and gives some coverage criteria which can be used and measured on the AltaRica model.

Méthodes pour la validation de modèles formels pour la sûreté de fonctionnement et extension aux problèmes multi-physiques

Pour certifier un système aéronautique, des études de Sûreté de Fonctionnement (SdF) visent à démontrer qu'il répond à des exigences de sécurité de haut niveau. La complexité des systèmes étudiés ayant évolué, les exigences à démontrer devenant toujours plus nombreuses, les analyses actuelles (e.g. arbre de défaillance) peuvent aujourd'hui présenter des limites d'utilisation.
Pour aller à l'encontre de ces limites, l'Ingénierie Dirigée par les Modèles s'est développée et s'intéresse aux études de SdF. L'objectif est alors de 1) modéliser dans un langage adapté (le langage AltaRica a ici été utilisé) les comportements fonctionnels et dysfonctionnels d'un système et de ses composants en présence de défaillances, 2) s'assurer que le modèle est une abstraction valide du système réel et 3) vérifier la tenue des exigences du système par le modèle.
Les travaux effectués dans cette thèse se sont intéressés aux deux premiers points. Une méthodologie a été proposée pour spécifier l'abstraction du comportement de composants de systèmes multi physiques. Des bibliothèques AltaRica ont été réalisées pour modéliser des sous-systèmes d'un turbomoteur d'hélicoptère. Les résultats obtenus via le modèle ont été comparés avec ceux des analyses existantes de SdF. Pour les projets futurs où celles-ci ne seraient plus disponibles, un processus de validation a été proposé pour caractériser le degré de revue atteint lors de la simulation d'un jeu de tests sur le modèle. Inspiré du « génie logiciel », ce processus s'étend sur trois niveaux de validation (unitaire ; intégration des composants ; modèle complet) et propose des critères de couvertures applicables et mesurables sur un modèle AltaRica.

Mots clés : Sûreté de fonctionnement – Modèles formels – Langage AltaRica – Modélisation – Validation – Couverture de modèles AltaRica – Critère de couverture de modèle AltaRica – Modélisation multi-physique

Methods for the validation of formal models for safety analysis and extension to multi-physical concerns

To certify an aeronautical system, safety studies aim to demonstrate several requirements on this system. Because of the increasing complexity of systems and of the growing number of requirements, classical studies (such as Fault Tree Analysis) have shown limits.
To overcome these limits, Model-Driven Engineering, initially developed for software systems, deals nowadays with the achievement of safety analyses. The development process of a model can be divided into three main parts: 1) Modelling activities: system components and their behaviours are described in an adapted formal language (in the thesis, AltaRica has been used); 2) Validation of the model: ensuring the model is a valid abstraction of the real system; 3) Verification of the model: checking if the system requirements are satisfied by the model.
The two first points have been addressed during this thesis. A methodology (for different kinds of physical systems) has been proposed to specify AltaRica libraries. Some sub-systems of a turboshaft engine have been modelled. The results given by the models were compared with the results extracted from classical safety analyses. For future projects, where these analyses won't be available anymore, a validation process aims at giving a coverage measure of a test set simulated on the model. Inspired from several works about software testing, this validation process is divided in three levels (unit testing to validate the AltaRica library, integration testing to validate the I/O of components, system testing to validate the global model) and gives some coverage criteria which can be used and measured on the AltaRica model.

Keywords : Dependability – Safety – Formal models – Formal Modelling – AltaRica Language – Models validation – Coverage of AltaRica models – Coverage criteria for AltaRica models – Multiphysical modelling