Contribution à l'analyse de testabilité des systèmes réactifs temps-réel : Aide à la validation et à la vérification de systèmes

Fassely Doumbia

Université de Grenoble

N° attribué par la bibliothèque
|__|__|__|__|__|__|__|__|__|__|

T H E S E

pour obtenir le grade de

DOCTEUR DE L'Université de Grenoble

Spécialité : *« Mathématiques et Informatique »*

préparée au Laboratoire de Conception et d'Intégration des Systèmes (LCIS)

dans le cadre de **l'Ecole Doctorale** *« Mathématiques, Sciences et Technologies de l'Information, Informatique »*

présentée et soutenue publiquement

par

Fassely DOUMBIA

le 02 mars 2010

Contribution à l'analyse de testabilité des systèmes réactifs temps réel : Aide à la validation et à la vérification de systèmes

Directeur de thèse : ***Chantal ROBACH (LCIS)***

Co-directeur de thèse : ***Odile LAURENT (AIRBUS OPERATIONS)***

JURY

M.	Ioannis PARISSIS, Grenoble INP	, Président
Mme.	Virginie WIELS, ONERA Toulouse	, Rapporteur
M.	Yves LE TRAON, Université du Luxembourg	, Rapporteur
M.	Bruno MARRE, CEA centre de Saclay	, Examinateur
Mme.	Françoise DE LA CRUZ, Thales Avionics	, Examinateur
M.	Olivier AVONDET, Sagem	, Examinateur

« Le peu que je sais, c'est à mon ignorance que je le dois »

Sacha Guitry (1885 – 1957)

A ma famille, à tous ceux qui me sont chers, …

Remerciements

Les travaux présentés dans ce mémoire ont été menés au département « Recherche System, Methods and Tools » à AIRBUS OPERATIONS S.A.S – Toulouse et au sein de l'équipe CTSYS du Laboratoire de Conception et d'Intégration de Systèmes (LCIS) de l'Institut Polytechnique de Grenoble.

Je tiens à remercier Monsieur Ioannis Parissis, professeur à Grenoble INP, pour m'avoir fait l'honneur d'être président du jury de ma thèse. Je remercie également Madame Virginie Wiels, Professeur/Chercheur à l'ONERA de Toulouse et Monsieur Yves Le Traon, Professeur à l'université du Luxembourg, pour avoir accepté d'être rapporteurs de mes travaux de thèse.

Mes remerciements vont aussi à Madame Françoise De La Cruz, Direction Ligne Programmes à Thalès Avionics de Valence ainsi que Messieurs Olivier Avondet, Responsale Pôle Logiciels à SAGEM Défense de Valence et Bruno Marre, Chargé de Recherche CNRS au CEA-LIST à Saclay, qui ont bien voulu examiner mon travail.

J'exprime toute ma reconnaissance à Madame Odile Laurent pour l'accueil au département « Recherche Methods and Tools » et pour m'avoir fait profiter de son immense connaissance sur les systèmes avioniques, tant au niveau théorique qu'expérimental, pour ses conseils précieux et pour m'avoir fait confiance tout au long de cette thèse en me laissant orienter ce travail selon mes aspirations.

Je remercie particulièrement Madame Chantal Robach pour l'accueil au sein de l'équipe CTSYS, pour m'avoir fait confiance et pour avoir créé les conditions nécessaires à l'adaptation et à l'évolution des concepts théoriques développés par le LCIS pour la réalisation des objectifs de cette thèse.

Que Monsieur Huy Vu Do, Ingénieur de recherche au LCIS, soit remercié pour les conseils et pour toute l'aide qu'il m'a apportée dans le cadre de mes travaux de thèse. Je remercie aussi Monsieur Michel Delaunay, Ingénieur de recherche au CNRS, pour avoir contribué au développement d'outils tout au long de mon travail de thèse.

Je n'oublierai pas les aides permanentes reçues du personnel administratif, des enseignants et des indispensables du service informatique du LCIS et de l'ESISAR (Ecole supérieure des systèmes avancés et réseaux) tout au long de cette thèse.

J'associe à mes remerciements les chercheurs, les doctorants et les stagiaires du LCIS pour avoir contribué à maintenir une ambiance agréable le temps de ma thèse.

A toute ma famille et à mes amis qui, de près comme de loin m'ont aidé et encouragé aux moments opportuns ; Merci …

Enfin, j'ai une pensée émue à mes parents pour leur soutien au cours de ces longues années d'études et sans lesquels je n'en serai pas là aujourd'hui.

Table des matières

Table des matières

Table des figures

Table des figues

Introduction

L'évolution croissante des exigences en termes de fonctionnalités des systèmes (informatiques) embarqués se traduit, aujourd'hui, par une augmentation significative de la complexité de leur cycle de développement. Les systèmes réactifs temps-réel, largement utilisés dans les domaines critiques tels que l'avionique, le spatial, le nucléaire, le ferroviaire, l'automobile, etc., sont une parfaite illustration de cette tendance. La criticité de ces systèmes est liée au fait que toute défaillance peut conduire à des conséquences catastrophiques en termes de vies humaines, de coût économique et environnemental. De ce fait, le développement des systèmes avioniques est soumis à de fortes contraintes de sûreté de fonctionnement (dictées par les recommandations standardisées par des normes telles que la RTC/DO–178B [RTC94]) en vue de la certification avant la mise en service opérationnel.

Les systèmes avioniques assurent les fonctions avion désirées : le guidage, le pilotage, le confort passager, le freinage, la gestion du fuel, etc. La complexité de ces systèmes ne cesse de croître. Il suffit d'examiner les évolutions du nombre de calculateurs embarqués, de la taille du code logiciel et du nombre de bus numériques entre par exemple l'A310 et l'A380 pour s'en convaincre. En même temps, on assiste à une évolution des technologies de traitement et de communication conduisant à une interdépendance grandissante entre les systèmes. L'AMI (Avionique Modulaire Intégrée ou « avionique nouvelle ») symbolise cette tendance en ajoutant de nouvelles contraintes fonctionnelles, de sûreté de fonctionnement et de performance temps-réel [Bel02, Par03].

Ceci explique l'importance de la part du coût de développement des systèmes avioniques dans le coût d'un avion (30 à 35%). Maîtriser les coûts, fournir les preuves exigées pour la certification passe nécessairement par la maîtrise de la complexité de ces systèmes. D'où le besoin de modèles formels, évaluables pour les systèmes avioniques au cours de leur développement.

L'étude, la définition des modèles permettant la description des systèmes, leur évaluation (fonctionnel, sûreté de fonctionnement et performance temps-réel) et la production de codes logiciels est un processus long et coûteux. En effet, la nature critique et la complexité de ces systèmes (la taille, l'utilisation des nombre réels et l'explosion combinatoire liée aux phases de transitions) exigent la mise en place d'une méthodologie rigoureuse depuis la définition des exigences fonctionnelles jusqu'à la fin de la conception.

Plusieurs approches intégrées dans les outils ou environnements de développement ont été introduites pour aider à la modélisation des systèmes critiques. Ces environnements, proposant souvent une interface graphique, offrent la possibilité de construire un modèle en introduisant un aspect hiérarchique entre ses différents composants et de décrire le système sous de nombreux formalismes. Parmi ces environnements nous pouvons citer : SCADE/Lustre [Est05a, Est05b], Matlab/Simulink [Mat05], Sildex/Signal [TNI], StateFlow/StateChart [Mat09]. Ces environnements s'appuient sur différents langages de description des systèmes. Simulink et Scicos [Nou96] sont classiquement utilisés pour décrire les opérations portant sur les données discrètes et/ou continues. StateChart [Har87] est utile pour construire les modèles comportementaux. Les nouvelles notions apportées par UML 2.0 [Rum04] aident à la définition des systèmes numériques et peuvent être utilisées pour la description des logiciels ou des composants logiciels/matériels. Lustre [Hal91a] est utilisé pour la description flot de données synchrone et Signal [Bou91, Ben01, Bau04] permet une description flot de contrôle d'un système.

Afin d'être évalués, les modèles formels subissent une étape de validation et de vérification (V&V). La validation est le processus qui consiste à s'assurer que les exigences d'un système ou logiciel sont correctes et complètes. Autrement dit, elle permet de répondre à la question suivante : « sommes-nous en train de concevoir le bon produit ? ». La vérification consiste à s'assurer que le logiciel ou le système a un comportement conforme à ce qui est attendu. Autrement dit, elle permet de répondre à la question suivante : « sommes-nous en train de concevoir le produit correctement et conformément à ses exigences ? ». Certains environnements de développement proposent des fonctionnalités de simulation du modèle et de réalisation de preuves formelles sur certaines parties du modèle pour aider à la validation et à la vérification.

Par ailleurs, le cycle de développement des systèmes avioniques intègre une phase de vérification à chacune des étapes. Ceci permet la détection et la correction des incohérences et erreurs le plus tôt possible afin de maîtriser la complexité et d'atteindre un niveau de maturité satisfaisant du système à la fin de son développement.

De ce fait, différentes techniques ont été développées ou adoptées pour la validation et la vérification des modèles formels : la preuve formelle sur certaines propriétés du système, les méthodes de relecture croisée de la spécification et le test qui représente la principale technique utilisée dans le processus de V&V de ces systèmes. Or, les méthodes de test présentent des limites : un test exhaustif est une option qui est très souvent écartée pour ces systèmes à cause de leur taille et de leur complexité. En effet, il est quasiment impossible de définir les jeux de test permettant de réaliser un test exhaustif. Dans la pratique, les méthodes de test fonctionnel sont largement utilisées et l'effort de test est de l'ordre de 30 à 40 % du temps passé sur l'ensemble d'un projet (développement des applications logicielles), pour les niveaux de criticité les plus élevés. La testabilité devient, par conséquent, un facteur de qualité pour les méthodes de test car elle représente la facilité à tester un système ou un composant du système. En effet, elle permet de quantifier l'effort de test en termes de définition des jeux de test et de diagnostic.

Dans ce contexte, nous avons développé des méthodologies basées sur des techniques d'évaluation de la testabilité des systèmes avioniques et des stratégies de test. Les enjeux de la maîtrise de l'effort de test (complexité et coût) sont majeurs, mais les exigences de qualité sont très grandes ; autant sur l'élaboration des cas à tester et les données de test (entrées et sorties attendues) que sur le diagnostic aboutissant à l'identification avec certitude de l'erreur introduite dans le système. Il est question dans ces travaux de thèse, de proposer ou de développer des méthodes permettant d'une part, d'aider à la validation des modèles ou spécifications formelles des systèmes avioniques et d'autre part, d'aider à la vérification des systèmes installés à bord d'un avion sur la chaîne d'assemblage finale. Avant de détailler les démarches méthodologiques proposées, il est important de présenter les techniques utilisées pour la validation et la vérification des modèles formels et de faire un bilan sur les différentes approches d'analyse de testabilité développées pour les systèmes.

Le premier chapitre pose la problématique liée à la validation et à la vérification des modèles formels des systèmes critiques au cours de la conception. Les principes ainsi que les techniques permettant une bonne maîtrise du processus V&V seront exposés. Enfin, quelques approches d'analyse de testabilité, disponibles dans la littérature, seront discutées en spécifiant leur contexte d'application ainsi que les informations fournies par chacune d'entre elle.

Dans le second chapitre, nous présentons le contexte dans lequel les travaux décrits dans ce manuscrit ont été menés. Il s'agit du développement des systèmes avioniques AIRBUS. Nous présentons d'abord les systèmes avioniques d'une manière générale. Ensuite, nous introduirons le cycle de développement ainsi que les contraintes réglementaires qui régissent le processus de développement. Nous poursuivrons par la présentation du cycle de validation et de vérification (V&V) de ces systèmes avant de nous focaliser sur les phases de validation des spécifications

formelles au cours de la conception et de la vérification des systèmes installés à bord d'un avion sur la chaine d'assemblage finale (FAL) en mettant en évidence les principales difficultés rencontrées.

Le troisième chapitre portera sur la définition de la méthodologie d'aide à la validation des spécifications flot de données des systèmes. Cette méthodologie se base sur les concepts d'analyse de testabilité de logiciels flot de données et une stratégie de test adaptée au contexte de validation des systèmes au cours de la conception. Les systèmes étudiés ont été fournis par le département des « commandes de vol ». Les techniques d'adaptation pour l'application de l'analyse de testabilité et de la stratégie de test sont définies. Les informations proposées par cette méthodologie permettent d'aider à l'évaluation de la couverture : d'une part, de la spécification détaillée vis-à-vis des exigences fonctionnelles du système ; et d'autre part, des tests définis vis-à-vis de la spécification détaillée (implicitement des exigences fonctionnelles).

Dans le quatrième chapitre, nous proposons une méthodologie d'aide à la vérification des systèmes installés à bord d'un avion sur la FAL. Les approches développées se basent sur des concepts d'analyse de l'effort du diagnostic à partir d'une stratégie de test adaptée au contexte de vérification sur la FAL. Les méthodes définies permettent de fournir, dès la phase de conception des tests, des informations sur les tests pertinents, les tests redondants, et les fonctions pour lesquelles des tests supplémentaires sont nécessaires pour améliorer l'étape du diagnostic. Cette méthodologie propose également des informations permettant d'aider à l'identification de la ressource défectueuse et de guider les activités de vérification pour améliorer le diagnostic après la détection d'une faute.

Chapitre 1
Etat de l'art

Dans ce premier chapitre, nous présentons brièvement les problématiques de la validation et de la vérification (V&V) de systèmes : quels sont les objectifs, quels sont les principes, quelles sont les différentes techniques utilisées dans l'accomplissement des étapes du développement de systèmes. Le test étant la technique de V&V la plus utilisée, nous nous intéressons : aux principaux types de test et aux différentes approches d'évaluation de la testabilité, facteur de qualité, des systèmes. Nous insistons plus particulièrement sur les systèmes réactifs décrits à l'aide de spécifications formalisées (modèles formels).

Dans la suite, nous considérons les notions suivantes (définies selon *Larousse*) :

- Un programme « ou code » est défini comme un ensemble d'instructions et de données représentant un algorithme et susceptible d'être exécuté par un ordinateur.
- Un logiciel est un ensemble de programmes, procédés et règles, et éventuellement de la documentation, relatifs au fonctionnement d'un ensemble de traitement de données.
- Un modèle « d'un système » est une représentation abstraite des relations entre les paramètres caractéristiques du système.
- Un système est un ensemble de moyens d'acquisition, de traitement, de stockage et de restitution de données, et de moyens de télécommunication mis en œuvre pour une application ou un ensemble d'applications spécifiées.

Les systèmes réactifs ont la particularité de réagir continûment avec leur environnement, à une vitesse imposée par cet environnement [Ben91, Hal91b]. Les systèmes réactifs sont souvent utilisés dans les domaines critiques comme l'aéronautique, la téléchirurgie, le ferroviaire, le contrôle nucléaire, etc. Ceci justifie le fait que ces systèmes soient : déterministes (avec les mêmes entrées, on obtient toujours les mêmes sorties) ; soumis à des contraintes temporelles strictes et à des contraintes de sûreté sévères.

1.1 Objectifs de V&V

Un système ne peut être sûr que s'il peut être « validé » et « vérifié ». La validation est le processus permettant d'affirmer que le modèle construit est le « bon ». La vérification est le processus permettant d'affirmer que le modèle est construit « correctement » [Mor07]. La validation et la vérification des systèmes réactifs sont complexes (taille, fonctions, contraintes de sûreté, etc.). Un moyen pour répondre à cette complexité, largement utilisé au cours du développement de ces systèmes, consiste à « valider » et à « vérifier » le modèle comportemental du système simulé dans les phases amont de la conception. En effet, la simulation est le processus permettant de construire un modèle du système correspondant à un problème précis et de conduire des expérimentations avec ce modèle afin de résoudre le problème [Bal95]. La fiabilité des résultats d'une simulation ne dépend pas uniquement de la conformité du modèle, mais aussi de la précision de la formulation du problème posé. Par conséquent, les techniques de V&V doivent être utilisées au cours de la simulation.

La validation d'un modèle permet de montrer que le modèle, dans son domaine d'applicabilité, se comporte avec une précision satisfaisante en accord avec les exigences du système [Bal95]. En d'autres termes, la validation, a priori relative à la spécification formelle d'un système, consiste à montrer que cette spécification répond au cahier de charges [Cia94]. En effet, la validation démontre que le calcul des sorties à partir des entrées du modèle est suffisamment précis pour représenter la transformation entrées-sorties du système. L'activité de validation d'un modèle consiste à exécuter ce modèle sous les mêmes conditions d'entrée (environnement et contraintes) que le système dans un environnement réel et ainsi comparer le comportement du système avec celui du système.

La vérification d'un modèle permet de montrer que le modèle est transformé en une autre formalisation du système, comme prévu, avec précision suffisante [Bal95]. En d'autres termes, la vérification consiste à prouver que le code généré et les équipements développés, à partir de la spécification formelle, sont conformes au cahier des charges [Cia94]. La précision de

transformation du modèle formel d'un système en un programme exécutable est évaluée au cours de la vérification du modèle.

1.2 Principes de V&V

L'importance des activités de validation et de vérification des modèles, au cours du développement des systèmes, est capitale dans le contrôle de la qualité qui est un facteur essentiel de ces systèmes. Plus rigoureuses sont les méthodes de validation et de vérification, meilleure sera la confiance dans le produit en vue de sa certification. La certification est l'opération administrative qui permet d'attester que les procédures légales ont été respectées au cours de la conception ou de la réalisation du système. Pour atteindre un degré de confiance satisfaisant, certains principes basés sur les retours d'expériences ont été décrits dans la littérature [Mor07, Bal95, Sar98, Kit98]. Nous présentons, ci-dessous, quelques uns de ces principes.

V&V à travers le cycle de conception

Les activités de validation et de vérification ne constituent pas une phase ou une étape de conception d'un système, mais des activités qui s'étendent sur tout le cycle. Conduire les activités de V&V lorsque le prototype ou le modèle expérimental du système est complet est comparable à la situation où le professeur sanctionne son cours en effectuant un unique examen aux étudiants, sans contrôle intermédiaire, à la fin du semestre. Pas d'autre opportunité pour l'étudiant, tout au long du semestre, d'identifier ses points faibles dans la compréhension du cours. De sévères problèmes peuvent ne pas être détectés à l'avance et à temps au niveau de l'étudiant et par conséquent, conduiront à son échec. De contrôles fréquents (écrit ou oral), à travers le semestre, permettent aux étudiants d'identifier leur faiblesse et d'œuvrer dans le sens de l'amélioration de leur connaissance au fil de la progression du cours. Cette situation explique l'importance de mener les activités de V&V à travers le cycle de conception des systèmes afin d'identifier, très tôt, toute incohérence introduite au cours de la conception du système. Ceci permet de réduire l'effort de correction lorsqu'une incohérence est détectée au cours des activités de V&V.

Interprétation non binaire des résultats de V&V

Le modèle étant une abstraction du système, une représentation parfaite n'est pas envisageable. « Il n'est pas du tout certain qu'il soit théoriquement possible d'établir que nous avons un modèle valide ; même si cela était possible, peu de chefs de projet accepteraient de payer le prix de cette démonstration » Shannon [Sha75]. A partir de ce constat, le résultat de

V&V d'un modèle devrait être considéré comme un degré de fiabilité sur une échelle de 0 à 100, où 0 représente un modèle totalement incorrect et 100 représente un modèle totalement correct. Plus élevé est ce degré, meilleure sera la confiance dans la qualité de développement du modèle.

Indépendance des activités de V&V

Certaines activités de V&V ne sont significatives que lorsqu'elles sont menées de façon indépendante par des intervenants non biaisés. En effet, le développeur d'un modèle, qui possède le plus de connaissance sur le modèle, est moins indépendant pour mener les activités nécessitant une totale impartialité. Les développeurs ne sont pas souvent impartiaux parce qu'il d'une part, il n'est pas évident de prendre suffisamment de recul par rapport à un travail qu'on a réalisé et d'autre part, ils pourront avoir peur qu'un résultat négatif ne joue à l'encontre de l'appréciation de performance.

Conformité du modèle

La précision de la transformation entrée-sortie d'un modèle est influencée par les conditions d'entrée. La transformation correcte pour un ensemble de conditions d'entrée peut produire une sortie erronée lorsqu'elle est effectuée sous un autre ensemble de conditions d'entrée. La conformité de l'exemple du modèle de simulation du trafic au niveau d'un carrefour pendant les heures de pointe ne peut être évaluée qu'en considérant les conditions d'entrée correspondant aux heures de pointe (taux d'arrivage constant, etc.). L'évaluation de la conformité de ce modèle conduira à un échec lorsqu'elle est effectuée en considérant les conditions d'entrée des heures creuses. L'ensemble des conditions d'évaluation de la conformité d'un modèle est nommé domaine d'applicabilité du modèle au cours des activités de V&V [Sch79]. La conformité d'un modèle ne peut être affirmée que dans son domaine d'applicabilité.

Planification et documentation des activités de V&V

Les activités de validation et de vérification sont menées tout au long du cycle de conception du système. Par conséquent, ces activités doivent être identifiées ; les informations et moyens permettant la réalisation de ces activités doivent être préparés ; un ordonnancement cohérent des activités doit être effectué ; et l'ensemble de processus de V&V doit être documenté afin d'assurer une meilleure traçabilité, indispensable pour des activités de V&V au cours de la conception de systèmes complexes.

Influence de la formulation du problème

Un problème correctement formulé est à moitié résolu [Wat76]. L'objectif ultime de la validation et de la vérification n'est pas juste de produire une solution à un problème ; mais de fournir celle qui est suffisamment crédible, acceptable et implantable. La précision dans la formulation du problème a une grande influence sur l'appréciation des résultats de validation et de vérification. La formulation correcte du problème est tout aussi cruciale que sa solution (Albert Einstein). Il est important de reconnaitre que lorsque la formulation du problème est mal conduite, le résultat des activités de V&V ne sera pas pertinent.

1.3 Techniques de V&V

Différentes techniques ont été développées pour la validation et la vérification de systèmes. Beizer [Bei90] et Balci [Bal94] ont proposés différentes classifications de ces techniques. Nous proposons, dans cette thèse, une classification qui regroupe ces techniques en deux ensembles complémentaires : les techniques dynamiques et les techniques statiques. Une description non exhaustive de ces techniques est proposée ci-dessous.

1.3.1 V&V dynamique

Les techniques dynamiques sont essentiellement basées sur des techniques de test. Elles consistent en l'exécution du modèle ou du système sur un sous-ensemble fini de ses entrées possibles appelé un jeu de tests. Ces techniques posent alors quatre problèmes :

- quels sont les critères permettant de choisir un bon jeu de tests (*le problème de la sélection*) ;
- comment soumettre le jeu de tests sélectionné (*le problème de la mise en œuvre*) ;
- comment décider du succès ou de l'échec d'un jeu de tests (*le problème de l'oracle*) ;
- à partir de quel moment peut-on estimer que le modèle n'a plus besoin d'être testé (*le problème de l'arrêt du test*).

Ces différents problèmes sont abordés en considérant les deux principales techniques de test : le test structurel [Chu87, Har91, Las83, Woo80] et le test fonctionnel [Ber91, Den91, Ost86, Wey80].

Sélection d'un jeu de tests

Lorsque la sélection est guidée par le code du modèle, on parle de test de structurel ou test « boîte blanche ». Par ailleurs, lorsque les spécifications du système sous test sont utilisées pour guider cette sélection, on parle de test fonctionnel ou test « boîte noire ».

Si l'on considère que le test logiciel a pour but de détecter des « fautes », les besoins du test logiciel et ceux du test de matériel peuvent sembler voisins. Malheureusement, si pour le test matériel des modèles de fautes efficaces peuvent être relativement aisément construits, ce n'est pas toujours le cas pour le test dynamique de logiciel (c'est en cas insuffisant). C'est ce que démontre Howden [How81].

Mise en œuvre des jeux de tests

Les techniques de mise en œuvre utilisées sont essentiellement empiriques, et il ne semble pas exister de méthodologie générale permettant de résoudre le problème de soumission des tests. En effet, l'instrumentation est souvent spécifique aux applications envisagées et donc fort peu généralisable. L'insertion de codes supplémentaires ou « sondes » dans un modèle exécutable dans l'objectif de collecter des informations sur le comportement du modèle au cours de son exécution est appelée « instrumentation du modèle ». L'emplacement de ces sondes est déterminé de façon manuelle ou automatique en se basant sur une analyse statique de la structure du modèle. Les bancs de test, les simulateurs, etc. sont les moyens utilisés pour la mise en œuvre des jeux de tests.

Construction de l'oracle

Le problème de l'oracle consiste à décider du succès ou de l'échec d'un jeu de tests. Lorsque la méthode de sélection est fonctionnelle, la spécification du système permet souvent de prévoir le résultat attendu des tests sélectionnés. Souvent, ces résultats attendus sont consignés dans une « matrice de test » ainsi que les moyens à mettre en œuvre pour comparer ces résultats avec ceux effectivement obtenus. Lorsque la méthode de sélection employée est structurelle, il peut être très difficile de prévoir les résultats corrects que devraient retourner les tests sélectionnés. L'usage de spécifications formelles semble être le moyen le plus approprié pour dégager des oracles fiables [Ber91].

Critère d'arrêt du test

Généralement, l'arrêt du test est un arrêt sur objectif (formalisé ou non). Selon la méthode de sélection utilisée, il existe des critères adaptés.

Dans le cas du test structurel, ce critère d'arrêt est directement déterminé par les critères qui ont servi à la sélection : couverture de toutes les instructions (ou un certain pourcentage), de toutes les portions linéaires de code suivies d'un saut (ou un certain pourcentage), etc. Ceux-ci conviennent dans un contexte de test unitaire. Pour le test d'intégration l'arrêt peut être déterminé par des critères similaires de couverture du graphe d'appels du logiciel.

Pour le test fonctionnel, le critère évident est couverture de toutes les fonctionnalités mentionnées dans les spécifications quelle que soit la phase de test considérée (unitaire, intégration, etc.). Il doit de plus être affiné par une couverture satisfaisante des cas d'utilisation de ces fonctionnalités : tests nominaux, aux limites et éventuellement hors limite (robustesse).

Lorsque la sélection est fonctionnelle (on dispose des spécifications), il est possible d'affiner cette couverture des fonctionnalités par une analyse « structurelle » (automatique).

1.3.2 V&V statique

Contrairement aux techniques dynamiques, les techniques statiques ne nécessitent pas d'exécution du modèle. Les techniques statiques peuvent être reparties selon qu'elles soient basées sur le raisonnement humain guidé par des règles (informelles) ou qu'elles soient supportées des outils d'analyse (formelle).

Techniques informelles

Les techniques informelles de V&V sont parmi les plus utilisées. Elles permettent d'évaluer les caractéristiques du code du modèle par rapport à une norme ou un standard d'écriture des logiciels. L'appellation informelle est liée au fait que les approches utilisées se basent essentiellement sur le raisonnement humain et la subjectivité, sans une formalisation mathématique rigoureuse. L'étiquette « informelle » n'implique aucun manque de structure pour l'utilisation de cette technique. Quelques exemples de techniques informelles sont : les activités de revue, d'inspection de code.

Techniques formelles

Les techniques formelles utilisent des outils pour analyser le modèle (ou code du modèle). Cette analyse peut être basée sur la structure du modèle pour fournir des informations sur les flots de données, flots de contrôle et la cohérence ainsi que la complétude de l'implantation du modèle. Elle peut également être basée sur des méthodes mathématiques de vérification. Parmi ces techniques, nous décrivons, ci-dessous, les techniques symboliques et les techniques de preuve. Les techniques de preuve permettent de fournir des preuves de conformité. Lorsqu'elles sont « atteignables », ces preuves formelles de conformité constituent les moyens les plus efficaces de V&V d'un modèle. Les techniques de validation et de vérification formelles peuvent être utilisées pour prouver que le système ne contient pas une certaine faute ou que le système possède une certaine propriété. Les fautes et les propriétés doivent être formellement spécifiées. Toutefois, il n'est pas possible de prouver qu'un système est exempt de fautes, car il n'est pas

possible de spécifier formellement la propriété « pas de fautes » [Mor07]. Les techniques de preuve peuvent être utilisées à différents niveaux de formalisation du système [Bow06, For].

- *Techniques d'analyse symbolique* : Ces techniques permettent d'évaluer les instructions (le code) d'un modèle avec des données d'entrée symboliques, le long des chemins activés au cours de la transformation conduisant à la production des données de sortie. Ce procédé entraîne le calcul de conditions de chemin qui tendent à être simplifiées ou résolues dans le but de trouver des données de test qui sensibilisent des chemins activés ou de démontrer la non-excitabilité de ces chemins [Kar76, Cou77, Cou78, Gra89].
- *« Model-checking »* : Cette technique de preuve consiste à vérifier des propriétés par une énumération exhaustive et astucieuse des états accessibles. L'efficacité de cette technique dépend en général de la taille de l'espace des états accessibles (selon l'algorithme) du système. Elle trouve donc ses limites dans les ressources de l'ordinateur pour manipuler l'ensemble des états accessibles. Des techniques d'abstraction (éventuellement guidées par l'utilisateur) peuvent être utilisées pour améliorer l'efficacité des algorithmes. Le model-checking est la technique la plus appropriée pour les systèmes critiques. Il existe des outils interactifs (Prover [Pro] et LESAR [Hal92]) qui permettent à l'utilisateur de guider la preuve.
- *« Automated Theorem proving »* : Le principe de cette technique consiste à laisser l'ordinateur prouver les propriétés automatiquement, un ensemble d'axiomes et un ensemble de règles d'inférences sur la cohérence et intégrité du système. Cependant, la recherche de preuve est connue pour être un problème non décidable en général, c'est-à-dire qu'il n'existe aucun algorithme permettant de décider en temps fini si une propriété est vraie ou fausse. Toutefois, lorsqu'il existe des cas où le problème est décidable, le temps et les ressources nécessaires pour que les propriétés soient vérifiées peuvent dépasser les délais acceptables ou les ressources dont dispose l'ordinateur. Par conséquent, cette technique peut être très couteuse, mais elle pourrait être utilisée lorsque le coût des fautes est extrêmement élevé.

Des progrès significatifs ont été réalisés sur les techniques formelles dans le milieu de la recherche. Cependant, ces méthodes ne sont pas encore complètement maîtrisées dans l'industrie. Nombre de recommandations existent sur la manière et le moment d'utilisation des techniques formelles. Bowen et Hinchey ont proposé « Ten Commandments of Formal Methods [Bow95] » et « Ten Commandments of Formal Methods – Ten Years Later [Bow06] », qui conseillent sur les bonnes pratiques permettant de produire des systèmes corrects et fiables avec les méthodes formelles. Les études de comparaison ont été également menées entre des techniques de test et de preuve afin de déterminer si l'utilisation des méthodes formelles est plus efficace que le test [Kin00]. Guy et Philippa Broadfoot ont identifié certaines raisons du

manque d'utilisation des techniques formelles dans l'industrie [Bro03]. Il faut noter que de plus en plus d'environnements de développement s'appuyant sur des sémantiques formelles (SCADE, etc.) et d'outils de preuve de propriété associés sont accessibles aux ingénieurs de conception.

1.4 Test et testabilité

Incontestablement, les techniques ou méthodes de V&V font la part belle au test au cours de la conception des systèmes réactifs. Cependant, l'activité de test est loin être aisée d'autant plus que la complexité croissante de ces systèmes rend les activités de vérification très difficiles. La testabilité apparaît dans ce contexte comme un attribut de qualité permettant d'évaluer l'effort de test.

Dans les paragraphes qui suivent, nous proposons d'abord différentes définitions attribuées au test et à la testabilité, puis nous présentons une description non exhaustive des principales techniques de sélection des données de test et des différentes approches d'évaluation ou d'analyse de testabilité proposées dans la littérature. Enfin, nous abordons la problématique liée à la localisation de fautes après la détection au cours d'une campagne de test.

1.4.1 Définitions et objectifs

Test

D'une manière générale, le test peut être défini comme l'activité dont l'intention est de révéler des fautes en exécutant le système par comparaison de la sortie à celle attendue [14 Mye79, Lin97].

Le test d'un modèle au cours de la validation et de la vérification de systèmes, par rapport à ses exigences, consiste à démontrer l'existence d'incohérences ou de révéler l'existence d'erreurs [Bal95]. Dans ce même contexte, Hetzel a défini le test comme toute activité dont le but est d'évaluer un attribut du système et de déterminer si le système est conforme à ses exigences [Het88]. Le modèle est soumis au test afin de vérifier s'il se comporte correctement. Quelle que soit la technique de test, tester un système consiste à l'exécuter avec des entrées prédéterminées (définition ou la génération des données de test) et observer le comportement du système afin de détecter, localiser et corriger les fautes.

L'objet du test est d'acquérir une confiance suffisante dans le système en mettant en œuvre le temps et les moyens appropriés et disponibles. En effet, il faut bien s'assurer que le

système réalise correctement et dans toutes les circonstances les fonctions attendues conformément aux exigences fonctionnelles, aux contraintes d'implantation et aux exigences liées au cas limites. Or, le test n'apporte pas de réelles certitudes dans la correction du système mais il doit cependant être rendu aussi efficace que possible afin d'atteindre un niveau de confiance proche de la « certitude ».

La génération des données de test constitue une problématique générale pour toutes les techniques de test. En effet, les difficultés sont liées : aux critères de sélection (choix des données de tests pouvant couvrir certaines classes de fautes), ainsi qu'aux critères d'adéquation (évaluation de la qualité des données de test). La problématique liée à l'observation du comportement du système est liée à celle de la construction de l'oracle. Un oracle est un moyen automatique permettant de vérifier le comportement du système. La construction d'un oracle peut être très difficile. En effet, un bon oracle doit reconnaître tous les comportements corrects du système.

Trois grandes phases de test peuvent être réalisées selon le niveau de développement d'un système :

- le test unitaire : cette phase consiste à tester indépendamment les unes des autres toutes les unités élémentaires ou modules qui composent le système. Le test unitaire assure que les modules sont conformes aux exigences ;
- le test d'intégration : cette phase intervient lors de l'assemblage des modules du système. Elle consiste à tester la cohérence des interfaces des modules entre elles et à détecter des erreurs provenant de l'interaction entre unités et n'ayant pas été détectées au niveau du test unitaire. Cette phase permet de vérifier que les modules assemblés réalisent les fonctionnalités prévues ;
- le test système : cette phase consiste à tester le système dans sa totalité. Elle permet de valider les fonctionnalités du système avant sa mise en exploitation. L'échec d'un test système est lourd de conséquences et doit par conséquent être exceptionnel.

D'autres phases de test peuvent être planifiées, en fonction de la nature du système et des objectifs, telles que : le test de non régression, le test de performance, le test d'acceptation, etc.

Testabilité

La testabilité, définie intuitivement comme la « facilité à tester », est une notion ambigüe et spécifique. En effet, cette notion, initialement utilisée dans le domaine matériel, a reçu plusieurs définitions et chacune de ces définitions repose sur une manière spécifique d'aborder le

problème. Ce problème consiste à chercher à qualifier ainsi qu'à apprécier les paramètres influant sur la facilité à tester tel ou tel système (ou partie du système).

Historiquement, les principaux challenges du test ont été : le contrôle de l'effort de test, la réduction du temps de test et l'efficacité du test. L'automatisation des moyens de test constitue une solution à ce problème tout en réduisant également des erreurs liées au facteur humain [Amm08]. Une autre approche consiste à considérer que plusieurs solutions (conceptions) peuvent être élaborées à partir d'un problème [Bin94] ; le but étant de trouver la (les) conception(s) facile(s) à tester : d'où le terme « conception testable » d'un système. La question que l'on se pose naturellement à ce niveau est : quand dit-on qu'un système est testable ?

De nombreuses définitions plus ou moins abstraites de la testabilité ont été proposées dans la littérature. Parmi les plus abstraites figurent celles de :

- Clure et Martin [Clu86], qui définissent la testabilité comme la facilité avec laquelle on peut démontrer la correction d'un programme ;
- IEEE [IEE90], qui considère la testabilité comme : (1) la facilité d'élaborer des critères de test d'un système (ou d'un composant) et des règles pour vérifier que ces critères sont satisfaits ; (2) la facilité avec laquelle les objectifs de test et les tests peuvent être définis lorsque de tels critères sont satisfaits ;
- Jungmayr [Jun99], qui définit la testabilité comme la facilité à tester un logiciel dans un contexte de test donné (contraintes de coût, qualité exigée, critères de test appliqués, moyens de test, etc.) ;
- la norme ISO/IEC 9126 [ISO06], qui définit la testabilité comme des attributs du logiciel liés à l'effort nécessaire pour valider le logiciel en question.

Les définitions les moins abstraites, quantifiables, ont été proposées par :

- Mohanty [Moh79], qui présente la testabilité comme une mesure de l'effort nécessaire pour tester un module, un chemin ou un programme en entier ;
- Voas, Morell et Miller [Voa91, Voa92, Voa3a], qui proposent les définitions suivantes : (1) la prédiction de la capacité du logiciel à cacher des fautes lorsque celui-ci est testé à l'aide d'une technique de test fonctionnel avec des données tirées aléatoirement ; (2) la tendance à observer des défaillances pendant le test du logiciel en présence de fautes ;
- Bertolino et Strigini [Ber96], qui proposent la définition suivante : la testabilité d'un programme est la probabilité qu'un test soit rejeté par un oracle spécifié, en supposant qu'un tel oracle existe et que le logiciel contienne une faute ;

– Le Traon [LeT95a, LeT95b, LeT97], qui présente la testabilité comme la facilité à tester un logiciel en appliquant des stratégies de test structurelles, cette facilité est à la fois une propriété intrinsèque du logiciel et une propriété corrélée à la stratégie de test utilisée.

L'enjeu de la testabilité apparaît entre deux défis majeurs a priori contradictoires qui sont : l'efficacité du test conduisant à la construction de systèmes fiables et la réduction du coût qu'elle occasionne (sachant qu'il faut bien tester pour obtenir une « bonne » confiance dans le système). Le but de la testabilité est de rendre compatibles ces deux exigences de confiance et de réduction des coûts : un système plus testable doit pouvoir être testé plus efficacement, un système plus testable doit aussi être moins coûteux à tester. La testabilité devient alors une des caractéristiques de qualité les plus souhaitables puisqu'elle tend à rendre la validation et la vérification plus efficaces.

1.4.2 Approches d'analyse de la testabilité selon les techniques de test

Indéniablement, la testabilité est liée à l'objectif visé par le test d'un système. Lorsqu'elle est évaluée à partir des critères externes du système, elle peut être considérée comme une approche purement fonctionnelle comme le test. Par ailleurs, l'évaluation de la testabilité à partir des critères internes du système ne peut pas forcement se réduire à une analyse purement structurelle comme le test. En effet, elle peut prendre en compte les flots de données traversant le système (qui sont étroitement liés avec les fonctionnalités du système) en plus de l'analyse de la structure de contrôle du système.

Deux grandes techniques ou méthodes peuvent s'appliquer pour la génération de données de test : le test structurel (boîte blanche) et le test fonctionnel (boîte noire). Pour chaque technique de test, nous présentons les approches d'évaluation de testabilité appropriées qui ont été proposées dans la littérature.

Test structurel et approches d'analyse de testabilité

Le test structurel consiste à se baser sur la connaissance de la structure interne du programme sous test pour générer des données de test et définir des critères d'arrêt. Cette technique de test présuppose que l'on dispose du code source du programme et non pas seulement de son interface ou de sa spécification. La plupart des techniques de génération de test structurelles se basent sur une abstraction du programme qui est ensuite couverte selon des critères. Les modèles utilisés sont par exemple les graphes de flot de contrôle [Bei90] ou les graphes de flot de données [Rap85] et sont couverts par des critères de couverture des arcs, des

nœuds, ou des chemins du graphe. Les outils de génération de données de test tels que GaTeL [Mar00, Mar04] et AuGuSTe [Gou04] sont utilisés dans un contexte de test structurel.

Quelques approches d'analyse de testabilité basées sur la structure interne d'un programme ont été proposées et sont brièvement exposées ci-dessous.

Nombre cyclomatique

Cette approche mathématique a été proposée par McCabe [McC76]. Elle permet d'identifier les modules ou composants du logiciel qui sont difficiles à tester ou à maintenir. La complexité d'un composant est évaluée en termes de chemins élémentaires qui, une fois combinés permettent de générer tous les chemins possibles contenus dans le composant. Le nombre de chemins élémentaires est en fait le nombre maximum de circuits linéairement indépendants, le nombre cyclomatique. Myers [Mye77] et Sheppard [She88] ont notamment contribué à l'amélioration de cette approche qui ne s'applique qu'aux programmes impératifs.

NPATH

Nejmeh [Nej88] pense qu'en dehors des facteurs (tels que la conception modulaire et la taille du domaine d'entrée) qui affectent la testabilité du logiciel, un facteur important est le nombre de chemins traversant les composants. En d'autres termes, les composants ayant plus de chemins d'exécution sont plus difficiles à tester que les composants qui en ont moins. Dans ce contexte McCabe et Nejmeh partagent le même point de vue qui est : la détermination du nombre de chemins d'exécution dans un composant. Cependant, Nejmeh propose une autre approche prenant en compte certains critiques liées à la méthode de calcul du nombre de chemins de l'approche de McCabe et applicable à d'autres types de programmation.

PIE : Propagation – Infection – Execution

Voas a proposé une technique dynamique appelée PIE (Propagation – Infection – Execution) [Voa91, Voa92, Voa95a, Voa96]. Cette technique permet d'estimer de manière probabiliste la testabilité d'un programme en se basant sur la structure et la sémantique du programme. La probabilité calculée correspond à celle qu'une faute contenue dans une instruction cause une défaillance du programme sous une distribution des entrées spécifiées. Cette probabilité est appelée la « *sensibilité* » d'une instruction dans le programme. La testabilité du programme est donc mesurée au sens de la sensibilité dans toutes les instructions.

Approches de testabilité basées sur les flots de données

De nombreuses approches d'évaluation de la testabilité basées sur les flots de données, telles que [Ovi80, Tai84, Mun93, Tan98], ont été proposées pour estimer la complexité, la

difficulté de compréhension et le test d'un logiciel. Les travaux d'Oviedo nous semblent aborder la difficulté du test logiciel [Ovi80]. L'approche proposée se base sur les relations entre les définitions et les utilisations des variables d'un programme. Cette approche, assez simple, montre l'effort du test basé sur le flot de données. Par ailleurs, elle n'est qu'une mesure complémentaire avec d'autres mesures, comme celles basées sur le flot de contrôle.

Le Traon [LeT95a, LeT95b, LeT97] et Do [Do06] proposent d'analyser la testabilité de spécification flot de données de systèmes réactifs. L'approche se base sur l'analyse du transfert d'information travers le logiciel. Elle propose des mesures calculées à partir de l'évaluation de la perte d'information dans le système.

Test fonctionnel et approches d'analyse de la testabilité

Le test fonctionnel consiste à considérer le système comme une boîte noire dont on ne connaît que les spécifications. Pour la génération de données de test, ces spécifications sont la plupart du temps exprimées de manière la plus formelle possible. Le test fonctionnel ne prend pas en compte la structure du programme, et les tests générés sont ainsi indépendants de l'implantation du programme sous test, ce qui n'est pas le cas pour le test structurel. Il est ainsi possible de réappliquer les mêmes données de test à la suite d'une modification du système qui ne remet pas en cause sa spécification. Dans ce contexte, les tests de non régression peuvent être effectués afin de vérifier que les modifications n'ont pas introduit d'erreurs. Lutess [Par96] et Lurette/Lucky [Ray98] sont par exemple des outils de générations de données utilisés dans un contexte de test fonctionnels.

Des approches d'analyse de testabilité basées sur les spécifications du système ont été proposées et sont exposées ci-dessous.

Testabilité de domaine

Freedman a introduit la notion de domaine de testabilité d'un composant en termes de propriété d'observabilité et de contrôlabilité [Fre91]. Un composant devient observable lorsqu'il produit des sorties distinctes depuis des entrées distinctes. De la même manière, un composant est contrôlable si le domaine de sortie spécifié est égal au domaine de sortie produit par le composant. Enfin, un composant est domaine-testable lorsqu'il est observable et contrôlable. Cette approche ne peut pas être appliquée aux systèmes réactifs pour deux principales raisons. La première raison est que les composants des systèmes réactifs contiennent souvent des états, ils ne sont donc pas souvent observables. Et la modification d'un composant avec états afin qu'il devienne observable est difficile et parfois impossible. La seconde raison est la complexité des systèmes réactifs qui rend l'étude de domaines contrôlables très difficile.

DRR : Domain/Range Ratio

Partant de l'observation que les fautes logicielles affectant rarement des sorties causent des problèmes dans la mise au point des logiciels, Voas et Miller proposent d'étudier les fautes cachées du logiciel pour mettre en évidence les parties (ou composants) les plus douteuses [Voa93a]. Notamment lors de la phase de conception des logiciels critiques, les auteurs pensent qu'ont peut isoler les composants susceptibles de cacher des fautes. Cela peut d'une part réduire l'effort de test et d'autre part renforcer la sûreté de fonctionnement du logiciel. Cette approche propose une métrique qui exhibe les composants ayant une tendance à cacher des fautes.

Le « *rapport domaine/image* » (DRR – domain/range ratio), dérivable depuis la spécification du composant, est un rapport entre la cardinalité du domaine d'entrée et la cardinalité du domaine de sortie (image des entrées). Cette métrique ne dépend que du nombre de valeurs du domaine d'entrée et du domaine de sortie d'un composant. Pour les auteurs, l'approche du DRR se place parmi les facteurs qui déterminent si un composant semble cacher des fautes dans le sens où : lorsque le DRR d'un composant est très élevé, alors il cache très probablement des fautes.

Cependant, dans le domaine logiciel, les domaines des entrées et des sorties peuvent ne pas être dénombrables. Il devient alors difficile d'appliquer l'approche du DRR. De plus, cette métrique est assez simpliste, puisqu'elle ne procède pas à l'analyse de chaque paramètre d'entrée et de sortie du composant. A titre d'exemple, pour un composant avec une entrée de type entier et une sortie de type entier, le DRR est ($\infty : \infty$) ; pour un autre composant avec dix entrées de type entier et une sortie de type entier, le DRR est aussi ($\infty : \infty$). On peut regretter que ces deux composants soient qualifiés dans la même testabilité.

1.4.3 Problématique de la localisation de fautes

La localisation de fautes ou le diagnostic de fautes, bien qu'inévitable, n'est souvent pas identifiée parmi les activités cruciales à considérer dans les différentes phases de test au cours de la validation et de la vérification. D'ailleurs, il est rare de voir les concepteurs introduire le critère de diagnostic parmi les principaux critères à considérer au cours de la conception des systèmes ou des tests. La tendance est, le plus souvent, de se focaliser sur la capacité ou l'aptitude d'un système ou du test à révéler les fautes contenues dans le système. Il est clair qu'on ne peut pas parler de diagnostic de fautes sans une détection préalable. Il n'en demeure pas moins qu'une attention particulière doit être portée au diagnostic au vu des exigences de qualité imposées par les systèmes de plus en plus complexes au cours de leur développement.

En effet, les activités d'exécution du système à l'aide de données de test et la détection des fautes sont complémentaires de l'activité de diagnostic de fautes.

Le diagnostic est défini comme le processus qui détermine la partie défectueuse, responsable d'une anomalie de fonctionnement qui a induit une défaillance [Kle93]. Une différence entre les résultats obtenus et ceux attendus pour les entrées communiquées au système est une défaillance. Pour déterminer qu'il y a une défaillance, il faut avoir au préalable déterminé un oracle, c'est-à-dire la sortie attendue comme correcte à l'exécution des données de test.

Une défaillance apporte la preuve que l'implantation du système testée est incorrecte. En d'autres termes, une défaillance témoigne de la présence dans l'implantation du logiciel de défauts ou fautes introduits lors de la conception ou du codage. Il s'agit alors de corriger le défaut, après l'avoir localisé (diagnostic et correction).

Selon le contexte (test unitaire, test d'intégration et test système), il existe plus ou moins de corrélation entre la génération ou la définition des données de test (ou tests) et le diagnostic. L'intention du test unitaire est de définir des tests capables de détecter les fautes et permettre leur localisation afin de les corriger. Dans ce cas, la définition des tests peut prendre en compte l'effort de diagnostic. Quant aux phases de test d'intégration ou système, elles visent à vérifier les fonctionnalités du système. Il est difficile, dans ce contexte, d'établir une quelconque corrélation entre la définition des tests et le diagnostic.

En effet, même s'il est facile ou aisé de détecter des fautes au cours de l'exécution du système sous test, il n'y a pas de raison qu'il soit facile de localiser et de corriger ces fautes [Voa95b]. Par conséquent, l'isolation d'une faute peut nécessiter beaucoup plus de minutie et de procédures coûteuses que la détection de faute. La mise en œuvre de solutions permettant de réduire le coût du diagnostic peut être d'une grande importance dans la validation et la vérification de systèmes. Les principes de base des différentes techniques de diagnostic des systèmes ont été présentés par Sheppard et Simpson [She94].

De nombreux travaux ont été réalisés dans le domaine matériel sur la problématique du diagnostic. Deux approches complémentaires sont généralement considérées selon que l'objectif soit d'améliorer la « puissance » de diagnostic des tests ou d'améliorer l'analyse des résultats de test. Dans la première approche, le but est d'améliorer les techniques d'ATPG (Automatic Test-Pattern Generation) afin de localiser plus facilement les fautes au cours du test [Gir96, Gui91]. La seconde approche propose l'exploitation de la structure du circuit et les résultats de test afin de localiser les fautes au lieu d'intervenir au niveau des générateurs de tests. Cette approche de

diagnostic, après test, est généralement applicable aux circuits séquentiels et ne se base pas sur une analyse du comportement du système ; excepté le cas de l'utilisation d'un « système expert » pour le diagnostic [Xia86]. Un système expert désigne un programme qui, à partir de connaissances intégrées, limitées et spécifiques au domaine d'application, doit réagir face à un problème donné exactement comme l'expert du domaine.

Technique de recoupement « dicing et slicing »

La plupart des travaux portant sur la localisation des fautes en logiciel concernent la technique de découpage (ou « slicing ») [Gal91, Hor94, Agr95, Kam95, Kor97]. Cette technique est aussi bien utilisée pour le test logiciel que pour le diagnostic proprement dit. Le programme est décomposé en coupes (ou slices), telles que chacune contient toutes les instructions qui peuvent affecter la valeur d'une variable d'un programme. Parmi les différentes techniques de diagnostic élaborées à partir des méthodes de découpage, nous nous intéressons à la technique de recoupement.

La technique, appelée « *program dicing* », a été développée par Gallagher et Lyle [Gal91]. Elle compare les coupes dynamiques, la différence entre deux coupes étant appelée « *dice* ». Une coupe dynamique isole le seul chemin de données affectant une instruction pour une exécution donnée.

Afin de localiser les fautes, on considère qu'on a deux groupes de coupes : les coupes qui donnent les résultats incorrects et les coupes qui donnent les résultats corrects. Lorsqu'on a deux coupes A et B telles que A est la coupe est fautive et B la coupe correcte, le dice correspondant est (*A*–*B*), et c'est dans ce domaine restreint que les fautes sont localisées. Il s'agit de réduire l'ensemble suspect par recoupements des résultats de l'oracle et donc de diminuer le temps du diagnostic. La **Figure 1.1** représente un programme pour tester les saveurs. La **Figure 1.2** montre deux coupes et le dice résultant. La coupe A représente la variable « sucré » dans l'instruction 19 et coupe B représente la variable « amer » dans l'instruction 19.

N° de l'instruction	Programme
1	program savoures ;
2	main ()
3	{
4	int rouge, vert, bleu, jaune ;
5	int sucré, aigre, salé, amer ;
6	int i ;
7	rouge = fetch() ;
8	bleu = fetch() ;
9	vert = fetch() ;
10	jaune = fetch() ;
11	rouge = 2 * rouge ;
12	sucré = rouge * vert ;
13	aigre = 0 ;
14	for (i = 0; i < rouge; i++)
15	aigre += vert;
16	sale = bleu + jaune;
17	vert = vert + 1;
18	amer = jaune + vert ;
19	**printf ("%d %d %d %d\n", sucré, aigre, sale, amer) ;**
20	exit(0)
21	}

Figure 1.1 : Programme « savoures »

Coupe A	Coupe B	Dice (Coupe A – Coupe A)

main ()
{
int rouge, vert, bleu, jaune ;
int sucré, aigre, salé, amer ;
 	
rouge = fetch() ;
 	
vert = fetch() ;
 	
rouge = 2 * rouge ;
sucré = rouge * vert ;
 	
printf ("%d %d %d %d\n", sucré, aigre, sale, amer) ;
exit(0)
}
	main ()
	{
	int rouge, vert, bleu, jaune ;
	int sucré, aigre, salé, amer ;
	
	vert = fetch() ;
	jaune = fetch() ;
	
	vert = vert + 1;
	amer = jaune + vert ;
	printf ("%d %d %d %d\n", sucré, aigre, sale, amer) ;
	exit(0)
	}
		
		rouge = fetch() ;
		
		rouge = 2 * rouge ;
		sucré = rouge * vert ;

Figure 1.2 : Dice « Coupe A – Coupe B »

1.5 Conclusion

Le processus de validation et de vérification entraine un coût important dans le développement de systèmes réactifs. L'automatisation des techniques utilisées reste une problématique cruciale. Dans ce chapitre, nous avons tout d'abord rappelé les objectifs, les principes majeurs ainsi que les différentes techniques de V&V, avant de nous concentrer sur le test, les approches d'analyse de testabilité de systèmes et la problématique liée à la localisation de fautes, le test étant la technique la plus utilisée au cours de la validation et de la vérification de systèmes. Par ailleurs, les techniques de test structurel ne sont pas appliquées au cours des activités de V&V des systèmes critiques (DO-178B). Cependant, la couverture de la structure sert de critère d'arrêt et permet d'identifier d'éventuel code mort. De ce fait, les méthodologies proposées dans cette thèse se basent sur l'hypothèse de définition des tests fonctionnels. Il faut noter que l'on ne peut pas vraiment s'affranchir de la structure du programme, au cours de la définition des tests, car la définition purement fonctionnelle des tests peut conduire la non-détection et certains types de faute. Ceci permet de s'assurer que le système sous test répond bien à ses exigences.

La réduction du coût du test tout en assurant la meilleure efficacité possible représente un enjeu majeur du processus de V&V. Parmi les outils de génération automatique de test et approches d'analyse de testabilité proposés, peu sont appliqués au niveau de la validation et de la vérification de systèmes réactifs dans un contexte industriel.

En effet, la plupart des outils et approches « académiques » se basent soit sur un modèle relativement petit, soit sur un modèle très détaillé dont la complexité rend difficile l'application au niveau d'un système complexe. Si la plupart de ces outils et approches ont de bonnes bases théoriques, peu d'entre eux sont transférés vers l'industrie.

Tout comme les outils de génération de test, pour pouvoir être appliquée, une approche d'analyse de testabilité doit prendre en compte des pratiques industrielles. Dans ce contexte, nous proposons :

- d'une part, une méthodologie d'aide à la validation basée sur une approche d'analyse de testabilité de systèmes réactifs spécifiés selon une approche flot de données synchrones au cours de la conception. Pour cela, nous avons retenu la définition de la testabilité et l'approche proposée par Le Traon [LeT97] et Do [Do06].
- d'autre part, une méthodologie d'aide à la vérification de systèmes avioniques installés à bord basée sur une stratégie de diagnostic au cours de la phase d'assemblage finale d'un avion. Cette stratégie implante la technique de recoupement (dicing et slicing) pour

guider la localisation de fautes après la détection de défaillance au cours de la vérification du bon fonctionnement de ces systèmes.

Ces deux méthodologies prennent en compte les pratiques industrielles (rédaction des exigences en langage naturel, utilisation de cycles de validation et de vérification classiques, les moyens ou ressources existants, etc.).

Chapitre 2
Contexte industriel

L'objet de ce chapitre est tout d'abord de situer le contexte dans lequel les travaux décrits dans ce manuscrit ont été menés. Nous commençons par présenter d'une manière générale les systèmes avioniques et particulièrement ceux embarqués dans les avions AIRBUS. Puis, nous introduirons le développement des systèmes AIRBUS dans le but d'évoquer certaines contraintes réglementaires et de présenter globalement le cycle de développement en se focalisant sur les aspects de validation et de vérification. Enfin, cette brève introduction conduira à la formulation des motivations des travaux réalisés afin de donner l'idée directrice de ce manuscrit et d'en faciliter la lecture.

2.1 Systèmes avioniques

D'une manière générale, un système avionique (civil ou militaire) est l'ensemble des équipements électroniques et des moyens informatiques à bord d'un avion, satellite ou lanceur [Moi02]. Les systèmes avioniques disposent, pour réaliser leurs fonctions, de capteurs, de calculateurs et de logiciels qu'ils exécutent, de bus et d'actionneurs à bord qui définissent les

systèmes réalisant les fonctions avion. L'avionique (ensemble des équipements électroniques et des logiciels) représente environ 33% du coût d'un avion civil. Dans le domaine militaire, le coût de l'avionique dépasse même ce ratio d'un tiers du coût global de l'aéronef.

Parmi les systèmes avioniques civils, nous pouvons citer les systèmes ci-dessous (en termes de fonctionnalités).

- Guidage : il permet la planification et le contrôle de la trajectoire avion (suivi de plan de vol, tenue de cap, d'altitude, suivi axe radioguidage, etc.) ;
- Pilotage : ce système contrôle des mouvements avion autour de son centre de gravité (contrôle assiette, facteur de charge, roulis, etc.) ;
- Asservissement des organes de pilotage (gouvernes et moteurs) ;
- Protection du domaine de vol ;
- Interface avion / équipage : il gère des écrans du cockpit, les alarmes ;
- Maintenance ;
- Gestion des moteurs, gestion du carburant, gestion de l'alimentation électrique, anticollision.

En dehors des systèmes avioniques énumérés ci-dessus, l'avionique militaire se distingue du civil par les systèmes dits de « mission » qui comportent les systèmes suivants : planification des missions, gestion des senseurs tactiques, autoprotection, communication liaison de données et attaque.

Dans ce manuscrit, nous nous intéressons à l'avionique civile. Nous présentons notamment les systèmes de commande de vol et le système d'Orientation Roues Avant (ORA) dans les sections suivantes. En effet, certaines parties de ces systèmes ont été utilisées au cours des expérimentations réalisées dans le cadre de nos travaux.

2.1.1 Systèmes de commande de vol

Les systèmes de commande de vol (CDV), englobant le pilote automatique, ont pour objectif de piloter l'avion autour de son centre de gravité, de contrôler sa trajectoire. L'avion dispose de surfaces mobiles disposées sur la voilure et l'empennage. L'empennage est l'ensemble de plans fixes et mobiles qui assure la stabilité et la gouverne en tangage (profondeur) et en lacet (direction) de l'avion. La **Figure 2.1**, ci-dessous, représente les surfaces dites primaires dans l'A380. Ces surfaces primaires sont composées des ailerons, spoilers, gouvernes de profondeur, gouvernes de direction, plan horizontal réglable. Elles sont destinées à modifier l'attitude de l'avion.

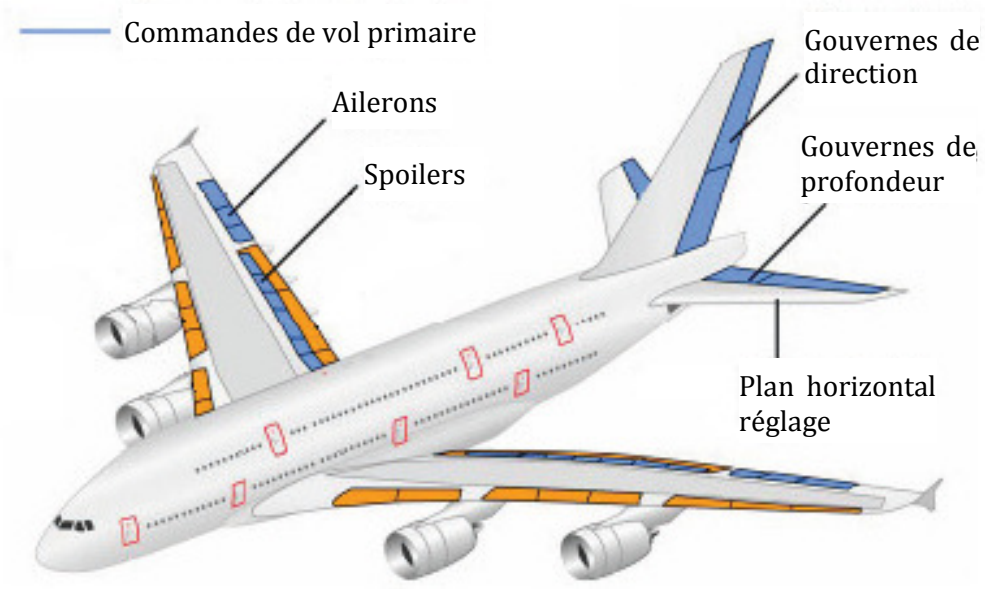


Figure 2.1 : Surfaces de contrôle primaires de l'A380

Le pilote dispose de commandes dans le cockpit qui lui permettent de commander l'attitude et la trajectoire de l'avion. Ces commandes incluent les mini manches (ou sidesticks), le palonnier, le levier des aérofreins, les boutons d'équilibrage latéral (lacet) et profondeur (tangage) et le bandeau du pilote automatique. Dans le cas de l'A380 (**Figure 2.2**), les informations provenant de ces organes de pilotage sont traitées par six calculateurs appelés calculateurs de commande de vol primaires (FCPC – Flight Control Primary Computer).

Les calculateurs calculent l'angle à appliquer aux surfaces primaires afin de répondre à la demande du pilote. Ces calculateurs prennent en compte un certain nombre de paramètres liés à la configuration (tels que la masse et le centrage) de l'avion et à son environnement pour synthétiser les ordres. Ils ont aussi une fonction de protection qui limite les trajectoires de l'avion dans le domaine de vol.

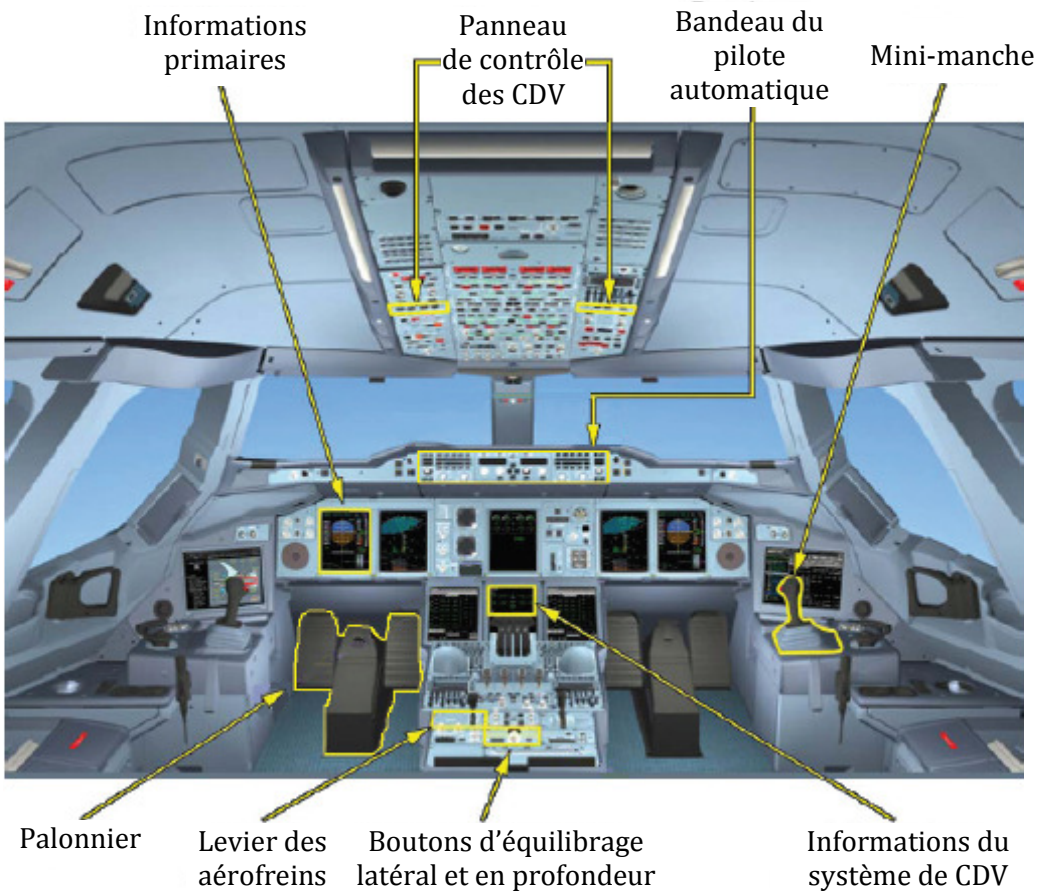


Figure 2.2 : Organes de pilotage primaires de l'A380

2.1.2 Système d'Orientation Roues Avant « ORA »

Le système « ORA » fournit les moyens de contrôle de direction au sol afin de permettre les mouvements latéraux de l'avion. La figure ci-dessous présente un schéma mécanique du train avant d'un avion de type A330-340. Ce système est hébergé dans le calculateur BSCU (Braking and Steering Control Unit) qui gère l'alimentation et les actionneurs hydrauliques en fonction des signaux électriques d'entrée. Les commandes d'orientation proviennent en priorité des volants actionnés par le pilote ou le copilote. Chacun de ces deux opérateurs a un contrôle total sur l'ensemble des mouvements des roues avant. Par ailleurs, les ordres de commandes peuvent aussi provenir (avec un débattement limité en fonction de la vitesse) des pédales de gouverne ou du système de pilote automatique à travers les calculateurs FCPC (Flight Control Primary Computers).

Le BSCU commande les valves hydro-électriques montées sur la jambe de train avant. Ces valves contrôlent les fluides actionnant les vérins de direction. Deux capteurs de position transmettent les informations sur la position des roues au BSCU : un pour les commandes et un

pour le monitoring. Le BSCU surveille aussi le mouvement ordonné par la servovalve (commande électrique). Lorsqu'il détecte un mouvement incohérent, il isole l'hydraulique, désactive le contrôle du mouvement et configure les roues pour le remorquage.

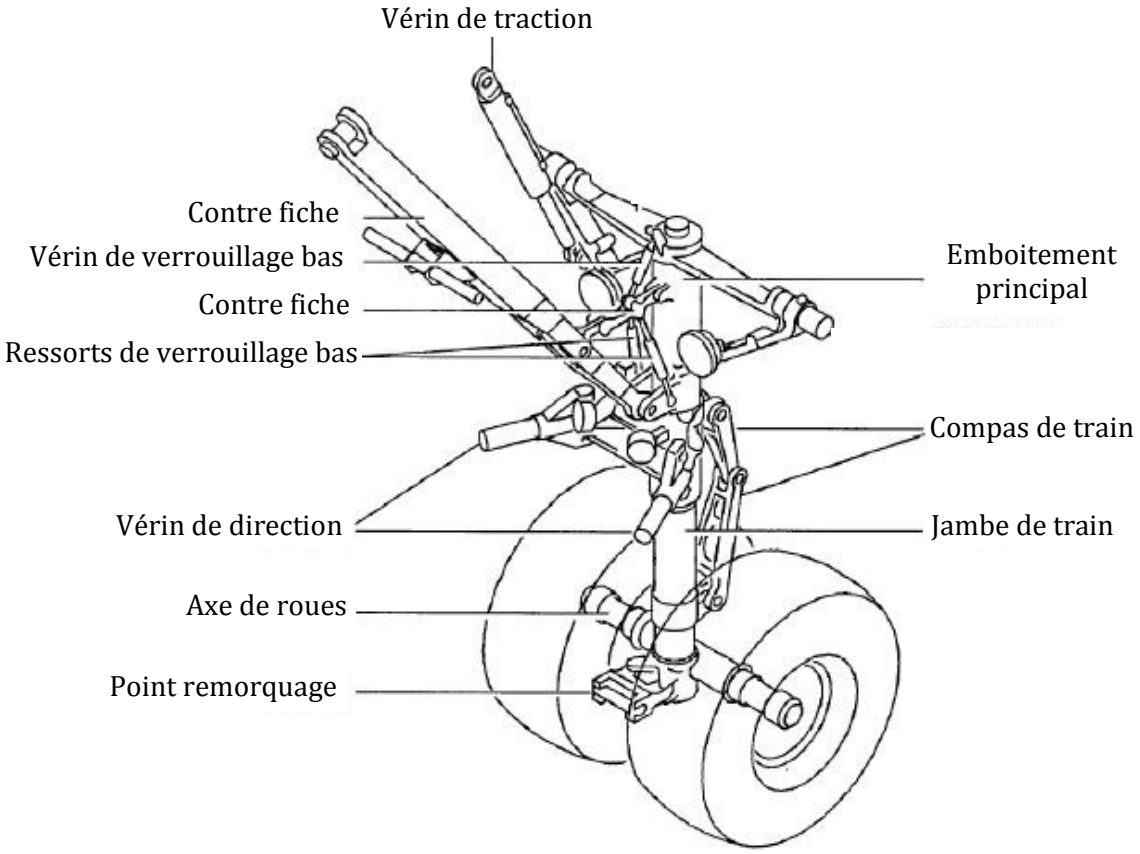


Figure 2.3 : Roues avant d'un avion de type A330-340

Les principales fonctionnalités du BSCU liées au système ORA permettent de :

- Contrôler l'angle de braquage des roues en fonction de l'axe de l'avion lors de ses mouvements au sol.
- Calculer les ordres provenant des volants ou des pédales du palonnier (pilote ou copilote) en fonction de la vitesse de l'avion.
- Rendre possible le remorquage en désactivant l'alimentation hydraulique.

2.2 Développement des systèmes AIRBUS

Les systèmes avioniques développés par AIRBUS peuvent être regroupés en trois grandes catégories de systèmes selon leurs fonctions :

- des systèmes de type contrôle/commande (commandes de vol, train d'atterrissage, gestion des fonctions moteur, etc.) ;
- des systèmes de gestion de l'information des données (calculateur d'alarmes, calculateur de maintenance, etc.) ;
- des systèmes gérant les communications (gestion des données bord/sol, l'avionique modulaire intégrée, etc.).

Dans la suite de cette section, nous exposons les contraintes réglementaires qui doivent être prises en compte au cours du développement des systèmes AIRBUS, puis nous présentons les activités de validation et de vérification du cycle de développement des systèmes et nous insistons sur les différentes étapes de validation et de vérification. Nous nous intéressons par la suite aux deux axes qui ont guidé les travaux de cette thèse : le cycle de validation des spécifications et le processus de vérification au cours des essais au sol des systèmes.

2.2.1 Contraintes réglementaires

Le développement des systèmes avioniques est soumis à des exigences de qualité. Le rôle des autorités de certification est de formuler ces exigences et de s'assurer que les avions construits sont conformes à la règlementation. L'autorité de certification est l'*EASA* (European Aviation Safety Agency) pour l'Europe et la FAA (Federal Aviation Administration) pour les USA. Ainsi, les règles que l'avionneur devra suivre durant toute la phase de développement sont définies. Lorsque les exigences de l'autorité sont respectées, celle-ci délivre un certificat de navigabilité qui permettra à l'avion de voler dans l'espace aérien qu'elle régit.

Le document ARP4754 [SAE96] définit les recommandations sur le processus de développement des systèmes avioniques. AIRBUS se base sur ces recommandations pour définir les règles de développement en concordance avec les exigences des autorités de certification. La directive AIRBUS ABD200 [ABD06] décrit les exigences de développement de ses systèmes.

Ces recommandations s'organisent autour d'un processus d'évaluation de la sûreté de fonctionnement d'un système au cours du développement. Ce processus est basé sur trois principales analyses : la *FHA* (Functionnal Hazard Assessment), la *PSSA* (Preliminary System

Safety Assessment) et la *SSA* (System Safety Assessment). Ces analyses s'intéressent aux *FC* (Failure Conditions) correspondant aux différentes situations redoutées associées aux fonctions réalisées par le système. Ces FC sont classées selon leur gravité en termes de sûreté de fonctionnement.

La norme DO-178B [RTC92], document de référence pour la certification des logiciels, se base sur la classification des FC pour définir les différents niveaux de criticité des systèmes (logiciels) embarqués. Le tableau ci-dessous propose cette classification avec le niveau de criticité associé.

Tableau 2.1 : Classification des défaillances

Gravité	Effet de la défaillance	Objectifs	Niveau
Catastrophique	L'avion ne peut voler de façon sûre, perte de l'avion, de l'équipage ou de passagers	10^{-9}	A
Dangereux	Réduction importante des marges de sécurité ou des fonctions, augmentation importante de la charge de travail pour l'équipage, blessures sévères	10^{-7}	B
Majeur	Réduction significative des marges de sécurité ou des fonctions, augmentation de la charge de travail pour l'équipage, inconfort ou blessures possibles	10^{-5}	C
Mineur	Réduction légère des marges de sécurité ou des fonctions, augmentation de la charge de travail pour l'équipage, inconfort	10^{-3}	D

Par conséquent, les activités de validation et de vérification des systèmes requises par la règlementation dépendent du niveau de criticité (DO-178B). En effet, Cette norme définit les objectifs et les activités du processus de V&V des systèmes (logiciels) embarqués. En considérant les deux systèmes présentés dans la section précédente (section 2.1), les systèmes de commande vol sont de niveau A et le système d'orientation des roues avant est de niveau B.

2.2.2 Objectifs du processus de V&V selon la DO-178B

Le but du processus de validation et de vérification des logiciels est de détecter et de signaler les erreurs qui ont pu être introduites au cours du développement. La correction des erreurs est une activité de processus de développement des logiciels. D'une manière générale, les objectifs du processus de V&V sont de vérifier que :

- Les exigences du système allouées au logiciel ont été prises en compte et développées dans les exigences « haut niveau » du logiciel.

- Les exigences « haut niveau » du logiciel ont été prises en compte et développées dans l'architecture et les exigences « bas niveau » du logiciel. Il existe des processus où un ou plusieurs niveaux d'exigences du logiciel peuvent être développés entre les exigences « haut niveau » et les exigences « bas niveau ». Dans ce cas, les niveaux successifs des exigences sont développés tels que successivement chaque niveau inferieur satisfait les exigences du niveau supérieur. Cet objectif n'est pas considéré lorsque le code est généré directement à partir des exigences de « haut niveau ».
- L'architecture et les exigences « bas niveau » ont été prises en compte et développées dans le code du logiciel.
- Le code exécutable satisfait les exigences du logiciel.

Les moyens techniques déployés à travers les activités de V&V pour l'atteinte de ces objectifs sont définis en fonction du niveau de criticité du logiciel.

2.2.3 Activités du processus de V&V selon la DO-178B

Les objectifs du processus de V&V sont réalisés à travers une combinaison de revues, d'analyses, de définition et d'exécution de tests et de procédures. Les revues et les analyses permettent l'évaluation, la précision, la complétude et la vérifiabilité des exigences, de l'architecture et du code source du logiciel. La définition des tests permettent, en plus, une évaluation de la cohérence et la complétude interne des exigences. L'exécution des procédures de test permet une démonstration de la conformité avec les exigences.

Le processus de V&V permet de réaliser deux types de traçabilité entre l'implantation du logiciel et la vérification des exigences :

- la traçabilité entre les exigences du logiciel et les tests est réalisée à l'aide d'une analyse de couverture des exigences afin d'assurer la prise en compte de toutes les exigences du logiciel et seulement ces exigences ;
- la traçabilité entre la structure du code et les tests est réalisée à l'aide d'une analyse de couverture structurelle afin d'évaluer les critères de couverture des tests et d'identifier d'éventuel code « mort ».

2.3 V&V et processus de développement AIRBUS

Le cycle de développement complet des systèmes avioniques d'AIRBUS est représenté sur la **Figure 2.4**. Trois principaux niveaux peuvent être identifiés sur ce cycle : le niveau Avion, le niveau Système et le niveau Equipement. Les différentes activités de validation et de vérification menées au cours des principales étapes de chaque niveau y sont également associées. Les activités des niveaux Avion et Système sont essentiellement réalisées par AIRBUS (le maître d'ouvrage) ; celles liées au niveau Equipement sont déléguées aux équipementiers (les maîtres d'œuvres). Dans la suite de cette section, nous présentons, dans un premier temps, les moyens de validation et de vérification, puis nous proposons une brève présentation, par niveau, du cycle de développement des systèmes AIRBUS.

2.3.1 Moyens de V&V

Les principaux moyens utilisés durant les activités de vérification et validation menées au cours du développement d'un système sont les suivants :

- **Le simulateur au niveau système** : le système considéré est validé dans un environnement avion simulé. Ces simulateurs sont appelés simulateurs de bureau. Ils permettent au concepteur du système de valider le bon fonctionnement de sa spécification logicielle sur des machines temps réel dotées d'un pupitre de commandes représentatif des actions pilote (mini–manche, manette des gaz, palonnier) ;
- **Le simulateur au niveau avion** : un ensemble de systèmes est validé dans un environnement avion simulé. Ces moyens de simulation sont similaires aux moyens utilisés pour la validation au niveau système, à ceci près qu'on cherche à valider le bon fonctionnement de plusieurs systèmes entre eux ;
- **Les bancs d'essais système** : ils permettent l'exécution des tests pour la vérification de la conformité d'un système dont on dispose des éléments matériels à sa spécification ;
- **Les bancs d'essais multi-systèmes** : ce moyen permet la vérification, à l'aide de tests, du bon fonctionnement de plusieurs systèmes. Il est utilisé lorsqu'on dispose des équipements réels constituant les différents systèmes. L'objectif est alors de démontrer que les produits constituant les systèmes répondent bien à leurs spécifications ;
- **Le banc général** : Il est utilisé lorsqu'on dispose des équipements réels et d'une bonne représentation de l'environnement avion. Ce moyen est mis en œuvre au cours des derniers essais réalisés avant les essais au sol ou en vol qui eux, sont effectués sur le produit final, l'avion.

2.3.2 Niveau Avion

Ce niveau correspond aux phases initiale et finale du développement d'un avion. Il est essentiellement constitué des trois parties décrites ci-dessous.

La première partie consiste à définir les exigences avion et de tous les systèmes qui le composent. Ces exigences (Haut Niveau – Client) sont initialement décrites dans le TLRD (Top Level Requirements Document). Les exigences contenues dans le TLRD sont par la suite dérivées dans les documents FRD (Functional Requirements Document) et FDD (Functional Description Document). Les prototypes des différents systèmes avioniques sont réalisés. Ces prototypes sont généralement décrits sans le respect de certaines normes de qualité (les règles de codage, typage incomplet, options de génération de code non strictement appliquées, etc.). Ils servent à tester la viabilité de ces systèmes face au simulateur. Les concepts et tests choisis à l'issue de cette étape seront utilisés au cours du développement de la partie logicielle des systèmes avioniques. A partir du prototype est écrite une documentation (TLSRD - Top Level System Requirements Document) associée à chaque système logiciel.

La seconde partie du niveau Avion correspond à l'intégration virtuelle de l'ensemble des systèmes avioniques. Elle doit être effectuée après la validation de chaque système et a pour objectif de valider les spécifications et la conception de ces systèmes au niveau avion. Le moyen de validation est le simulateur avion. Un simulateur avion permet de valider le bon fonctionnement de plusieurs systèmes dans un environnement avion simulé.

La dernière partie est composée des étapes finales du développement de l'avion avant son entrée en service.

- Intégration niveau avion : elle correspond à l'intégration de l'ensemble des systèmes avioniques. A cette étape, on dispose des équipements réels et d'une bonne représentation de l'environnement avion. Les activités de V&V sont réalisées à l'aide du banc général. Les essais au sol sont effectués sur le produit final, l'avion. Ils consistent à vérifier le bon fonctionnement des systèmes installés à bord ainsi que leur interconnexion. Cette vérification est réalisée sur la chaîne d'assemblage finale des avions (FAL – Final Assembly Line). Elle est essentiellement basée sur l'exécution des tests fonctionnels. Nous nous intéresserons au processus de réalisation des essais au sol dans le **chapitre 4**.
- Route proving : Cette étape consiste à mettre l'avion dans les conditions opérationnelles le plus longtemps possible avant la livraison. Les essais en vol sont le moyen de validation utilisé à ce niveau.

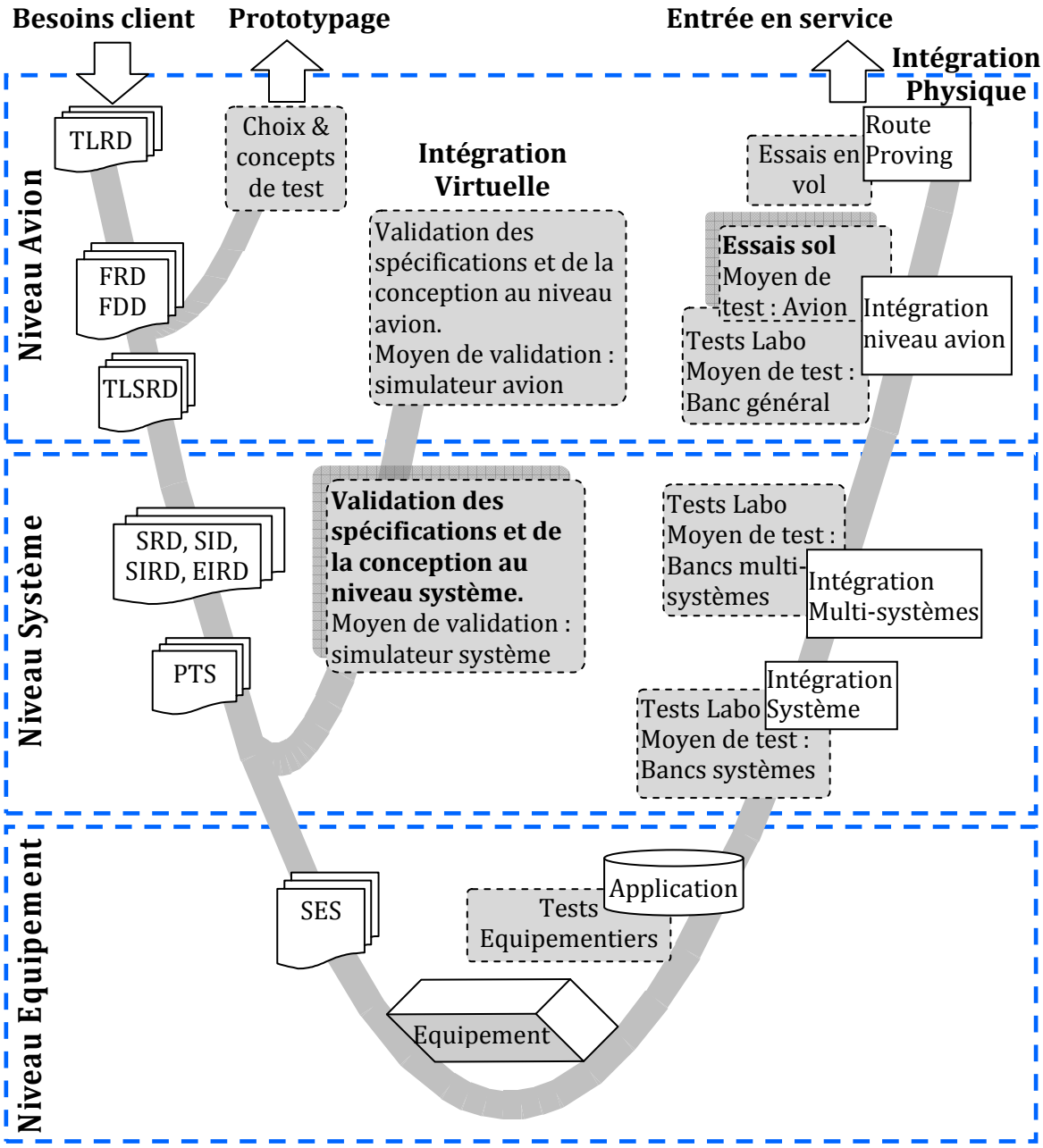


Figure 2.4 : Activités de V&V et le cycle de développement d'un système AIRBUS

2.3.3 Niveau Système

Le niveau système a pour objectif de développer le système selon les exigences du niveau avion. Ce niveau est composé principalement de deux parties : la phase conception et les différentes phases d'intégration système.

Conception

Le processus de conception est essentiellement constitué d'une part, des étapes de définition et de validation des exigences du système ; d'autre part, de la définition et choix de l'architecture conforme aux exigences du système. Nous nous intéressons à la définition et à la validation des exigences fonctionnelles du système logiciel.

Les exigences fonctionnelles sont définies dans le SRD (System Requirements Document). Le SRD est le document de référence « haut niveau » pour la conception d'un système. Il est conçu pour être le seul point de référence des concepteurs au cours de la phase de conception du système. Le SRD identifie toutes les exigences auxquelles le système doit répondre pour satisfaire les spécifications avion qui sont entre autres : les fonctionnalités du système, les performances à atteindre, les interfaces avec d'autres systèmes, les exigences opérationnelles, les exigences de certification, les exigences de sûreté de fonctionnement, etc. Il est considéré comme document de référence dans l'élaboration du SID (System Interface Document), SIRD (System Installation Requirements Document) et EIRD (Equipment Installation Requirements Document).

Le SRD sert de base dans la définition du PTS (Purchaser Technical Specification). Le PTS est une spécification technique contractuelle entre AIRBUS et les équipementiers. Il prend en compte la définition, le cadre d'échange et le contrat pour la réalisation d'un équipement (nouveau ou existant).

L'étape de validation des exigences fonctionnelles au cours du développement d'un système consiste à montrer que les exigences définies sont suffisamment complètes et correctes. Le simulateur système est le moyen utilisé pour cette phase de validation des spécifications. Dans le contexte de nos travaux les exigences sont spécifiées à l'aide de l'environnement SCADE. Nous nous intéresserons au cycle de validation des systèmes dans le **chapitre 3**.

L'environnement SCADE a été défini pour assister le développement des systèmes réactifs critiques. Il est composé de plusieurs fonctionnalités : édition graphique [Edi09], simulation [Sim09], génération de code C ou ada et model-checking [Ver09]. Nous présentons brièvement le langage SCADE dans la suite.

Langage SCADE

SCADE est une notation graphique formelle dérivée du langage Lustre [Hal91a, Hal91b, Hal02] basée sur le formalisme « flot de données synchrone ». Lustre est un langage de type déclaratif ou fonctionnel.

Le formalisme flot de données synchrone consiste à introduire une dimension temporelle dans le modèle flot de données. Un flot (entité de base) comprend deux parties : une séquence des valeurs d'un type donné et une horloge représentant une séquence d'instants (sur l'échelle temporelle discrète). L'hypothèse de synchronisme stipule que le système réagit à un événement instantanément. Cela sous-entend que le temps de calcul des algorithmes est nul. En pratique, il faut que le système réagisse à un événement externe avant l'acquisition de tout nouvel événement. Cette hypothèse est réaliste dès lors que l'on démontre que les algorithmes respectent leurs échéances.

Les systèmes spécifiés en SCADE ont un comportement cyclique. En effet SCADE propose une horloge qui est une variable booléenne. Une description SCADE possède une horloge de base qui est vraie à chaque cycle. Elle est structurée en un réseau de nœuds ou planches et représente les relations entre les entrées et les sorties d'un système. Ces relations sont exprimées au moyen d'un diagramme d'opérateurs. Ce diagramme est constitué d'un ensemble d'équations permettant de définir les variables (effectives ou locales ou globales) à l'aide d'opérateurs (arithmétique, booléen, conditionnel ou composé) et de constantes. Un nœud et une variable SCADE n'ont pas la même horloge. L'horloge d'une variable est la fréquence à laquelle cette variable est définie. L'horloge effective d'un nœud est la fréquence d'appel de ce nœud.

Dans ce contexte, une expression X représente un flot. Un flot est une séquence infinie $(x_0, x_1, \ldots, x_n, \ldots)$ de valeurs où x_n est la valeur de x à l'instant n (au $n^{ième}$ cycle) d'exécution. Une équation est un invariant temporel ; par exemple $x = y + z$ définit le flot $(x_0, x_1, \ldots) = (y_0 + z_0, y_1 + z_1, \ldots)$. La **Figure 2.5** ci-dessous représente cette équation décrite à l'aide de l'éditeur SCADE.

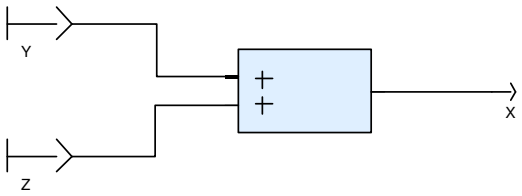


Figure 2.5 : Représentation graphique d'une équation d'addition

Intégration

Il existe deux niveaux d'intégration au niveau « système » du développement des systèmes AIRBUS :

– l'intégration système : elle correspond à l'intégration des différents éléments d'un système. L'activité de vérification menée consiste à évaluer la conformité du système ainsi

assemblé par rapport à sa spécification. Le banc de test système est utilisé pour cette phase de vérification ;

– l'intégration multi-systèmes : à cette étape du cycle de développement, l'intégration des équipements réels constituant les différents systèmes est réalisée. Elle permet de vérifier le bon fonctionnement de plusieurs systèmes. Le banc de test multi-systèmes est utilisé pour cette dernière phase de vérification au niveau système.

2.3.4 Niveau Equipement

Les activités menées à ce niveau sont de la responsabilité des équipementiers. L'ensemble des informations permettant la réalisation des équipements implémentant le système sont fournis aux équipementiers par le biais du PTS (Purchaser Technical Specification). Ceux-ci proposent le SES (Supplier Equipement Specification) en réponse au PTS. Le SES correspond en quelque sorte à l'offre développée par l'équipementier en prenant en compte l'ensemble des exigences définies dans le PTS. Les équipements (matériel et logiciel) sont par la suite développés pour aboutir aux applications nécessaires au fonctionnement du système. Les activités de vérification sont définies et réalisées par les équipementiers. Toutefois, cette vérification peut être incomplète, faute de jeux d'essais suffisamment complets. AIRBUS peut donc être amené à effectuer pour certains équipements une partie de la vérification.

2.4 Motivations

Les activités de validation et de vérification, menées au cours du développement des systèmes AIRBUS, mêlent revues, activités de traçabilité, analyse statique de code et test. L'activité de test est réalisée à tous les niveaux (Avion, Système, Equipement) du processus de développement (**Figure 2.4**). A chaque phase de test correspond une problématique spécifique liée aux objectifs de validation ou de vérification définis. Nous nous intéressons aux problématiques liées à la génération de test de validation de la spécification détaillée des systèmes et à la vérification des systèmes au cours des essais au sol sur la chaîne d'assemblage finale de l'avion.

Validation de la spécification détaillée des systèmes

La spécification détaillée formelle des systèmes AIRBUS est écrite en utilisant l'environnement SCADE. Certains comportements physiques peuvent être analysés sur des modèles SIMULINK. AIRBUS utilise le générateur de code qualifié SCADE pour les systèmes décrits en SCADE. Par conséquent, les tests unitaires sur le code généré ne sont pas effectués.

Par ailleurs, quel que soit le formalisme utilisé et le type de système traité, seuls des tests fonctionnels sont réalisés sur l'ensemble des activités de test réalisées au cours de cette étape de validation. Les tests sont réalisés sur le simulateur système de bureau dit « OCASIME » ; des activités de validation peuvent être menées à l'aide d'outils disponibles au niveau de l'environnement SCADE, à savoir :

- tests à l'aide du simulateur SCADE,
- preuve de propriétés à l'aide de SCADE Verifier.

La réalisation des activités de V&V dynamique, au cours de la validation, nécessite la définition de vecteurs ou données de tests pertinents puisqu'il s'agit d'effectuer des campagnes de tests dans des environnements de simulation. Or, la définition de jeux de d'essai (les données de test et l'oracle) permettant de valider une spécification détaillée formelle nécessite un effort important. En effet, cette définition est confrontée aux problèmes suivants :

- comment être certain que toutes les fonctions ont été correctement testées ?
- comment s'assurer que le système ne se retrouve pas dans certaines configurations redoutées qui pourraient nuire à la sûreté de fonctionnement de l'avion ?

Afin de répondre à ces questions, AIRBUS propose de mener des activités de vérification et validation guidées par les fonctions du système et de définir explicitement des critères d'arrêt, c'est à dire des critères capables de s'assurer qu'un nombre suffisant de tests a été réalisé afin de garantir le niveau recherché de sûreté de fonctionnement du système.

Dans ce contexte, sera très bénéfique toute méthode ou outil permettant d'aider, au cours du processus de définition des tests et aux activités d'analyse de couverture, à l'évaluation de la couverture des :

- spécifications détaillées formelles vis-à-vis des exigences liées aux fonctions du système ;
- jeux d'essai définis dans le LTR vis-à-vis des spécifications détaillées formelles.

Nous proposons, dans le **chapitre 3**, une méthodologie d'aide à la validation des systèmes basée sur une approche d'analyse de testabilité des spécifications flot de données.

Vérification des systèmes sur la FAL (Final Assembly Line)

Le gain de temps est un facteur très important au cours de la vérification des systèmes sur la chaîne d'assemblage finale. En effet, toute activité raccourcie permet de réduire le temps de livraison de l'avion, ce qui est un facteur important de réduction des coûts. C'est ainsi qu'AIRBUS cherche sans cesse à améliorer et à automatiser ses outils de test sur la FAL des différents programmes (types d'avion).

Cependant, la conception des essais et une partie de l'activité de diagnostic est basée sur l'expertise humaine (compréhension, expériences, …). Toutefois, la méthodologie adoptée pour la conception des essais consiste à :

- limiter le nombre de configurations au cours de la vérification des systèmes installés à bord. L'idée est de pouvoir exploiter au maximum une configuration, c'est à dire, exécuter un maximum d'essais avec une configuration donnée. Par exemple, une configuration peut être requise pour tester le système hydraulique de l'avion ; une pour les commandes de vol ; et une autre pour les bus d'acquisitions;
- rationaliser les essais au cours de leur définition. Cette phase passe par une bonne répartition et un bon ordonnancement des tests à exécuter.

Quant à la partie non automatisée de la phase diagnostic, elle dépend du technicien de test assurant les activités de vérification d'un système. Selon le vécu de la personne, l'approche de diagnostic adoptée face à une défaillance du système peut être différente. De ce fait, l'efficacité d'une approche de diagnostic ne dépend que de l'expérience du testeur.

Dans ce contexte, il est important de définir des concepts permettant, au cours des activités de définitions de tests et de diagnostic de fautes, de :

- aider à la définition d'essais pertinents ;
- aider à la détection et à l'identification de tests redondants ;
- guider le processus d'identification des composants (ou ressources) défaillants du système ;
- aider à l'amélioration de la précision du diagnostic d'un système.

Nous proposons une méthodologie basée sur certains concepts d'analyse de testabilité dans l'objectif de fournir, pour un système, des informations sur la qualité des essais depuis leur conception jusqu'au diagnostic des ressources tout au long du processus de vérification sur la FAL dans le **chapitre 4**.

Chapitre 3
Aide à la génération de tests de validation

La validation d'un système AIRBUS, au cours de sa conception, est le processus qui permet d'assurer que les exigences spécifiées sont suffisamment correctes et complètes afin qu'il réponde aux exigences de navigabilité requises [ABD06]. Les techniques de V&V menées au cours de cette phase, notamment la technique dynamique (test), ont pour objectif de montrer que l'ensemble des fonctions du système sont correctement spécifiées et de fournir des assurances sur le comportement du système en termes de sûreté de fonctionnement. De ce fait, les jeux d'essai ou tests fonctionnels définis pour atteindre ces objectifs doivent être les plus pertinents possibles.

Or, la définition des tests fonctionnels pour la validation se fait de façon manuelle à partir des exigences fonctionnelles textuelles du système. Ceci conduit à un processus de test coûteux en termes de ressources. En effet, le manque d'outillage pour cette phase fait que certains tests sont redondants ou oubliés dans les phases initiales de définition des tests, malgré l'existence de guides méthodologiques. De plus, les critères d'arrêt du test sont empiriques : couverture fonctionnelle des tests vis-à-vis des exigences du système difficilement évaluable, couverture structurelle de la spécification non établie. Malgré tout, les tests définis sont complets à l'issue des différentes campagnes de test réalisées tout au long du développement mais au prix d'un effort conséquent.

Dans ce contexte, nous proposons, dans ce chapitre, une méthode d'analyse de testabilité des spécifications flot de données permettant d'aider la phase de définition des jeux d'essai pour la validation des spécifications détaillées des systèmes. Nous présentons d'abord le cycle de validation des spécifications d'un système ; puis nous introduisons l'analyse de testabilité proposée et nous décrivons les évolutions nécessaires à l'application de cette méthode d'analyse de testabilité dans le contexte AIRBUS. La méthodologie d'évaluation de la testabilité des systèmes est par la suite décrite. Enfin, nous exposons les résultats des différentes expérimentations.

3.1 Cycle de validation des spécifications d'un système

Le processus de spécification et de validation des spécifications des systèmes AIRBUS est présenté dans la **Figure 3.1** ci-dessous. Il est composé des étapes de définition du SRD (System Requirements Document), DFS (Detailed Functional Specification), LTR (Lab Test Requirements), de formalisation de la spécification et de validation de la spécification du système. La gestion de la traçabilité entre les documents issus des différentes étapes est assurée par l'outil DOORS [Tel].

Ce processus est essentiellement basé sur une hiérarchie documentaire contenant un ensemble d'exigences textuelles pour la partie descendante du cycle de développement. Ces exigences ne deviennent formelles qu'au moment de leur transcription dans un modèle formel.

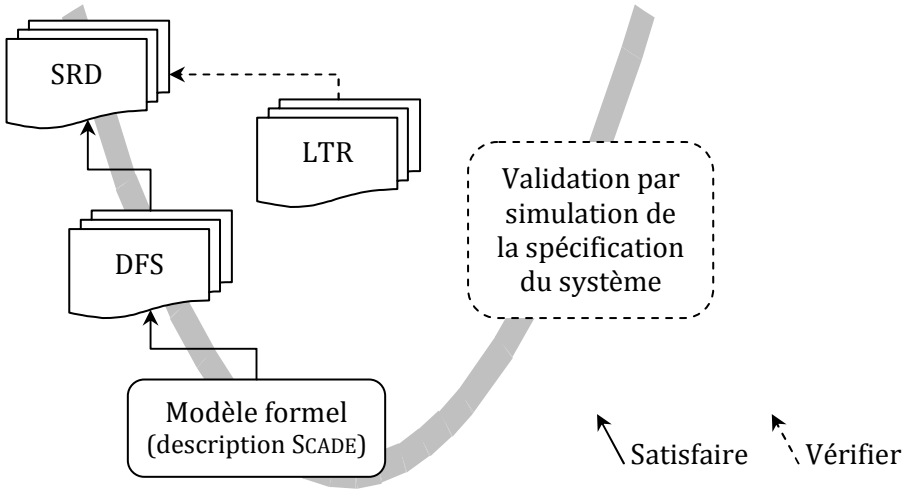


Figure 3.1 : Cycle de spécification et de validation

3.1.1 SRD (System Requirements Document)

Pour rappel, le SRD est un document qui définit les exigences implantées dans les fonctions du système. Conformément aux directives ABD200 [ABD06] (*Module 2 – Section 3*), la description des exigences doit être guidée par les règles suivantes :

- *la définition des exigences fonctionnelles système et l'affectation d'un identificateur à chaque exigence*. Cette définition doit être non ambiguë et vérifiable. Les identificateurs permettent de garder la traçabilité des exigences et de maintenir le lien entre les activités de validation et de vérification ;
- *l'identification de la source de chaque exigence fonctionnelle (document d'exigences avion, standards industriels, etc.)*. Cette information est utilisée comme une partie de la démonstration de la complétude de l'exigence. Elle est aussi utilisée pour identifier les liens entre différentes étapes du cycle de développement pour faciliter la traçabilité et l'analyse de l'impact d'une modification ;
- *la justification de chaque exigence dont les origines ne peuvent pas être déduites directement des exigences avion ou de l'exigence elle-même*. Ceci permet d'évaluer si une telle exigence est nécessaire et pertinente pour la validation. Cela permet également d'accroître la compréhension de l'objectif de l'exigence et d'éviter une mauvaise interprétation.

3.1.2 LTR (Lab Test Request)

Les LTR sont des documents contenant les jeux d'essai ou de test définis pour la validation des spécifications des fonctions décrites dans les DFS (Detailed Functional Specification) à partir des exigences du SRD à l'aide du simulateur de bureau ou sur les bancs d'essais. Un LTR est associé à chaque DFS. Cette étape est réalisée conformément aux directives « *ABD200* » qui recommandent « l'*identification des activités nécessaires pour montrer que chaque exigence est correcte* ». Les LTR contribuent à la vérification de la conformité de la spécification détaillée par rapport aux exigences système.

3.1.3 DFS (ou CDF – Cahier De Fonction)

La spécification des systèmes est une phase très critique du processus de développement. En effet, un système mal spécifié a toutes les chances de ne pas être conforme à ses exigences.

Le CDF est un ensemble de documents intermédiaires au niveau système entre les SRD des systèmes et les spécifications détaillées des calculateurs qui les hébergeront. Il contient la spécification de toutes les exigences fonctionnelles définies dans le SRD. Par ailleurs, le contenu du CDF doit être cohérent avec celui d'autres documents d'exigences décrits à partir du SRD comme le SID (System Installation Document).

La transformation correcte des exigences fonctionnelles en spécification détaillée doit être contrôlée. Ce contrôle est réalisé conformément aux directives « ABD200 » qui recommandent : *« l'identification des activités nécessaires pour montrer que chaque exigence est correcte. Une exigence doit être validée et plus d'une activité de validation peut être nécessaire pour s'assurer que l'exigence est correcte* ». Elle permet de démontrer que la spécification détaillée décrit un ensemble de fonctions homogènes répondant aux exigences fonctionnelles (SRD) ainsi qu'aux exigences liées à l'architecture du système. Cette validation est effectuée à l'aide des jeux d'essai définis dans le LTR.

Description du modèle formel

Les spécifications détaillées du CDF, pour être validées, sont décrites dans un modèle formel. La validation des exigences du logiciel des systèmes est effectuée à partir de ce modèle décrit à l'aide d'un environnement tel que SCADE [Est05a, Est05b] ou Simulink [Mat05]. Nous nous intéressons aux modèles formels SCADE dans ce manuscrit. Les systèmes implantés dans les calculateurs sont spécifiés dans des planches SCADE. Les spécifications détaillées des systèmes de commande de vol implantés dans un calculateur primaire de l'A380 comportent environ 5000 planches. Cet ensemble de planches est découpé en fonction des systèmes, sous systèmes et fonctions. Ce découpage est géré à l'aide des notions de chapitre, de livre et de feuille dans l'organisation des planches SCADE. Les activités de validation sont réalisées en tenant compte de ce découpage.

Validation des spécifications détaillées

Le modèle formel implémenté à partir des spécifications détaillées est soumis à deux phases (contrôle et simulation) de vérification au cours de la validation.

Contrôle

La phase de contrôle consiste en l'application de certaines techniques de vérification statique :

- Dans un premier temps, les diagrammes d'opérateurs SCADE sont soumis à un contrôle syntaxique et sémantique au niveau de chaque nœud SCADE défini. Une relecture croisée

de ces schémas d'opérateurs est réalisée entre les différentes équipes afin de compléter cette étape de contrôle ;

- Ensuite, la spécification formelle est soumise à une analyse des aspects flot de données et ordonnancement (annexe, section B.2).

Simulation

Avant d'être fournies aux équipementiers, les spécifications détaillées sont validées sur les simulateurs de bureau. L'outil DAMAS (Data Manager for specifications) est utilisé pour la génération de code simulé fonctionnellement conforme aux systèmes embarqués. Ce code est utilisé par les simulateurs de bureau pour simuler les systèmes tel qu'OCASIME (Outil de Conception Assistée par Simulation Multi-Equipements). OCASIME est un simulateur « système » qui simule les spécifications détaillées (formalisées) dans un environnement avion. Ce simulateur propose un mode interactif et dispose de commandes réelles tel que le mini-manche pour représenter l'environnement. Les jeux d'essai définis dans les LTR (Lab Test Request) pour la validation des exigences fonctionnelles sont exécutés sur ces simulateurs. OCASIME est également utilisé pour l'exécution des tests de non-régression sur la partie modifiée du modèle à l'issue de l'évolution des spécifications détaillées. L'outil DOORS gère la traçabilité des modifications de la spécification détaillée et les informations sur leur validation doit être conservée.

3.2 Analyse de testabilité des spécifications flot de données

La méthode d'analyse de testabilité, dont il est question dans cette section, a été initialement proposée pour les systèmes matériels [Rob79, Dam85]. Le Traon [LeT97] et Do [Do06] ont étudié une adaptation des principes de cette méthode d'analyse, implantée par la technologie SATAN (System's Automatic Testability ANalysis), aux spécifications flot de données des logiciels. Cette méthode d'analyse de testabilité est basée sur un modèle de testabilité construit à partir de la spécification formelle flot de données d'un système (section 3.2.1). Le processus d'analyse est constitué par : la détermination des chemins d'information ou écoulements contenus dans le système à partir du modèle de testabilité (section 3.2.2) ; une stratégie de test est par la suite utilisée pour la sélection des écoulements pertinents du système (section 3.2.3) ; les mesures de testabilité sont calculées en se basant sur ces écoulements afin d'aider à l'identification des parties du système susceptibles d'être difficilement testables (section 3.2.4).

3.2.1 Modèle de testabilité

Le modèle de testabilité est fondé sur le transfert de l'information dans la spécification flot de données d'un système. Ce modèle est un graphe orienté appelé *Modèle de Transfert d'Information* (MTI) du système. La présentation du modèle de testabilité d'un système nécessite l'introduction et la définition de certaines notions élémentaires.

Définitions des notions

Définition 1 ; Modes de transfert d'information

Soit $G=(P,T,A)$ un graphe biparti orienté où :

- $P=(I\cup M\cup O)$ correspond à l'ensemble de places avec I l'ensemble non vide de sources, M l'ensemble non vide de modules et O l'ensemble non vide de puits du graphe ;
- T correspond à l'ensemble non vide de transitions ;
- A correspond à un ensemble non vide d'arcs tel que les extrémités initiale et terminale de chaque arc appartiennent aux deux ensembles différents P et T. Les éléments des ensembles P et T représentent les sommets du graphe.

Un mode de transfert d'information définit les règles d'acheminement de l'information d'une place à une autre à travers les transitions. Trois modes de transfert d'information sont utilisés dans cette thèse (**Figure 3.2**) :

- *attribution* ; seule l'information provenant d'une des places sources est nécessaire à l'activation de la place destinataire
- *sélection* ; l'information provenant de la place source est nécessaire à l'activation des places destinataires
- *jonction* ; l'information provenant de l'ensemble des places sources est nécessaire à l'activation de la place destinataire.

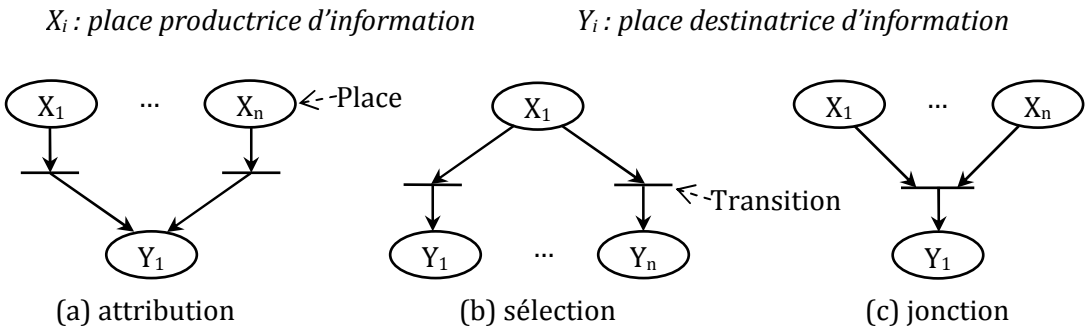


Figure 3.2 : Modes de transfert d'information

Définition 2 : MTIe – Modèle de Transfert d'Information élémentaire

Soient : $\Gamma_G^+(x)$ ensemble des successeurs du sommet x d'un graphe G
$\Gamma_G^-(x)$ ensemble des prédécesseurs du sommet x d'un graphe G
$\Gamma_G(x)$ ensemble des voisins du sommet x d'un graphe G

Un MTIe est un graphe biparti orienté $G=(P,T,A)$ tel que :

$$\forall t \in T \quad \Gamma_G^+(t) \neq \varnothing,\ \Gamma_G^-(t) \neq \varnothing; \quad (1)$$

$$\forall i \in I \quad \Gamma_G^-(i) = \varnothing\ ; \quad (2)$$

$$\forall o \in O \quad \Gamma_G^+(o) = \varnothing\ ; \quad (3)$$

$$\forall m \in M \quad \Gamma_G^+(m) \neq \varnothing,\ \Gamma_G^-(m) \neq \varnothing; \quad (4)$$

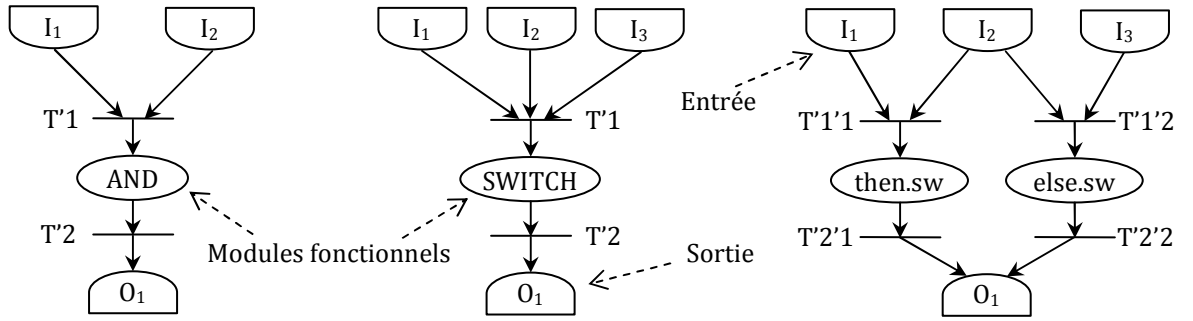
$$\forall t \in T \quad \Gamma_G(t) \subset Z\ ; \quad (5)$$

$$\forall p \in P \quad \Gamma_G(p) \subset T\ ; \quad (6)$$

$$\forall t \in T \quad \Gamma_G^+(t) \cap \Gamma_G^-(t) = \varnothing\ ; \quad (7)$$

$$\forall i \in I \quad \exists o \in O \ /\ i \overset{*}{\Rightarrow} o \quad (8)$$

Le MTIe représente le modèle de testabilité élémentaire d'un opérateur utilisé dans la spécification flot de données d'un système. Il représente le modèle transformationnel de l'opérateur. En effet, pour un opérateur donné, la modélisation consiste à représenter les flots de données concourant à la définition des variables de sortie des opérateurs. Au cours de cette représentation, les modules fonctionnels correspondent aux opérations effectuées pour le calcul les sorties (puits) à partir de l'ensemble ou d'une partie des flots de données en entrée (sources) de l'opérateur. Le choix de la modélisation du MTIe est lié au critère de couverture de la structure du modèle visé au cours de la validation. La **Figure 3.3** représente le MTIe simple de l'opérateur AND logique et les MTIe simple et détaillé de l'opérateur d'aiguillage SWITCH. En effet, le MTIe simple de l'opérateur SWITCH de la **Figure 3.3 (b)** est utilisé lorsque le critère de sélection est la couverture des branches. Le MTIe détaillé de la **Figure 3.3 (c)** du même opérateur est choisi lorsque le critère de couverture est celui des branches et des décisions (la commande de décision est « O_2 »). De ce fait, la modélisation du MTI d'un opérateur évolue en fonction du critère de couverture « structurel » visé par le test de validation (fonctionnel) du système.



(a) AND logique (b) Modèle simple d'un SWITCH (c) Modèle détaillé d'un SWITCH

Figure 3.3 : Représentation graphique des MTI élémentaires

MTI – Modèle de Transfert d'Information du système

Les spécifications flot de données des systèmes décrites en SCADE représentent les relations entre les entrées et les sorties à l'aide des opérateurs prédéfinis dans l'environnement SCADE ou définis dans les bibliothèques par les concepteurs du système. La construction du modèle de testabilité d'un système consiste en la composition (essentiellement par opération de concaténation) des MTIe associés aux opérateurs utilisés dans le système.

Le MTI d'un système, résultat de la composition de MTIe, est un graphe biparti orienté défini par un ensemble de places, un ensemble de transitions et un ensemble d'arcs où :

- les sources représentent sont les entrées du système (entrées effectives, constantes, points d'injection de données de test, etc.) ;
- les puits représentent les sorties du système (sorties effectives, point d'observation, etc.) ;
- les modules correspondent aux modules fonctionnels des opérateurs du système ;
- les transitions caractérisent les modes de transfert d'information entre les places.
- les arcs connectent les places et les transitions. Ils représentent le support d'information indiquant le sens du transfert de l'information entre les places et les transitions.

Dans la représentation graphique du MTI d'un système, les sources et les puits sont représentés par les demi-cercles, les modules sont représentés par les cercles et les transitions sont représentées par les barres. L'extrait d'une planche SCADE de la spécification des modes latéraux du pilote automatique d'un avion AIRBUS est utilisé pour l'illustration de la notion de modèle de testabilité d'un système (**Figure 3.4**).

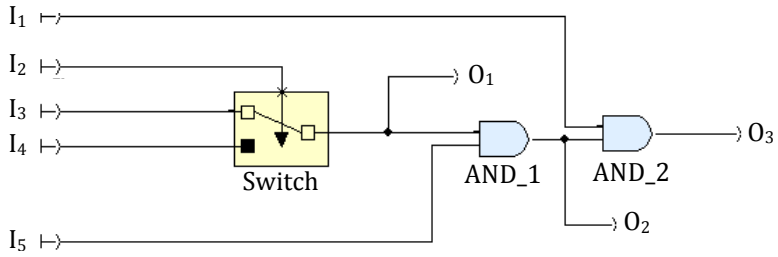


Figure 3.4 : Extrait d'une planche SCADE d'un avion AIRBUS

Cette description SCADE contient un opérateur d'aiguillage (Switch) et deux opérateurs AND logique (AND_1 et AND_2). Le modèle de testabilité associé à ce diagramme d'opérateurs décrit en SCADE est représenté ci-dessous (**Figure 3.5**).

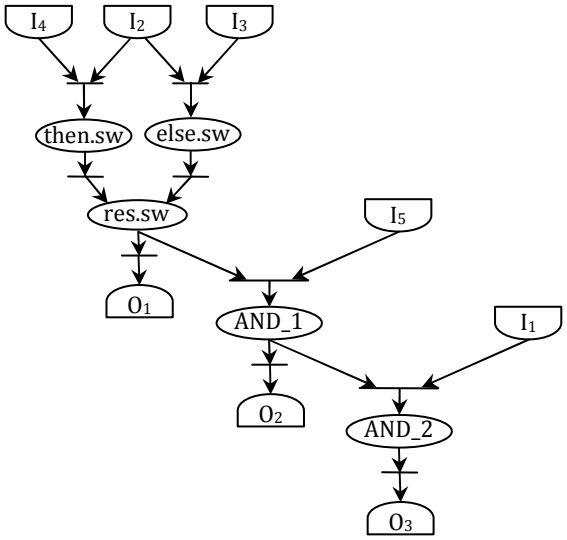


Figure 3.5 : Modèle de transfert d'information associé à l'extrait de planche SCADE

3.2.2 Ecoulements

Définition 3 : Chemin d'un graphe

Soient $v[x,y]=(u_1,u_2,\cdots,u_i,\cdots,u_p)$ un chemin du sommet x au sommet y et S l'ensemble des sommets d'un graphe. Le chemin $v[x,y]$ est inclus dans S si et seulement si toutes les extrémités des arcs u_i $(i<p)$ appartiennent à S.

Définition 4 : Ecoulement

Un écoulement correspond à un chemin d'information du MTI qui conduit l'information à partir d'une ou plusieurs sources, à travers les modules et les transitions, jusqu'à un puits. Un écoulement peut être considéré comme une « fonction élémentaire » du système.

Soient $G=(P,T,A)$ le MTI d'un système (graphe biparti orienté) et Ecl un sous-ensemble de (P,T) tels que :

- P_{Ecl} le sous-ensemble de places de Ecl ;
- T_{Ecl} le sous-ensemble de transitions de Ecl ;
- M_{Ecl} le sous-ensemble de modules de P_{Ecl} ;
- I_{Ecl} le sous-ensemble de sources P_e ;
- $\{o\}$ le seul puits de P_{Ecl}.

On dit que Ecl est un écoulement si et seulement si :

$\forall t \in T_{Ecl}$ $\quad \Gamma_G(t) \subset P_{Ecl}$;

$\forall m \in M_{Ecl}$ $\quad \Gamma_G^-(m) \cap T_{Ecl} \neq \varnothing, \Gamma_G^+(m) \cap T_{Ecl} \neq \varnothing$;

$\forall m \in M_{Ecl}$ $\quad \exists i \in I_{Ecl} \ / \ i \overset{*}{\Rightarrow} m$ et $\exists o \in P_{Ecl} \ / \ m \overset{*}{\Rightarrow} o$ $\ (i \overset{*}{\Rightarrow} o) \subset Ecl$ (il existe un chemin inclus dans Ecl partant de i passant par m au puits o);

De manière informelle, soit $S = P \cup T$ l'ensemble des sommets représentant toutes les places et toutes les transitions du MTI, un écoulement est construit à partir d'un sous-ensemble S' tel que :

- Si une source appartient à S', alors elle admet au moins une transition en aval dans S' ;
- Si un puits appartient à S', alors il admet au moins une transition en amont dans S' ;
- Si un module appartient à S', alors il admet au moins une transition en amont et une transition en aval dans S' ;
- Si une transition appartient S', alors toute place qui lui est adjacente appartient à S' ;
- Quelle que soit la place p appartenant à S', il existe dans S' un chemin allant d'une source à un puits via cette place.

Les écoulements sont utilisés pour l'évaluation de la perte d'information dans le système en considérant les propriétés suivantes :

- tout arc arrivant à une place permet de contrôler toutes les données nécessaires au test de cette place ;
- tout arc partant d'une place permet d'observer tous les résultats issus de cette place.

La description SCADE de la **Figure 3.4** contient six (6) écoulements. Chaque écoulement Ecl_i est décrit ci-dessous à l'aide des sources, des modules qu'il couvre et le puits qui lui est associé.

$$Ecl_1 = \{ I_2 , I_3 \mid else.sw \mid O_1 \}$$

$$Ecl_2 = \{ I_2, I_4 \mid then.sw \mid O_1 \}$$
$$Ecl_3 = \{ I_2, I_3, I_5 \mid else.sw, AND\_1 \mid O_2 \}$$
$$Ecl_4 = \{ I_2, I_4, I_5 \mid then.sw, AND\_1 \mid O_2 \}$$
$$Ecl_5 = \{ I_2, I_3, I_5 \mid else.sw, AND\_1, AND\_2 \mid O_3 \}$$
$$Ecl_6 = \{ I_2, I_4, I_5 \mid then.sw, AND\_1, AND\_2 \mid O_3 \}$$

Une représentation graphique des écoulements Ecl_1 et Ecl_3 (à l'aide de traits épais) est donnée par la **Figure 3.6** ci-dessous.

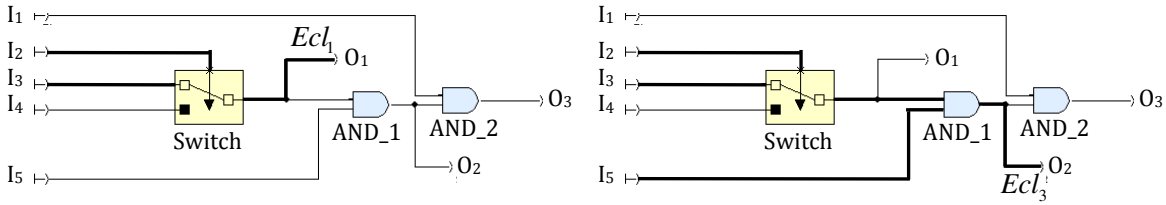


Figure 3.6 : Représentation graphique des écoulements

Cette notion d'écoulement qui correspond à la représentation d'une dépendance (en termes de flots de données) entre un ensemble d'entrées et une sortie va :

- aider à la définition de jeux de test car un écoulement va a priori correspondre à une forme de représentation de la fonction liée à la sortie qui lui est associée ;
- aider à l'estimation de la couverture des tests vis-à-vis d'un ensemble d'exigences fonctionnelles.

L'écoulement est donc un élément « *clé* » de la méthode d'analyse de testabilité des spécifications flot de données proposée dans ce chapitre.

3.2.3 Stratégie de test : Start-Small

Une stratégie de test définit la méthodologie de test correspondant à la manière d'appliquer les tests et d'analyser les résultats. L'application d'une stratégie de test au cours de l'analyse de testabilité permet la sélection des écoulements pertinents pour le test de validation de la spécification selon le critère suivant : toutes les places du modèle de testabilité doivent être activées au moins une fois afin d'assurer la couverture des opérateurs. Les écoulements sélectionnés correspondent à ceux pour lesquels les jeux de test doivent être définis, pour l'application de la stratégie choisie, au cours de la phase de test. Une stratégie définit également le degré de précision du diagnostic, lorsqu'une faute est détectée. Différents types de stratégie de test peuvent être appliqués, à ce niveau, selon le contexte et l'objectif recherché par le test [Rob89]. Dans ce chapitre, nous nous intéressons à la stratégie Start-Small qui est une

stratégie incrémentale adaptée à la détection de fautes multiples au cours de la phase de conception du système.

Start-Small est une approche structurelle progressive. Elle cherche à couvrir l'ensemble de toutes les places du modèle de testabilité. Appliquer Start-Small revient à exécuter à l'aide de jeux de test, dans un premier temps, l'écoulement qui contient une « quantité minimale » de modules fonctionnels ; puis le prochain écoulement exécuté concerne un minimum de modules non activés ; ainsi de suite jusqu'à l'activation de tous les modules du modèle de testabilité. Lorsqu'une faute est détectée, l'effort de diagnostic se concentre sur le nombre minimal de modules nouvellement activés. Cette faute doit être corrigée avant l'exécution du jeu de test suivant. Pour cela, l'ordre de sélection des écoulements doit être respecté au cours de leur activation afin minimiser l'effort de diagnostic. La **Figure 3.7** ci-dessous illustre les principes de Start-Small.

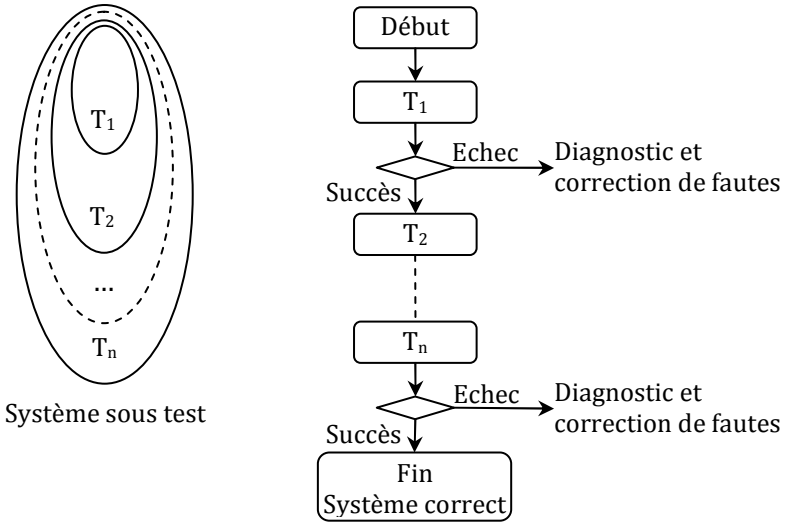


Figure 3.7 : Illustration des principes de la stratégie Start-Small

3.2.4 Mesures de testabilité

Le calcul des mesures de testabilité est basé sur l'évaluation de la perte d'information dans le système. Cette perte est déterminée à l'aide de certaines notions de la théorie de l'information. Pour la définition de ces notions, nous considérons les hypothèses ci-dessous.

A. Hypothèses et définitions

Hypothèse 1 : Cardinalité du domaine d'un flot de donnée de type entier et réel

Le domaine des entiers et des réels n'est pas dénombrable. Chacun des deux domaines peut être représenté comme suit : $\{-\infty,\cdots,x_0,x_1,\cdots,x_N,\cdots,+\infty\}$. Dans le contexte de nos travaux, les flots de type réel sont issus de données physiques et donc bornées. Ces bornes pourraient être

spécifiées dans un document d'exigences (de bas niveau). Ceci permet de procéder à un échantillonnage et de se ramener à des ensembles « quasiment finis ».

Dans ce manuscrit, nous considérons l'intervalle $[x_0, x_N]$ pour les méthodes de calcul proposées et nous fixons à 2^8 et 2^{12} respectivement les cardinalités des flots de données de type entier et réel. Si nous pouvons supposer une loi de distribution dans cet intervalle, ceci n'est probablement pas le cas pour les deux autres intervalles non bornés $]-\infty, x_0[$ et $]x_N, +\infty[$.

Hypothèse 2 : Indépendance de l'occurrence des différentes valeurs d'un flot de donnée

Dans un contexte de synchronisme, les valeurs des flots de données contenus dans le système sont calculées à chaque cycle d'exécution. Nous considérons, dans ce manuscrit, l'occurrence des valeurs des flots de données d'un cycle à un autre comme deux évènements indépendants. En effet, la complexité liée à la production de ces entrées en fonction de l'utilisation des opérateurs rend toute étude de corrélation, entre les différentes valeurs, relative. Selon qu'un opérateur temporel soit utilisé près des entrées, à l'intérieur ou près des sorties d'un diagramme d'opérateurs (plus ou mois complexe) du système, cette hypothèse d'indépendance est plus ou moins explicite. Afin d'homogénéiser les méthodes de calcul des coefficients de perte d'information (CPI) associés à chaque opérateur, nous supposons l'indépendance de l'occurrence des valeurs des entrées effectives des opérateurs.

De façon formelle, deux évènements A et B sont dits indépendants l'un de l'autre lorsque la probabilité d'occurrence ou la distribution de B dans l'univers « entier » est identique à celle de B dans le sous-univers A.

$$P(B) = P_A(B) = P(A/B) = \text{Probabilité d'occurrence de B sachant A}$$

La définition de la probabilité conditionnelle de B sachant A est :

$$P(A/B) = \frac{P(B \cap A)}{P(A)} \text{ d'où } P(B \cap A) = P(B) \times P(A) \text{ lorsque } A \text{ et } B \text{ sont indépendants.}$$

Par ailleurs, nous associons une loi de Bernoulli à la distribution aux entrées de type booléen des opérateurs : la probabilité d'occurrence de $1\,(vrai)$ est $P_1 = p$ et la probabilité d'occurrence de $0\,(faux)$ est $P_0 = 1 - p$.

<u>Définition 5 :</u> Quantité d'information d'une variable

Soit Var une variable représentant un flot de données de type T_D tel que :

- k est le cardinal (nombre d'éléments) de T_D ;
- $p(e_j)$ est la probabilité d'occurrence de l'élément $e_j \; avec \; j \in (1 \ldots k)$ au niveau de Var ;
- $I_{e_j} = -\log_2 \lfloor p(e_j) \rfloor$ est la quantité d'information liée à un élément e_j de Var.

La quantité d'information q associée à Var est donnée par la formule de Shannon :

$$q = H(Var) = -\sum_{j=1}^{k} p(e_j) \times \log_2[p(e_j)]$$

Cette quantité est maximale lorsqu'il y a équiprobabilité : $p(e_j) = \frac{1}{k}$.

Tableau 3.1 : Quantité d'information maximale associée à une variable

Type d'information	Probabilité $p(e_j)$	Capacité d'information
Booléen (1 bit)	$\frac{1}{2}$	1
Entier (8 bits)	$\frac{1}{2^8}$	8
Réel (12 bits)	$\frac{1}{2^{12}}$	12

Définition 6 : Capacité d'une place d'un MTI

Selon que la place soit une source, un module fonctionnel ou un puits, sa capacité n'est pas déterminée de la même manière.

- Lorsqu'il s'agit d'une source, la capacité de la place correspond à la quantité d'information maximale du type d'information associé. Donc, la capacité d'une source S_i produisant une information de type T_D est : $C(S_i) = k$ avec k le nombre de bits nécessaires à l'expression de tous les éléments de T_D.
- La capacité d'un module fonctionnel correspond à la quantité d'information associée à sa sortie. Donc, la capacité d'un module M_i est : $C(M_i) = H(O_i)$, O_i représentant la sortie du module.
- Pour un puits, la capacité correspond à celle de son entrée. Donc, la capacité d'un puits P_i est : $C(P_i) = H(E_i)$, E_i représentant la variable d'entrée du puits.

Définition 7 : Transinformation entre deux variables d'un MTI

Un canal d'information est un système qui produit une sortie X à partir d'une entrée Y. La capacité de ce canal notée $C(X,Y)$ est la valeur maximum de la transinformation $T(X,Y)$. La valeur maximum de la transinformation est calculée à l'aide de la formule suivante :

$$T(X,Y) = H(X) - H(X \mid Y) = H(Y) - H(Y \mid X)$$

Où $H(X \mid Y)$ représente la quantité d'information de la variable X sachant Y.

Définition 8 : Capacité d'un arc et de son extrémité finale d'un MTI

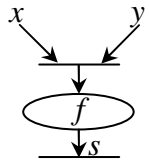
Un arc du MTI d'un système peut être considéré comme un canal d'information qui réalise la fonction identité. De ce fait, sa capacité est égale à la quantité d'information de la sortie de la place correspondant à son extrémité initiale. Donc, la capacité d'un arc Arc_i est : $C(Arc_i) = C(EI_i)$, EI_i la place représentant extrémité initiale de Arc_i.

La capacité de l'entrée de la place correspondant à l'extrémité finale de l'arc est à son tour égale à celle de l'arc. Soit EF_i l'extrémité finale d'un arc Arc_i, $C(EF_i) = C(Arc_i)$.

Définition 9 : CPI – Coefficient de Perte d'Information

Soit M un module fonctionnel représentant une sous fonction f de l'opérateur auquel il est associé en calculant une sortie s à partir d'un sous ensemble d'entrées. La quantité d'information effective à la sortie du module est calculée en considérant les ensembles finis discrets des domaines d'entrée et de sorite. Considérons la fonction f définie ci-dessous.

$$f : X \times Y \to S$$
$$x \times y \to s$$



- soit k le cardinal de l'ensemble S ;
- soit n_{z_i} le nombre de triplet (x, y, s_i) tel que $s_i = f(x, y)$;
- soit $N = \sum_{i=1}^{K} n_{s_i}$ avec K le nombre de sorties distinctes de f
- la probabilité d'occurrence de s_i est $p_i = \frac{n_{s_i}}{N}$

La quantité d'information effective à la sortie du module M est $H(M) = -\sum_{i=1}^{k} p_i \times \log_2(p_i)$

La quantité d'information maximum de la sortie s est : $H(s) = -\sum_{i=1}^{k} \frac{1}{k} \times \log_2\left(\frac{1}{k}\right)$ en supposant une équiprobabilité entre les différents éléments de S.

Le coefficient de perte d'information (CPI) d'un module représente le rapport entre la quantité d'information effective $H(M)$ et la quantité d'information maximum $H(s)$. Sa valeur est située dans l'intervalle $[0, 1]$.

$$CPI(M) = \frac{H(M)}{H(s)}$$

Pour illustration, le CPI associé au module fonctionnel d'un opérateur OR logique, avec deux entrées (e_1 , e_2) et une sortie s de type booléen, est calculé de la manière suivante :

$$P_0 = \frac{1}{4} ; \quad P_1 = \frac{3}{4}$$
$$H(OR) = -\sum_{i=0}^{1} p_i \times \log_2(p_i) \approx 0{,}81$$
$$H(s) = 1$$
$$CPI(OR) = \frac{0{,}81}{1} = 0{,}81$$

e_1	e_2	s
0	0	0
0	1	1
1	0	1
1	1	1

Table de vérité

B. Définition des mesures de testabilité

D'une manière informelle, la testabilité d'un système est liée à la testabilité de ses composants ou opérateurs. L'évaluation de la testabilité d'un système conduit à l'évaluation de la testabilité de ses composants. La méthode d'analyse de flot de données associe deux mesures de testabilité (la contrôlabilité et l'observabilité) à un composant.

- La contrôlabilité d'un composant détermine la facilité à acheminer les données à partir des entrées du système vers les entrées du composant (**Figure 3.8**) ;
- L'observabilité d'un composant détermine la facilité de propager un comportement incorrect du composant jusqu'aux sorties du système (**Figure 3.8**).

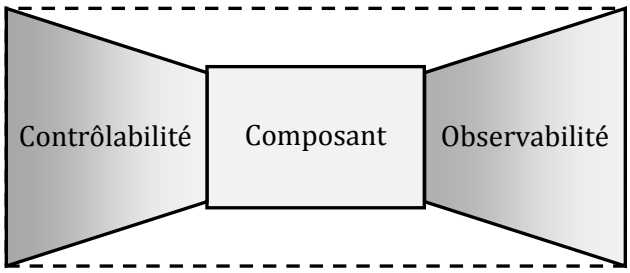


Figure 3.8 : Contrôlabilité et observabilité d'un composant

Les mesures de testabilité d'un composant sont déduites de celles des modules fonctionnels qui le représentent. Ces mesures sont calculées pour chaque écoulement déterminé du système. De ce fait, plusieurs mesures de testabilité peuvent être associées à un composant. Soit E un écoulement du modèle de testabilité, soit M un module appartenant à l'écoulement E. Les variables de E et M sont dénotées comme suit :

- I_E est l'entrée (ensemble des sources) de E ;
- O_E est la sortie (puits associé) de E ;
- I_M est l'entrée (ensemble des entrées) de M ;
- O_M est la sortie (sortie du module appartenant à l'écoulement) de M.

Contrôlabilité

Soit M un module du MTI appartient à l'écoulement E, la quantité d'information que le module M peut recevoir à travers E est la transinformation entre la variable I_E et la variable I_M notée $T(I_E, I_M)$. Cette quantité est inférieure ou égale à $C(I_M)$. La mesure de contrôlabilité d'un module M est calculée comme suit :

$$Cont_E(M) = \frac{T(I_E, I_M)}{C(I_M)}$$

Si le module M est isolé, toutes les valeurs de ses entrées peuvent être générées. La quantité d'information qu'un module M peut recevoir est donc la capacité maximum théorique $C(I_M)$ de son entrée.

D'une manière intuitive, l'information d'entrée du système qui est nécessaire à l'activation d'un composant, ne peut pas être acheminée entièrement aux entrées du composant lorsque celui-ci n'est pas isolé. Ceci rend difficile le contrôle du composant et par conséquent augmente la difficulté de contrôler le fonctionnement du système.

Observabilité

De la même manière, la quantité d'information maximum qu'un module M peut produire est la capacité maximum théorique $C(O_M)$ de sa variable sortie. La quantité d'information que le module peut délivrer à la sortie de l'écoulement E est la transinformation entre la variable O_M et la variable O_E notée $T(O_M, O_E)$. Cette quantité est inférieure ou égale à $C(O_M)$. L'observabilité du module M dans l'écoulement E est calculée par la formule suivante :

$$Obs_E(M) = \frac{T(O_M, O_E)}{C(O_M)}$$

D'une manière intuitive, l'information calculée par un composant interne n'arrive pas complètement aux sorties du système. Cette perte d'information rend l'observabilité du composant difficile et par conséquent l'observabilité des sorties du système est moins complète (erreur masquée par exemple).

C. Simulation du transfert d'information

Le calcul des mesures de testabilité s'appuie sur le principe de la simulation du transfert de flots d'information des entrées aux sorties du système. Cette simulation est basée sur l'évaluation de la perte d'information causée par les opérateurs utilisant ces flots d'information en entrée pour calculer les sorties du système. Cette perte d'information est représentée par le CPI (coefficient de perte d'information) associé à chaque opérateur. La simulation est réalisée à l'aide du modèle de transfert d'information pondéré avec des quantités d'information associées aux arcs et aux places. Ce modèle pondéré est appelé le Réseau de Transfert d'Information (RTI).

Réseau de Transfert d'Information (RTI)

Le RTI est un graphe de transfert d'information enrichi à l'aide des capacités maximales associées aux arcs et des capacités effectives associées aux places. De même que pour le modèle de transfert d'information, un RTI élémentaire est associé à chaque opérateur utilisé dans le

système. Le réseau de transfert d'information associé au système est obtenu par composition de tous ses RTI élémentaires (**Figure 3.9**).

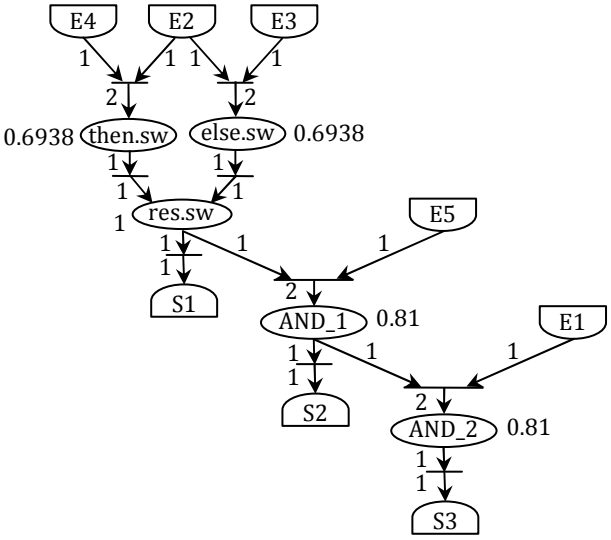


Figure 3.9 : Réseau de transfert d'information associé à l'extrait de planche SCADE

Simulation du réseau

La simulation du réseau de transfert d'information consiste à déterminer la quantité d'information effective associée à chacun de ces éléments à partir des quantités d'information théoriques.

Soit $RS = (X, U)$ avec $X = Z \cup T$, un réseau de transfert d'information auquel deux sommets (x_1, x_n) sont ajoutés tels que :

- $\Gamma_G^-(x_1) = \varnothing$ (x_1 est appelé sommet source et connecté à l'ensemble des places « sources » du RTI) ;
- $\Gamma_G^+(x_n) = \varnothing$ (x_n est appelé sommet destination et connecté à l'ensemble des places « puits » du RTI) ;
- c_{ij} (un entier positif) comme la capacité d'information associée à chaque arc (x_i, x_j) $\in U$. Le triplet $T = (X, U, C)$ où $C = \{c_{ij}, (x_i, x_j) \in U\}$ est appelé réseau avec capacités.

Un *flot* dans le réseau T est une fonction qui associe à chaque arc (x_i, x_j) $\in U$ une quantité φ_{ij} qui représente la quantité de flot qui passe par cet arc, en provenance de la source x_1 et en destination du puits x_n. Ce flot est *réalisable* s'il satisfait les contraintes sur la capacité des arcs (1) et sur la conservation des flux (2) ci-dessous.

$$(1) \qquad \varphi_{ij} \le c_{ij}, (x_i, x_j) \in U$$

$$(2) \quad \begin{cases} \sum_{x_j \in \Gamma_T^+(x_i)} \varphi_{ij} = \sum_{x_j \in \Gamma_T^-(x_i)} \varphi_{ji}, \ avec \ i = 2, \ldots, n-1 \\ \sum_{x_j \in \Gamma_T^+(x_1)} \varphi_{1j} = \sum_{x_j \in \Gamma_T^-(x_n)} \varphi_{jn} = \phi(\varphi) \end{cases}$$

La quantité $\phi(\varphi)$ représente le volume total d'information que met en circulation le flot réalisable φ ; elle est appelée valeur du flot φ. Un flot est dit de *valeur maximale* lorsqu'il maximise φ dans tous les flots réalisables de T. Ce problème peut être traduit sous forme d'un programme linéaire et peut donc se résoudre au moyen de l'algorithme du simplexe. Le théorème « *flot maximum et coupe minimum* » de Ford et Fulkerson assure l'existence et l'unicité d'un tel flot [Corn02].

Le réseau de transfert d'information simulé correspond donc au flot maximum du réseau de capacité associé sans les sommets x_1 et x_n. La **Figure 3.10** ci-dessous illustre cette notion de RTI simulé.

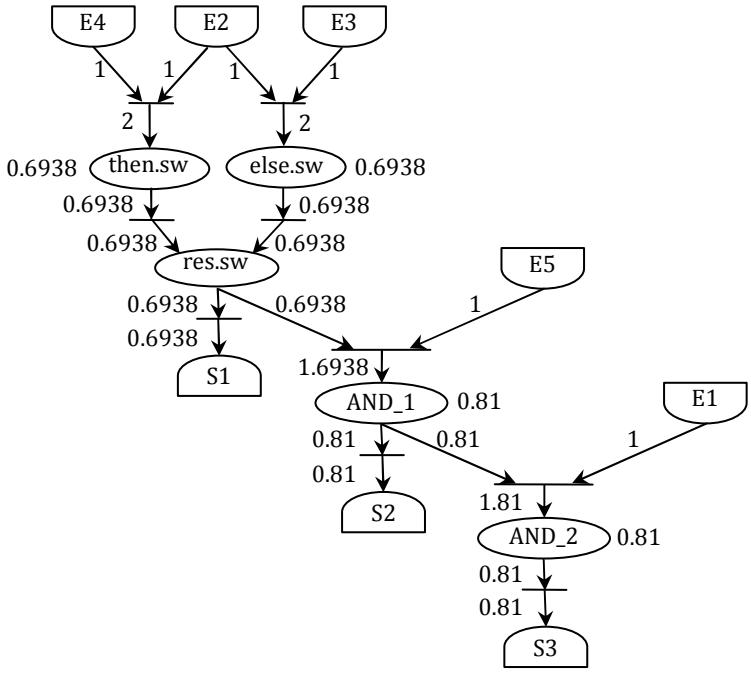


Figure 3.10 : RTI simulé associé à l'extrait de planche SCADE

3.2.5 Synthèse

Dans cette section, nous avons présenté l'approche d'analyse de testabilité des spécifications flot de données. Nous avons rappelé le modèle de transfert d'information d'un système qui représente « la brique » de base de cette analyse. La notion d'écoulement a été introduite. Les écoulements correspondent aux fonctions élémentaires du système en termes de flots

d'information. La stratégie de test Start-Small, adaptée à un contexte de test incrémental au cours de la conception, a également été présentée. Elle est utilisée pour sélectionner les écoulements pertinents pour la couverture des fonctions décrites dans une spécification flots de données. Nous avons fini par introduire les notions de contrôlabilité et d'observabilité qui sont des mesures définies à partir d'évaluation de la perte d'information causée par les opérateurs utilisés dans le système. L'estimation de cette perte d'information se base sur la théorie de l'information.

3.3 Adaptation de l'analyse de testabilité au contexte AIRBUS

L'analyse de testabilité des spécifications détaillées formelles SCADE des systèmes AIRBUS nécessite une adaptation de certains concepts de testabilité. Cette adaptation a nécessité des extensions permettant la prise en compte de certaines spécificités liées aux opérateurs utilisés pour la spécification des systèmes AIRBUS. Ces opérateurs sont décrits dans les bibliothèques « métiers ». Les bibliothèques « métier » AIRBUS décrivent les « opérateurs de base » utilisés dans les spécifications détaillées SCADE des systèmes. Ces opérateurs de base correspondent d'une part, à ceux prédéfinis dans l'environnement de spécification SCADE et d'autre part, aux opérateurs spécifiques définis par AIRBUS pour ses systèmes.

Les extensions des concepts de l'analyse de testabilité, réalisées dans cette thèse, se traduisent d'une part, par la modélisation des opérateurs d'aiguillage à commande multiple (section 3.3.1) et la modélisation des opérateurs temporels (section 3.3.2), décrits dans les bibliothèques « métier », utilisées pour la spécification des systèmes de commande de vol. Le comportement des systèmes réactifs synchrones (comme ceux de type « commande de vol ») évolue dans le temps et est très souvent lié au passé (cycles précédents). De ce fait, la spécification de ces systèmes est basée sur la notion de *« logique temporelle du passé strict »*. Un des moyens de représentation de ce comportement consiste à utiliser des opérateurs à mémoire ou temporels. Les opérateurs temporels sont des opérateurs dont le comportement à un cycle donné dépend des informations liées aux cycles précédents. D'autre part, les extensions passent par l'introduction de la notion de classification des écoulements conformément aux différentes phases d'exécution des systèmes réactifs synchrones.

Par ailleurs, les concepts du calcul des mesures de contrôlabilité et d'observabilité des composants du système se basent sur la simulation de la perte d'information dans le système. Cette simulation considère les quantités d'information théorique disponibles à l'entrée et à la sortie de chacun des composants. Ces mesures ont été évaluées dans des travaux de recherches

précédents. Les résultats de ces évaluations ont abouti à des conclusions prometteuses. Partant de ce constat, nous avons décidé de procéder à l'évaluation des CPI des opérateurs modélisés afin de mener des expérimentations avant de conclure sur la pertinence des mesures de testabilité dans le contexte AIRBUS.

3.3.1 Modélisation des opérateurs d'aiguillage à commande multiple

Les opérateurs d'aiguillage à commande multiple sont des opérateurs spécifiques définis par AIRBUS. Ils proposent les mêmes fonctionnalités qu'un opérateur d'aiguillage à commande unique (de type *si … alors … sinon …*) en considérant au moins deux commandes. Plusieurs types d'opérateurs sont définis dans les bibliothèques « métier » en fonction du type d'information qu'ils traitent. En effet, on distingue les opérateurs de type :

- $AIGx$ $(x \in [3,6]$ correspondant au nombre de choix possible$)$: cet opérateur traite les informations de type réel ;
- $AIGx\_B$ $(x \in [3,6]$ correspondant au nombre de choix possible$)$: cet opérateur traite les informations de type booléen ;
- $AIGx\_V$ $(x \in [3,6]$ correspondant au nombre de choix possible$)$: cet opérateur traite les informations de type vecteur.

Ces opérateurs permettent de gérer la priorité de prise en compte des flots d'information disponibles au niveau de leurs entrées effectives. De ce fait, cet aspect doit être intégré dans leur modélisation pour l'analyse de testabilité afin d'assurer une cohérence dans l'interprétation des résultats. La modélisation proposée pour ces types d'opérateurs, dans cette section, permet de :

- décrire l'ensemble des flots d'information qu'ils contiennent en tenant compte de leur priorité à l'aide d'un modèle de transfert d'information (MTI) élémentaire ;
- évaluer le coefficient de perte d'information associé à chaque module fonctionnel défini pour la modélisation de ces types d'opérateurs.

Dans un premier temps, le comportement des opérateurs d'aiguillage à commande multiple est décrit à l'aide d'un exemple d'illustration. Ensuite, les techniques de modélisation du MTI élémentaire et d'évaluation du CPI des modules fonctionnels sont présentées.

A. Description du comportement

La sémantique des trois types d'opérateurs d'aiguillage est illustrée à l'aide de la **Figure 3.11** ci-dessous.

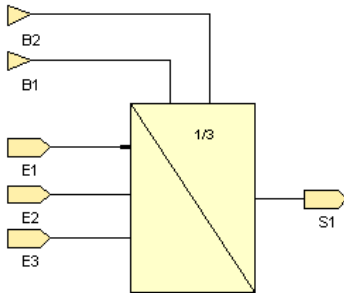


Figure 3.11 : Représentation SCADE de l'opérateur AIG3

Le comportement de l'opérateur $AIG3$ peut être décrit comme suit :

$$\begin{aligned} &Si\ B1(k) = True\ Alors\ S1(k) = E1(k) \\ &\quad Sinon\ Si\ B2(k) = True\ Alors\ S1(k) = E2(k) \\ &\quad\quad Sinon\ S1(k) = E3(k) \end{aligned}$$

La description sémantique de l'opérateur $AIG3$ ci-dessus peut s'étendre aux différents types d'opérateurs d'aiguillage $AIGn$ quelque soit le nombre ($n \in [0, 6]$) de commandes. Cette sémantique est décrite à l'aide l'expression généralisée ci-dessous.

$$\begin{aligned} &Si\ B_1(k) = True\ Alors\ S_1(k) = E_1(k) \\ &\quad Sinon\ Si\ B_2(k) = True\ Alors\ S_1(k) = E_2(k) \\ &\quad\quad \ldots \\ &\quad\quad Sinon\ Si\ B_{n-1}(k) = True\ Alors\ S1(k) = E_{n-1}(k) \\ &\quad\quad\quad Sinon\ S1(k) = E_n(k) \end{aligned}$$

B. Modélisation du MTI (Modèle de Transfert d'Information)

Le modèle de transfert d'information des opérateurs d'aiguillage à commande multiple est constitué d'une « cascade » de modèles de l'opérateur d'aiguillage à commande unique. Pour rappel, le MTI correspondant à un opérateur d'aiguillage à commande unique est représenté par la **Figure 3.12**.

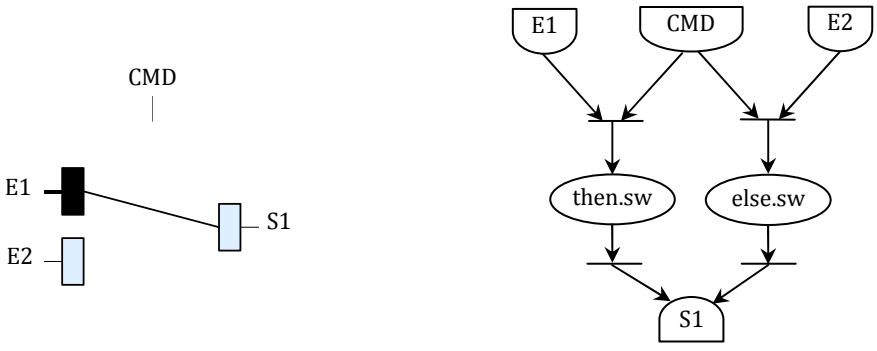


Figure 3.12 : Représentation SCADE et MTI de IF_THEN_ELSE

La modélisation du MTI des opérateurs d'aiguillage à commande multiple est illustrée ci-dessous en utilisant l'opérateur *AIG*3.

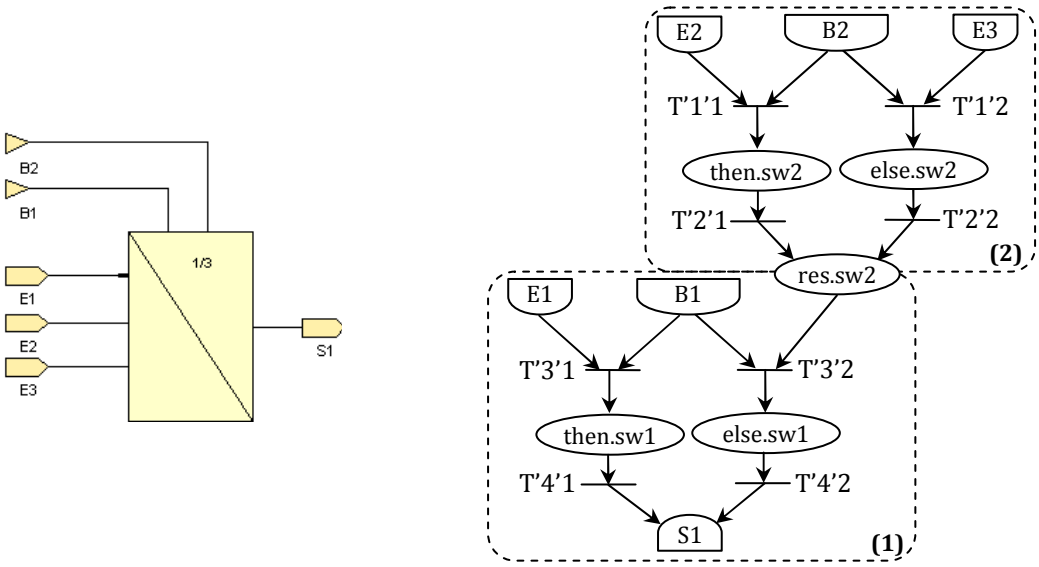


Figure 3.13 : Représentation SCADE et MTI de AIG3

Le modèle de transfert d'information de l'opérateur *AIG*3 est composé de deux principales parties. La première (1) correspond à la description des flots gérés par la commande *B*1 et la seconde (2) décrit ceux gérés par la commande *B*2. Cette modélisation représente les flots d'information contenus dans l'opérateur *AIG*3 conformément à son comportement. Par extrapolation, les différents types d'opérateurs d'aiguillage à commande multiple sont modélisés de la même manière en représentant les flots correspondant aux différentes commandes. La description textuelle du MTI de l'opérateur *AIG*3 est présentée ci-dessous.

```
1 (*
2   Description du MTI de "AIG3"
3 *)
4
5 type AIG3_III_I
6 debut
7       debut_entrees
8               IN1 : 1; (* E1 *)
9               IN2 : 2; (* E2 *)
10              IN3 : 3; (* E3 *)
11              IN4 : 4; (* B1 *)
12              IN5 : 1; (* B2 *)
13      fin_entrees
14      debut_sorties
15              OUT1 : 1000 ; (* S1 *)
16      fin_sorties
17      debut_modele
18              T'1'1 / IN2, IN5 / then.sw2 /
19              T'1'2 / IN3, IN5 / else.sw2 /
20              T'2'1 /  then.sw2 / res.sw2 /
```

```
21              T'2'2 /  else.sw2 / res.sw2 /
22              T'3'1 /  IN1, IN4 / then.sw1 /
23              T'3'2 /  res.sw2, IN4 / then.sw1 /
24              T'4'1 /  then.sw1 / OUT1 /
25              T'4'2 /  else.sw1 / OUT1 /
26      fin_modele
27 fin
```

C. Evaluation du CPI (Coefficient de Perte d'Information)

L'évaluation du CPI de ces opérateurs correspond au calcul du coefficient de perte d'information de chacun des couples de modules fonctionnels « $then.sw_x$ *ou* $else.sw_x$ » décrits dans leur MTI. Ce calcul revient à considérer les flots décrits par chaque couple et la commande qui assure la fonction d'aiguillage.

Pour un couple de modules fonctionnels « $then.sw_x$ *ou* $else.sw_x$ », la méthode d'évaluation du CPI associé à chaque module est décrite ci-dessous.

Module fonctionnel $then.sw_x$ **:**

$then.sw_x$ représente l'opération effectuant le calcul de la valeur de la sortie S1 (**Figure 3.11**) lorsque la commande a la valeur « *True* ». Le CPI associé à ce module est calculé en fonction de :

- $P(CMD)$: la probabilité d'avoir la valeur « *True* » au niveau de la commande ;

Ce module renvoie une valeur du flot « $E1$ » lorsque la commande « *CMD* » a la valeur « *True* » et « *Null* » sinon. De ce fait, deux types d'information ($E1$ et *Null*) peuvent être observés au niveau de la sortie de ce module. Soit « $E1$ » un domaine de données de cardinalité K $(avec\ K\ un\ entier)$.

- La quantité d'information théorique à la sortie du module est alors :

$$Q = \log_2(K+1)$$

- La probabilité d'occurrence d'un élément e_{1i} (avec $i \in [1, K]$) dans le flot de l'entrée $E1$ est notée p_{1i}. La probabilité d'occurrence de cet élément au niveau de la sortie du module est :

$$P(e_{1i}) = P(occurrence\ e_{1i}\ dans\ E1\ et\ CMD\ est\ True) = p_{1i} \times P(CMD)$$

- La probabilité d'occurrence de Null à la sortie du module est :

$$P(Null) = 1 - P(CMD)$$

- La quantité d'information effective à la sortie du module est alors :

$$q = -\left(\sum_{i=1}^{K} P(e_{1i}) \log_2 (P(e_{1i})) + P(Null) \log_2 (P(Null)) \right)$$

Le CPI du module « $then.sw_x$, en fonction de $P(CMD) \neq 0$ et 1, est exprimé à l'aide de l'expression ci-dessous.

$$CPI(then.sw_x) = \frac{q}{Q} = \frac{-\left(\sum_{i=1}^{K} p_{1i} \times P(CMD) \log_2 (p_{1i} \times P(CMD)) + (1 - P(CMD)) \log_2 (1 - P(CMD))\right)}{\log_2 (K+1)}$$

Le tableau ci-dessus présente quelques valeurs de CPI du module $then.sw_x$ en fonction de la cardinalité K. Nous considérons une équiprobabilité entre les différents éléments de $E1$ ($p_{1i} = \frac{1}{K}$) et $P(CMD) = \frac{1}{2}$.

Tableau 3.2 : Valeurs du CPI du module "then" de IF_THEN_ELSE

K	$CPI(then.sw_x)$
$2\ (booléen)$	0,9464
$2^8\ (entier)$	0,6246
$2^{12}\ (réel)$	0,5833

Module fonctionnel $else.sw_x$:

$else.sw_x$ correspond à l'opération effectuant le calcul de la valeur de la sortie S1 (**Figure 3.11**) lorsque la commande à la valeur « *False* ». Le CPI associé à ce module est calculé en fonction de :

- $P(CMD)$: la probabilité d'avoir la valeur « *True* » au niveau de la commande ;

Ce module renvoie le flot « $E2$ » lorsque la commande « *CMD* » à la valeur « *False* » et « *Null* » sinon. De ce fait, deux informations ($E2$ et *Null*) peuvent être observées au niveau de la sortie de ce module. Nous allons considérer « $E2$ » comme un domaine de données avec une cardinalité K $(avec\ K\ un\ entier)$. La méthode de calcul du CPI de $else.sw_x$ est similaire à celle du module $then.sw_x$ avec :

$$P(e_{2i}) = P(occurrence\ e_{2i}\ dans\ E2\ et\ CMD\ est\ False) = p_{2i} \times (1 - P(CMD)) \text{ et } P(Null) = P(CMD).$$

De ce fait, le CPI est défini à l'aide des expressions ci-dessous.

$$CPI(else.sw_x) = \frac{q}{Q} = \frac{-\left(\sum_{i=1}^{K} p_{2i} \times (1 - P(CMD)) \log_2 (p_{2i} \times (1 - P(CMD))) + P(CMD) \log_2 (P(CMD))\right)}{\log_2 (K+1)}$$

Le tableau ci-dessus présente quelques valeurs de CPI du module $else.sw_x$ en fonction de la cardinalité K. Nous considérons une équiprobabilité entre les différents éléments de $E2$ ($p_{2i} = \frac{1}{K}$) et $P(CMD) = \frac{1}{2}$.

Tableau 3.3 : Valeurs du CPI du module "else" de IF_THEN_ELSE

K	$CPI(else.sw_x)$
2 $(booléen)$	0,9464
2^8 $(entier)$	0,6246
2^{12} $(réel)$	0,5833

3.3.2 Modélisation des opérateurs temporels

Il existe des opérateurs temporels de type « confirmation », de type « bascule », de type « détecteur d'impulsion », de type « stabilisateur de front » et de type « retard ». Nous présentons les techniques de modélisation sur les opérateurs de type « confirmation ». La modélisation des autres types est proposée en annexe (section A.). Les opérateurs de confirmation de front généralement utilisés dans les systèmes de commande de vol AIRBUS sont : les opérateurs de confirmation d'un front montant ($CONF1$) et les opérateurs de confirmation d'un front descendant ($CONF2$). Nous détaillons la modélisation de l'opérateur $CONF1$ dans cette section (celle de $CONF2$ étant similaire). L'opérateur $CONF1$ est utilisé pour décrire la modélisation des opérateurs de confirmation de front dans cette section. $CONF1$ confirme un changement de front à l'état $True$ de l'entrée $E1$. La sortie est « $True$ » si et seulement si l'entrée $E1$ est $True$ pendant au moins Nb_cycles cycles (**Figure 3.14**). Les entrées liées à l'initialisation sont :

- B_Init : entrée d'initialisation de type booléen
- $Init$: nombre de cycles d'initialisation appartenant à l'intervalle $[0, Nb\_cycles]$

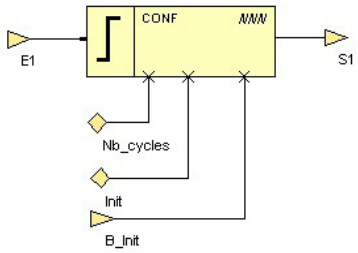


Figure 3.14 : Représentation SCADE de CONF1

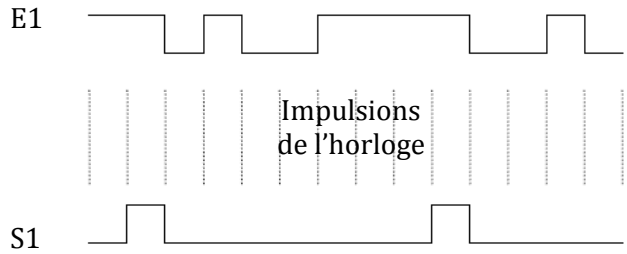
Le comportement de l'opérateur $CONF1$ peut être décrit comme suit :

Initialisation
Si $B\_Init(k) = True$
Alors $E1(k-i) = E1(k)$ *pour i allant de* 0 *à Init*
$E1(k - Init - 1) = False$
"*Calcul normal de l'état de la sortie* $S1$"
Sinon
"*Calcul normal de l'état de la sortie* $S1$"

Calcul normal de l'état de la sortie $S1$
Si $E1(k) = E1(k-1) = \ldots = E1(k - Nb\_cycles) = True$
Alors $S1(k) = True$
Sinon $S1(k) = False$

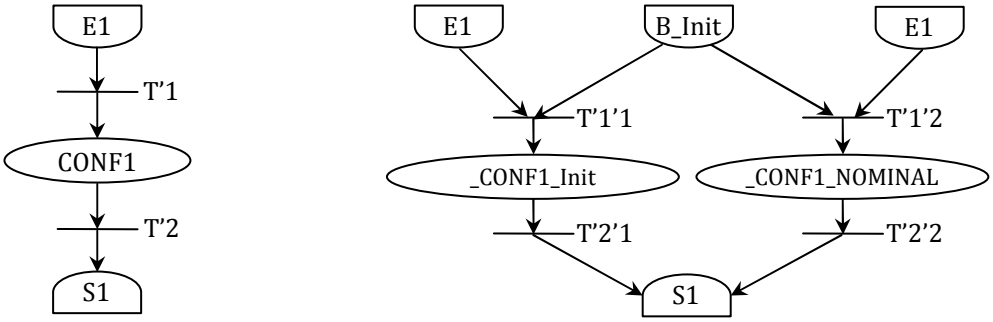
Le diagramme ci-dessous illustre le fonctionnement de l'opérateur $CONF1$ avec les paramètres suivants :

$Nb\_cycles = 3$
$Init = 2$
$B\_Init(k) = [1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0]$
$E1(k) = [1\ 1\ 0\ 1\ 0\ 0\ 1\ 1\ 1\ 1\ 0\ 0\ 1\ 0]$
$S1(k) = [0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0]$



A. Modélisation du MTI élémentaire

Nous proposons deux différentes modélisations pour le MTI de l'opérateur $CONF1$: sans et avec la prise en compte de l'initialisation de l'opération au cours du fonctionnement du système (**Figure 3.15**). La première (a) est constituée d'un module fonctionnel produisant la sortie $S1$ à partir de l'entrée $E1$. La deuxième (b) comporte deux modules fonctionnels. Le premier module $\_CONF1\_Init$ permet de calculer l'état de la sortie $S1$ à l'occurrence d'une initialisation ; le module $\_CONF1\_NOMINAL$ produit l'état de la sortie S1 lorsqu'il n'y a pas d'initialisation. Les entrées de délai ($Init$ et Nb_cycles) ne sont pas représentées dans le MTI. En effet, elles n'apportent pas plus d'information sur la dépendance des flots contenus dans la spécification.



(a) Sans initialisation **(b) Avec initialisation**

Figure 3.15 : Représentation graphique du MTI de CONF1

La description textuelle associée à chacune des deux représentations du MTI de l'opérateur *CONF*1 pour l'analyse de testabilité est décrite ci-dessous.

– Représentation sans initialisation (a)

```
1 (*
2   Description du MTI de "CONF1" sans initialisation
3 *)
4
5 type CONF1_I_I
6 debut
7       debut_entrees
8               IN1 : 1; (* E1 *)
9       fin_entrees
10      debut_sorties
11              OUT1 : 1000 ; (* S1 *)
12      fin_sorties
13      debut_modele
14              T'1 / IN1 / CONF1 /
15              T'2 / CONF1 / OUT1 /
16      fin_modele
17 fin
```

– Représentation avec initialisation (b)

```
1 (*
2   Description du MTI de "CONF1" avec initialisation
3 *)
4
5 type CONF1_II_I
6 debut
7       debut_entrees
8               IN1 : 1; (* E1 *)
9               IN2 : 2; (* B_Init *)
10      fin_entrees
11      debut_sorties
12              OUT1 : 1000 ; (* S1 *)
13      fin_sorties
14      debut_modele
15              T'1'1 / IN1, IN2 / _CONF1_Init /
```

```
16              T'1'2 / IN1, IN2 / _CONF1_NOMINAL /
17              T'2'1 / _CONF1_Init / OUT1 /
18              T'2'2 / _CONF1_NOMINAL / OUT1 /
19      fin_modele
20 fin
```

B. Evaluation du CPI (Coefficient de Perte d'Information)

Nous proposons dans cette section les méthodes d'évaluation des CPI associés aux modules des MTI décrits dans la **Figure 3.15** selon les modes de fonctionnement sans ou avec initialisation. Dans la suite, l'expression $E(k)$ représente la valeur du flot E au cycle k d'exécution de l'opérateur.

Calcul du CPI sans initialisation

Soient $P(S1=1)$ et $P(S1=0)$ respectivement les probabilités d'occurrence de 1 et 0 au niveau de la sortie $S1$ à un cycle d'exécution de l'opérateur $CONF1$.

$$P(S1=1) = P(E1(k)=True\ et\ E1(k-1)=True\ et \ldots et\ E1(k-Nb\_cycles)=True)$$

Sachant que les évènements $E(k)$ sont indépendants d'un cycle à l'autre,

$$P(S1=1) = P(E(k)=1) \times P(E(k-1)=1) \times \ldots \times P(E(k-Nb\_cycles)=1)$$

$$\Rightarrow P(S1=1) = p^{Nb\_cycles+1}$$

Or, $P(S1=1) + P(S1=0) = 1$. De ce fait, $P(S1=0) = 1 - p^{Nb\_cycles+1}$

A partir de ces deux expressions associées aux probabilités d'occurrence des deux états possibles de $S1$, la valeur du CPI sachant que la quantité d'information théorique au niveau de $S1, Q=1$ en fonction de p est :

$$CPI(CONF1) = -\left[p^{Nb\_cycles+1} \times \log_2\left(p^{Nb\_cycles+1}\right) + \left(1-p^{Nb\_cycles+1}\right) \times \log_2\left(1-p^{Nb\_cycles+1}\right)\right]$$

Calcul du CPI avec initialisation

Calculer le CPI de l'opérateur $CONF1$ avec initialisation possible revient à calculer celui des deux modules fonctionnels $\_CONF1\_Init$ et $\_CONF1\_NOMINAL$.

a. *Module* $\_CONF1\_Init$

Ce module fournit l'information *Null* lorsque $B\_Init=0$ et calcule l'état de $S1$ lorsque $B\_Init=1$. De ce fait, trois différents états peuvent être observés au niveau de la sortie de ce module $(Null, 0, 1)$. Par conséquent, la quantité d'information théorique disponible à la sortie du module est :

$$Q = \log_2(3)$$

Il existe deux cas pour le calcul du CPI de ce module : le premier correspond au cas où $Init \in [0,\ Nb\_cycles[$ et le deuxième correspond à celui où $Init = Nb\_cycles$. P_B est la probabilité de l'évènement $B\_Init = 1$. Soient $P_{s\_m\_i}(Null)$ la probabilité d'occurrence de la valeur $Null$, $P_{s\_m\_i}(0)$ la probabilité d'occurrence de la valeur 0 et $P_{s\_m\_i}(1)$ la probabilité d'occurrence de la valeur 1 à la sortie du module $\_CONF1\_Init$ représentant la phase initialisation de l'exécution de $CONF1$.

1er cas :

Dans ce contexte, $P_{s\_m\_i}(1) = P(B\_Init = 1\ et\ S1 = 0) = 0$. En effet, il ne peut pas avoir de confirmation au cours du cycle d'initialisation lorsque $Init \in [0,\ Nb\_cycles[$. En effet, il y a toujours un changement de front dans la fenêtre des Nb_cycles précédents.

Par ailleurs :

$$P_{s\_m\_i}(0) = P(B\_Init = 0) = P_B$$
$$P_{s\_m\_i}(Null) = P(B\_Init = 1) = 1 - P_B$$

Donc :

$$CPI(\_CONF1\_Init) = -\frac{1}{\log_2(3)} \times \sum_{j=\{Null,\ 0,\ 1\}} P_{s\_m\_i}(j) \log_2\left(P_{s\_m\_i}(j)\right)$$

2ième cas :

Les probabilités associées à l'occurrence des trois différents états possibles de $S1$ sont exprimées ci-dessous en fonction P_B et p.

$$P_{s\_m\_i}(Null) = P(B\_Init = 0) = 1 - P_B$$
$$P_{s\_m\_i}(0) = P(B\_Init = 1\ et\ E(k) = 0)$$
$$= P(B\_Init = 1) \times P(E(k) = 0) = P_B \times (1 - p)$$
$$P_{s\_m\_i}(1) = P(B\_Init = 1\ et\ E(k) = 1)$$
$$= P(B\_Init = 1) \times P(E(k) = 1) = P_B \times p$$

Donc :

$$CPI(\_CONF1\_Init) = -\frac{1}{\log_2(3)} \times \sum_{j=\{Null,\ 0,\ 1\}} P_{s\_m\_i}(j) \log_2\left(P_{s\_m\_i}(j)\right)$$

b. *Module* $\_CONF1\_NOMINAL$

Ce module fournit l'information $Null$ lorsque $B\_Init = 1$ et calcul l'état de $S1$ lorsque $B\_Init = 0$. De ce fait, trois différents états peuvent être observés au niveau de la sortie de ce

module $(Null, 0, 1)$. Par conséquent, la quantité d'information théorique disponible à la sortie du module est :

$$Q = \log_2(3)$$

Soient $P_{s\_m\_n}(Null)$ la probabilité d'occurrence de la valeur $Null$, $P_{s\_m\_n}(0)$ la probabilité d'occurrence de la valeur 0 et $P_{s\_m\_n}(1)$ la probabilité d'occurrence de la valeur 1 à la sortie du module $\_CONF1\_NOMINAL$ représentant la phase nominale de l'exécution de $CONF1$.

De ce fait, $P_{s\_m\_n}(Null) = P(B\_Init = 1) = P_B$

$P_{s\_m\_n}(1)$ correspond à aux évènements suivants :

- Il n'y a pas eu d'initialisation pendant les Nb_cycles cycles précédents et au cycle courant et puis $E(k=1)$ pendant ces $Nb\_cycles+1$ cycles. Cet évènement peut être exprimé comme suit :

$$((1-P_B)\times p)^{Nb\_cycles+1}$$

- La dernière initialisation a eu lieu dans l'intervalle $[Nb\_cycles,\ Nb\_cycles-Init]$ cycles précédents et $E(k=1)$ depuis ce cycle. Cet évènement peut être exprimé comme suit :

$$\sum_{i=Nb\_cycles-Init}^{Nb\_cycles} P_B\times(1-P_B)^i\times p^{i+1} = (P_B\times p)\times\left[((1-P_B)\times p)^{Nb\_cycles-Init}\times\frac{1-((1-P_B)\times p)^{Init+1}}{1-((1-P_B)\times p)}\right]$$

Donc,

$$P_{s\_m\_n}(1) = ((1-P_B)\times p)^{Nb\_cycles+1} + (P_B\times p)\times\left[((1-P_B)\times p)\times\frac{1-((1-P_B)\times p)^{Init+1}}{1-((1-P_B)\times p)}\right]$$

La somme des probabilités d'occurrence des états $(Null, 0, 1)$ au niveau de la sortie du module $\_CONF1\_NOMINAL$ étant égale à 1 :

$$P_{s\_m\_i}(0) = 1-(P_{s\_m\_i}(Null) + P_{s\_m\_i}(1))$$

$$\Rightarrow P_{s\_m\_i}(0) = 1-\left(P_B + ((1-P_B)\times p)^{Nb\_cycles+1} + (P_B\times p)\times\left[((1-P_B)\times P)\times\frac{1-((1-P_B)\times P)^{Init+1}}{1-((1-P_B)\times P)}\right]\right)$$

A partir de ces trois expressions associées aux probabilités d'occurrence des trois états possibles de $S1$, la valeur du CPI sachant que la quantité d'information théorique au niveau de $S1, Q = \log_2(3)$ est :

$$CPI(\_CONF1\_Init) = -\frac{1}{\log_2(3)}\times\sum_{j=\{Null,\,0,\,1\}} P_{s\_m\_n}(j)\log_2\left(P_{s\_m\_n}(j)\right)$$

3.3.3 Classification des écoulements

Les écoulements constituent un élément fondamental de la méthode d'analyse de testabilité proposée dans cette thèse. En effet, les flots d'information représentés dans le MTI du système sont analysés pour déterminer l'ensemble des écoulements. Pour rappel, un écoulement correspond à un chemin d'information conduisant l'information depuis un sous-ensemble d'entrées à travers les modules fonctionnels et transitions jusqu'à une sortie du système. Il représente une fonction « élémentaire » du système.

L'introduction de l'aspect temporel et particulièrement de la notion d'initialisation dans le processus d'analyse de testabilité conduit à l'apparition de deux différentes classes d'écoulements. Dans les sections suivantes, nous présentons ces deux différentes classes d'écoulements ; puis nous identifions les opérateurs dont la modélisation conduit à cette classification ; enfin, nous proposons une technique de classification des écoulements. Ceci permettra d'analyser les écoulements déterminés à l'issue de l'analyse de testabilité en tenant compte des différentes phases du fonctionnement des systèmes réactifs.

Les deux classes d'écoulements

Le fonctionnement des systèmes réactifs comporte généralement deux principales phases d'exécution :

- La phase d'initialisation qui correspond à l'étape d'initialisation de la mémoire du programme. Au cours de cette phase, l'ensemble des points de mémorisation définis dans le programme est initialisé. Ceci permet d'assurer un comportement déterministe durant l'exécution du programme.
- La phase nominale qui représente la phase périodique d'exécution du programme. Elle est composée de trois étapes : la lecture des entrées, le calcul des sorties et la mise à jour de la mémoire du programme. Dans un contexte synchrone, une période d'exécution du programme correspond à un cycle.

Afin de rendre l'analyse de testabilité la plus représentative de ce fonctionnement, nous proposons une méthodologie d'intégration de ces deux phases (initialisation et nominale) au cours de la détermination des écoulements. Ceci passe par la définition des deux classes d'écoulements correspondant respectivement aux écoulements activés au cours de l'initialisation et ceux activés au cours du fonctionnement nominal du système.

Identification des opérateurs « sensibles » aux deux phases

Les opérateurs utilisés pour la spécification des systèmes dans l'environnement SCADE peuvent être répartis en deux principales catégories. La première est constituée d'opérateurs « strictement combinatoires ». Le comportement de ces opérateurs dépend uniquement de l'information qu'ils reçoivent au niveau de leur entrée au cours d'un cycle d'exécution du système. En effet, ces types d'opérateur ne possèdent pas de mémoire interne. La seconde catégorie correspond aux opérateurs temporels présentés dans les sections précédentes. L'influence de la mémoire interne sur le comportement des opérateurs temporels leur confère une particularité au niveau de la spécification détaillée d'un système. L'état de l'ensemble des mémoires internes de ces opérateurs, en un cycle, représente l'état de la mémoire du système en ce cycle. De ce fait, l'initialisation du système revient à initialiser une partie de la mémoire interne de l'ensemble des opérateurs temporels utilisés pour sa spécification.

Par conséquent, les opérateurs temporels sont les opérateurs « sensibles » aux phases d'initialisation et nominale des systèmes. Ils sont les opérateurs à considérer pour l'intégration de la classification des écoulements dans l'approche d'analyse de testabilité. Les opérateurs de type bascule sont utilisés pour illustrer cette sensibilité aux différentes phases d'exécution d'un système.

Opérateurs de type bascule : BASCR, BASCS

L'opérateur BASCR (respectivement BASCS) est une bascule avec l'entrée RESET (respectivement SET) prioritaire (**Figure 3.16**).

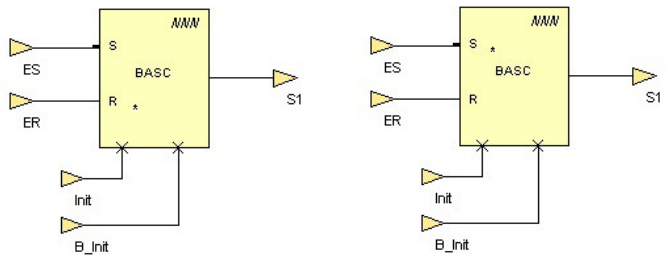


Figure 3.16 : Représentation SCADE de BASCR et BASCS

Pour chacun de ces opérateurs, l'influence des entrées d'initialisation ($Init$ et B_Init) peut être décrite comme suit :

$$Si\ B\_Init(k) = True\ Alors\ S1(k) = Init(k)$$
$$Sinon\ "Calcul\ normal\ de\ l'état\ de\ la\ sortie\ S1"$$

Cette description signifie que les entrées SET et $RESET$ des deux bascules n'influent pas sur le calcul de la sortie $S1$ lorsque l'entrée d'initialisation B_Init est *True*.

Tout comme les opérateurs de type bascule, les d'opérateurs temporels, considérés dans cette thèse, comportent des entrées qui n'interviennent dans le calcul de l'état de la sortie que lors de l'occurrence d'initialisation. Ceci permet de dissocier les flots d'information selon la phase d'exécution du système.

Technique de classification des écoulements

Dans notre approche d'analyse de testabilité, les écoulements sont déterminés à partir du modèle de transfert d'information (MTI) construit pour le système. Ce MTI est obtenu par la composition des MTI élémentaires des opérateurs utilisés dans la spécification du système. La technique de classification des écoulements proposée dans cette section est basée sur la modélisation des MTI élémentaires des opérateurs « sensibles » (identifiés dans la section précédente) aux phases d'exécution de la spécification d'un système. En effet, la modélisation du MTI de ces opérateurs temporels prend en compte les deux aspects (initialisation et nominal). La **Figure 3.17** ci-dessous, représentant celle de l'opérateur de type bascule « *BASCS* », sera utilisée pour illustrer cette technique de classement des écoulements.

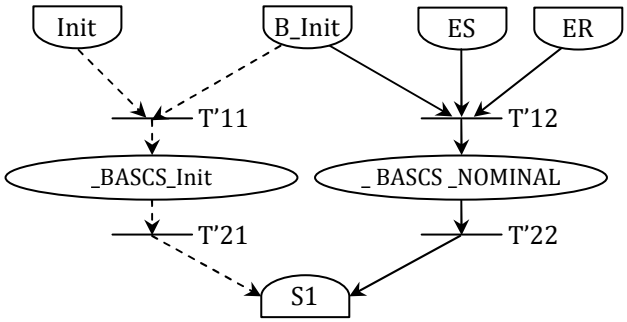


Figure 3.17 : MTI de BASCS avec initialisation

Le MTI de l'opérateur « *BASCS* » est composé de deux modules fonctionnels. Chaque module représente une des deux phases d'exécution (initialisation et nominale) de l'opérateur. En considérant cette modélisation, nous observons que l'activation de la sortie S1 nécessite un seul flot d'information provenant soit du module « *_ BASCS _ Init* » ou du module « *_ BASCS _ NOMINAL* » (la mode attribution, section 3.2.1). En effet, selon la phase d'exécution, un seul parmi les deux flots est pris en compte par la sortie S1 de l'opérateur. De ce fait, la classification des écoulements consiste à dissocier l'ensemble des écoulements contenant les modules représentant la phase d'initialisation des opérateurs et l'ensemble des écoulements contenant ceux de la phase nominale.

En considérant l'exemple ci-dessus (**Figure 3.18**), la classification consiste à regrouper d'une part, l'ensemble des flots d'information dépendant de la place « *Init* » (lorsque la valeur de la commande d'initialisation « *B _ Init* » est « *True* ») et d'autre part, celui dépendant des

places « ES et ER ». Ces deux ensembles ont en commun les flots d'information dépendant de la place « B_Init » qui joue le rôle de la « commande » d'initialisation.

La figure ci-dessous est utilisée pour illustrer la technique de classification des écoulements. Elle représente le diagramme d'opérateurs SCADE définissant la variable « Var_S1 ».

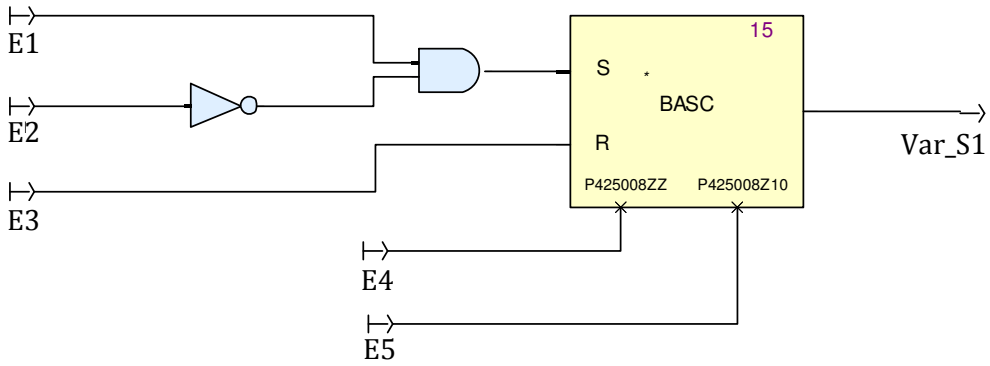


Figure 3.18 : Exemple SCADE d'illustration de la classification des écoulements

Le modèle de transfert d'information correspondant à ce diagramme est représenté ci-dessous.

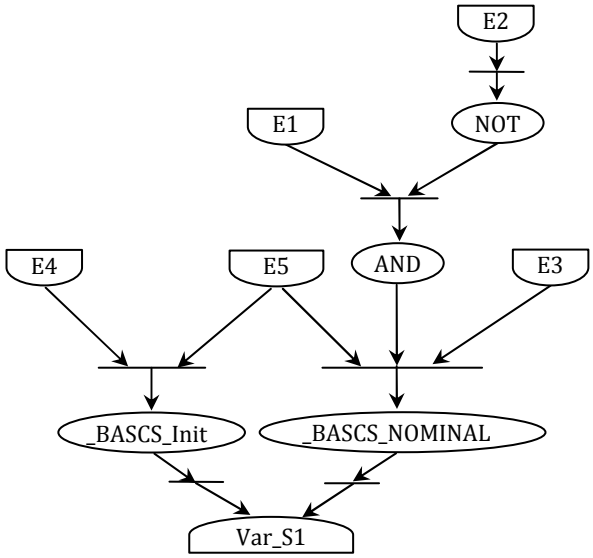


Figure 3.19 : MTI de l'exemple d'illustration de la classification des écoulements

Deux écoulements peuvent être déterminés à partir de ce MTI. Ils sont repartis selon les deux phases d'exécution :

- L'écoulement correspondant à la phase d'initialisation contient les places suivantes : $\{E4\ ;\ E5\ ;\ \_BASCS\_Init\ ;\ Var\_S1\}$
- L'écoulement correspondant à la phase nominale contient les places suivantes : $\{E2\ ;\ NOT\ ;\ E1\ ;\ AND\ ;\ E5\ ;\ E3\ ;\ \_BASCS\_NOMINAL\ ;\ Var\_S1\}$

3.3.4 Synthèse

Dans cette section, nous avons proposé des méthodes de modélisation, d'une part pour les opérateurs d'aiguillage à commande multiple quel que soit le nombre de commandes, et d'autre part pour les opérateurs temporels, en vue de l'analyse de testabilité.

Les opérateurs d'aiguillage ont été considérés comme une cascade d'opérateurs d'aiguillage à commande unique. Une méthode d'évaluation du CPI a été définie pour ces types d'opérateur. Elle permet de déterminer le coefficient de perte d'information associé à chacun des modules fonctionnels définis dans leur modèle de transfert d'information élémentaire.

Deux approches de modélisation ont été développées (avec et sans la prise en compte des entrées liées à la (ré-) initialisation des opérateurs) pour les opérateurs temporels. Elles fournissent, chacune, l'ensemble des informations permettant l'intégration de l'aspect temporel à l'analyse de testabilité. Cette intégration passe par :

- la prise en compte des modèles de transfert d'information élémentaire associés aux opérateurs temporels dans le modèle global construit pour un système ;
- la mise en œuvre des techniques de calcul des coefficients de perte d'information proposées pour chaque type d'opérateurs temporels.

Une méthode de classification des écoulements a été également proposée afin de dissocier les écoulements de la phase d'initialisation des écoulements de la phase nominale du fonctionnement des systèmes.

Les modélisations et techniques décrites dans cette section assurent la cohérence entre le comportement fonctionnel et la représentation « flots d'information » de ces opérateurs au cours de l'analyse. Par conséquent, les écoulements déterminés donneront une représentation plus fidèle des fonctions élémentaires contenues dans le système. Les méthodes de calcul des CPI pour chacun des types d'opérateur permettront de mener des expérimentations avant de conclure sur la pertinence des mesures de testabilité dans le contexte AIRBUS.

3.4 Méthodologie d'aide à la validation

La validation des spécifications formelles est le processus permettant d'assurer que la description des exigences fonctionnelles est suffisamment correcte et complète afin que le système soit conforme aux exigences de navigabilité en vigueur. Ceci revient à répondre à la question suivante : « sommes-nous en train de construire le bon produit ? ». Nous avons

présenté le cycle de validation des spécifications d'un système explicitant les liens entre les différentes activés dans la section 3.1 (**Figure 3.1**). Ces liens se traduisent par des activités de couverture basée sur des informations de traçabilité tout au long du cycle de validation des systèmes. La **Figure 3.20** représente ce cycle avec les activités de couverture entre les différentes étapes.

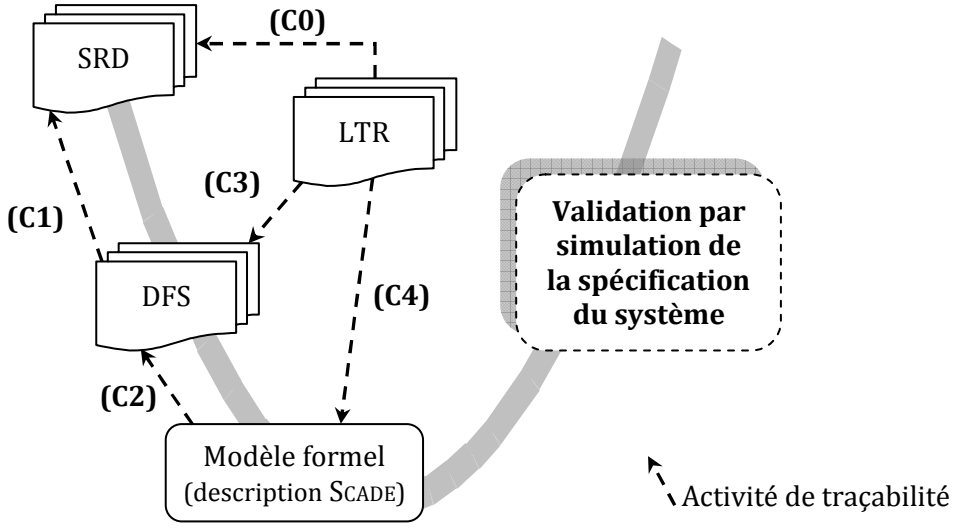


Figure 3.20 : Activités de couverture du cycle de validation du système

- **(C0)** représente l'activité d'évaluation de la couverture des tests définis dans le LTR (Lab Tests Request) vis-à-vis des exigences décrites dans le SRD (System Requirements Document). L'objectif de cette analyse est de vérifier que des tests ont été définis pour la vérification des exigences fonctionnelles du système. (C0) est réalisée à partir de documents textuels et des informations de traçabilité gérées par l'outil DOORS [Tel].
- **(C1)** correspond à l'activité d'évaluation de la couverture de la spécification des fonctions spécifiées dans le DFS (Detailed Functional Specification) ou cahier de fonctions vis-à-vis des exigences définies dans le SRD. Elle consiste à vérifier que l'ensemble des exigences fonctionnelles et les exigences dérivées a été pris en compte dans le DFS. Cette activité de couverture est réalisée à partir de documents textuels et des informations de traçabilité gérées par l'outil DOORS.
- **(C2)** correspond à l'activité d'évaluation de la couverture du modèle formel vis-à-vis des exigences spécifiées dans le DFS. Elle consiste à vérifier que le modèle formel décrit uniquement les fonctions décrites dans le DFS. Ceci permet d'une part, de détecter les fonctions qui ne sont pas complètement décrites dans le modèle et d'autre part, de détecter d'éventuelles parties du modèle qui n'ont pas de rapport avec les fonctions du DFS. Elle est également réalisée à partir de documents textuels, du modèle formel et des informations de traçabilité gérées par l'outil DOORS.

- **(C3)** et **(C4)** correspondent respectivement aux activités d'évaluation de la couverture des tests définis dans le LTR vis-à-vis des fonctions du DFS et du modèle formel. Ceci permet d'assurer d'une part, de la définition de tests pertinents et non redondants pour la phase de validation par simulation des spécifications du système. D'autre part, ces activités permettent de détecter des tests manquants pour la validation lorsqu'il n'existe pas de tests pour la validation de certaines fonctions. Elles sont réalisées à partir de documents textuels, le modèle formel et des informations de traçabilité gérées par l'outil DOORS.

La méthodologie d'aide à la validation proposée dans cette section décrit les méthodes permettant de fournir des informations, à l'aide de techniques de couverture basées sur les écoulements : d'une part, sur la complétude du modèle formel spécifiant les exigences définies dans le DFS et d'autre part sur la pertinence des tests définis dans le LTR pour la validation. Cette méthodologie est composée de trois principales parties : l'application d'une approche d'analyse de testabilité (section 3.4.1), l'évaluation de la couverture (C2) du modèle formel vis-à-vis des exigences à l'aide des écoulements à l'issue de l'analyse de testabilité (section 3.4.2) et l'évaluation de la couverture (C3) et (C4) des tests vis-à-vis des écoulements associés aux exigences (section 3.4.3). Par ailleurs, les activités de couverture (C0) et (C1) ne sont pas couvertes par la méthodologie définie. La **Figure 3.21** ci-dessous donne une vue globale de la représentation des différentes étapes de cette méthodologie. Elle présuppose un lien entre les écoulements et les exigences fonctionnelles du système.

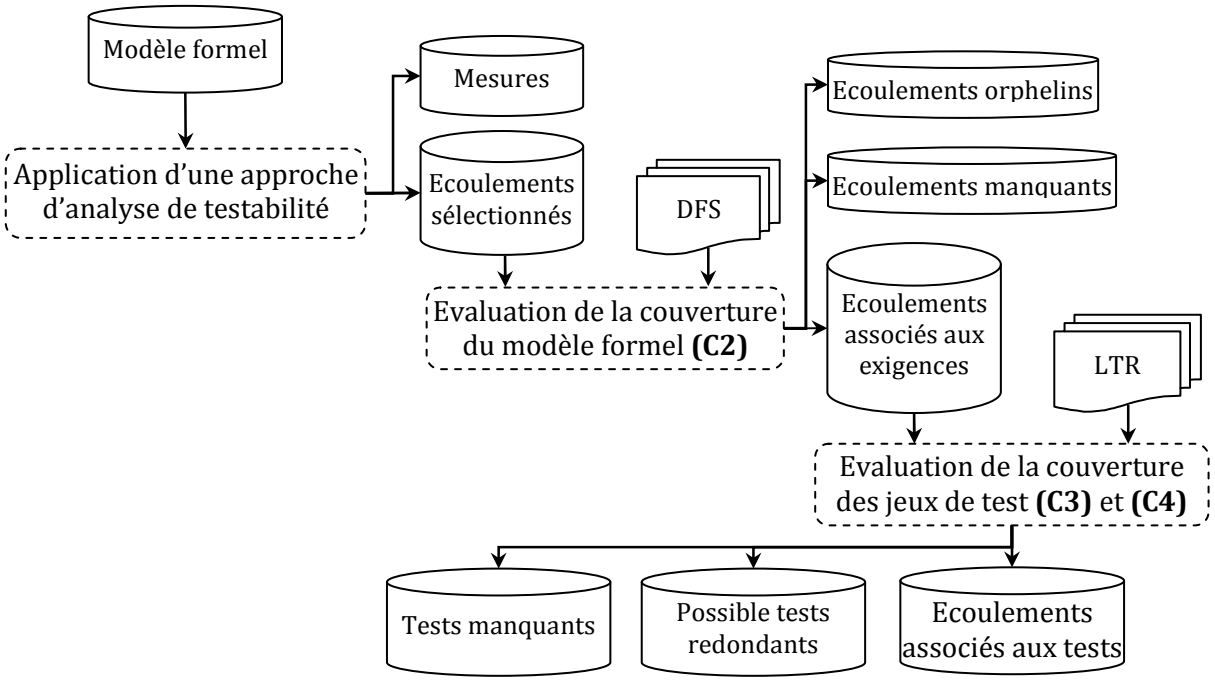


Figure 3.21 : Méthodologie d'aide à la validation des systèmes

3.4.1 Application d'une approche d'analyse de testabilité

L'analyse de testabilité proposée est composée de deux principales étapes : l'analyse des flots de données contenus dans le système pour déterminer les écoulements et la simulation des pertes d'information dans le système pour calculer les mesures. Nous nous concentrons sur la détermination des écoulements dans cette section. La pertinence des mesures dans le contexte AIRBUS est évaluée dans la section 3.5 dédiée aux expérimentations.

Les informations liées aux écoulements à l'issue de l'analyse de testabilité ne sont utiles que si la méthode de détermination de ces écoulements reflète le processus de test pour la validation des systèmes. En d'autres termes, ces informations ne sont utiles que s'il est possible d'établir une relation entre les fonctions du système et les écoulements afin de pouvoir aider les activités de couverture matérialisées par les liens C2, C3 et C4 de la **Figure 3.20**. Nous avons défini trois approches d'analyse de testabilité en considérant respectivement chaque variable de sortie définie, les fonctions décrites et les parties indépendantes en termes de flot de données dans le modèle formel. Ces approches sont appliquées sur la partie des spécifications détaillées du système à valider. Les approches sont illustrées à l'aide de la **Figure 3.22**. Elle correspond à une spécification SCADE définissant trois variables de sortie (O1, O2, O3).

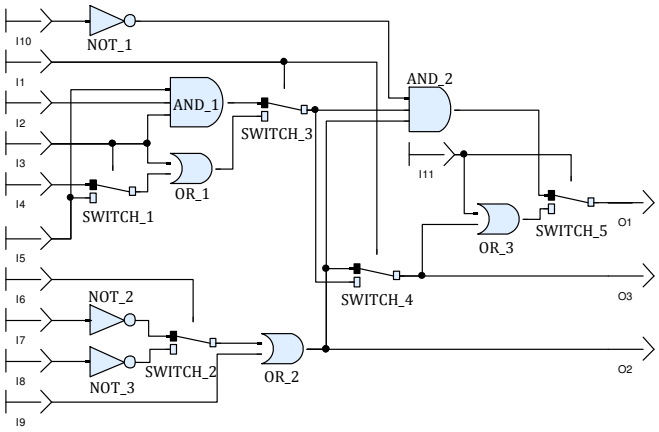


Figure 3.22 : Modèle SCADE pour l'illustration des approches d'analyse de testabilité

Cette spécification contient seize écoulements associés aux trois sorties définies. Dix écoulements pour la sortie 01, deux pour O2 et quatre pour O3. Nous proposons, ci-dessous, une description des écoulements « simples » Ecl_7, Ecl_{11} et Ecl_{15} associés respectivement aux sorties 01, 03 et 02.

$$Ecl_7 = \{I_1, I_6, I_7, I_9, I_{11} \mid NOT\_2, then.SWITCH\_2, OR\_2, then.SWITCH\_4, OR\_3, else.SWITCH\_5 \mid O1\}$$

$$Ecl_{11} = \{I_1, I_6, I_7, I_9 \mid NOT\_2, then.SWITCH\_2, OR\_2, then.SWITCH\_4 \mid O3\}$$

$$Ecl_{15} = \{I_6, I_7, I_9 \mid NOT\_2, then.SWITCH\_2, OR\_2 \mid O2\}$$

Dans la suite, nous considérons les annotations suivantes :

- O représente l'ensemble des sorties définies par une spécification $Spec$. Cet ensemble est composé des sorties O1, O2 et O3 dans la **Figure 3.22**.
- $O_i(Ecl)$ désigne l'ensemble des écoulements associés à la sortie O_i définie dans $Spec$. L'ensemble $O1(Ecl)$ contient 10 écoulements, $O2(Ecl)$ contient 2 écoulements et $O3(Ecl)$ contient 4 écoulements dans la **Figure 3.22**.
- $Spec(O_i, O_j \cdots)$ désigne la partie de la spécification $Spec$ dont dépendent le sous-ensemble de sorties $\{O_i, O_j \cdots\}$ en termes d'opérateurs utilisés et les flots de données. Par exemple, $Spec(O2)$ peut être représentée par l'ensemble d'opérateurs défini comme suit : $\{NOT\_2, NOT\_3, SWITCH\_2, OR\_2\}$.
- $STM(Ecl)$ représente le sous-ensemble des écoulements choisis par la stratégie Start-Small (section 3.2.3) à partir d'un ensemble d'écoulements Ecl.

A. Approche par variable de sortie (AVS)

Cette approche propose une analyse de la spécification formelle d'un système en considérant les variables de sortie indépendamment les unes des autres. La partie de la spécification liée à chaque variable de sortie est extraite et analysée. Les écoulements déterminés à l'issue de chaque analyse reflètent la testabilité locale à une variable sortie de la spécification formelle. L'approche par variable de sortie est adaptée au test unitaire. Une description algorithmique de cette approche se trouve ci-dessous.

Procédure AVS
Début
. *Pour chaque* $O_i \in O$ *faire*
. . *Extraire* $Spec(O_i)$
. . *Déterminer* $O_i(Ecl)$
. . *Appliquer la stratégie Start-Small sur* $O_i(Ecl)$ *pour obtenir* $STM[O_i(Ecl)]$
. *Fin_Pour*
Fin

Le résultat de l'application de cette approche sur le modèle de la **Figure 3.22** en termes de nombre d'écoulements sélectionnés est présenté dans le tableau ci-dessous.

Tableau 3.4 : Nombre d'écoulements avec l'approche par variable de sortie

	Variables de sortie		
	O1	O2	O3
Nombre d'écoulements sélectionnés	6	2	4

B. Approche par fonction (AF)

Cette approche consiste à analyser l'ensemble des variables de sortie associées à une fonction du système. Une fonction est définie dans le Cahier de Fonctions, elle correspond à un sous-ensemble de fonctionnalités que doit assurer d'une exigence fonctionnelle du système. La partie de la spécification liée à chaque fonction est extraite et une analyse est effectuée pour chacune de ces fonctions. Les écoulements déterminés à l'issue de chaque analyse reflètent la testabilité des fonctions décrites dans la spécification formelle. Une description algorithmique de cette approche se trouve ci-dessous.

Soient $O(fonc_i)=\{O_j, O_k, \cdots\}$ l'ensemble des variables de sortie $(O_j, O_k, \cdots)$ définies pour la description d'une fonction $fonc_i$ dans la spécification $Spec$ et $F=\{fonc_i, \cdots\}$ l'ensemble des fonctions décrites dans $Spec$.

```
Procédure AF
  Début
  .   Pour chaque fonc_i ∈ F faire
  .   .   FEcl ← ∅ /* Ensemble vide d'écoulements */
  .   .   Extraire Spec(O(fonc_i))
  .   .   Pour chaque O_j ∈ O(fonc_i) faire
  .   .   .   Déterminer O_j(Ecl)
  .   .   .   FEcl ← FEcl ∪ Oj(Ecl)
  .   .   Fin_Pour
  .   .   Appliquer la stratégie Start-Small sur FEcl pour obtenir STM(FEcl)
  .   Fin_Pour
  Fin
```

Le résultat de l'application de cette approche sur le modèle de la **Figure 3.22** en termes de nombre d'écoulements sélectionnés est présenté dans le tableau ci-dessous. Nous supposons que la variable de sortie O1 réalise une fonction et les variables O2 et O3 réalisent une autre fonction. De ce fait, la stratégie Start-Small est appliquée indépendamment les deux ensembles d'écoulements : $O1(Ecl)$ et $O2(Ecl)\cup O3(Ecl)$.

Tableau 3.5 : Nombre d'écoulements avec l'approche par fonction

	Fonctions		
	Première fonction	Deuxième fonction	
	O1	O2	03
Nombre d'écoulements par variable	6	2	3
Nombre d'écoulements par fonction	6	5	

C. Approche par composant (AC)

Cette troisième approche propose une analyse en considérant les composantes connexes de la spécification formelle d'un système. En effet, la spécification flot de données est considérée

comme un graphe. Une composante connexe d'un graphe représente un sous-ensemble de ses éléments interdépendants en termes de flots de données. L'approche consiste à extraire et à analyser de façon indépendante les différentes composantes connexes. Les écoulements déterminés à l'issue de chaque analyse reflètent la testabilité des fonctions décrites dans la spécification formelle. Une description algorithmique de cette approche se trouve ci-dessous.

Pour cette description, nous utilisons une fonction $CompConnexes$ qui détermine les différentes composantes connexes d'une spécification $Spec$. Cette fonction retourne la liste des sous-ensembles de sorties $O(CC_i)$ liées à chaque composante connexe CC_i.

```
Procédure AC
  Début
  .   ListeCC ← CompConnexes(Spec)
  .   Pour chaque O(CC_i) de ListeCC faire
  .   .   CCEcl ← ∅ /* Ensemble vide d'écoulements */
  .   .   Extraire Spec(O(CC_i))
  .   .   Pour chaque O_j ∈ O(CC_i) faire
  .   .   .   Déterminer O_j(Ecl)
  .   .   .   CCEcl ← CCEcl ∪ Oj(Ecl)
  .   .   Fin_Pour
  .   .   Appliquer la stratégie Start-Small sur CCEcl pour obtenir STM(CCEcl)
  .   Fin_Pour
  Fin
```

Le résultat de l'application de cette approche sur le modèle de la **Figure 3.22** en termes de nombre d'écoulements sélectionnés est présenté dans le tableau ci-dessous. La spécification est composée d'une seule composante connexe. De ce fait, la stratégie Start-Small est appliquée sur l'ensemble d'écoulements $O1(Ecl) \cup O2(Ecl) \cup O3(Ecl)$.

Tableau 3.6 : Nombre d'écoulements avec l'approche par composant

	Composant		
	O1	O2	O3
Nombre d'écoulements par variable	5	2	1
Nombre d'écoulements par composante	8		

D. Synthèse

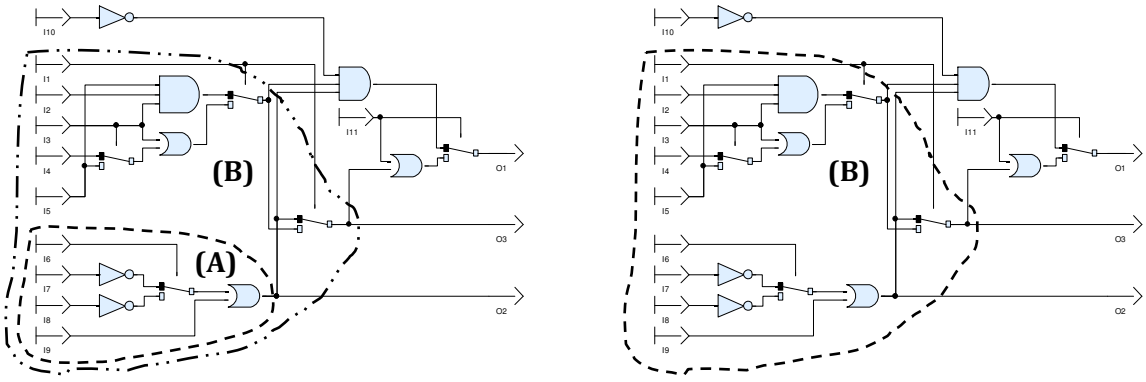
De façon globale, le nombre des écoulements choisis décroît de l'approche par variable à l'approche par composant (**Tableau 3.4**, **Tableau 3.5** et **Tableau 3.6**). Ceci s'explique par le fait que la description de certaines variables de sortie partage des parties dites communes du modèle formel. Deux variables de sortie O_i et O_j partage une partie commune de la spécification $Spec$ lorsque $Spec(O_i) \cap Spec(O_j) \neq \emptyset$. En effet, cette dépendance, du point de vue

opérateurs et flot de données, entre les variables de sortie a un impact sur le nombre d'écoulements déterminés et sélectionnés à l'issue de l'analyse de testabilité.

Dans l'approche par variable, ces parties communes sont analysées au cours du traitement de chaque variable de sortie qui les utilise. De ce fait, les écoulements, déterminés à l'issue de chaque analyse de testabilité, prennent en compte ces parties communes, d'où le nombre important d'écoulements choisis pour l'ensemble des variables de sortie. La **Figure 3.23** illustre cette notion de partie commune entre variables de sortie en représentant la partie **(A)** intervenant dans le calcul des trois sorties et **(B)** intervenant dans le calcul des sorties O1 et O3. En d'autres termes, $(A) = Spec(O_1) \cap Spec(O_3) \cap Spec(O_2)$ et $(B) = Spec(O_1) \cap Spec(O_3)$.

L'utilisation de l'approche par fonction permet une seule analyse des parties communes à l'ensemble de variables de sortie associées à une fonction du système. Cependant, les parties communes partagées par les différentes fonctions du système sont analysées autant de fois qu'il y a de fonctions qui les utilisent. Ceci explique la réduction du nombre d'écoulements déterminés par rapport l'approche par variable. La **Figure 3.23** illustre cette notion de partie commune entre fonctions en représentant la partie **(B)** intervenant dans le calcul des deux fonctions réalisées respectivement par O1 et (O2, O3). En d'autres termes, $(B) = Spec(O_1) \cap Spec(O_2, O_3)$.

L'approche par composant propose une méthode permettant d'analyser l'ensemble des variables de sorties définies par la même composante. En utilisant cette technique, l'ensemble du modèle SCADE est pris en compte une seule fois au cours de l'analyse testabilité. De ce fait, le nombre d'écoulements déterminés à l'issue de l'application de cette approche est inférieur à celui des deux premières approches.



Approche par variable **Approche par fonction**

Figure 3.23 : Illustration des parties communes pour les approches d'analyse

Par ailleurs, la mise en œuvre de ces trois approches introduit une nouvelle fonctionnalité dans le processus d'analyse de testabilité : l'analyse d'un certain nombre de variables de sortie ou de fonctions sans considérer le système dans sa globalité.

3.4.2 Méthode d'analyse de couverture du modèle (C2)

L'évaluation de la couverture du modèle formel, spécification SCADE dans notre contexte, vis-à-vis des exigences fonctionnelles permet d'associer les écoulements pertinents, déterminés et sélectionnés à l'issue de l'application d'une approche d'analyse de testabilité, aux exigences fonctionnelles décrites dans le DFS (Detailed Functional Specification). Elle est constituée de deux principales étapes : l'identification des variables de sortie SCADE définies pour chaque exigence et l'analyse de la couverture des exigences à partir des écoulements sélectionnés à l'aide de la stratégie Start-Small. La figure ci-dessous illustre ce processus d'évaluation de la couverture du modèle.

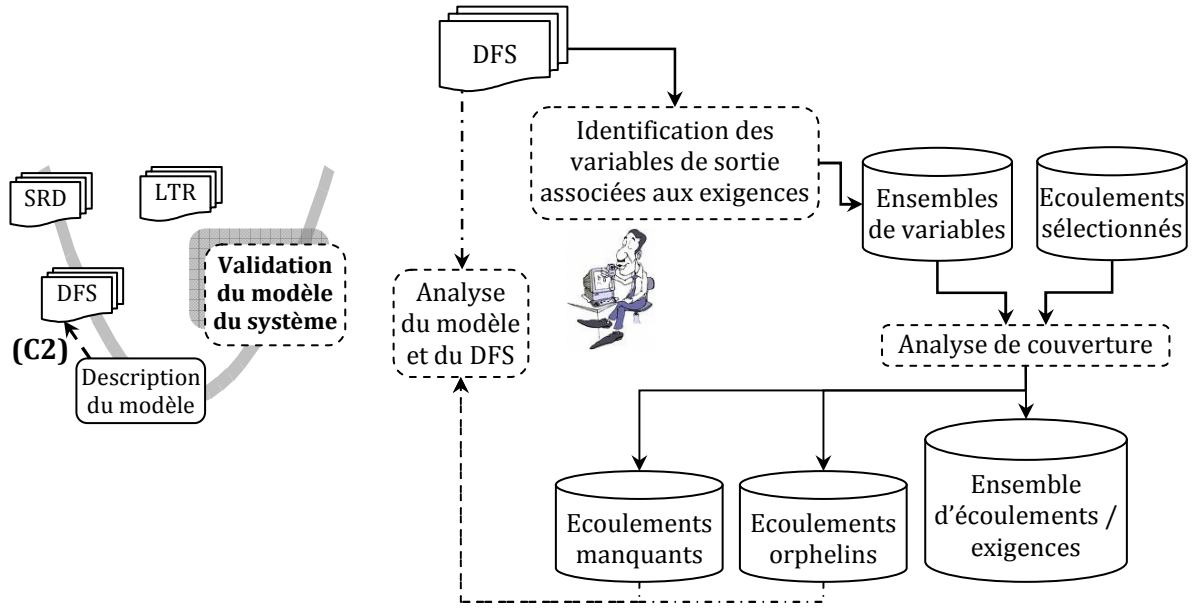


Figure 3.24 : Analyse de couverture du modèle formel

Identification des variables de sortie associées aux exigences

Une exigence fonctionnelle du système, du point de vue de la modélisation, peut être définie comme une combinaison de conditions sur les entrées correspondant à une décision affectant les sorties du système. Plusieurs fonctions du système peuvent être allouées à une exigence.

Une exigence fonctionnelle peut donc être associée à d'un ensemble de variables de sortie du système. Dans notre contexte, cette information est décrite dans le DFS. En effet, l'identification et la description d'une exigence sont suivies par la définition de l'ensemble des

variables de sortie SCADE qui lui est associée dans le DFS. Cette association est représentée à l'aide de la figure ci-dessous. Dans cette relation, une exigence fonctionnelle peut être associée à un ensemble de variables de sortie ; mais une variable de sortie ne peut pas être associée à plusieurs exigences à la fois afin de répondre aux exigences de spécification des systèmes AIRBUS. Le contraire conduirait à l'introduction de sources d'ambigüité non recommandée par la directive ABD200 d'AIRBUS [ABD06] (*Module 2 – Section 3*). Cette étape d'identification est manuelle et basée sur le DFS. Par ailleurs, elle peut être réalisée systématiquement au cours de la définition des exigences fonctionnelles.

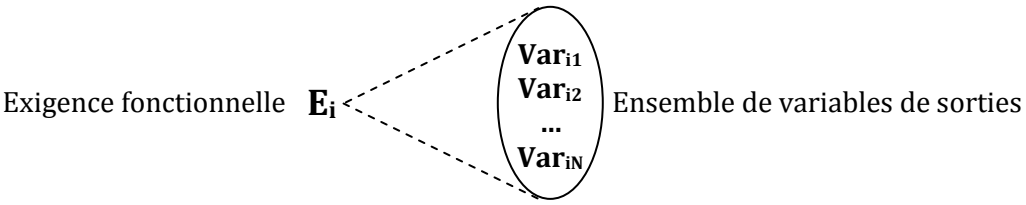


Figure 3.25 : Relation entre exigence et variables de sortie

Analyse de couverture du modèle formel

Les exigences fonctionnelles doivent être correctement et complètement modélisées dans le modèle SCADE. Cette étape d'analyse consiste à évaluer la couverture du modèle SCADE vis-à-vis des exigences en se basant sur les écoulements déterminés à l'étape d'application d'une des trois approches (par variable de sortie, par fonction ou par composant) définies dans la section 3.4.1. En effet, les écoulements assurent la couverture des fonctions élémentaires décrites dans le modèle. L'analyse permet d'associer à chaque exigence fonctionnelle les écoulements pertinents correspondant à ses variables de sortie. Elle fournit des informations sur :

- Les sous-ensembles d'écoulements associés à chaque exigence fonctionnelle décrite dans le DFS. La **Figure 3.26** illustre cette notion d'écoulements associés à une exigence. Ces écoulements permettent de donner des indications sur l'effort de test de chacun des exigences en termes de la complexité des opérateurs qu'ils activent.
- Les **écoulements orphelins** : un écoulement est dit orphelin lorsqu'il n'est associé à aucune exigence fonctionnelle. L'existence de ce type d'écoulements est liée à la présence de variables de sortie dans le modèle SCADE non identifiées dans le DFS. Deux interprétations possibles peuvent être faites de cette situation : soit, les parties du modèle SCADE correspondant à ces écoulements sont liées à certaines exigences dérivées ; soit ces parties du modèle SCADE sont superflues. L'écoulement « $\mathbf{Eclt_x}$ » représente un écoulement orphelin dans la **Figure 3.26**. En effet, cet écoulement n'est associé à aucune variable de sortie identifiée dans le DFS.

– Les **écoulements manquants** : Cette situation est provoquée par l'absence d'écoulements associés à certaines variables de sortie définies dans le DFS. Elle peut être interprétée par la définition d'exigences n'ayant pas de contrepartie dans le modèle SCADE. La variable « $\mathbf{Var_{j1}}$ » de la **Figure 3.26** est une illustration de l'existence des écoulements manquants.

Analyse du modèle et du DFS

Cette étape, manuelle, correspond à l'analyse du modèle SCADE et du DFS par les concepteurs afin de vérifier les informations fournies à l'issue de l'évaluation de la couverture. Ceci permet : soit de supprimer certaines parties du modèle SCADE lorsque les écoulements orphelins sont superflus ; soit de compléter le modèle lorsque la définition de certaines variables de sortie est manquante.

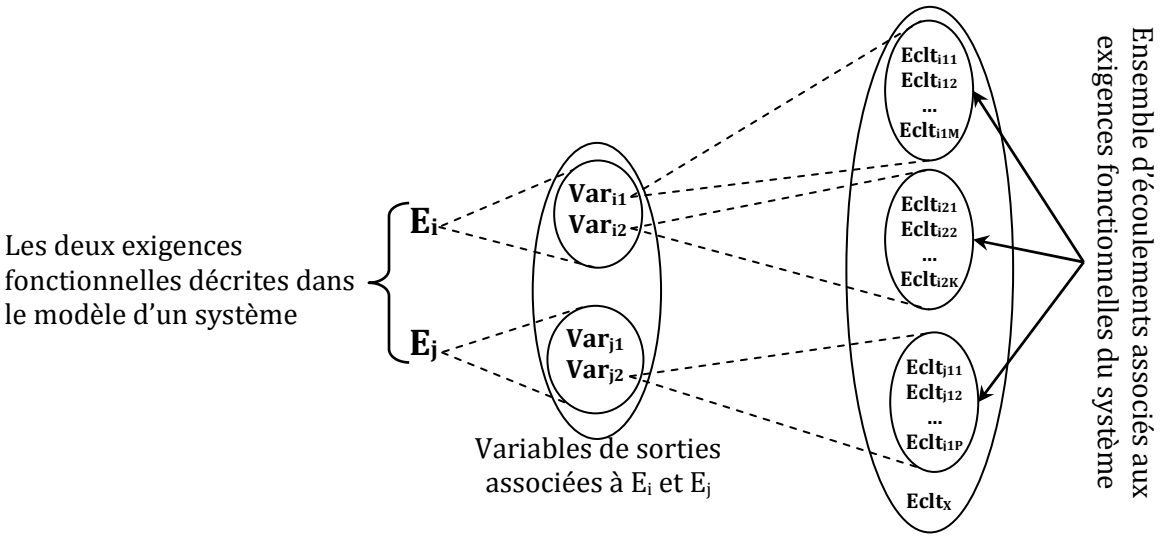


Figure 3.26 : Illustrations des informations d'analyse de couverture du modèle formel

3.4.3 Méthode d'analyse de couverture des jeux de test (C3 et C4)

L'évaluation de la couverture des jeux de test vis-à-vis du modèle formel et implicitement des exigences fonctionnelles permet d'associer les tests définis dans le LTR aux écoulements correspondant aux exigences fonctionnelles décrites dans le DFS. Cette activité est réalisée à la suite de l'évaluation de la couverture du modèle vis-à-vis des exigences. Elle est constituée de deux principales étapes : l'identification des tests définis dans le LTR pour chaque exigence et l'analyse de la couverture des tests à partir des écoulements associés aux exigences. La figure ci-dessous illustre ce processus d'évaluation de la couverture des tests.

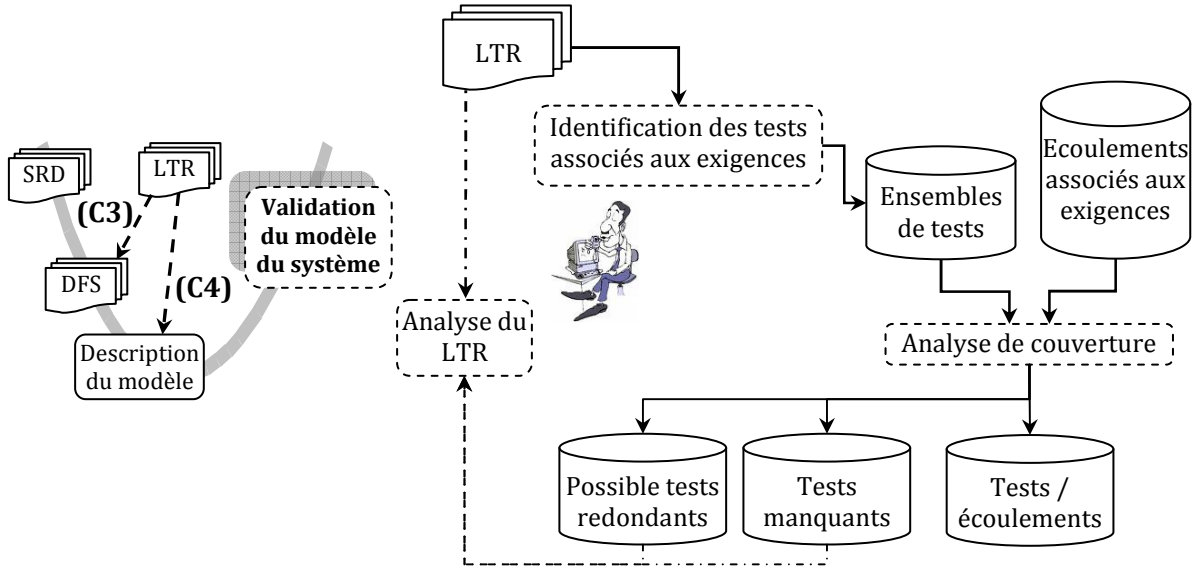


Figure 3.27 : Analyse de couverture des tests

Identification des tests associés aux exigences

Cette étape consiste à identifier les tests associés à chaque exigence décrite dans le DFS. En effet, les informations de traçabilité disponibles au niveau du LTR permettent d'établir ce lien entre les tests et les exigences. Dans notre contexte, plusieurs jeux de test peuvent être nécessaires à la vérification d'une exigence. Par ailleurs, un jeu test est défini et exécuté pour la vérification d'une exigence. La figure ci-dessous représente cette relation entre les exigences fonctionnelles et les tests.

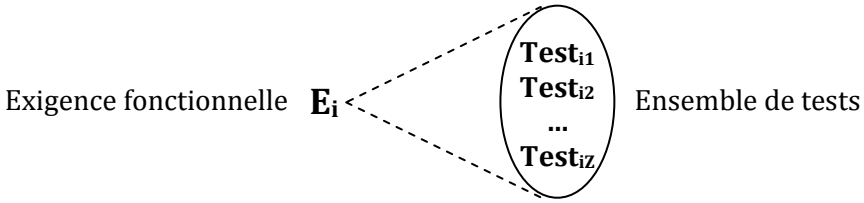


Figure 3.28 : Relation entre les exigences et les tests

Analyse de couverture des tests

L'ensemble des exigences fonctionnelles modélisées doit être vérifié au cours de la validation du modèle formel. L'analyse de couverture des tests est réalisée à l'aide de l'ensemble des tests et de l'ensemble des écoulements associés à chaque exigence. Elle consiste à associer les écoulements aux tests qui les activent. Cette notion de couverture des tests est liée au critère de couverture atteignable à partir du type de modélisation des MTI élémentaires choisi pour les opérateurs du système. Le critère considéré au cours de la modélisation des MTI élémentaires dans ce manuscrit est celui de la couverture des branches et des décisions de type commande d'aiguillage (section 3.2.1). Cette étape d'analyse fournit les informations sur :

- Les tests associés aux écoulements identifiés pour les exigences fonctionnelles. Ceci permet d'évaluer la couverture des tests vis-à-vis des écoulements et par conséquent des exigences fonctionnelles. Les tests « **$Test_{i1}$** et **$Test_{ij}$** » sont associés à l'écoulement « **$Eclt_{i1}$** » dans la **Figure 3.29** ci-dessous.
- Les tests manquants lorsqu'il n'existe pas de tests associés à certains écoulements. L'écoulement « **$Eclt_{i2}$** » illustre cette notion de tests manquants dans la **Figure 3.29** ci-dessous. En effet, aucun test n'est associé à cet écoulement.
- Les probables tests redondants lorsque plusieurs tests sont associés à un même écoulement. Les tests « **$Test_{i1}$**, **$Test_{ij}$** » et « **$Test_{i2}$**, **$Test_{iZ}$** » illustre cette notion de probables tests redondants dans la **Figure 3.29** ci-dessous. Cependant, une analyse fonctionnelle doit être réalisée dans le but de vérifier si ces tests sont fonctionnellement redondants.

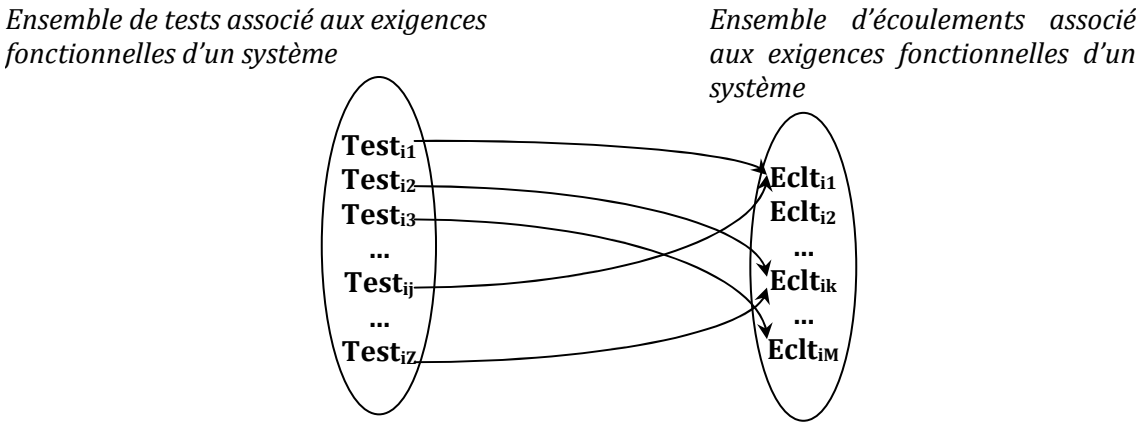


Figure 3.29 : Illustrations des informations d'analyse de couverture des tests

Analyse du LTR

Cette étape, manuelle, correspond à l'analyse du LTR par les concepteurs de test afin de vérifier les informations fournies à l'issue de l'évaluation de la couverture des tests. Cette activité permet la définition de tests additionnels afin de combler les tests manquants et la suppression des tests lorsqu'ils se révèlent fonctionnellement redondants.

3.5 Expérimentations

Dans ce sous chapitre, nous présentons la démarche ainsi que le résultat des expérimentations réalisées autour de l'aide à la validation des spécifications systèmes dans le cadre de la thèse. Les systèmes considérés dans cette étude sont ceux qui sont spécifiés sous le formalisme flot de données synchrones « SCADE ». Les travaux se sont effectués en collaboration avec le département des « commandes de vol » d'AIRBUS OPERATIONS (site de Toulouse). Le cas

d'étude fourni est constitué de deux systèmes extraits de la spécification détaillée des systèmes du pilote automatique de l'A350XWB. L'objectif de ces expérimentions est d'aider à l'évaluation de :

- la couverture des exigences fonctionnelles du système vis-à-vis de la spécification détaillée SCADE ;
- la couverture des tests définis pour la validation des exigences vis-à-vis de la spécification détaillée SCADE ;
- la pertinence des mesures de contrôlabilité et d'observabilité calculées à partir de la quantification de la perte d'information dans la spécification détaillée.

Les deux systèmes qui composent le cas d'étude sont : le système gérant les transitions entre les modes latéraux (LM1) et le système gérant les transitions entre les modes longitudinaux (LM2) du pilote automatique de l'A350XWB. Pour cette étude, les éléments suivants ont été fournis :

- un Cahier de Fonctions (DFS) contenant une description détaillée et synthétique des fonctions relatives à la transcription des exigences liées aux transitions entre ces deux modes. Ce document contient également les informations de traçabilité entre, d'une part les exigences et les fonctions et d'autre part, les fonctions et les planches SCADE issues de la spécification détaillée ;
- un ensemble de planches SCADE correspondant à la description des fonctions décrites dans le Cahier de Fonctions ;
- un fichier « TEMPO » contenant la séquence d'exécution des planches SCADE (aspect multi-cycles) ;
- un LTR (Lab Test Request) contenant les jeux d'essais définis pour la validation de ces planches SCADE.

3.5.1 Démarche des expérimentations

Les principales étapes de la démarche définie pour les expérimentations menées dans le cadre de notre étude sont présentées ci-dessous (**Figure 3.30**). Elle est composée d'une phase d'assemblage des planches SCADE, de la traduction de la description SCADE pour l'analyse de testabilité, de la phase d'application de la méthodologie définie dans ce chapitre.

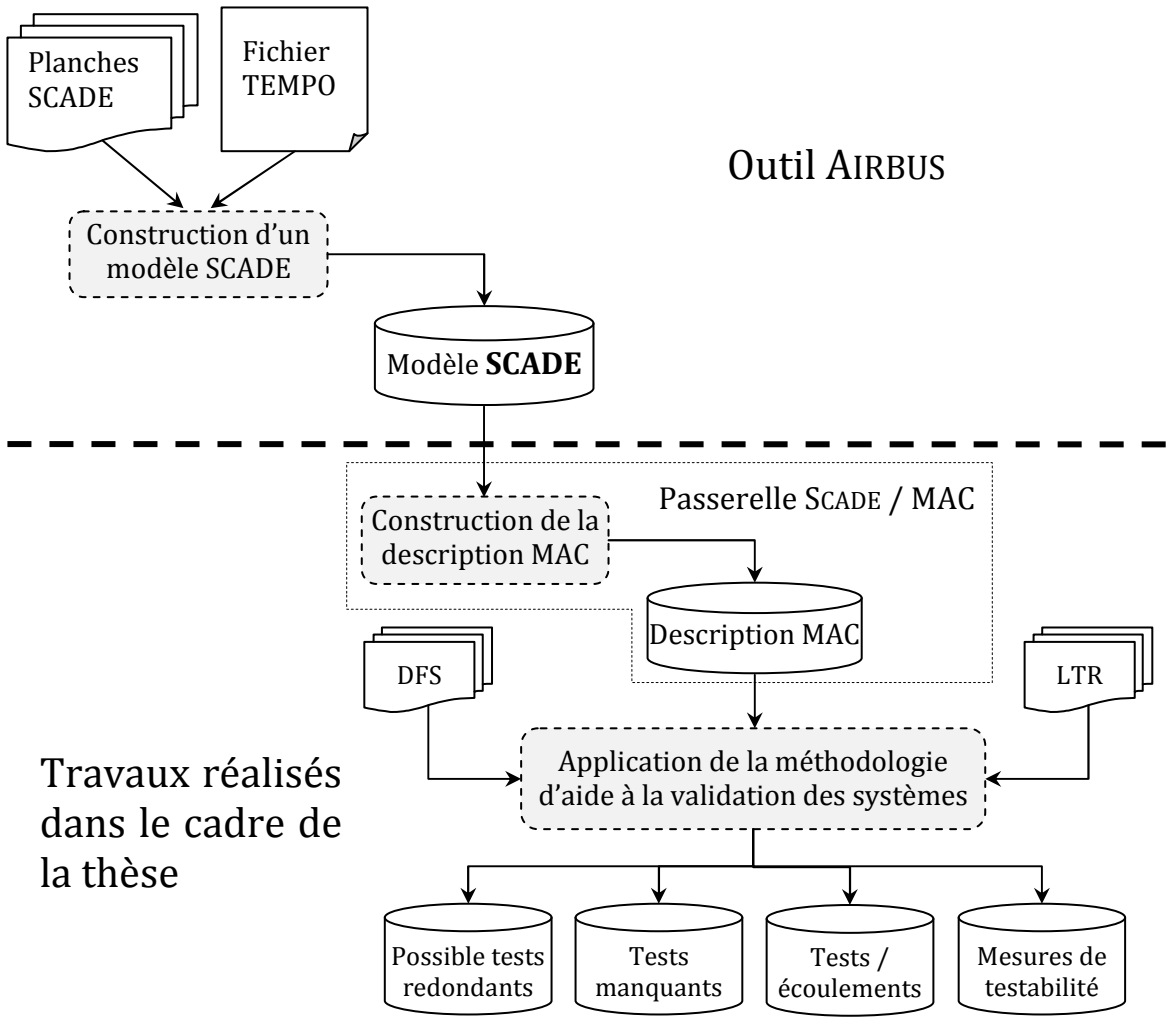


Figure 3.30 : Démarche d'analyse des systèmes

Assemblage des planches SCADE

Cette étape correspond à la phase d'assemblage des planches ou nœuds SCADE décrits au cours de la transcription des exigences liées aux systèmes à analyser. Les principes de cette phase d'assemblage sont décrits en annexe (section B.). Cet assemblage est effectué à l'aide d'un outil AIRBUS « *BuildSystem* » développé par Esterel Technology pour les systèmes de commande de vol. Cet outil permet de créer un modèle SCADE avec un nœud principal à partir d'un ensemble de nœuds SCADE pour reproduire le comportement global d'un système. Afin de respecter les contraintes de causalité dans le modèle global, un fichier « TEMPO » contenant la séquence d'exécution des planches SCADE modélisant l'aspect multi-cycles est fourni à l'outil « *BuildSystem* ».

Deux modèles ont été construits, l'un pour les modes latéraux (LM1) et le second pour les modes longitudinaux (LM2), dans le cadre de nos expérimentations. Le tableau ci-dessous présente quelques indications relatives à ces deux modèles.

Tableau 3.7 : Nombre de planches SCADE et de variables de sortie de LM1 et LM2

	LM1	LM2
Nombre de planches SCADE	69	146
Nombre de variables de sortie	98	132

Construction de la description MAC

Les spécifications flot de données sont décrites à l'aide de différents environnements de développement. Chaque environnement a son propre langage de description de système. Afin de faciliter l'extension de l'outil MAC (Module d'Analyse de complexité), qui encapsule la technologie SATAN (System's Automatic Testability ANalysis) pour l'analyse de testabilité, vers un nouvel environnement de développement utilisant un formalisme flot de données, une représentation commune nommée « description MAC » est utilisée. De ce fait, il suffit de traduire les spécifications flot de données d'un système en cette représentation commune pour effectuer une analyse de testabilité de ce système. Les trois approches d'analyse de testabilité (par variable de sortie, par fonction et par composant), définies dans la section 3.4.1, sont mises en œuvre à partir de cette description MAC du système. Le MTI (Modèle de Transfert d'Information) du système est également construit à partir de cette description MAC. Cette phase constitue une étape intermédiaire et nécessaire pour l'analyse de testabilité.

La représentation MAC conserve l'aspect hiérarchique, les flots de données, les informations concernant les opérateurs et les types de données des variables utilisés dans les spécifications. Elle correspond à une image de la spécification en termes de flots de données. Le langage MACDOT est le langage utilisé pour décrire la description MAC d'un système. Ce langage est un sous-langage du langage DOT. DOT est un langage de description des graphes [DOT06]. Le langage MACDOT modélise chaque nœud SCADE à l'aide d'un graphe. Un graphe DOT est composé de nœuds et d'arcs ou arêtes selon que le graphe soit orienté (pour MACDOT) ou non-orienté. Chacun de ces éléments possède des attributs. DOT présente l'avantage d'offrir la possibilité de pouvoir étendre les attributs en définissant de nouveaux en cas de besoin. Cette possibilité est exploitée par MACDOT pour représenter l'ensemble des informations nécessaires à l'analyse de testabilité à l'aide des attributs associés aux nœuds et arcs. La syntaxe ainsi que la sémantique de ce langage sont décrites dans le manuel de référence du langage MACDOT [Dou06]. La passerelle de traduction SCADE vers MACDOT a été développée dans le cadre de cette étude. Elle peut être intégrée graphiquement dans l'environnement de développement SCADE. Un exemple de traduction d'une planche SCADE en une description MAC est proposé en annexe (section C.).

Application de la méthodologie

L'application de la méthodologie définie a pour objectif d'évaluer la couverture de la spécification formelle vis-à-vis des exigences fonctionnelles, la couverture des tests vis-à-vis de la spécification formelle et de calculer les mesures de testabilité. A l'issue de cette étape, sont déterminés :

- les écoulements orphelins, les écoulements manquants et les écoulements associés aux exigences ;
- les possibles tests redondants, les tests manquants et les tests associés aux écoulements ;
- les mesures de contrôlabilité et d'observabilité.

3.5.2 Synthèse des résultats d'analyse de couverture

Les différentes propositions d'adaptation pour l'analyse de testabilité (section 3.3) ainsi que la méthode définie pour l'évaluation des couvertures ont été expérimentées sur les systèmes LM1 (modes latéraux) et LM2 (modes longitudinaux). Ces adaptations concernent : la modélisation des opérateurs temporels, la modélisation des opérateurs d'aiguillage et la méthode de classification des écoulements.

Les deux systèmes (LM1 et LM2) implantent respectivement 34 et 57 exigences fonctionnelles décrites dans le DFS (Detailed Functional Specification) à l'aide des 58 et 92 variables de sortie définies dans leur modèle SCADE. Ces variables de sortie peuvent être différentes des sorties du calculateur. Le nombre de tests définis dans le LTR (Lab Test Request) est respectivement de 70 et 114 pour les deux systèmes.

Pour illustration, nous présentons ci-dessous un exemple d'exigence fonctionnelle décrite dans le DFS et un exemple de description de test dans le LTR.

Exigence E_1 : *Id* (identificateur)

- *Description :* Engagement standard du mode A lorsque l'avion atteint la zone de capture du « leg » actif du plan de vol. Cette transition est autorisée à partir des modes B, C, D et F.
- *Variable de sortie associée :* Var_E1

Test T_1 : Id (identificateur)

- *Description :* Le mode G doit être automatiquement engagé lorsque l'avion atteint l'altitude N.
- *Données de test et oracle*
- *Moyen de vérification :* Preuve, Simulateur de bureau, ou autres bancs d'essais.

– *Identificateur de l'exigence vérifiée*

Les résultats de l'application des différentes approches d'analyse de testabilité définies dans le contexte AIRBUS sont présentés ci-dessous.

Approche par variable de sortie

Les informations fournies par l'application de cette approche sur les systèmes LM1 et LM2 sont présentées dans le tableau ci-dessous.

Tableau 3.8 : Synthèse des résultats de l'approche par variable de sortie

	Nombre			
	Exigences fonctionnelles	Variables de sortie	**Ecoulements nominaux choisis**	**Ecoulements d'initialisation choisis**
LM1	34	58	**614**	**155**
LM2	57	92	**565**	**203**

L'analyse des systèmes à l'aide de cette approche a permis de déterminer 614 écoulements nominaux et 155 écoulements d'initialisation pour LM1. Pour le système LM2, 565 écoulements nominaux et 203 écoulements d'initialisation ont été déterminés. Ces écoulements, sélectionnés à l'aide de Start-Small, sont répartis entre les variables de sortie de ces systèmes. Les écoulements d'initialisation ne sont déterminés que pour des variables de sortie dépendant des opérateurs temporels. Le détail de ces résultats est proposé en annexe (section D.1).

Approche par fonction

Le tableau, ci-dessous, présente les informations fournies à l'issue de l'application de cette deuxième approche sur les systèmes LM1 et LM2.

Tableau 3.9 : Synthèse des résultats de l'approche par fonction

	Nombre			
	Exigences fonctionnelles	Variables de sortie	**Ecoulements nominaux choisis**	**Ecoulements d'initialisation choisis**
LM1	34	58	**198**	**83**
LM2	57	92	**202**	**132**

L'application de cette approche a permis de déterminer 198 écoulements nominaux et 83 écoulements d'initialisation pour LM1. L'analyse du système LM2 a produit 202 écoulements nominaux et 132 écoulements d'initialisation. Ces écoulements, sélectionnés à l'aide de Start-Small, sont répartis entre les ensembles de variables de sortie associés aux fonctions identifiés

dans le cahier de fonctions (DFS – Detailed Functional Specification) des systèmes. Le détail de ces résultats est proposé en annexe (section D.2).

Approche par composant

Les informations fournies par l'application de cette dernière approche sur les systèmes LM1 et LM2 sont présentées dans le tableau ci-dessous.

Tableau 3.10 : Synthèse des résultats de l'approche par composant

	Nombre			
	Exigences fonctionnelles	Variables de sortie	**Ecoulements nominaux choisis**	**Ecoulements d'initialisation choisis**
LM1	34	58	**84**	**73**
LM2	57	92	**127**	**112**

L'analyse des systèmes à l'aide de cette dernière approche a permis de déterminer 84 écoulements nominaux et 73 écoulements d'initialisation pour LM1. Pour le système LM2, 127 écoulements nominaux et 112 écoulements d'initialisation ont été déterminés. Ces écoulements ont été également sélectionnés à l'aide de Start-Small. Le détail de ces résultats est proposé en annexe (section D.3).

Evaluation de couverture

Les deux méthodes d'évaluation de couverture des exigences fonctionnelles et des tests vis-à-vis de la spécification formelle détaillée (modèle SCADE) ont été expérimentées à partir des résultats de l'application des trois approches de détermination des écoulements. Les résultats de ces expérimentations sur les deux systèmes ont conduit aux conclusions suivantes :

- Aucun écoulement orphelin n'a été détecté à l'issue des évaluations de couverture du modèle SCADE vis-à-vis des exigences définies pour les systèmes. En effet, tous les écoulements ont été associés à une exigence décrite dans le CDF. Ceci a permis d'affirmer que les modèles SCADE des deux systèmes (LM1 et LM2) ne contiennent ni de spécification superflue, ni de transcription d'exigences dérivées non décrites dans le CDF.
- Aucun écoulement manquant dû à une absence de transcription d'exigences fonctionnelles dans le modèle SCADE des deux systèmes n'a été trouvé. En effet, un sous-ensemble d'écoulements a été identifié pour chaque variable de sortie associée à une exigence décrite dans le CDF.
- L'analyse de couverture des tests a permis d'associer plusieurs tests à certains écoulements choisis par la stratégie Start-Small. Ceci permet d'indiquer, en se basant sur

le critère de couverture proposée par l'analyse de testabilité, que ces tests peuvent être considérés comme potentiellement redondants.

– En ce qui concerne les tests manquants, l'analyse des résultats a permis d'identifier certains écoulements associés aux exigences qui n'ont pas de correspondant au niveau des tests. Nous nous intéressons aux tests nominaux dans cette analyse. En effet, les tests d'initialisation ne sont explicitement définis dans LTR (Lab Test Request) même s'ils sont réalisés avant l'exécution des tests nominaux qui nécessitent une initialisation préalable. Le tableau ci-dessous présente les informations sur les tests manquants par rapport aux écoulements de la phase nominale selon les trois approches d'analyse de testabilité.

Tableau 3.11 : Nombre de tests manquant en fonctions des approches d'analyse

	Approche par variable de sortie		Approche par fonction		Approche par composant	
	LM1	LM2	LM1	LM2	LM1	LM2
Nombre d'écoulements déterminés	614	565	198	202	84	127
Nombre de tests nominaux définis	70	114	70	114	70	114
Nombre de tests manquants	**544**	**451**	**128**	**88**	**14**	**13**

Ces résultats montrent que le nombre de tests manquants décroît selon les trois approches d'analyse de testabilité de 544 et 451 (approche par variable de sortie) respectivement pour LM1 et LM2 à 14 et 13 (approche par composant). Cette décroissance du nombre de tests manquants s'explique par le fait que l'approche d'analyse par composant est celle dont le principe correspond à la méthodologie de définition de tests pour la validation des systèmes AIRBUS [Dou09a]. En effet, les tests fonctionnels construits pour la validation doivent être non redondants et assurer la couverture optimale des exigences et de la spécification SCADE. En considérant l'approche par composant, la présence des 14 et 13 tests manquants identifiés respectivement pour LM1 et LM2 s'explique par le fait que les systèmes analysés étaient en cours de développement lors de nos expérimentations et le processus de définition des tests pour certaines exigences n'était pas complètement achevé.

3.5.3 Synthèse d'évaluation de la pertinence des mesures

Les expérimentations ont concerné également l'évaluation de la pertinence des mesures de contrôlabilité et d'observabilité proposées à l'issue de l'analyse de testabilité dans le contexte AIRBUS. En effet, l'interprétation de ces mesures devrait fournir des indications sur la difficulté de test, liée à la perte d'information, introduite par l'utilisation de certains opérateurs au cours de la génération des jeux d'essai pour la validation des spécifications formelles détaillées de systèmes. Les méthodes de calcul dynamique des CPI (Coefficient de Perte d'Information) définies pour les opérateurs temporels ont été implantées au cours de cette évaluation. L'idée

est d'établir une corrélation entre les mesures prédictives de contrôlabilité et d'observabilité évaluées au cours l'analyse de testabilité et la difficulté de test pour la validation des spécifications des systèmes. Les résultats de ces expérimentations ont révélé la difficulté d'établir une telle corrélation en termes d'interprétation dans le contexte de validation basée sur le test fonctionnel. Les sections suivantes permettront d'analyser les principales raisons de cette difficulté en se basant sur quelques résultats d'expérimentation.

Mesures de contrôlabilité

Pour rappel, la contrôlabilité d'un composant du système selon l'analyse de testabilité proposée dans ce chapitre est calculée à l'aide de l'expression suivante :

$$Cont_E(M) = \frac{T(I_E, I_M)}{C(I_M)}$$

où $T(I_E, I_M)$ est la quantité d'information que le composant M peut recevoir à travers un écoulement E et $C(I_M)$ est la quantité d'information disponible à l'entrée du composant M lorsqu'il est isolé (toutes les entrées sont directement accessibles). Un composant est parfaitement contrôlable lorsque $T(I_E, I_M) = C(I_M)$. Ces quantités d'informations sont calculées à l'aide de la théorie de l'information.

Nous utilisons le résultat de l'analyse d'un diagramme d'opérateurs, figure ci-dessous, extrait du système LM1 (modes latéraux, planche SCADE N420401) pour illustrer la difficulté d'interprétation des mesures de contrôlabilité. Dans cette analyse, nous considérons que les entrées I_1, I_2, I_3 et I_4 font partie des entrées effectives du système (parfaitement contrôlables) et que les sorties O_1 et O_2 font partie des sorties effectives du système (parfaitement observables). *PULSE*1 et *PULSE*2 sont des opérateurs de type détecteur d'impulsion respectivement d'un front montant et d'un front descendant du flot de type booléen en entrée. *PREV* retarde (d'un cycle dans ce cas d'utilisation) le flot type booléen en entrée.

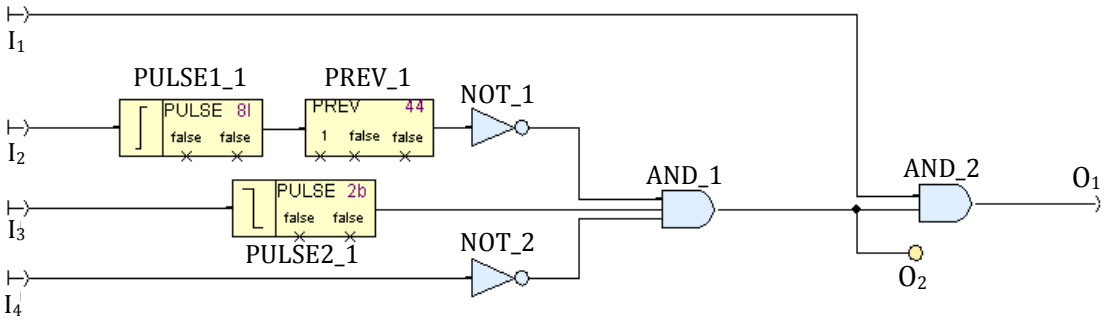


Figure 3.31 : Extrait du système LM1 (modes latéraux)

Les mesures de contrôlabilité calculées pour ce diagramme d'opérateurs sont présentées, pour chaque opérateur, dans le tableau ci-dessous.

Tableau 3.12 : Mesures de contrôlabilité

Opérateurs	Contrôlabilité
PULSE1_1	1,0
PREV_1	0,9539
PULSE2_1	1,0
NOT_1	0,9443
NOT_2	1,0
AND_1	*1,0*
AND_2	*0,8754*

Dans le tableau ci-dessus, nous constatons une différence de contrôlabilité d'environ 13% entre les opérateurs « AND_1 » et « AND_2 » (1,0 et 0,8754). Cette différence s'explique par le fait que l'opérateur « AND_1 » est à l'origine d'une perte d'information « théorique » sur une des deux entrées de l'opérateur « AND_2 ». Cette perte d'information diminue la quantité d'information effective disponible par rapport à la quantité d'information maximum (théorique) disponible à l'entrée de « AND_2 ». Par conséquent, la contrôlabilité se trouve diminuée au niveau de « AND_2 ».

Or, au cours de la définition des jeux d'essai dans un contexte de test fonctionnel, la difficulté d'activer « AND_2 » est au plus égale à celle de « AND_1 ». En effet, l'effort de test conduisant à la couverture de « AND_2 » correspond à celui de la production des valeurs « *true* » et « *false* » au niveau de la sortie de « AND_1 » (la deuxième entrée de « AND_2 » étant directement connectée à l'entrée du système).

Par ailleurs, les quantités d'information considérées au cours de l'analyse de testabilité expriment le nombre de quantités élémentaires d'information nécessaire pour représenter les valeurs théoriques ou effectives. Par conséquent, elles ne peuvent fournir aucune indication sur la difficulté de production d'une quelconque donnée de test fonctionnelle. De ce fait, cette notion de contrôlabilité ne peut être interprétée dans le contexte de validation des systèmes AIRBUS.

Mesures d'observabilité

De même, l'observabilité d'un composant du système selon l'analyse de testabilité proposée dans ce chapitre est calculée à l'aide de l'expression suivante :

$$Obs_E(M) = \frac{T(O_M, O_E)}{C(O_M)}$$

où $C(O_M)$ désigne la quantité d'information disponible à la sortie du composant M lorsqu'il est isolé (toutes les sorties sont directement accessibles) et $T(O_M, O_E)$ correspond à la quantité d'information disponible à la sortie de l'écoulement E à partir de M. Le composant est observable lorsque $T(O_M, O_E) = C(O_M)$.

Nous utilisons le résultat de l'analyse le même diagramme d'opérateurs, **Figure 3.31**, extrait du système LM1 (modes latéraux, planche SCADE N420401) pour illustrer la difficulté d'interprétation des mesures d'observabilité. Le tableau ci-dessous représente les mesures d'observabilité calculées pour chaque opérateur du diagramme d'opérateurs (**Figure 3.31**).

Tableau 3.13 : Mesures d'observabilité

Opérateurs	Observabilité
PULSE1_1	0,9273
PREV_1	0,9271
PULSE2_1	0,9177
NOT_1	*0,9271*
NOT_2	*0,8754*
AND_1	1,0
AND_2	1,0

Dans le tableau ci-dessus, nous constatons une différence d'observabilité d'environ 5% entre les opérateurs « NOT_1 » et « NOT_2 » (0,9271 et 0,8754). Cette différence s'explique par le fait que la quantité d'information effective disponible au niveau de la sortie de l'opérateur « NOT_1 » est inférieure est à celle de « NOT_2 ». Par conséquent, la perte d'information, à partir de ces opérateurs jusqu'à la sortie du système, est moindre pour « NOT_1 » par rapport à « NOT_2 ».

Or, au cours de la définition des jeux d'essai dans un contexte de test fonctionnel, la difficulté d'observer « NOT_1 » est égale à celle de « NOT_2 ». En effet, les flots d'informations à la sortie de ces deux opérateurs subissent les même traitements jusqu'à la sortie du système.

Pour les mêmes raisons (liées à la signification des quantités d'information) évoquées pour les mesures de contrôlabilité, cette notion d'observabilité ne peut également être interprétée dans le contexte de validation des systèmes AIRBUS.

3.6 Conclusion

Ce chapitre décrit les concepts de modélisation et les approches d'analyse de testabilité pour le développement d'une méthodologie d'aide à la génération des jeux d'essai au cours de la

validation de systèmes AIRBUS. Ces concepts et approches proposés visent à prendre en compte les spécificités liées aux méthodes de spécification et de test appliquées au cours de la phase de validation des systèmes. Ceci permet de proposer une méthodologie d'analyse pertinente et cohérente par rapport aux pratiques industrielles.

Les modélisations réalisées pour l'analyse de testabilité concernent des opérateurs d'une bibliothèque « métier » AIRBUS utilisée dans la spécification formelle détaillée de certains systèmes de « commande de vol ». Elles ont consisté à définir un modèle de transfert d'information (MTI) et à définir une méthode d'évaluation des coefficients de perte d'information (CPI) pour chacune des opérateurs. Ces modélisations ont permis la mise en place et l'intégration de la notion de classification des écoulements selon la phase d'initialisation et la phase nominale d'exécution des systèmes.

Les expérimentations menées sur le cas d'étude « commande de vol » de l'A350XWB ont permis la mise en œuvre de la méthodologie d'analyse définie ainsi que l'interprétation des résultats d'évaluation automatique d'une part, de la couverture de la spécification formelle détaillée vis-à-vis des exigences fonctionnelles et d'autre part, de la couverture des jeux d'essai vis-à-vis de la spécification formelle détaillée d'un système.

Ces expérimentations ont également permis de démontrer que les mesures de testabilité, telles qu'elles sont définies actuellement, ne sont pas adaptées au contexte de validation des systèmes AIRBUS. L'inadéquation de ces mesures peut être expliquée, d'une part par la méthode de calcul du coefficient de perte d'information (CPI), et d'autre part par la méthode de simulation du transfert d'information dans le système.

- *Calcul du CPI* : le CPI (élément clé du processus de calcul des mesures) associé à un opérateur est évalué de façon statique en supposant l'opérateur isolé (libéré de toute contrainte liée à son environnement). Ce coefficient, évalué une seule fois au cours de l'analyse de testabilité pour chaque type d'opérateur, est associé aux opérateurs quelque soit leur utilisation dans la spécification du système. Ce choix d'évaluation statique des CPI n'est pas adapté au contexte de test fonctionnel. En effet, un opérateur peut être responsable d'une perte d'information plus ou moins importante selon son utilisation (en fonction de la quantité d'information disponible à son entrée). Or, l'information disponible à l'entrée d'un opérateur est liée aux contraintes introduites partie du système en amont de cet opérateur. De ce fait, avec la méthode d'évaluation du CPI actuelle, la perte d'information causée par un opérateur peut se trouver surévaluée ou sous-évaluée selon son utilisation.

– *Simulation du transfert d'information* : au cours de la simulation du transfert d'information réalisée pour le calcul des mesures, la valeur des flots d'informations n'est pas prise en compte. Or, les fonctions associées aux les écoulements sont liées aux contraintes exprimées sur les valeurs des différents flots d'information participant à leur réalisation. La transparence de la simulation du transfert d'information entraine un manque de lisibilité lié à la part de la quantité d'information, disponible à la sortie d'un écoulement, provenant des différents flots intermédiaires participant au calcul de la sortie. Ceci conduit à une mauvaise appréciation sur l'origine de la quantité d'information effective disponible à la sortie d'un écoulement.

Chapitre 4
Aide au diagnostic au cours de la vérification sur la FAL

Les activités menées sur la chaîne d'assemble finale (FAL) des avions AIRBUS consistent à assembler l'ensemble des éléments de la structure (sections, équipements, ailes, etc.) d'un avion et à installer les systèmes et les moteurs avant son premier vol. La vérification des systèmes, au cours de cette phase d'installation, constitue une étape importante dans la phase d'assemblage d'un avion. Cette étape, essentiellement basée sur du test fonctionnel, a pour objectif de vérifier le ***fonctionnement des systèmes installés à bord*** ainsi que leur ***interconnexion***. Les systèmes sont implantés dans des calculateurs et communiquent entre eux et avec leur environnement (capteurs, commandes, etc.) à l'aide de bus d'acquisition.

Au cours de cette étape critique de la production de l'avion, en dehors de l'exécution des tests qui est essentiellement automatisée, les activités de conception des tests et de diagnostic après la détection d'une faute sont essentiellement manuelles et basées sur l'expertise humaine. De ce fait, la définition des tests et le diagnostic dépendent de l'expérience du concepteur de test et du testeur sur la FAL. Or, quelles que soient les méthodes de réalisation, ces activités sont soumises à certaines règles (décrites ci-dessous) permettant de réduire l'effort de vérification et par conséquent de maîtriser le coût de cette étape importante de la production d'un avion.

- La définition de tests pertinents et non redondants capables de couvrir les exigences des systèmes vérifiés au cours de la phase de conception des tests fonctionnels.
- La mise en place de techniques de diagnostic efficaces permettant d'identifier, soit la partie du système responsable d'une faute, soit le manque d'information disponible au moment du diagnostic.

Dans ce contexte, nous proposons une méthodologie d'aide au diagnostic au cours de la vérification des systèmes en FAL basée sur une stratégie de test. Nous présentons d'abord le processus de vérification des systèmes au cours des essais au sol ; puis nous introduirons la stratégie de test Multiple-Clue utilisée et nous décrirons la méthodologie définie pour la FAL. Le prototype implémentant cette méthodologie est par la suite présenté. Enfin, nous exposons les résultats de l'expérimentation réalisée sur un système AIRBUS.

4.1 Processus de vérification des systèmes sur la FAL

Le processus de vérification des systèmes au cours des essais au sol sur la chaîne d'assemblage finale d'un avion (FAL) consiste à vérifier le *bon fonctionnement des systèmes installés à bord* ainsi que leur *interconnexion*. En effet, les activités menées sur la FAL consistent à assembler les différents éléments de la structure de l'avion (sections équipées des câbles appropriés et équipements, voilure, etc.) et à installer les systèmes avioniques et les moteurs. Les systèmes installés sont implantés par des calculateurs et communiquent entre eux et avec leur environnement (capteurs, commandes, etc.) à l'aide de circuits (bus d'acquisition), d'hydrauliques, etc. Ces moyens d'interconnexion servent à récupérer les sorties ou à fournir les entrées des systèmes. Ils sont considérés comme des ressources à vérifier au même titre que les systèmes installés à bord au cours de la phase de test sur la FAL.

La vérification des systèmes sur la FAL se base sur les GTR (Ground Test Requirements) et les GTI (Ground Test Instructions). Ces deux documents contiennent l'ensemble des informations nécessaires (en termes d'exigences) à cette phase de vérification des systèmes installés à bord de l'avion avant le premier vol. Ce processus de vérification est présenté dans la **Figure 4.1**. Il comprend deux parties : la phase de conception des essais au sol et la phase d'exécution des essais et le diagnostic après la détection de fautes.

Conception des essais au sol

La phase de conception des essais au sol consiste en la définition des exigences de test (GTR) pour un avion ainsi que la description des objectifs et instructions de test (GTI) pour un avion sur la FAL.

GTR – Ground Test Requirements

Le GTR est un document issu du bureau d'étude, responsable de la définition des exigences de tests pour un avion complet. Il a pour objectif d'assurer que le système a été bien installé en vérifiant le bon fonctionnement de certaines fonctions et que les aspects de sûreté de fonctionnement, qui ne peuvent pas être vérifiés par les procédures d'inspection, marchent correctement. La vérification de ces exigences permet de démontrer la cohérence de l'ensemble des systèmes par rapport à la spécification de l'avion. Elle permet également de fournir les informations pour approbation de ces systèmes et de préparer l'avion pour le premier vol.

Dans ce contexte, le GTR constitue une trame de base de tous les essais à réaliser pour un système embarqué à bord de l'avion. Quelques exemples d'essais définis dans le GTR sont : le test du poignet Pilote ou Copilote ; le test d'un système à partir de deux voies d'alimentation différentes, les aspects redondance et Sûreté de Fonctionnement ; etc.

GTI – Ground Test Instructions

Les GTI sont décrits par les concepteurs d'essais. Ils sont produits à partir d'une interprétation des exigences définies dans le GTR pour vérifier un système sur la chaîne d'assemblage finale. Des objectifs de test sont définis pour chaque essai décrit dans le GTR. Afin de compléter, dans certains cas, la couverture de l'ensemble des interconnexions entre ressources d'un système, des exigences supplémentaires peuvent être définies. Elles peuvent être liées, soit à l'environnement, soit à la structure du système. Le processus de description des GTI est contraint afin de permettre l'exécution des tests par l'outil ESAO (Essai Assisté par Ordinateur).

Un GTI est constitué de deux principales parties :

- Phase de préparation du système : cette partie définit la configuration de base que doit avoir l'avion pour la vérification du système. Une large partie des ressources impliquées dans la vérification du système est identifiée à ce niveau.
- Descriptions des tests : pour chaque objectif de test, le GTI décrit la configuration requise ainsi que l'ensemble des instructions ou « pas de tests » à exécuter, nécessaires pour atteindre cet objectif.

L'objectif visé par un GTI est la vérification de l'ensemble des fonctions du système ainsi que des liens (câbles) qui relient les ressources entre elles. Pour ce faire, certains tests spécifiques sont créés pour tester les câbles non couverts par la vérification des exigences d'un système. Ces tests spécifiques peuvent être définis par : des tests préliminaires, par le biais des mesures de continuité et d'isolement ; des tests fonctionnels en dehors des exigences décrites dans le GTR ; une adaptation de l'outillage afin de compléter la couverture de l'exigence concernée.

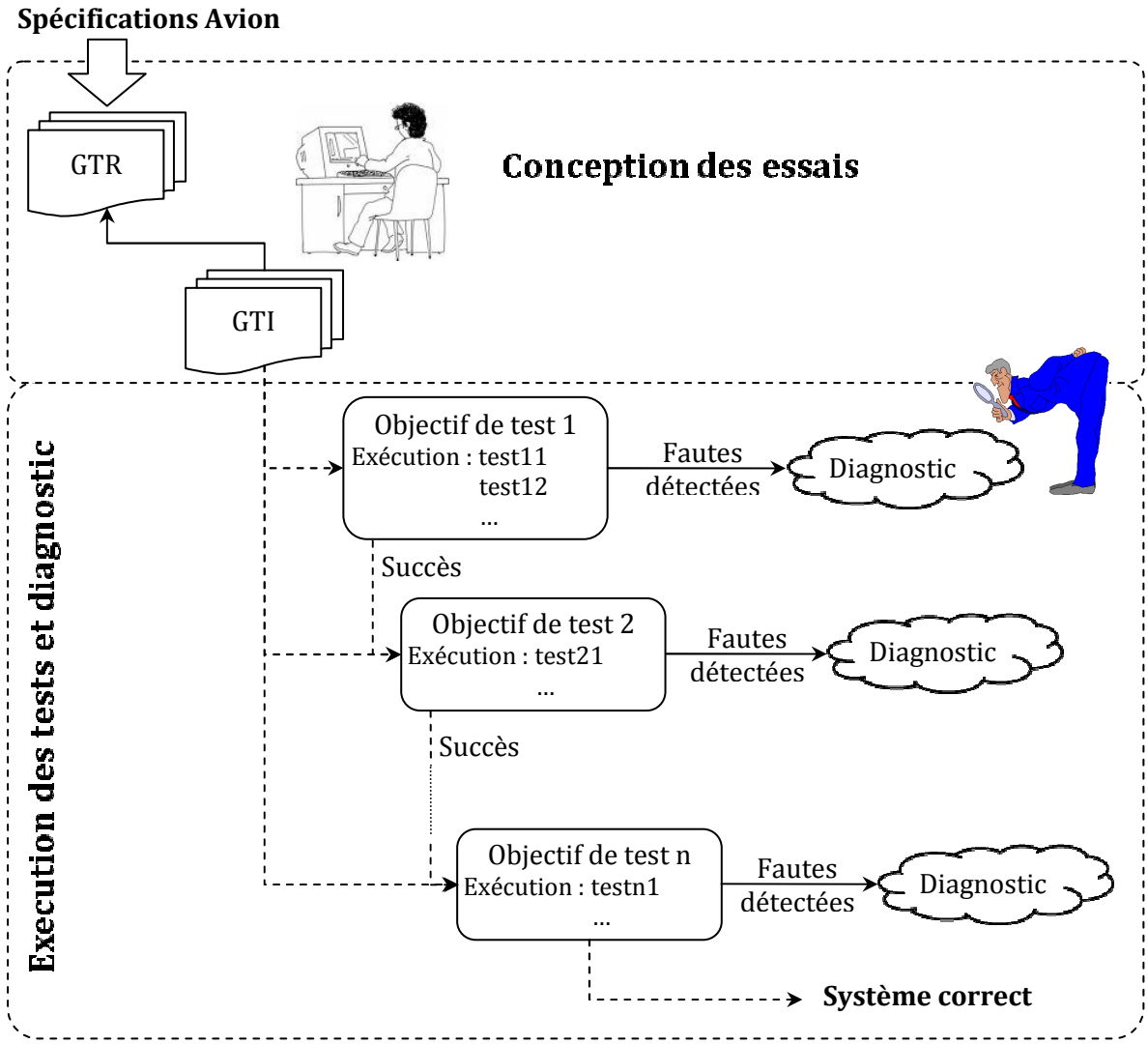


Figure 4.1 : Processus de vérification des systèmes sur la FAL

Exécution des tests et diagnostic

La phase d'exécution des tests et de diagnostic consiste à vérifier successivement chaque objectif de test à l'aide des instructions de test définies dans un GTI. Le diagnostic est effectué au niveau d'un objectif de test à l'issue de la détection d'une faute avant de poursuivre la vérification. Il existe des moyens et outils de tests permettant d'automatiser une grande partie des instructions de test à exécuter et de fournir le verdict à ce niveau. Cependant, l'expertise

humaine est souvent nécessaire pour réaliser certains tests. Par ailleurs, le processus de diagnostic de faute est essentiellement basé sur les retours d'expérience acquis par les testeurs sur la FAL. Selon la technologie utilisée dans les ressources des systèmes, les moyens de diagnostic utilisés diffèrent.

Moyens de diagnostic sur la FAL

Les ressources utilisées par les fonctions d'un système peuvent être classées dans deux catégories : les ressources dites « outillage » et les ressources dites « fonctionnelles ». Pour chaque catégorie, il existe des méthodes et moyens de diagnostic appropriés sur la FAL.

Ressource outillage

Une ressource outillage est une ressource électrique discrète ou analogique dont l'état peut être modifié à l'aide des outils de test disponibles. Le comportement de ces ressources peut être vérifié de façon indépendante. Ceci offre une très grande souplesse au niveau du diagnostic de ces ressources. En effet, en cas de soupçon porté sur une ressource outillage, il est possible d'aller vérifier directement le fonctionnement (interne) de la ressource à l'aide des outils de vérification. Considérons trois ressources outillage $(R_1, R_2 \, et \, R_3)$ réalisant une fonction donnée. La **Figure 4.2** illustre la méthode et les moyens de diagnostic utilisés en cas de détection de faute.

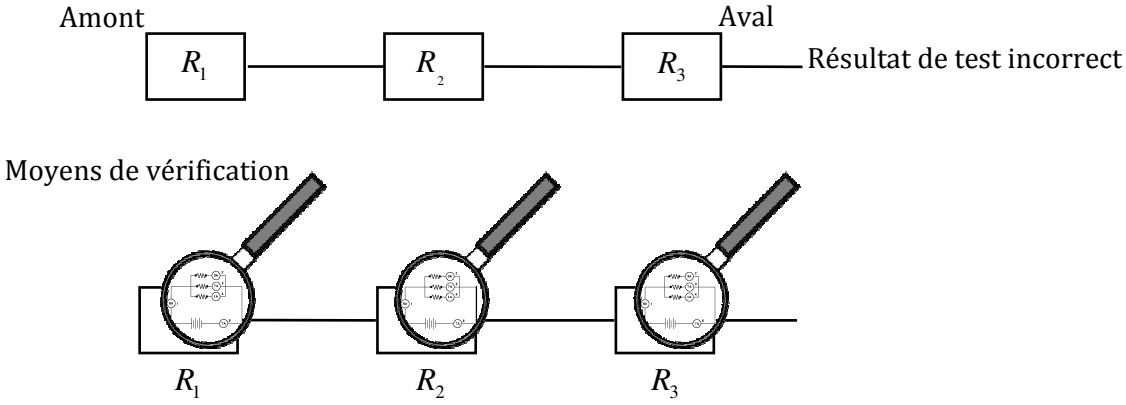


Figure 4.2 : Méthode de diagnostic de ressources outillage

Ressource fonctionnelle

Une ressource fonctionnelle est une ressource dont il n'est pas possible de changer l'état à l'aide des outils de vérification disponibles. Ces ressources sont considérées comme des boites noires dont on ne connait pas la structure interne. De ce fait, le diagnostic des ressources de cette catégorie se fait de deux manières différentes selon les caractéristiques de la ressource concernée.

La première méthode consiste à envisager une intervention fonctionnelle au niveau de la ressource. Cette intervention est de nature à observer le comportement de la ressource fonctionnelle suspectée à l'aide d'un point de piquage. Les observations faites à partir de ce point de piquage permettront d'incriminer ou pas la ressource suspectée. Considérons trois ressources fonctionnelles $(R_1, R_2 \; et \; R_3)$ réalisant une fonction donnée. La **Figure 4.3** illustre cette première méthode et les moyens de diagnostic utilisés en cas de détection de faute.

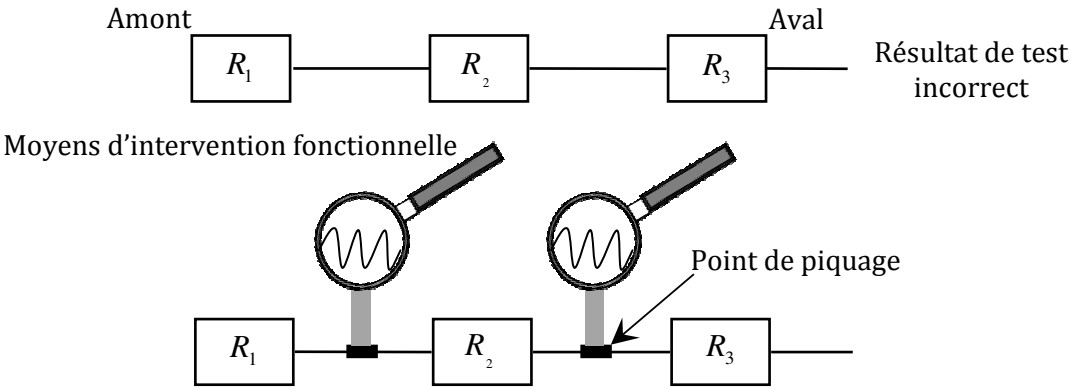


Figure 4.3 : Première méthode de diagnostic de ressources fonctionnelles

La seconde méthode consiste à procéder par remplacement de la ressource suspectée au cas où l'intervention fonctionnelle (par point de piquage) n'est pas retenue. En effet, dans certaines conditions, il est plus facile et moins coûteux de procéder par remplacement que d'envisager une intervention fonctionnelle. Au cours de ce remplacement les ressources sont supposées être correctes (comportement vérifié à l'avance). Considérons trois ressources fonctionnelles $(R_1, R_2 \; et \; R_3)$ réalisant une fonction donnée. La **Figure 4.4** illustre cette seconde méthode et les moyens de diagnostic utilisés en cas de détection de faute.

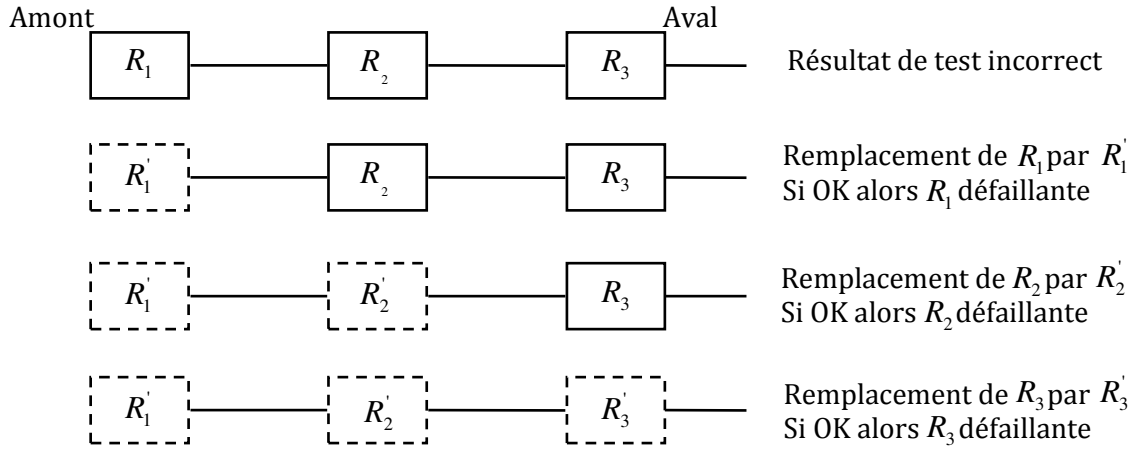


Figure 4.4 : Seconde méthode de diagnostic de ressources fonctionnelles

4.2 Stratégie de test Multiple-Clue

Pour rappel, une stratégie de test définit la méthodologie de test correspondant à la manière d'appliquer les tests et d'analyser les résultats. Multiple-Clue est une stratégie de test combinatoire basée sur une approche par recoupements pour le diagnostic d'une faute à l'issue de la vérification d'un système [Gal91]. Dans le cadre de nos travaux, un système peut être considéré comme un ensemble de ressources R_i (matériels ou logiciels) implantant des fonctions. Au cours de la phase de vérification, des tests T_i sont définis pour vérifier ces fonctions. Le principe de la stratégie Multiple-Clue est de sélectionner un ensemble minimal de tests nécessaires et suffisants pour :

- activer toutes les ressources du système ;
- assurer le diagnostic le plus précis possible à partir des informations disponibles sur le résultat des tests exécutés au cours de la vérification.

Dans la suite de cette section, nous introduisons les notions et termes utilisés (section 4.2.1) ; nous présentons les hypothèses d'application de Multiple-Clue (section 4.2.2) et nous décrivons les informations fournies à l'issue de l'application de cette stratégie de test (section 4.2.3).

4.2.1 Définition des notions et termes

D'une manière générale, un système S peut être considéré comme un ensemble de ressources R (matérielles et logicielles) implantant des fonctions. Au cours de la phase de vérification, un ensemble de tests T sont définis et exécutés pour vérifier ces fonctions.

Définition 10 : Ressources

Une ressource R_i correspond à une partie du système S qui peut être responsable d'une faute.

Définition 11 : Test ou cas de test

Un test T_i correspond à une combinaison d'entrées du système permettant l'activation d'un ensemble de ressources pour la vérification d'une fonction. L'exécution d'un test peut révéler une faute parmi les ressources activées. On appelle trace du test T_i « $trace(T_i)$ » l'ensemble des ressources activées au cours de son exécution.

Définition 12 : Oracle

Un oracle décrit le résultat attendu de chaque test T_i défini. Il permet de déterminer si une fonction testée est correcte ou pas. Le verdict est alors rendu sur le résultat de l'exécution du test.

Définition 13 : Tests équivalents

Deux tests T_i et T_j sont dits équivalents ou redondants pour le diagnostic lorsqu'ils activent les mêmes ressources : $trace(T_i) = trace(T_j)$.

Définition 14 : Ressources indiscernables

Deux ressources R_i et R_j sont dites indiscernables pour le diagnostic lorsqu'elles sont toujours et uniquement activées par les mêmes tests :

$$\forall T_k \in T;\ si\ R_i \in trace(T_k)\ alors\ R_j \in trace(T_k)$$

Définition 15 : Ensemble de diagnostic

Un ensemble de diagnostic est un sous-ensemble de T nécessaire et suffisant pour activer au moins une fois l'ensemble des ressources du système et identifier soit la ressource défectueuse, soit l'ensemble de ressources indiscernables suspecté au cours du diagnostic.

4.2.2 Hypothèses d'application de Multiple-Clue

Au cours d'une campagne de test, la détection d'une faute à l'issue de l'exécution d'un test implique l'existence d'au moins une ressource défectueuse parmi les ressources activées. Les tests détectant une faute sont ceux qui fournissent des informations sûres au cours de la vérification. Une faute est détectée par un test lorsqu'il n'y a pas de cohérence entre le résultat effectif des tests exécutés et le résultat attendu constituant l'oracle. De ce fait, le diagnostic est basé sur la confiance en l'oracle et le verdict associé au résultat des tests exécutés. Un oracle est dit « *sûr* » lorsqu'il remplit la condition suivante : si le verdict d'un test correspond à celui indiqué dans l'oracle alors l'ensemble des ressources activées par ce test peuvent être considérées comme correctes. Inversement, un oracle est « *non sûr* » lorsque la non-détection d'une faute au cours l'exécution d'un test n'implique pas que l'ensemble des ressources activées soient correctes.

Cependant, deux paramètres inconnus sont utilisés pour réaliser un tel diagnostic : le nombre de fautes détectées à l'issue de l'exécution d'un ensemble de tests et la confiance attribuée aux tests n'ayant pas détecté de faute à l'issue de leur exécution (la probabilité d'infection d'un état défectueux et de propagation de cet état au niveau du résultat du test).

Considérant ces paramètres, les hypothèses d'application de Multiple-Clue sont les suivantes : une seule ressource est défectueuse parmi l'ensemble des ressources activées au cours de la vérification et l'oracle est « *sûr* ». En effet, un ensemble de tests qui ne détecte pas de faute implique que les ressources activées sont correctes (la probabilité d'infection et de propagation d'une faute lorsqu'une ressource défectueuse est activée est égale à 1). Le résultat de l'ensemble des tests sélectionnés est utilisé au cours du diagnostic.

Ces hypothèses conduisent à utiliser la stratégie Multiple-Clue dans un contexte d'existence de faute simple, contexte bien adapté aux phases de vérification finale des systèmes et de maintenance.

4.2.3 Informations fournies par Multiple-Clue

L'application de la stratégie Multiple-Clue [Kha98, Kha01a, Kha01b] permet d'évaluer l'effort de diagnostic d'un système en fournissant des informations sur :

- **les ressources indiscernables** : deux ressources sont indiscernables lorsqu'elles sont toujours activées de la même manière par les mêmes tests. En cas d'erreur, il est donc impossible de savoir laquelle de ces ressources est défectueuse ;
- **les tests équivalents pour le diagnostic** : deux tests sont équivalents pour le diagnostic lorsqu'ils activent le même ensemble de ressources ;
- **l'ensemble de diagnostic** : il correspond à un ensemble minimal de tests « pertinents » choisis qui assure l'activation de toutes les ressources et la localisation d'une faute. Cet ensemble est utilisé pour la mise en œuvre de l'approche par recoupements.

La figure ci-dessous est utilisée pour illustrer le principe de l'approche par recoupements proposée par la stratégie Multiple-Clue.

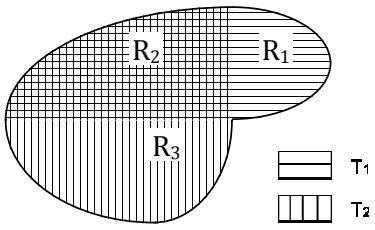


Figure 4.5 : Exemple pour l'illustration des principes de l'approche par recoupements

Le système représenté par la figure ci-dessus est composé de trois ressources (R_1, R_2 et R_3). Deux tests (T_1 et T_2) ont été sélectionnés par Multiple-Clue pour le diagnostic après la détection d'une faute au cours de la vérification du système. Le test T_1 active les ressources R_1, R_2 et le test T_2 active les ressources R_2, R_3. Le verdict d'un test T_i est un « succès » lorsque le résultat de

l'exécution correspond au résultat attendu et les ressources activées par ce test sont correctes. Le verdict est un « échec » sinon. Les différentes possibilités de diagnostic en fonction des résultats des tests T_1 et T_2 sont présentées à l'aide de la figure ci-dessous. Elle sera appelée par la suite « **arbre de test** ».

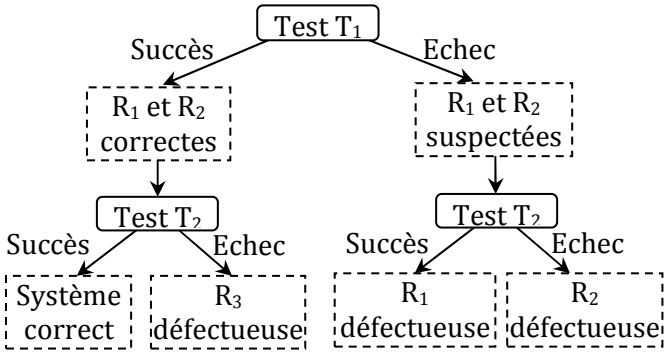


Figure 4.6 : Arbre de test du résultat de Multiple-Clue

4.2.4 Synthèse

Dans cette section, nous avons présenté la stratégie Multiple-Clue adaptée à la détection et la location de faute simple au cours de la vérification de systèmes. Les hypothèses d'application ainsi que les informations fournies par cette stratégie ont été décrites. Par ailleurs, l'application de la stratégie Multiple-Clue nécessite une modélisation du système. Cette modélisation consiste à mettre en relation l'ensemble des ressources avec l'ensemble des tests définis pour la vérification du système.

4.3 Méthodologie d'aide au diagnostic

La vérification des systèmes sur la chaîne d'assemble finale d'un avion AIRBUS permet d'assurer que les systèmes installés fonctionnent correctement ainsi que leur interconnexion. Ceci revient à répondre à la question suivante : « le produit réalisé est-il conforme à ses exigences ? ». Le processus de vérification des systèmes sur la FAL (Final Assembly Line) est composé de deux phases principales : la phase de conception des essais ou des tests fonctionnels et la phase d'exécution des tests et de diagnostic (**Figure 4.7**). La définition des tests fonctionnels pertinents et non redondants constituent les principaux défis de la conception des essais. L'identification de la ressource défectueuse ou l'identification d'une situation d'indiscernabilité impliquant un ensemble de ressources sont les difficultés auxquelles sont confrontés les testeurs au cours du diagnostic (essentiellement basé sur l'expertise humaine) sur la FAL.

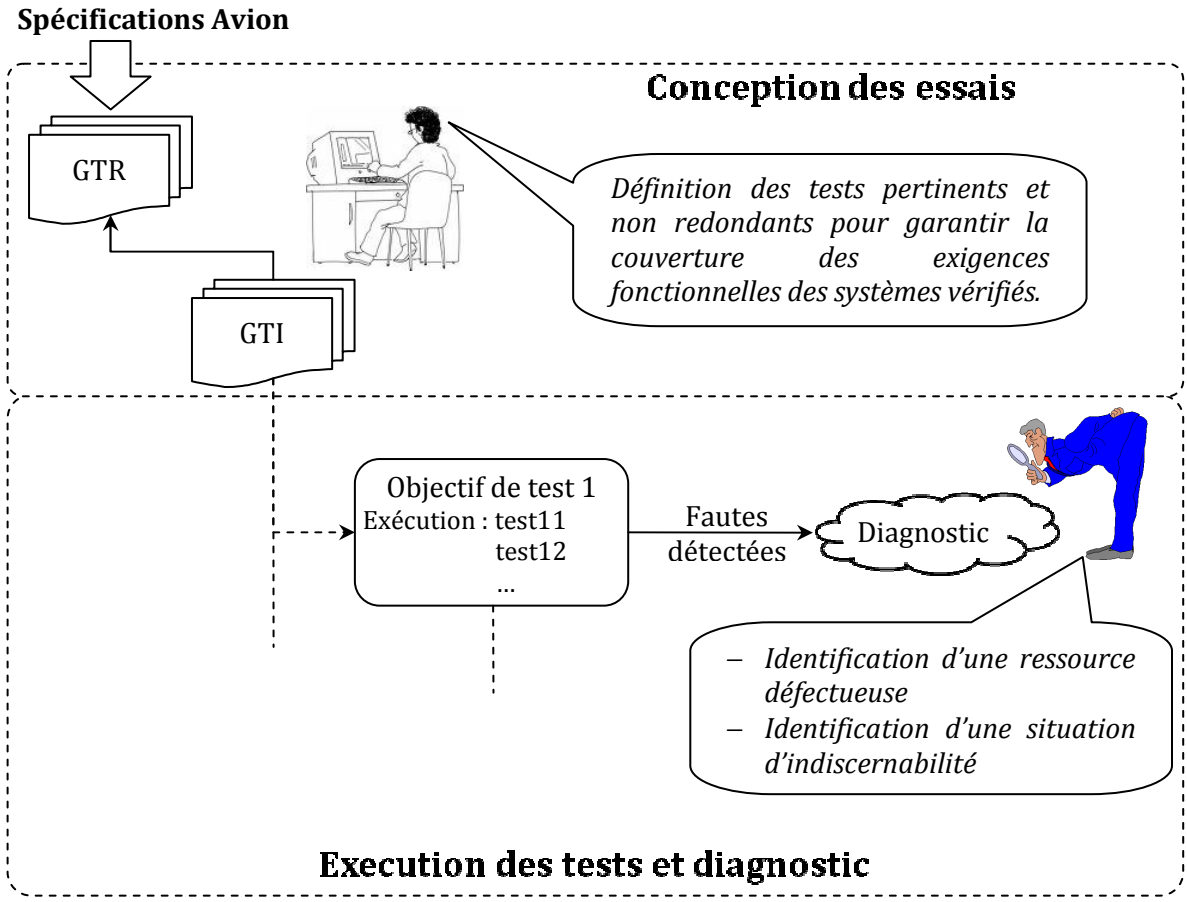


Figure 4.7 : Challenges de la vérification des systèmes sur FAL

La méthodologie d'aide au diagnostic, proposée dans cette section, décrit des méthodes fournissant des informations sur le diagnostic dès la phase de conception des tests. Ces méthodes fournissent également des informations permettant de guider la phase de localisation de la ressource défectueuse après la détection d'une faute au cours de l'exécution des tests. Cette méthodologie se base principalement sur la stratégie Multiple-Clue. Dans cette section, nous détaillons le processus d'application de Multiple-Clue (section 4.3.1) ; ensuite nous décrivons une méthode d'amélioration de la précision du diagnostic (section 4.3.2) ; enfin, nous présentons les avantages de l'application d'une telle méthodologie dans le processus de vérification des systèmes sur la FAL (section 4.3.3).

4.3.1 Processus d'application de Multiple-Clue

L'application de la stratégie Multiple-Clue nécessite une modélisation mettant en relation l'ensemble des ressources et l'ensemble des tests définis pour la vérification du système. Ce modèle abstrait correspond à une représentation matricielle impliquant les ressources et les tests du système. Dans le contexte de la vérification des systèmes sur la FAL, trois principales

étapes ont été définies pour l'application de Multiple-Clue : la modélisation des ressources et des tests ; la construction du modèle (nommé matrice de diagnostic) et la mise en œuvre de Multiple-Clue. La démarche globale d'application de cette stratégie, à partir du GTI, est représentée dans la **Figure 4.8** ci-dessous. En effet, les informations relatives aux ressources et tests sont décrites dans le GTI.

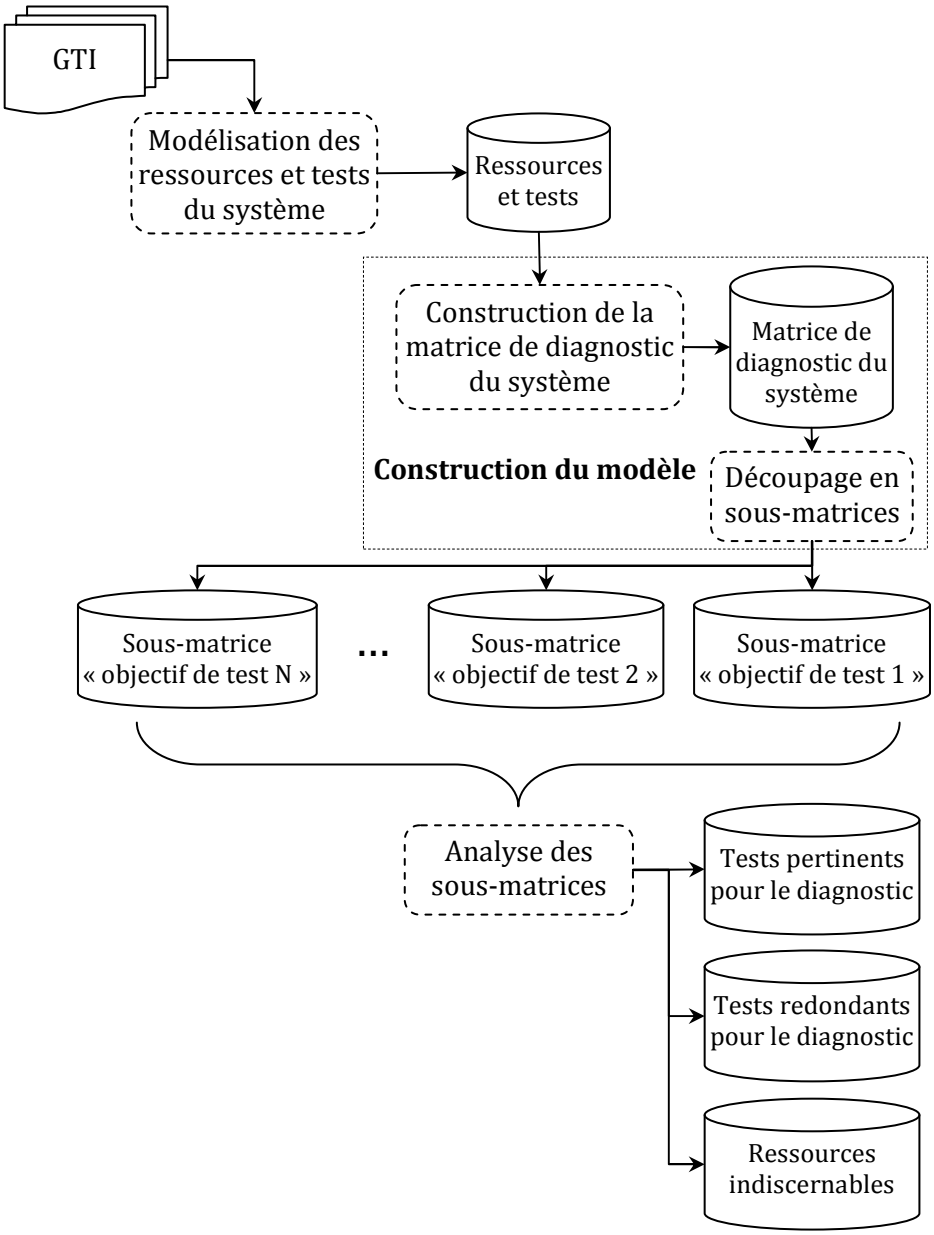


Figure 4.8 : Démarche d'application de la stratégie Multiple-Clue sur la FAL

Modélisation des ressources et des tests

Cette étape est essentielle dans le processus d'application de la stratégie Multiple-Clue. En effet, elle consiste à identifier l'ensemble des composants ou ressources fonctionnelles (calculateurs, disjoncteurs, câbles, etc.) qui peuvent se révéler défectueux au cours de la vérification du système. L'ensemble des tests exécutés pour la vérification du système sont

également identifiés dans cette étape. Cette phase de modélisation est réalisée en se basant sur les informations contenues dans le GTI (Ground Test Requirements) du système. Ce document est structuré comme suit :

- Une première partie est dédiée à la description de la phase de configuration qui consiste à réunir les conditions nécessaires au niveau du système et de son environnement pour la vérification. Une large partie des ressources utilisées par le système est définie dans cette section.
- Pour chaque objectif de test défini, la configuration « supplémentaire » requise est décrite. Les ressources spécifiques à l'objectif de test sont également définies. Les tests fonctionnels assurant la vérification de l'objectif de test sont aussi définis. Un identificateur est spécifié dans le GTI (Ground Test Instructions) pour chaque test défini. Dans l'approche de modélisation proposée, cet identificateur sera utilisé pour la modélisation des tests.

La modélisation des ressources pour l'application de la stratégie Multiple-Clue dépend de la précision recherchée au cours du diagnostic. En effet, le niveau d'abstraction ou la granularité de la modélisation des ressources du système joue un rôle important dans la détermination de la partie défectueuse après la détection d'une faute. Plus fine est cette granularité, meilleure sera la précision du diagnostic. Lorsqu'une ressource représente une partie du système pouvant révéler un ensemble de fautes, l'identification de la ressource défectueuse n'est pas suffisante pour la localisation précise de l'erreur responsable de la faute ; d'autres activités de diagnostic sont nécessaires pour atteindre un tel degré de précision dans le diagnostic. A titre d'exemple, considérons le composant assurant les fonctions « ANTI-SKID » des roues avant d'un avion de type A330-340. Ce composant peut être utilisé dans deux états différents (« ON » et « OFF ») et chacun de ces deux états peut causer une faute au cours de la vérification du système. Lorsque le composant « ANTI-SKID » est modélisé comme une seule ressource et que l'objectif du diagnostic est l'identification de l'état défectueuse d'un composant, l'incrimination de cette ressource ne permet pas d'obtenir un tel résultat à l'issue du diagnostic. Cependant, la modélisation de ce même composant en deux ressources distinctes (« ANTI-SKID ON » et « ANTI-SKID OFF ») permet d'atteindre le degré de diagnostic souhaité, à savoir l'identification de l'état défectueux.

Dans notre contexte, nous considérons une ressource comme un composant dans un état donné. Soient E_c l'ensemble des états d'un composant, C une partie du système et nR_c le nombre de ressources représentant le composant C dans le modèle construit pour l'application de Multiple-Clue. Nous pouvons écrire la relation suivante : $nR_c = |E_c|$. Ceci signifie qu'un

composant du système comportant n états différents est modélisé à l'aide de n ressources différentes.

Construction du modèle

La matrice de diagnostic constitue l'élément « clé » représentant le modèle du système pour l'application de Multiple-Clue. Elle correspond à une matrice dont les lignes représentent les tests fonctionnels à exécuter pour la vérification des fonctions du système. Ces tests sont notés $\{T_1, T_2, ..., T_n\}$. Les colonnes correspondent aux ressources utilisées par le système. Elles sont notées $\{R_1, R_2, ..., R_p\}$. La matrice de diagnostic est notée $M = |m_{ij}|$ où $i \in [1, n]$ est le nombre de tests et $j \in [1, p]$ est le nombre de ressources du système. Cette matrice est construite telle que, pour un élément m_{ij} :

- $m_{ij} = 1$ si le test T_i active la ressource R_j. On dit que la ressource R_j est dite couverte par le test T_i.
- $m_{ij} = 0$ sinon.

L'exemple ci-dessous représente la matrice de diagnostic d'un système.

	R_1	R_2	R_3
T_1	0	1	0
T_2	1	1	0
T_3	0	1	1
T_4	1	1	1

Figure 4.9 : Exemple d'illustration d'une matrice de diagnostic

Les informations fournies par ce modèle sur le système sont les suivantes :

- le test T_1 active la ressource R_2 ;
- le test T_2 active les ressources R_1 et R_2;
- le test T_3 active les ressources R_2 et R_3;
- le test T_4 active les ressources R_1, R_2 et R_3.

Sur la chaine d'assemblage finale, en dehors de la phase de configuration requise « commune » à tous les objectifs de test, le contenu d'un GTI est structuré de la manière suivante : pour chaque objectif de vérification ou de test, la configuration requise « spécifique » est mise en œuvre et les tests sont exécutés avant de passer à un autre objectif de test. De ce fait, les résultats de l'ensemble des tests exécutés pour la vérification d'un objectif de test sont interprétés indépendamment de ceux d'un autre objectif de test. Cette indépendance des tests

est illustrée à l'aide du modèle ci-dessous donnant une représentation de celui d'un système vérifié sur la FAL.

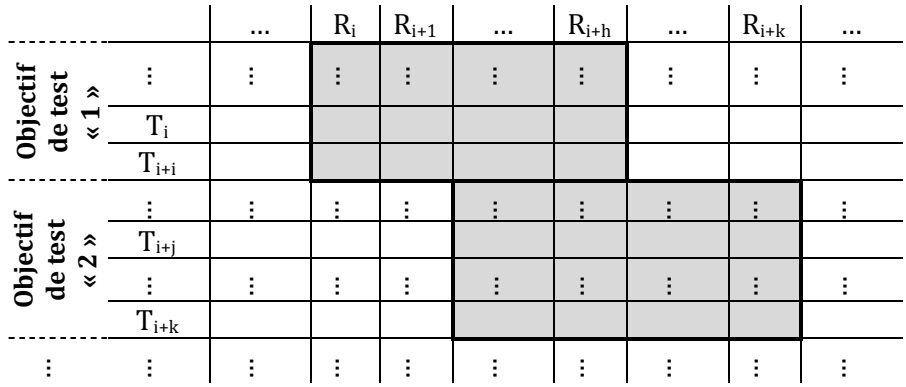


Figure 4.10 : Représentation d'une matrice de diagnostic dans le contexte de la FAL

Le modèle ci-dessus met en évidence le caractère disjoint des ensembles de tests définis pour la vérification des objectifs de test. Par conséquent, la matrice de diagnostic du système peut être découpée en sous-matrices associées à chaque objectif de test défini pour la vérification sur la FAL. Cette approche de « découpage » de la matrice de diagnostic globale du système en sous-matrices permet d'appliquer la stratégie Multiple-Clue pour l'analyse de chaque objectif de test. Ceci est essentiel pour l'adaptation du processus d'application de Multiple-Clue dans le contexte de vérification des systèmes sur la FAL.

Le modèle ci-dessous représente la sous-matrice d'un objectif de test correspondant à la « *vérification du système master lever* » au cours de la vérification du système « *d'Orientation Roues Avant* » d'un avion de type A330-340. Ce système est géré par deux moteurs différents. Chacun des deux moteurs peut être activé dans deux états différents. La matrice ci-dessous représente la « sous-matrice objectif de test » de ce système.

	ENG1 MASTER ON	ENG1 MASTER OFF	ENG2 MASTER ON	ENG2 MASTER OFF
3-1-16	0	1	0	1
3-1-17	1	0	0	1
3-1-18	0	1	0	1
3-1-19	0	1	1	0
3-1-20	0	1	0	1

Figure 4.11 : Sous-matrice de diagnostic de la « *vérification du système master lever* »

Analyse des sous-matrices

Cette étape consiste à analyser la matrice de diagnostic correspondant à chaque objectif de test. Le résultat de cette analyse fournit des informations permettant d'aider d'une part, à l'identification des tests pertinents (construction de l'arbre de test) pour l'incrimination d'une ressource défectueuse et l'identification des tests redondants pour le diagnostic. D'autre part,

cette analyse permet d'identifier les ressources indiscernables pour le diagnostic. En considérant la matrice de diagnostic, les notions de redondance, de pertinence des tests et d'indiscernabilité des ressources peuvent être définies comme suit :

- Deux tests T_i et T_j sont dits redondants lorsqu'ils activent le même ensemble de ressources. Ceci correspond à deux lignes identiques dans la matrice de diagnostic. Dans l'exemple de la **Figure 4.11**, les tests identifiés par « 3-1-18 » et « 3-1-20 » sont redondants par rapport au test « 3-1-16 ».
- Deux ressources R_i et R_j sont dites indiscernables lorsqu'elles sont activées exactement par les mêmes tests. Ceci correspond à deux colonnes identiques dans la matrice de diagnostic. De ce fait, le résultat du diagnostic ne peut pas incriminer l'une ou l'autre de ces ressources. Il ne peut que suspecter l'ensemble de ces ressources. Aucune ressource indiscernable n'a été identifiée dans l'exemple de la **Figure 4.11**.
- Les tests pertinents pour le diagnostic forment l'ensemble de diagnostic qui correspond au sous-ensemble de tests nécessaires et suffisants pour l'activation des ressources du système et la localisation, la plus précise possible, d'une faute avec les informations disponibles à l'issue de l'exécution des tests. Les tests « 3-1-16 », « 3-1-17 » et « 3-1-19 » représentent cet ensemble de diagnostic pour l'exemple de la **Figure 4.11**.

La figure ci-dessous met en évidence l'ensemble de diagnostic avec les ressources de l'exemple de la **Figure 4.11.**

	ENG1 MASTER ON	ENG1 MASTER OFF	ENG2 MASTER ON	ENG2 MASTER OFF
3-1-16	0	1	0	1
3-1-17	1	0	0	1
3-1-19	0	1	1	0

Figure 4.12 : Ensemble de diagnostic de la « *vérification du système master lever* »

Cet ensemble de diagnostic est utilisé pour la construction de l'arbre de test pour le diagnostic. La **Figure 4.13** représente l'arbre de test construit à partir des tests « 3-1-16 », « 3-1-17 » et « 3-1-19 ».

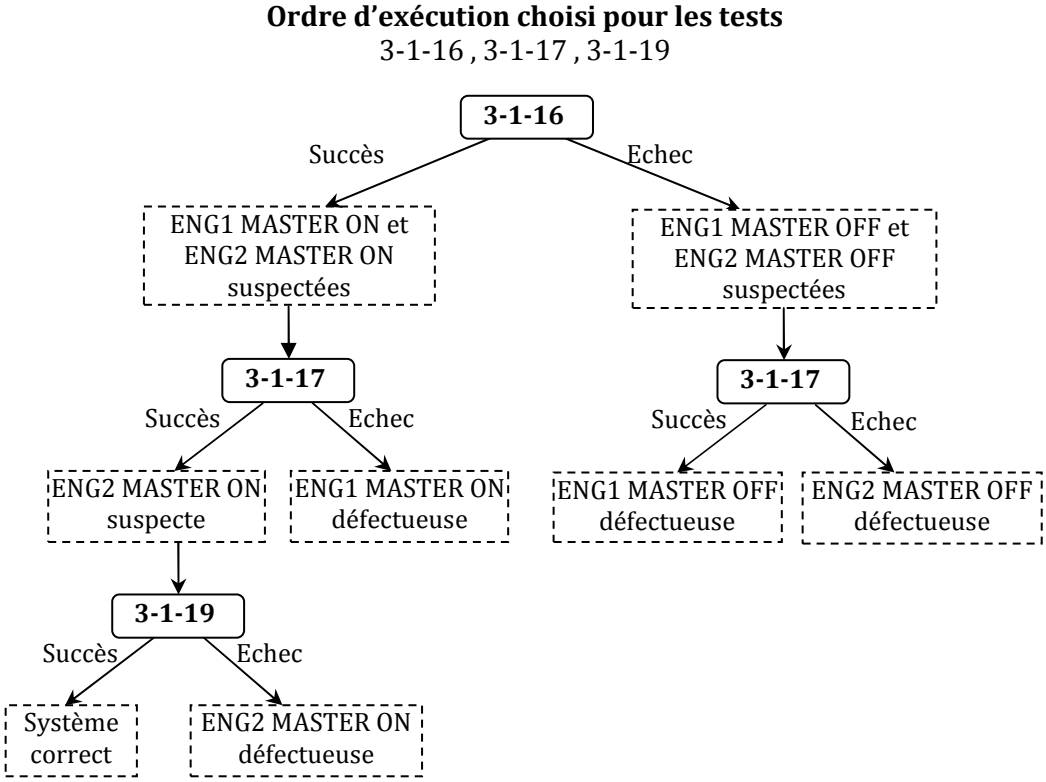


Figure 4.13 : Arbre de test pour la « *vérification du système master lever* »

Cet arbre, construit en considérant l'ordre de définition et d'exécution des tests, permet de guider le testeur pour l'incrimination d'une ressource après la détection d'une faute à l'issue de la « *vérification du système master lever* » qui constitue un objectif de test.

Cependant, l'ordre d'exécution des tests choisis par Multiple-Clue n'a a priori pas d'importance pour assurer l'incrimination de la ressource défectueuse. Cela veut dire que quel que soit l'ordre d'exécution ou d'interprétation des résultats des tests choisi, on saura détecter et localiser une ressource défectueuse du système en utilisant l'arbre de test. La figure ci-dessous propose un deuxième arbre de tests suivant un ordre d'exécution des tests différent.

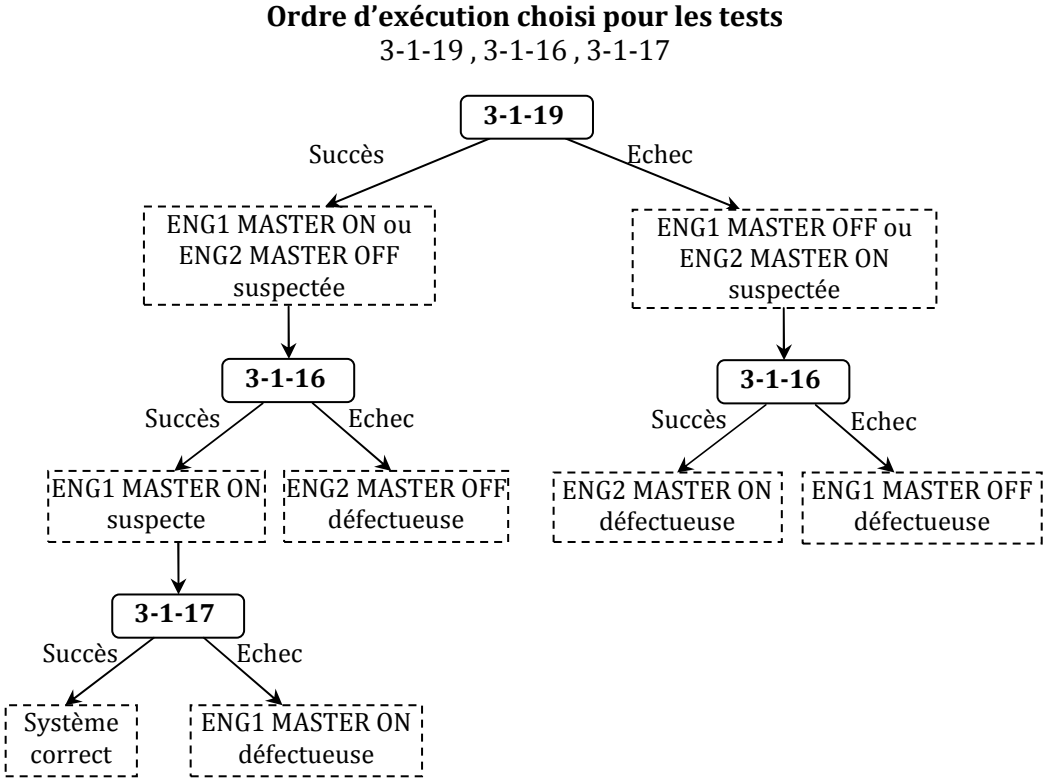


Figure 4.14 : Deuxième arbre de test pour la « *vérification du système master lever* »

Ce second arbre de test permet également d'incriminer toute ressource défectueuse dans le système. Par ailleurs, l'incrimination des ressources peut ne pas se faire sur la même partie ou au même niveau de profondeur de l'arbre de test (**Figure 4.13** et **Figure 4.14)**. Par conséquent, en se basant sur certaines informations du système, le choix d'un certain ordre d'exécution ou d'interprétation des tests (pour la construction de l'arbre) peut avoir un impact positif en termes de complexité ou d'effort d'incrimination d'une ou d'un sous-ensemble de ressources impliquées dans le diagnostic.

En théorie, l'arbre de la **Figure 4.14** est plus efficace pour l'incrimination de la ressource « ENG2 MASTER ON ». En effet, cette incrimination ne se fait pas au même niveau (3 pour la **Figure 4.13** et 2 pour **Figure 4.14**). Toutefois, l'implantation d'un tel algorithme est complexe et demanderait des connaissances sur la probabilité d'occurrence de panne des différents éléments. De ce fait, aucun travail n'a été réalisé sur cette thématique dans le cadre de cette thèse.

Synthèse

Dans cette section, nous avons présenté le processus d'application de la stratégie Multiple-Clue défini pour la vérification des systèmes sur la FAL (Final Assembly Line). Ce processus tient compte des différentes étapes de la vérification des systèmes sur la FAL et propose des

méthodes d'aide à l'identification des tests pertinents et non redondants dès la phase de conception des tests. Il fournit également une méthode d'interprétation du verdict de l'exécution des tests pour l'incrimination de la ressource défectueuse. Par ailleurs, la problématique liée aux ressources indiscernables au cours du diagnostic introduit une source de complexité supplémentaire dans la vérification des systèmes sur la FAL. Les prochaines sections décrivent et proposent une méthode d'amélioration du degré de diagnostic dans une situation d'indiscernabilité afin de réduire d'avantage l'effort d'incrimination d'une ressource.

4.3.2 Description d'une méthode d'amélioration du diagnostic

L'existence de ressources indiscernables met en évidence la nécessité de définir des tests supplémentaires ou de prévoir des moyens de diagnostic appropriés permettant d'incriminer toute ressource défectueuse dans le système. En effet, il est souvent très difficile de définir des tests fonctionnels supplémentaires dans le but de dissocier le comportement de toutes les ressources utilisées par un système à ce niveau de vérification sur la FAL. Ceci entraîne la rencontre (assez fréquente) de situations d'indiscernabilité au cours de la vérification des systèmes sur la chaîne d'assemblage finale. Les ressources indiscernables sont une cause de la complexification et de l'augmentation de l'effort du diagnostic. Lorsqu'une campagne de test révèle une faute et que le diagnostic a abouti à la suspicion d'un sous-ensemble de ressources à partir des informations récoltées à l'issue de la vérification, d'autres activités de diagnostic sont nécessaires pour incriminer la ressource défectueuse contenue dans l'ensemble des ressources indiscernables. Cette activité d'incrimination peut s'avérer plus ou moins complexe et coûteuse en fonction du nombre de ressources impliquées dans cette situation d'indiscernabilité et de la nature des ressources. On parle de degré de diagnostic du système. Le degré de diagnostic d'un système indique la facilité à incriminer une ressource défectueuse après la détection d'une faute au cours de la phase de vérification. Ce degré est meilleur lorsque toutes les ressources utilisées par le système sont discernables. Il devient faible lorsque le système contient au moins un ensemble de ressources indiscernables.

Dans ce contexte, toute méthode permettant d'améliorer le degré de diagnostic sera très intéressante au cours de la vérification sur la FAL. Le processus d'application de Multiple-Clue, défini dans la section précédente, permet une analyse indépendante des différents objectifs de test définis pour le système. Par ailleurs, une ressource peut être utilisée dans la vérification de plusieurs objectifs de test (R_{i+h} dans la **Figure 4.10**). Cette caractéristique permet de prévoir cinq scénarii possibles, dans le contexte de l'existence d'une ressource défectueuse, à l'issue du diagnostic de deux objectifs de test « Obj1 » et « Obj2 » ayant au moins une ressource en commun :

a. la ressource commune peut être mise hors de cause à l'issue de la vérification de l'objectif « Obj1 » et appartenir à un ensemble de ressources indiscernables suspecté de l'objectif « Obj2 » ;

b. la ressource commune peut être incriminée à l'issue de la vérification de l'objectif « Obj1 » et appartenir à un ensemble de ressources indiscernables suspecté de l'objectif « Obj2 » ;

c. la ressource commune peut appartenir à un ensemble de ressources indiscernables suspecté à l'issue de la vérification des deux objectifs de test ;

d. La ressource commune peut être mise hors de cause à l'issue de la vérification des deux objectifs de test ;

e. La ressource commune peut être incriminée à l'issue de la vérification des deux objectifs de test.

L'objectif de la méthode d'amélioration du degré de diagnostic est de considérer ces différentes possibilités de diagnostic offertes par l'existence des ressources communes à plusieurs objectifs de test pour :

- aider à l'incrimination de la ressource défectueuse dans un ensemble de ressources indiscernables suspecté ;
- aider à la mise hors de cause de certaines ressources contenues dans un ensemble de ressources indiscernables suspecté.

L'idée de la méthode proposée est de se servir des informations issues du diagnostic de différents objectifs de test (a priori indépendants les uns des autres par rapport aux tests) afin de rendre discernable l'ensemble ou une partie des ressources indiscernables suspectées. Ceci contribuera à l'amélioration du degré de diagnostic et par conséquent, réduira l'effort consacré à l'incrimination des ressources au cours de la vérification des systèmes sur la FAL. Nous nous intéresserons aux scénarii de diagnostic **a**, **b** et **c** décrits ci-dessus. Les scénarii **d** et **e** sont considérés comme triviaux par rapport à l'incrimination des ressources du système.

Illustration de la méthode d'amélioration du diagnostic

Dans cette section, nous illustrons la méthode d'amélioration du diagnostic en considérant la situation suivante : une faute est détectée au cours de la vérification de l'objectif de test « Obj1 » représenté par les ressources qu'il utilise et les tests sélectionnés pour le diagnostic dans la **Figure 4.15** ci-dessous. Le modèle de cet objectif de test est composé de cinq ressources (R_1, R_2, R_3, R_4 et R_5) dont trois (R_1, R_4 et R_5) forment un ensemble de ressources indiscernables noté « B1 » et deux tests pertinents (T_1 et T_2) constituent l'ensemble de diagnostic de « Obj1 ».

	R_1	R_2	R_3	R_4	R_5
T_1	0	1	1	0	0
T_2	1	1	0	1	1

Figure 4.15 : Ressources et ensemble de diagnostic de « Obj1 »

L'arbre de diagnostic correspondant à la vérification de l'objectif de test « Obj1 » est construit dans la **Figure 4.16**.

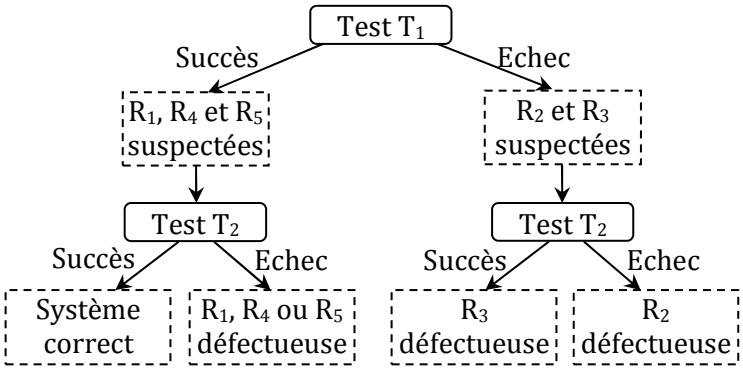


Figure 4.16 : Arbre de test de l'objectif de test « Obj1 »

Nous supposons que les informations récoltées à l'issue de l'exécution des tests indiquent le verdict suivant : T_1 est un « succès » et T_2 est un « échec ». Ceci permet de suspecter les ressources indiscernables (R_1, R_4 et R_5) et ce diagnostic ne peut être amélioré avec les informations disponibles sur « Obj1 ».

Afin d'améliorer cette situation, nous choisissons un autre objectif de test « Obj2 » composé de quatre ressources (R_1, R_5, R_6 et R_7) dont deux (R_1 et R_6) forment un ensemble de ressources indiscernables noté « B2 » et l'ensemble de diagnostic est formé de deux test (T_3 et T_4). Le modèle simplifié de l'objectif de test « Obj2 » est représenté ci-dessous. Les ressources « R_1 et R_5» sont utilisées dans la vérification des deux objectifs de test « Obj1 » et « Obj2 » ; par conséquent, elles représentent les ressources communes à ces deux objectifs de test.

	R_1	R_5	R_6	R_7
T_3	1	1	1	0
T_4	0	1	0	1

Figure 4.17 : Ressources et ensemble de diagnostic de « Obj2 »

L'arbre de diagnostic correspondant à la vérification de l'objectif de test « Obj2 » est construit dans la **Figure 4.18**.

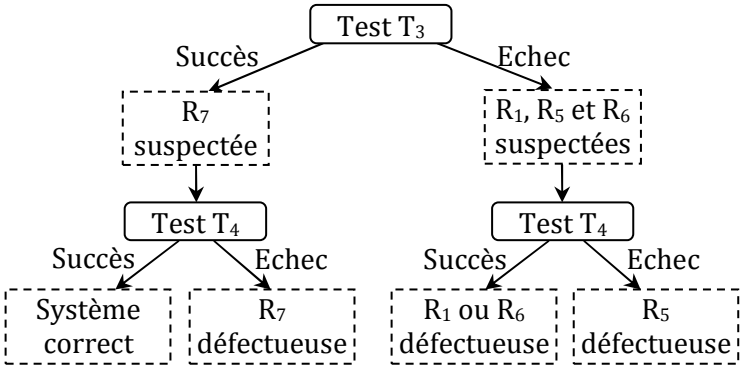


Figure 4.18 : Arbre de test de l'objectif de test « Obj2 »

A partir de ces informations, nous pouvons proposer des techniques d'amélioration du diagnostic de l'objectif de test « Obj1 » en fonction des informations du diagnostic de l'objectif de test « Obj2 » conformément aux scénarii **a**, **b** et **c** décrits dans la section précédente. L'hypothèse de l'existence d'une faute simple (une seule ressource défectueuse par objectif de test) est également considérée pour l'application de cette méthode d'amélioration.

Scénario (a) :

Ce scénario correspond à la situation représentée par la figure ci-dessous.

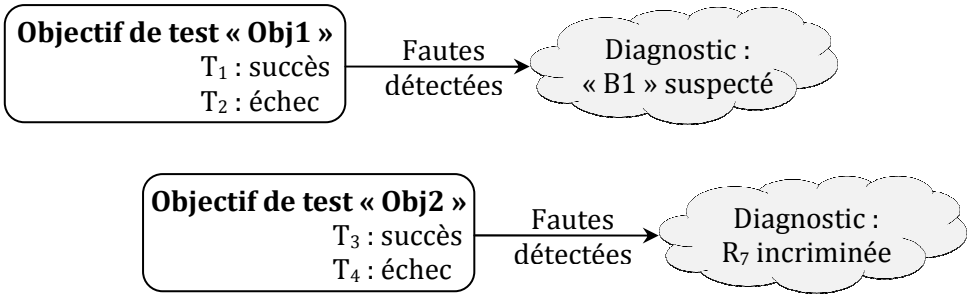


Figure 4.19 : Premier scenario d'amélioration du degré de diagnostic

L'analyse du verdict des tests de l'ensemble de diagnostic de l'objectif de test « Obj2 » donne lieu à l'incrimination de la ressource R_7. En considérant l'hypothèse de l'existence d'une ressource défectueuse dans le système, les ressources R_1 et R_5 peuvent être mises hors de cause dans le diagnostic de « Obj1 ». En effet, le diagnostic de l'objectif de test « Obj2 » montre que ces deux ressources sont correctes. De ce fait, le diagnostic de l'objectif « Obj1 » se trouve améliorer avec l'incriminant de la ressource « R_4 ».

Scénario (b) :

Ce scénario correspond à la situation représentée par la figure ci-dessous.

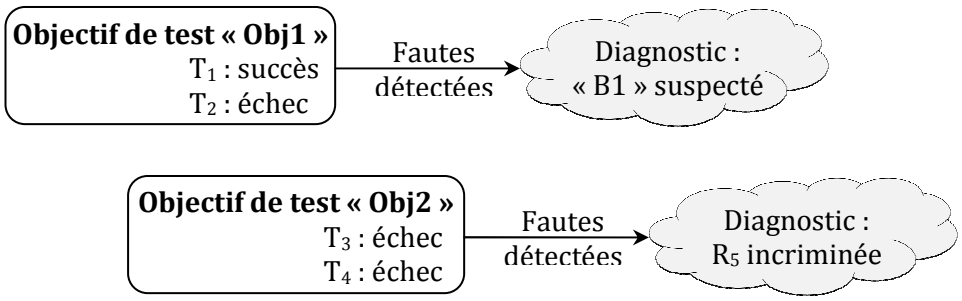


Figure 4.20 : Deuxième scénario d'amélioration du degré de diagnostic

Les informations sur le verdict des tests de l'ensemble de diagnostic de l'objectif de test « Obj2 » permettent d'incriminer R_5 à l'aide de son arbre de test (**Figure 4.20**). A partir de ces données, les ressources R_1 et R_4 peuvent être considérées comme correctes. En effet, le diagnostic de l'objectif « Obj2 » incrimine la ressource R_5. De ce fait, cette ressource peut également être identifiée comme la ressource défectueuse pour améliorer le diagnostic de l'objectif « Obj1 ».

Scénario (c) :

Ce scénario peut être illustré à l'aide de la figure ci-dessous.

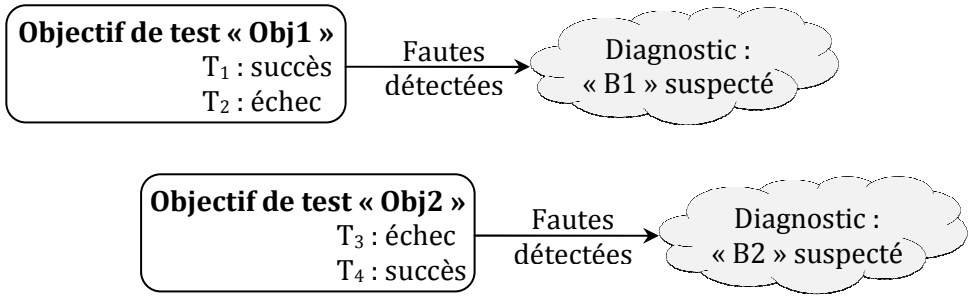


Figure 4.21 : Troisième scénario d'amélioration du degré de diagnostic

L'analyse du verdict des tests de l'ensemble de diagnostic de l'objectif de test « Obj2 » donne lieu à la suspicion de l'ensemble des ressources indiscernables « B2 ». A partir de ces informations, nous pouvons considérer que la ressource R_5 est correcte. En effet, le diagnostic de l'objectif de test « Obj2 » la met hors de cause. Par ailleurs, la ressource « R_1 » est commune aux deux ensembles de ressources suspectés. De ce fait, elle peut être incriminée pour améliorer le degré de diagnostic de l'objectif de test « Obj1 ».

Synthèse

Les différents scénarii, décrits et illustrés à l'aide de deux objectifs de test, montrent qu'il est possible d'améliorer le diagnostic à l'issue l'application de la stratégie Multiple-Clue dans le contexte de la FAL. La méthode d'amélioration permet de fournir des informations aidant à la

déduction de l'état d'une ou plusieurs ressources suspectées à l'issue du diagnostic d'un objectif de test. Elle permet de réduire le nombre de ressources suspectées ou d'incriminer la ressource défectueuse appartenant à un ensemble de ressources indiscernables suspecté. Cette méthode d'amélioration du degré de diagnostic consiste à améliorer le diagnostic d'un objectif de test en utilisant la technique par recoupements avec les informations issues des résultats de diagnostic de plusieurs objectifs de test. La section suivante (section 4.3.3) propose une formalisation permettant d'automatiser la mise en œuvre de la méthode d'amélioration.

4.3.3 Formalisation de la méthode d'amélioration du diagnostic

La formalisation de la méthode d'amélioration du degré de diagnostic consiste à représenter certaines étapes sous la forme d'opérations automatisables. Les étapes de la méthode d'amélioration peuvent être regroupées en deux principales phases : l'identification des objectifs de test utiles et l'application de la technique d'affinage permettant de réduire le nombre de ressources suspectées ou d'identifier la ressource défectueuse appartenant à un ensemble de ressources indiscernables suspecté.

Identification des objectifs de test utiles

Le degré de diagnostic d'un système doit être amélioré, si possible, lorsqu'un ensemble de ressources indiscernables est suspecté. Un objectif de test peut être utile pour l'amélioration du diagnostic dans une situation d'indiscernabilité lorsque sa vérification utilise certaines ressources contenues dans l'ensemble des ressources indiscernables suspecté. En d'autres termes, un objectif de test utile partage certaines ressources avec l'objectif de test en cours de vérification et notamment celles qui sont suspectées par le diagnostic. En rappelant la représentation de la matrice de diagnostic d'un système dans le FAL, dans la figure ci-dessous, la ressource R_{i+h} est partagée par les objectifs de test « Obj1 » et « Obj2 ».

		…	R_i	R_{i+1}	…	R_{i+h}	…	R_{i+k}	…
Objectif de test « Obj1 »	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
	T_i								
	T_{i+i}								
Objectif de test « Obj2 »	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
	T_{i+j}								
	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
	T_{i+k}								
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Les matrices de diagnostic des objectifs de test utiles sont appelées matrices exploitables pour un ensemble de ressources indiscernables. Lorsque la ressource R_{i+h} fait partie d'un ensemble de ressources suspecté au cours de la vérification de l'objectif de test « Obj1 », la matrice de diagnostic de l'objectif de test « Obj2 » fera partie des matrices exploitables qui permettront éventuellement d'améliorer ce diagnostic. R_{i+h} est appelée ressource commune aux matrices exploitables.

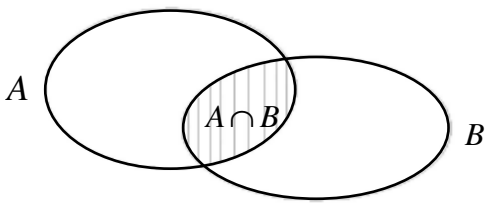
L'identification des objectifs de test utiles permet de guider les activités de vérification suivantes afin d'améliorer le degré de diagnostic à l'aide de la technique proposée dans la section suivante.

Mise en œuvre de la technique d'amélioration

La technique d'amélioration du degré de diagnostic permet de réduire le nombre de ressources suspectées ou d'isoler la ressource défectueuse d'un ensemble de ressources indiscernables. Elle est essentiellement basée sur deux opérations. En effet, cette approche consiste à utiliser le résultat de diagnostic des matrices exploitables pour réaliser les recoupements principalement à l'aide d'opérations sur les ensembles de ressources.

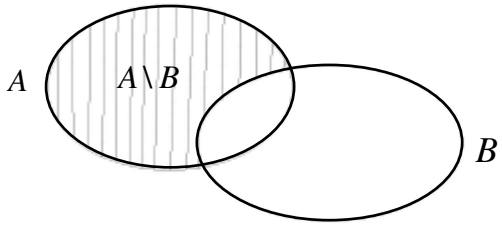
Soient « Obj_I » un objectif de test dont le diagnostic suspecte un ensemble de ressources indiscernables « ERIS » et « Obj_U » un objectif de test utile pour l'amélioration du degré de diagnostic de l'objectif de test « Obj_I ». Dans le contexte de l'existence d'une faute dans le système, la technique d'amélioration du diagnostic se fait comme suit :

- Lorsque le résultat du diagnostic de « Obj_U » suspecte un ensemble de ressources ayant au moins une ressource en commun avec « ERIS », le périmètre de recherche de la ressource défectueuse se limite au sous-ensemble de ressources commun aux deux ensembles de ressources suspectés respectivement pour « Obj_I » et « Obj_U ». Ceci permet de réduire le nombre de ressources impliquées dans le diagnostic. Le sous-ensemble de ressources commun est déterminé à l'aide d'une opération d'intersection entre deux ensembles. Cette opération est illustrée en considérant le schéma ci-dessous (A et B sont deux ensembles finis d'éléments).



$$A \cap B = \{x \ tels \ que \ x \in A \ \underline{et} \ x \in B\}$$

– De la même manière, lorsque le résultat du diagnostic de « Obj_U » met hors de cause l'ensemble des ressources partagées, le périmètre de recherche de la ressource défectueuse se restreint aux ressources suspectées et non partagées contenues dans l'objectif de test « Obj_I ». Ceci permet également de réduire le domaine de recherche de la ressource responsable de la faute. Ce domaine réduit est obtenu en réalisant une opération de différence entre « ERIS » et les ressources communes utilisées par « Obj_U ». L'opération de différence est illustrée à l'aide du schéma ci-dessous (A et B sont deux ensembles finis d'éléments).



$$A \setminus B = \{x \text{ tels que } x \in A \text{ et } x \notin B\}$$

Processus de diagnostic

Les différentes étapes du processus d'amélioration du degré de diagnostic guidé d'un objectif de test « Obj_I » sont représentées dans la **Figure 4.22** ci-dessous. Ce processus est principalement composé de sept étapes : A, B, C, D, E, F, G et H. Les conditions nécessaires au passage à certaines étapes sont présentées ci-dessous. Soient « ERIS » l'ensemble de ressources indiscernables suspecté à l'issue du diagnostic de « Obj_I », « ERS » l'ensemble de ressources suspecté à l'issue du diagnostic d'un objectif de test utile « Obj_U » et « ERNS » l'ensemble de ressources non suspecté à l'issue du diagnostic d'un objectif de test utile « Obj_U ».

– C_0 : la ressource défectueuse est incriminée à l'issue de la construction de l'arbre de test de l'objectif « Obj_I ».
– C_1 : Aucun objectif de test utile « Obj_U » n'a été trouvé ou tous les objectifs de test utiles ont été traités.
– C_2 : Certains objectifs de test utiles ont été déjà vérifiés avec un diagnostic établi.
– C_3 : Certaines ressources de « ERIS » sont également suspectées à l'issue du diagnostic de « Obj_U ».
– C_4 : les ressources partagées sont mises hors de cause à l'issue du diagnostic de « Obj_U ».
– C_5 : le nombre de ressources suspectées est réduit à une seule ressource.

Etape A : Cette étape correspond à la construction automatique de l'arbre de test de l'objectif de test « Obj_I » en utilisant le verdict de l'ensemble de diagnostic renseigné par le testeur.

Etape B : Cette activité consiste à chercher les objectifs de test utiles pour l'amélioration du degré de précision du diagnostic. Ces objectifs de test correspondent à ceux qui utilisent certaines ressources impliquées dans la situation d'indiscernabilité.

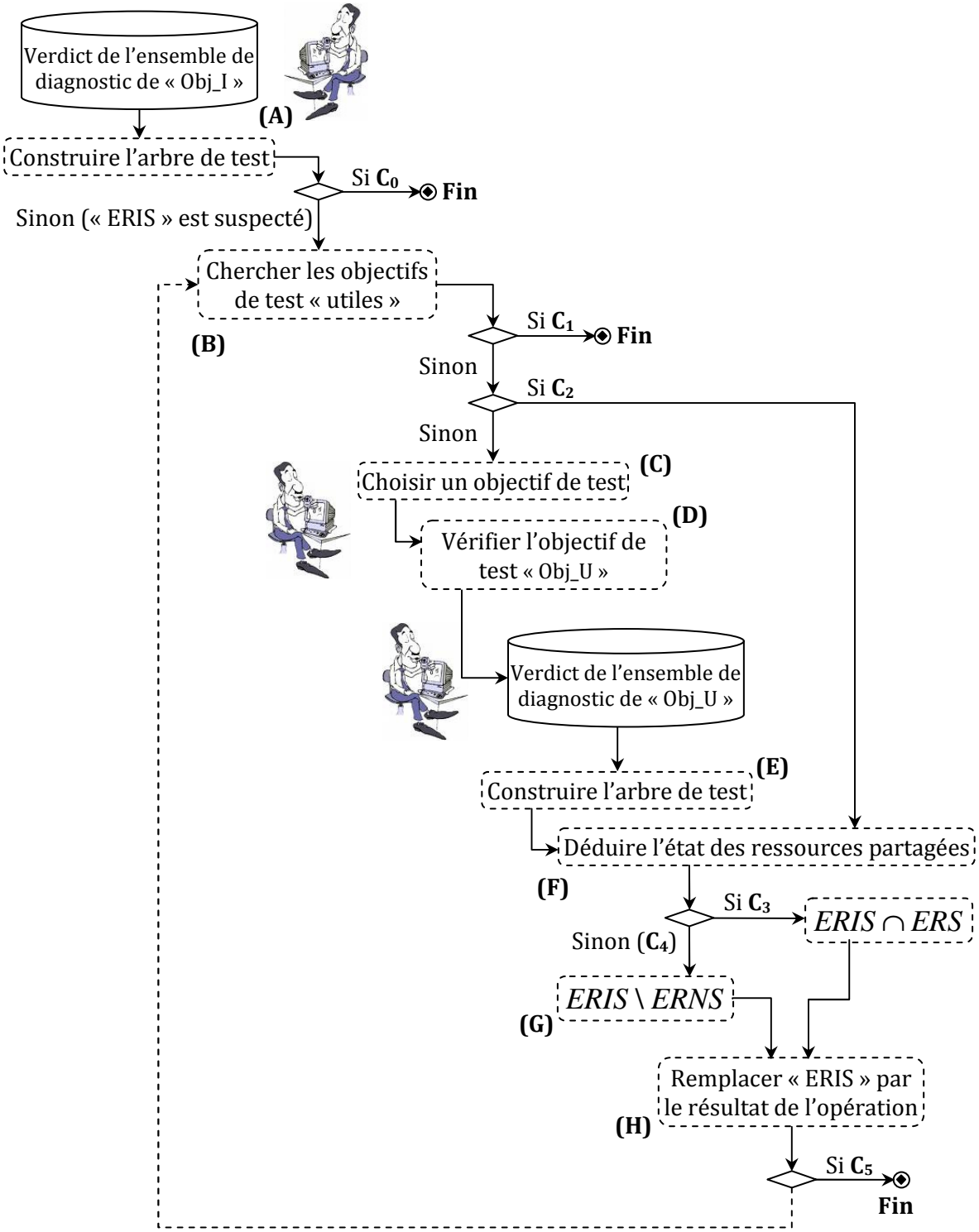


Figure 4.22 : Processus de diagnostic guidé

Etape C : Cette étape nécessite l'intervention d'un opérateur. En effet, le testeur doit choisir l'objectif de test utile « Obj_U », dans la liste proposée, le plus proche dans l'ordre de vérification des objectifs de test définis dans le document GTI (Ground Test Instructions) pour le diagnostic.

Etape D : l'objectif de test choisi est vérifié à l'aide des instructions de test définies dans le GTI au cours de cette étape. Cette phase de vérification est réalisée soit au moment du choix, dans ce cas l'objectif de test utile « Obj_U » suit le dernier objectif vérifié dans l'ordre du GTI ; soit ultérieurement lorsque les prochains objectifs à vérifier dans l'ordre du GTI ne peuvent pas aider à améliorer le diagnostic de « Obj_I ».

Etape E : Cette étape correspond à la construction de l'arbre de test de « Obj_U » à partir du verdict de l'ensemble de diagnostic extrait du verdict global de l'exécution de l'ensemble des instructions de tests qui lui sont associées.

Etape F : L'état des ressources partagées est déduit de la lecture de l'arbre de diagnostic au cours de cette étape.

Etape G : La réduction du périmètre de diagnostic est effectuée dans cette partie du processus. Elle consiste à réaliser automatiquement soit une opération d'intersection lorsque certaines ressources partagées utilisées dans « Obj_U » sont suspectées ; soit une opération de différence lorsque les ressources partagées utilisées dans « Obj_U » sont mises hors de cause.

Etape H : Cette dernière étape consiste à remplacer l'ensemble des ressources indiscernables suspecté dans « ERIS » par le résultat de l'opération réalisée dans l'étape précédente.

4.3.4 Implantation de la méthode de diagnostic guidée

Cette section propose une description algorithmique des principales procédures implantées pour la réalisation des fonctions définies dans les étapes du processus de diagnostic guidé (**Figure 4.22**). Ce processus peut être représenté par trois procédures : la première recherche les objectifs de test utiles, la deuxième réduit le périmètre de diagnostic et la troisième qui utilise les deux premières permet d'améliorer le diagnostic. Les notations ci-dessous seront utilisées dans les descriptions algorithmiques de ces procédures.

- ER_i représente l'ensemble des ressources utilisées dans la vérification d'un objectif de test OT_i.
- ERS_i correspond à l'ensemble des ressources suspectées à l'issue du diagnostic de OT_i. Cet ensemble est vide lorsque toutes les ressources sont correctes.

- $ERNS_i$ est l'ensemble des ressources mises hors de cause (ou non suspectées) à l'issue du diagnostic de OT_i. Cet ensemble est vide lorsque toutes les ressources appartiennent au même ensemble de ressources indiscernables suspecté.
- $LOTU_i$ représente la liste des objectifs de test utiles pour l'amélioration du diagnostic de OT_i.
- EOT correspond à l'ensemble des objectifs de test définis dans le GTI pour la vérification du système.

Procédure de recherche des objectifs de test utiles (ROTU)

La procédure ROTU cherche et fournit une liste des objectifs de test utiles utilisant certaines ressources contenues dans l'ensemble ERS_i d'un objectif de test OT_i. Elle prend en entrée ERS_i et retourne la liste éventuellement vide $LOTU_i$ des objectifs de test utiles. Cette procédure réalise les fonctions définies dans l'étape (B) du processus de diagnostic. Sa description algorithmique se trouve ci-dessous.

```
Procédure ROTU ( ERS_i)
    Début
    .    LOTU_i ← vide(liste)
    .    Pour chaque OT_k ∈ EOT faire
    .    .    Si ERS_i ∩ ER_k ≠ ∅  /* OT_k utilise certaines ressources de ERS_i */
    .    .    .    Alors Enfiler(LOTU_i, OT_k)
    .    .    Fin_Si
    .    Fin_Pour
    .    Retourner LOTU_i
    Fin
```

Procédure de réduction du périmètre de diagnostic (RPD)

La procédure RPD réduit le domaine de recherche de la ressource défectueuse contenue dans l'ensemble ERS_i en utilisant le résultat du diagnostic d'un objectif de test utile OT_j. Elle prend en entrée l'ensemble ERS_i et l'objectif de test OT_j et retourne le domaine réduit sur lequel il faudra se focaliser pour identifier la ressource défectueuse. Les étapes (G) et (H) sont considérées dans cette procédure. Sa description algorithmique se trouve ci-dessous.

```
Procédure RDP ( ERS_i , OT_j )
  Début
  .   S ← ∅
  .   Si |ERS_j| = 0      /* Toutes les ressources de OT_j sont correctes */
  .   .   Alors S ← ERS_i \ ERNS_j /*Différence entre ERS_i de OT_i et ERNS_j de OT_j */
  .   .   Sinon
  .   .   .   Si ERS_i ⊂ ERS_j /*Toutes les ressources de ERS_i sont suspectées dans OT_j */
  .   .   .   .   Alors S ← ERS_i
  .   .   .   .   Sinon S ← ERS_i ∩ ERS_j /*Intersection entre ERS_i de OT_i et ERS_j de OT_j */
  .   .   .   Fin_Si
  .   Fin_Si
  .   Retourner S
  Fin_RDP
```

Procédure d'amélioration du diagnostic d'un objectif de test (ADO)

La procédure ADO réalise les fonctionnalités permettant de guider les activités de diagnostic d'un objectif de test OT_i dans une situation d'indiscernabilité afin de l'améliorer. Elle réalise les étapes (A), (C), (D) et (F). Cette procédure prend comme paramètre d'entrée OT_i et retourne quatre types d'information décrites ci-dessous :

- « *Pas besoin d'amélioration* » : ce résultat signifie que soit toutes les ressources utilisées par OT_i sont correctes, soit la ressource défectueuse est déjà identifiée.
- « *Résultat incohérent* » : cette situation implique que les informations fournies par les objectifs de test utiles révèlent que les ressources suspectées dans ERS_i à l'issue du diagnostic de OT_i sont correctes. Dans ce contexte, les verdicts des tests exécutés pour la vérification de OT_i et des objectifs de test utiles doivent être réexaminés afin d'identifier et corriger l'incohérence.
- « *Amélioration optimale* » : ce résultat est obtenu lorsque le processus parvient à identifier la ressource défectueuse contenue dans l'ensemble ERS_i de OT_i. En d'autres termes, ceci revient à dire que le périmètre du diagnostic a été réduit à une seule ressource.
- « *Ressources récalcitrantes* » : lorsque la procédure n'est pas capable d'identifier la ressource défectueuse en utilisant les informations fournies par les objectifs de test utiles. Elle réduit, par conséquent, au mieux le nombre de ressources indiscernables suspectées.

La description algorithme (ci-dessous) de cette procédure nécessite la prise en compte : d'une fonction nommée $Est\_Vérifié(OT_i)$ qui retourne « vrai » lorsque toutes instructions de test définies pour OT_i ont été exécutées et que le diagnostic est déjà établi et « faux » sinon ; d'une activité nommée $Vérifier(OT_i)$ qui consiste à exécuter les instructions de test associées à OT_i et à

rendre le verdict pour le diagnostic ; et une dernière une fonction nommée $Arbre\_Test(OT_i)$ qui construit automatiquement de l'arbre de test de OT_i à partir de l'ensemble de diagnostic.

```
Procédure ADO ( OT_i )
  Début
  .   Arbre_Test(OT_i)
  .   Si |ERS_i| = 0 or |ERS_i| = 1
  .   .   Alors
  .   .       Retourner Afficher (« Pas besoin d'amélioration »)
  .   Fin_Si
  .   LOTU_i ← ROTU(OT_i)   /* Recherche des objectifs de test utiles */
  .   Tant que LOTU_i ≠ vide(liste)
  .   .   OT_j ← Défiler(LOTU_i) /*Enlever l'objectif de test choisi de la file LOTU_i */
  .   .   Si Est_Vérifié(OT_j) = faux
  .   .   .   Alors Vérifier(OT_j)
  .   .   Fin_Si
  .   .   ERS_i ← RDP(ERS_i, OT_j)
  .   .   Si |ERS_i| = 0
  .   .   .   Alors
  .   .   .   .   Afficher (« Résultat incohérent »)
  .   .   .   .   Arrêter (AD)   /* Arrêt de l'exécution de la procédure */
  .   .   Fin_Si
  .   .   Si |ERS_i| = 1
  .   .   .   Alors Arrêter (tant que)   /* Arrêt de la boucle */
  .   .   Fin_Si
  .   Fin_Tant_que
  .   Si |ERS_i| ≠ 1
  .   .   Alors
  .   .   .   Afficher (« ressources récalcitrantes »)
  .   .   .   Retourner ERS_i
  .   .   Sinon
  .   .   .   Afficher (« Amélioration optimale »)
  .   .   .   Retourner ERS_i
  .   Fin_Si
  Fin_ADO
```

4.3.5 Intégration de la méthodologie dans la vérification sur la FAL

La méthodologie décrite dans ce sous-chapitre propose de combiner la stratégie de test Multiple-Clue et la méthode de diagnostic guidée afin de fournir des informations pertinentes permettant d'aider le processus de diagnostic au cours de la vérification des systèmes sur la FAL. La **Figure 4.23** représente le processus de vérification des systèmes complété avec les différentes étapes de la méthodologie ainsi que les informations proposées au cours des deux

phases de conception des tests et de diagnostic. Ceci fournit une vue globale des avantages de cette méthodologie en termes de :

- aide à la définition des tests pertinents et non redondants pour la couverture des exigences fonctionnelles des systèmes au cours de la conception des tests ;
- aide à l'incrimination de la ressource défectueuse et d'aide à l'amélioration du degré de diagnostic dans une situation d'indiscernabilité au cours du diagnostic.

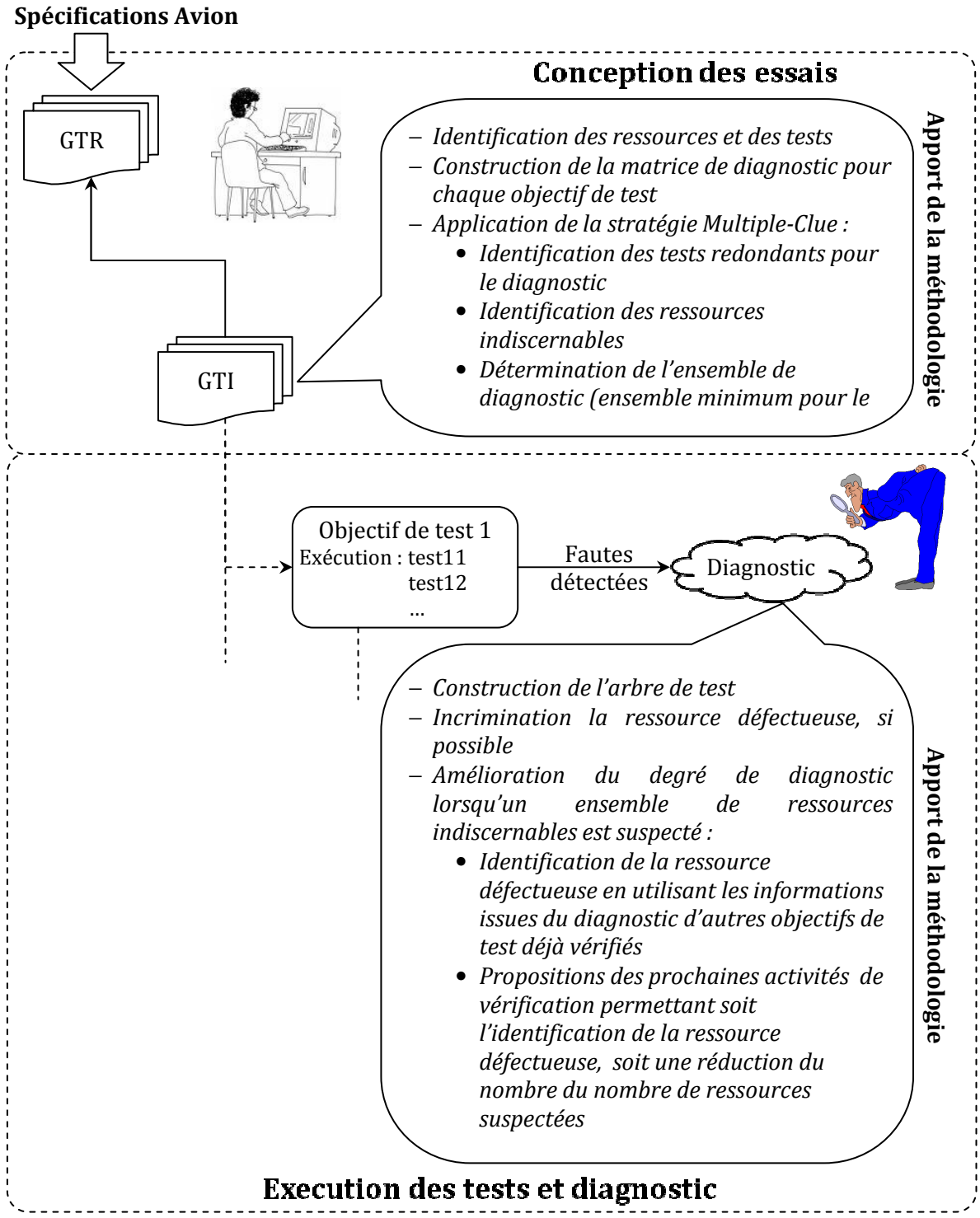


Figure 4.23 : Processus de vérification amélioré

4.3.6 Synthèse

Nous avons décrit la méthodologie développée pour l'aide à la vérification des systèmes sur la chaîne d'assemblage finale d'avions AIRBUS (FAL) en utilisant la stratégie de test Multiple-Clue. Nous avons défini les différentes étapes d'application de cette stratégie dans le processus de vérification des systèmes. Une méthode d'amélioration du degré de diagnostic adaptée à la méthodologie de vérification des systèmes sur la FAL a été proposée. Une formalisation et une implantation de cette méthode ont été également décrites afin de permettre son intégration dans la méthodologie d'aide à la vérification des systèmes sur la chaîne d'assemblage finale.

4.4 Prototype ADIS

Le prototype ADIS (**A**ide au **DI**agnostic de **S**ystèmes) a été développé pour l'implantation des méthodes définies dans la méthodologie d'aide au diagnostic des objectifs de test définis pour la vérification des systèmes sur la FAL. Les sections ci-dessous présentent les principales données de traitement et les fonctionnalités proposées par le prototype. L'analyse fonctionnelle réalisée au cours de la conception ainsi que les principales interfaces de cet outil sont présentées en annexe (section E et section F).

4.4.1 Données de traitement

Les données de traitement, bases d'information, pour l'aide au diagnostic au cours de la vérification des objectifs de test sont décrites ci-dessous.

- Ressources : Cette base contient les informations liées à l'ensemble de ressources utilisées par les différents objectifs de test.
- Tests : Cette base contient les informations liées à l'ensemble des tests définis pour la vérification des différents objectifs de test.
- Sauvegarde locale : Cette base contient les informations liées, pour chaque objectif de test traité, aux verdicts des tests exécutés ainsi que l'état des ressources utilisées à l'issue du diagnostic. La structure interne de cette base est présentée dans le chapitre de l'analyse fonctionnelle du prototype en annexe (section E).
- Sauvegarde globale : Cette base contient les informations liées à la synthèse des différents états de chaque ressource à partir des diagnostics déjà réalisés. La structure interne de cette base est présentée dans le chapitre de l'analyse fonctionnelle du prototype en annexe (section E).

4.4.2 Fonctionnalités

Le prototype ADIS permet principalement : de visualiser des informations sur un objectif de test, de supprimer les informations liées à un objectif de test, de réaliser le diagnostic d'un objectif de test, de proposer des activités dans le but d'améliorer le degré de diagnostic et de gérer les traces des différentes analyses.

Visualisation

Cette fonctionnalité est utile au niveau des deux phases de conception des essais et de diagnostic. Les informations issues de ce traitement permettent :

- A la conception des essais, d'indiquer les tests redondants, les tests pertinents (ensemble de diagnostic) ainsi que les ressources indiscernables pour le diagnostic d'un objectif de test ;
- Au diagnostic, de proposer le résultat du diagnostic d'un objectif de test lorsqu'il a été déjà réalisé.

La fonctionnalité de visualisation des informations d'un objectif de test utilise les bases de ressources, de tests et la sauvegarde locale.

Suppression d'un objectif de test

Cette fonctionnalité supprime les informations relatives à un objectif de test dans les différentes données de traitement du prototype. Elle permet de ne plus considérer l'objectif de test au cours des différentes analyses d'aide au diagnostic. Cette fonctionnalité peut être utile à la suite de la modification du GTI à la conception des essais et au cours du diagnostic. Elle utilise les bases de ressources, de tests. Cette suppression est notifiée dans la base de traces.

Diagnostic d'un objectif de test

La fonctionnalité de diagnostic d'un objectif de test permet de construire automatiquement l'arbre de test à partir des verdicts des tests de l'ensemble de diagnostic et de déduire l'état des différentes ressources utilisées. Elle utilise la base de ressources, la base de tests. Le résultat du diagnostic est enregistré dans la base de sauvegarde locale et de sauvegarde globale. Cette opération est notifiée dans la base de traces.

Amélioration du degré de diagnostic

Cette fonctionnalité propose des indications sur la suite des activités de diagnostic et de vérification permettant d'améliorer le résultat d'un diagnostic. Elle utilise les bases de ressources, de tests, de sauvegarde locale, de sauvegarde globale. Le résultat de l'amélioration est enregistré dans la base de sauvegarde locale et de sauvegarde globale. Cette action est notifiée dans la base de traces.

Gestion des traces

La fonctionnalité de gestion des traces met en place une politique de gestion des informations enregistrées sur l'ensemble des activités de diagnostic réalisées par le prototype. Elle utilise la base des traces.

4.4.3 Synthèse

Nous avons présenté, dans ce sous chapitre, les principales fonctionnalités du prototype développé pour l'implantation de la méthodologie d'aide à la vérification des systèmes proposée pour FAL. Les fonctionnalités implantées prennent en compte la méthodologie de vérification des systèmes sur la chaîne d'assemblage finale et fournissent des informations pertinentes permettant d'aider et de guider les phases de conception des tests et de diagnostic après la détection d'une faute au cours de l'exécution des tests.

4.5 Expérimentations

Les expérimentations menées autour de la définition de la méthodologie d'aide à la vérification des systèmes se sont déroulées en collaboration avec le département « Test au sol » sur la FAL du programme A330–340 d'AIRBUS OPERATIONS (site de Toulouse). Dans ce sous chapitre, nous présentons les principales étapes ainsi que les résultats de ces expérimentions. Le système qui a servi de cas d'étude est celui de « l'Orientation Roues Avant (ORA) » d'un avion de type A340 basic. Les activités de vérification concernant ce système consistent à contrôler son fonctionnement ainsi que l'interconnexion des composants qu'il utilise. Cette vérification est réalisée à l'aide de tests fonctionnels définis à partir des exigences exprimées par le bureau d'étude.

Les principales motivations de ces travaux d'expérimentation sur la méthodologie d'aide à la vérification des systèmes sur FAL sont :

- la modélisation d'un système « réel » sous une représentation matricielle en termes de ressources et tests ;
- l'analyse des matrices de diagnostic associées à chaque objectif de test défini dans le GTI à l'aide de la stratégie Multiple-Clue afin de fournir des informations pertinentes pour la conception des tests et le diagnostic ;
- la mise en œuvre de la méthode d'amélioration du degré de diagnostic.

Le GTI du système « ORA » a été fourni dans le cadre de ces expérimentations. Nous présentons d'abord les informations sur la modélisation du système ; ensuite nous exposons la synthèse des résultats de l'application de Multiple ; enfin les bénéfices de la méthode d'amélioration du diagnostic seront décrits.

4.5.1 Modélisation du système

Le GTI du système « d'Orientation Roues Avant » a été modélisé conformément aux règles de modélisation décrites dans la section 4.3.1. Les différentes phases de configuration du système n'ont pas été prises en compte au cours de cette modélisation. En effet, nous nous sommes intéressés aux instructions du GTI assurant la vérification des objectifs de test. La table ci-dessous donne une représentation du modèle issue de la modélisation du GTI du système « ORA ». Les ressources modélisées sont : d'une part, des composants physiques tels que les volants, pédales ou palonniers, boutons, disjoncteurs, etc. et d'autre part, les éléments tels que les sortie des calculateurs, les indicateurs (ou paramètres) des bus d'acquisition, les sorties des capteurs, etc.

Tableau 4.1 : Modèle du système « ORA » d'un A340 basic

	Nombre
Objectifs de test	26
Ressources utilisées	100
Tests modélisés	169
Nombre de ressources par objectif de test	de 2 à 18
Nombre de tests par objectif de test	de 2 à 18

Les informations fournies par le tableau ci-dessus indique que : la modélisation du système « ORA » a donné lieu à l'identification de 26 objectifs de test définis dans le GTI ; 100 ressources sont utilisées et 169 tests ont été définis pour la vérification de ces objectifs de test. Le nombre de ressources utilisées, par objectif de test, varie de 2 à 18 et le nombre de tests modélisés, par objectif de test, varie également de 2 à 18. Parmi les ressources modélisées, 57 ressources sont utilisées dans au moins deux objectifs de test différents. Ceci confirme le constat réalisé au cours de la description de la méthode de modélisation des systèmes. De ce fait, la méthode d'amélioration du degré du diagnostic pourra être utilisée, si nécessaire, pour réduire l'effort de la vérification des systèmes sur la FAL.

4.5.2 Synthèse de l'analyse des objectifs de test

La deuxième étape de la méthodologie consiste en l'analyse des matrices de diagnostic, extraites du modèle global, associées aux objectifs de test définis pour la vérification du système. A l'issue de cette phase d'analyse, les informations suivantes sont fournies pour chaque objectif de test :

- la liste des ensembles de ressources indiscernables ;
- la liste des tests redondants pour le diagnostic ;
- la liste des tests pertinents pour le diagnostic formant l'ensemble de diagnostic.

Ces informations peuvent donner quelques indications par rapport à la qualité des tests définis au cours de la conception des tests et à l'effort du diagnostic. Une synthèse des résultats de l'analyse des objectifs de test est présentée dans le tableau ci-dessous. Le détail de ces résultats se trouve en annexe (section G.1).

Tableau 4.2 : Synthèse des résultats de l'analyse des objectifs de test du système « ORA »

Informations fournies par Multiple-Clue	Min	Max
Nombre de test redondants (1)	0	6
Nombre d'ensembles de ressources indiscernables (2)	0	6
Taille des ensembles de ressources indiscernables (3)	2	7
Taille de l'ensemble de diagnostic (4)	2	6

Le tableau ci-dessus quantifie les informations fournies à l'issue de l'analyse des objectifs de test par Multiple-Clue en utilisant les bornes minimales et maximales. Cette synthèse indique les caractéristiques suivantes :

- (1) : le nombre de tests redondants pour le diagnostic identifiés au cours de l'analyse varie de 0 à 6 pour chaque objectif de test. En effet, il existe certains objectifs de test pour lesquels il n'existe aucun test redondant défini au cours de la conception des tests et certains pour lesquels jusqu'à 6 tests redondants ont été détectés au cours de l'analyse.
- (2) : le nombre d'ensembles de ressources indiscernables identifiés au cours de l'analyse varie aussi de 0 à 6 pour chaque objectif de test. En effet, il y a des objectifs de test pour lesquels toutes les ressources sont discernables à partir des tests définis donc ne contenant aucun ensemble de ressources indiscernables. Par ailleurs, il existe certains objectifs de test contenant jusqu'à 6 ensembles de ressources indiscernables différents. En considérant le système « ORA », 26 ensembles de ressources indiscernables différents impliquant 47 ressources ont été identifiés à l'issue de l'application de Multiple-Clue.
- (3) : la taille des différents ensembles de ressources indiscernables détectés se situe entre 2 et 7. L'analyse d'un objectif de test a montré l'existence d'un ensemble constitué de 7

ressources sur les 9 qu'il utilise. Par conséquent, le degré de diagnostic de cet objectif de test peut être qualifié de « mauvais ».

- (4) : le nombre des tests, sélectionnés par Multiple-Clue, constituant l'ensemble de diagnostic va de 2 à 6 selon les objectifs de test. En effet, les résultats d'analyse ont montré qu'il existe plus ou moins de tests non pertinents pour le diagnostic en fonction des objectifs de test. Pour illustration, seulement 6 tests ont été identifiés comme étant des tests pertinents pour le diagnostic sur les 18 initialement définis pour la vérification d'un objectif de test.

4.5.3 Mise en œuvre de la méthode d'amélioration

Les résultats de l'analyse à l'aide Multiple-Clue ont permis de mettre en évidence les informations suivantes sur les ressources indiscernables :

- 47 ressources modélisées appartiennent à au moins un ensemble de ressources indiscernables ;
- La taille moyenne des ensembles de ressources indiscernables est de 4 à 5 ressources.

Par ailleurs, 57 ressources sur les 100 modélisées sont utilisées par au moins deux objectifs de test différents. Ces informations montrent que la méthode d'amélioration du degré de diagnostic peut être utilisée, si nécessaire, pour réduire l'effort de diagnostic.

En considérant différents scénarii de diagnostic, la méthode d'amélioration a été mise en œuvre afin d'évaluer son efficacité sur un système « réel » dans le contexte de vérification des systèmes sur FAL. Le tableau ci-dessous présente le résultat global de l'application de cette méthode sur le système « ORA » en termes de ressources indiscernables. Le détail de ce résultat se trouve en annexe (section G.2).

Tableau 4.3 : Résultat global de la méthode d'amélioration du diagnostic

	Nombre de ressources indiscernables
Avant la mise en œuvre de la méthode d'amélioration	47
Après la mise en œuvre de la méthode d'amélioration	33

Globalement, le nombre de ressources indiscernables s'est trouvé réduit à l'issue de l'application de la méthode d'amélioration du degré du diagnostic. En effet, les expérimentations ont montré que 14 ressources peuvent devenir discernables en considérant certains scénarii de diagnostic au cours de la vérification du système « ORA ». Le nombre d'ensembles de ressources indiscernables différents a été également réduit à 13. La taille moyenne des ensembles de ressources indiscernables est aussi passée à 2 ou 3 ressources au lieu de 4 ou 5.

4.5.4 Synthèse des gains de l'application de la méthodologie

Dans cette section, nous décrivons les principaux avantages de l'application de la méthodologie d'aide au diagnostic en se basant sur les résultats obtenus à partir du système « ORA ». Le tableau ci-dessous quantifie les informations utiles sur le diagnostic avant et après la mise en œuvre de cette méthodologie. Dans ce tableau, la colonne « Avant » correspond aux informations recueillies à partir des retours d'expériences des ingénieurs et testeurs sur la FAL ; la colonne « Après » correspond aux informations fournies par la méthodologie définie et décrite dans ce chapitre.

Tableau 4.4 : Informations quantitatives sur l'effort de diagnostic du système « ORA »

	Nombre	
	Avant	Après
Tests pertinents pour le diagnostic	169	80
Tests redondants pour le diagnostic	inconnu	30
Ensembles de ressources indiscernables	26	13
Taille moyenne des ensembles indiscernables	4 ou 5	2 ou 3

Ce tableau met en évidence la réduction significative des informations à considérer pour le diagnostic du système « ORA ». Ces informations pourront être utilisées pour aider à la vérification des systèmes sur la FAL dès la phase de conception des tests ainsi que celle du diagnostic après la détection d'une faute.

En ce qui concerne la conception des tests, cette aide passe par la prise en compte des informations sur les tests redondants, les ensembles de ressources indiscernables et les tests pertinents pour le diagnostic.

- Les tests redondants : 30 tests parmi les 169 initialement définis sont considérés comme redondants pour le diagnostic. Ceci représente un gain de 17% des tests à considérer pour le diagnostic. Ces informations peuvent être exploitées par les concepteurs de test afin de vérifier si les tests redondants pour le diagnostic le sont également dans le contexte de test fonctionnel du système. Il n'y avait pas d'information recueillie sur les tests redondants pour ce cas d'étude avant ces expérimentations.
- Les ensembles de ressources indiscernables : l'application de la méthodologie proposée a permis de détecter 13 ensembles différents au lieu des 26 identifiés avant les expérimentations et de réduire le nombre de ressources contenues dans un ensemble en moyenne de 4 ou 5 à 2 ou 3. Cette information, généralement proposée après le diagnostic, est disponible au cours de la conception des tests. Ceci constitue un indicateur important au cours de la conception des tests. En effet, les concepteurs de tests peuvent

s'en servir pour la définition de tests supplémentaires ou prévoir des moyens ou outils de tests appropriés afin de lever les situations d'indiscernabilité.

- Les tests pertinents : ces tests constituent les différents ensembles de diagnostic des objectifs de test définis pour le système « ORA ». Les expérimentations ont permis de déterminer les tests pertinents dont l'exécution est nécessaire et suffisant pour activer au moins une fois toutes les ressources et assurer l'incrimination d'une ressource ou la suspicion d'un ensemble de ressources indiscernables dès la phase de conception des tests. Parmi les 169 tests initialement définis, 80 ont été choisis pour constituer les différents ensembles de diagnostic. Soit 48% des tests définis pour la vérification du système. Cette information peut être utilisée par les concepteurs de tests afin d'analyser les tests restants pour décider de leur utilité.

4.5.5 Retours des concepteurs d'essai et des testeurs sur la FAL

Les informations fournies sur le système ORA (Orientation des Roues Avant) à l'issue de ces expérimentations ont été analysées d'une part, par les concepteurs d'essai et d'autre part, par les testeurs qui assurent les activités de diagnostic suite à la détection de fautes au cours de l'exécution des tests sur la FAL (Final Assembly Line).

Concepteurs d'essai

Les concepteurs d'essai ont été particulièrement séduits par la nature des informations proposées sur les tests décrits dans le GTI (Ground Test Instructions) à partir de l'application de la méthodologie définie. Ces informations concernent notamment les ressources indiscernables, les tests redondants pour le diagnostic ainsi que les ensembles de diagnostic.

- Ressources indiscernables : cette information met en évidence les parties du système pour lesquelles suffisamment de tests n'ont pas été définis afin d'identifier le comportement des différentes ressources utilisées les unes par rapport aux autres. Les concepteurs d'essai ont pu vérifier que de tests supplémentaires pourront être définis dans ces cas de figure afin d'améliorer l'étape de diagnostic.
- Tests redondants pour le diagnostic : les concepteurs d'essai se sont montrés très intéressés par cette information. En effet, la détection et l'élimination des tests redondants sont parmi les principaux défis de la conception d'essai sur la FAL. Par ailleurs, l'analyse des tests redondants identifiés pour le diagnostic a révélé que 15 sur les 30 sont également redondants fonctionnellement. Cette analyse s'est basée sur les informations préalablement collectées à partir des jugements d'ingénieurs d'essai. Les concepteurs d'essai ont conclu que ces tests redondants pour le diagnostic peuvent être

utiles pour le processus d'identification et d'élimination de tests redondants au cours de la définition des essais.

- Ensembles de diagnostic : concernant cette information, elle permet de mettre en évidence, pour chaque objectif de test défini dans le GTI, le sous-ensemble de tests nécessaire et suffisant pour activer toutes les ressources et proposer un diagnostic optimal. Les concepteurs de d'essai trouvent que ce sous-ensemble peut servir d'une bonne base dans la recherche des tests pertinents au cours de la vérification des systèmes sur la FAL.

Testeurs

Parmi les informations proposées par la méthodologie définie, les testeurs sur la FAL ont montré leur intérêt pour les ensembles de diagnostic et la méthode d'amélioration du diagnostic afin de réduire l'effort du diagnostic en automatisant certaines pratiques essentiellement basées sur l'expérience humaine.

- Ensembles de diagnostic : cette information permet, au cours de la vérification de chaque objectif de test, de se focaliser sur les verdicts des tests de l'ensemble de diagnostic afin de construire automatiquement l'arbre de diagnostic. Ceci permet d'incriminer la ressource défectueuse ou de suspecter un ensemble de ressources indiscernables. Par conséquent, les testeurs ont apprécié de trouver, dans cette information, un moyen permettant de proposer un diagnostic automatique (utile dans le contexte de la FAL) à partir des verdicts d'un sous-ensemble de tests.
- Méthode d'amélioration du diagnostic : l'intérêt de cette méthode, pour les testeurs, est qu'elle permet de guider le processus de report du diagnostic d'un objectif de test. En effet, elle fournit automatiquement les informations sur les autres objectifs de test dont la vérification peut aider à améliorer le diagnostic lorsque plusieurs ressources sont suspectées. Ceci représente un apport majeur aux activités de vérification des systèmes sur la FAL.

4.6 Conclusion

Ce chapitre propose une méthodologie d'aide à la vérification des systèmes sur la chaîne d'assemblage final d'un avion AIRBUS. Cette méthodologie prend en compte le processus de vérification sur la FAL et a pour objectif de fournir des informations permettant de pallier les difficultés identifiées au cours des phases de conception des tests et de diagnostic. Cette méthodologie est basée d'une part sur l'application d'une stratégie de test (Multiple-Clue)

adaptée à la vérification des systèmes sur la FAL. D'autre part, elle propose une méthode d'amélioration du diagnostic du système.

Un processus d'application de la stratégie Multiple-Clue a été défini pour l'analyse des systèmes AIRBUS. Ce processus est composé d'une étape de modélisation du système et d'une étape de détermination d'informations pertinentes, en utilisant une approche par recoupements, destinées aux concepteurs de test et testeurs afin de réduire l'effort et prévoir les moyens de vérification sur la FAL. Ces informations sont fournies pour chaque objectif de test défini dans le GTI conformément au processus de vérification des systèmes sur la FAL.

Les différentes étapes de la méthode d'amélioration du degré du diagnostic, constituant la deuxième partie de la méthodologie décrite dans ce chapitre, ont été présentées. Cette méthode, dont le principe est également basé sur les approches par recoupement, permet de guider les activités de vérification (des objectifs de test) afin de traiter une situation d'indiscernabilité à l'issue du diagnostic d'un objectif de test.

Enfin, un prototype (ADIS) a été développé pour la mise en œuvre de la méthodologie d'aide à la vérification des systèmes. A l'issue de l'application de Multiple-Clue, ce prototype fournit automatiquement des informations pertinentes permettant d'aider d'une part, à la conception des tests et d'autre part, au diagnostic à partir de la construction de l'arbre de test. ADIS fournit également des indications pour la poursuite des activités de vérification afin d'améliorer le diagnostic.

Conclusions et perspectives

Un des principaux défis du développement des systèmes temps-réel critiques est la réduction du coût et temps des activités de validation et de vérification (V&V). Les travaux réalisés dans le cadre de cette thèse s'inscrivent dans cette optique puisqu'ils permettent, en définissant des méthodes adaptées, de partiellement automatiser certaines activités de V&V à savoir : les activités de validation des spécifications formelles et les activités de vérification des systèmes avioniques sur la chaîne d'assemblage finale. Les méthodologies proposées se basent sur les concepts d'analyse de testabilité de spécifications flot de données et des stratégies de test.

Les écoulements sélectionnés, par la stratégie Start-Small, à l'issue de l'analyse de testabilité des spécifications flot de données ont servi d'élément « clé » dans la mise en œuvre de la méthodologie d'aide à la validation des spécifications formelles des systèmes. En effet, ces écoulements ont permis d'aider à l'automatisation de certaines étapes d'évaluation de couverture. Les expérimentations ont montré que la méthodologie développée permet de fournir des informations utiles pour la validation des spécifications formelles. Notamment, l'identification des écoulements orphelins et des écoulements manquants liés respectivement à la transcription d'exigences fonctionnelles dérivées, non définies dans le cahier de fonctions (CDF – DFS) ou à une sur-spécification du système et à l'existence d'exigences fonctionnelles dans le CDF sans contre partie dans la spécification formelle. Cette méthodologie d'aide à la validation fournit également des informations sur l'évaluation de la couverture des tests définis pour la simulation du système. Ces informations concernent les possibles tests redondants et les tests manquants correspondant respectivement aux tests vérifiant les mêmes fonctionnalités (activant le même écoulement) et les écoulements pour lesquels aucun test n'a été défini pour la vérification des fonctionnalités qui leurs sont associées.

Les expérimentations ont aussi montré la difficulté d'interpréter les mesures de contrôlabilité et d'observabilité associées aux composants d'un système dans le contexte de validation des spécifications des systèmes AIRBUS basé sur le test fonctionnel. En effet, ces

mesures sont calculées à partir de la simulation de la perte d'information dans le système en considérant les quantités d'information théorique disponibles à l'entrée et à la sortie de chacun des composants. Ce calcul ne tient pas compte des contraintes associées à la valeur des flots de données traités par le système. Seule une estimation de la quantité d'information nécessaire du codage des informations à l'entrée ou à la sortie du composant est effectuée. De ce fait, ces mesures, en termes de difficulté à tester les composants du système, n'ont pas pu être interprétées dans le contexte AIRBUS.

La méthodologie d'aide à la vérification des systèmes sur la FAL, basée sur une modélisation abstraite (ressources et tests), a permis d'intégrer des méthodes d'analyse produisant des informations utiles pour la conception des tests et le diagnostic à l'issue de la détection d'une faute. Dès la conception des tests, les concepts développés dans le prototype ADIS (Aide au Diagnostic des Systèmes) fournissent des informations, à l'aide de la stratégie Multiple-Clue, sur les tests redondants, les tests pertinents et les ressources indiscernables pour le diagnostic. Ces informations sur la vérification sont déterminées de façon automatique à partir du modèle du système. Elles permettent aux concepteurs de réduire le nombre de tests définis lorsque ceux-ci se révèlent fonctionnellement redondants ou non pertinents. Les indications sur la définition de tests supplémentaires afin de traiter certaines situations d'indiscernabilité sont également proposées par cette méthodologie. Au niveau de la phase de diagnostic, les méthodes définies permettent d'automatiser le diagnostic à partir du verdict des tests pertinents à l'aide de l'arbre de test et de donner des indications sur la suite des activités de vérification afin d'améliorer le diagnostic dans une situation d'indiscernabilité. Les expérimentations menées dans le cadre de cette thèse ont abouti à des résultats encourageants et cohérents avec les informations recueillies disponibles sur le système.

Ces deux méthodologies ont été développées en considérant respectivement le cycle de validation des spécifications et le processus de vérification sur la FAL (Finale Assembly Line). Par ailleurs, on peut noter que la méthodologie d'aide à la validation des spécifications des systèmes est applicable à tout système spécifié à l'aide d'un formalisme flot de données. La méthodologie d'aide au diagnostic sur FAL est adaptable à tout système modélisable à l'aide d'une matrice de diagnostic.

Perspectives

Cette thèse a permis de montrer que des méthodologies basées sur une analyse prédictive peuvent fournir des informations pertinentes pour la définition des tests et le diagnostic tant pour la validation des spécifications que pour la vérification des systèmes. Cependant, il reste des pistes à explorer pour améliorer les méthodes que nous avons proposées.

Une piste de recherche à suivre pour une plus grande adaptabilité de la méthodologie d'aide à la validation à différents contextes de développement est la prise en compte des principaux critères de couverture des tests pour la sélection des écoulements. Ceci passe par la mise en œuvre de nouvelles techniques de modélisation des composants du système pour l'analyse de testabilité. En effet, le critère proposé actuellement est celui de la couverture des conditions d'activations et branches. Ceci n'est pas suffisant pour la validation de la spécification de certains systèmes critiques qui exigent d'autres critères comme MC/DC (Modified Condition/Decision Coverage).

Le développement d'une méthode de génération des données de test à l'aide des écoulements peut faire l'objet d'une autre piste de recherche. En effet, il existe des outils de génération des données de tests fonctionnelles pour la validation des spécifications flot de données tels que GATeL développé par le CEA (Commissariat à l'Energie Atomique). L'idée est de formuler les objectifs de test pour GATeL à partir des informations proposées par les écoulements sélectionnés à l'issue de l'analyse de testabilité afin de guider la génération des données de test.

Un point qui mérite d'être amélioré est la méthode de calcul des mesures de testabilité afin qu'elles soient pertinentes dans un contexte de test fonctionnel. Ceci nécessite la prise en compte des valeurs ainsi que les contraintes liées aux flots de données contenus dans la spécification du système. Une des solutions serait l'utilisation d'un moteur de résolution de contraintes permettant d'aider à la détermination de ces domaines. L'utilisation d'un tel moyen pour l'analyse de testabilité est-elle judicieuse ? L'effort de l'analyse de testabilité (calcul des mesures) risquerait de devenir comparable à celui de la génération des données de test.

La dernière piste que nous proposons dans cette thèse, en ce qui concerne l'aide à la validation, est l'extension de la méthodologie aux automates représentant l'aspect flot de contrôle de la spécification des systèmes. En effet, une étude d'applicabilité des concepts développés pour l'analyse des spécifications flot de données aux automates serait très

intéressante en termes de critère de couverture des différents modes et transitions et de génération des données de test. Ceci passe par la définition d'un nouveau modèle pour l'analyse des automates et la définition de critère de couverture et de mesures appropriées aux automates.

Concernant l'aide à la vérification des systèmes sur la FAL, les principales orientations sont le renforcement ou la consolidation de l'applicabilité de la méthodologie dans un contexte opérationnel. La phase de consolidation consiste à appliquer la méthodologie sur d'autres types de systèmes afin de confirmer les résultats obtenus dans le cadre de ces travaux. Ceci permettrait d'enrichir la méthode de modélisation des différents types de ressources utilisées dans les systèmes installés à bord d'un avion. Les expérimentations doivent aussi se poursuivre sur l'application de la méthodologie définie sur des systèmes de plus grande taille en termes de ressources et de tests afin de valider ses concepts sur le passage à l'échelle. Enfin, les futurs travaux devront porter sur la mise en place de méthodes de modélisation automatique des ressources et des tests au cours de la production du GTI (Ground Test Instructions). Ils permettront de conclure sur une possible intégration de la méthodologie dans les moyens ou outils de test des systèmes sur la FAL (Finale Assembly Line).

Les algorithmes implantés dans les prototypes développés dans le cadre de cette thèse, afin d'être utilisés dans un contexte opérationnel, devront être industrialisés pour être intégrés à l'environnement de développement des concepteurs systèmes.

Abréviations

ABD : AIRBUS Directive

BSCU : Braking and Steering Control Unit

CDV : Commandes de vol

DAMAS : Data Manager for specifications

DFS : Detailed Functional Specification

EIRD : System Installation Requirements Document

FAL : Final Assembly Line

FCPC : Flight Control Primary Computer

FDD : Functional Description Document

FHA : Functional Hazard Assessment

FRD : Functional Requirements Document

GTI : Ground Test Instructions

GTR : Ground Test Requirements

LTR : Lab Test Request

OCASIME : Outil de Conception Assistée par Simulation Multi-Equipements

OSMA : Outil de Simulation du Mouvement Avion

PSSA : Preliminary System Safety Assessment

PTS : Purchaser Technical Specification

SES : Supplier Equipment Specification

SIMPA : SIMulation du Pilote Automatique

SRD : System Requirements Document

SSA : System Safety Assessment

TLRD : Top Level Requirements Document

Glossaire

Programme : un programme « ou code » est définie comme un ensemble d'instructions et de données représentant un algorithme et susceptible d'être exécuté par un ordinateur. *(Larousse)*

Logiciel : un logiciel est un ensemble de programmes, procédés et règles, et éventuellement de la documentation, relatifs au fonctionnement d'un ensemble de traitement de données. *(Larousse)*

Modèle : un modèle « d'un système » est une représentation abstraite des relations entre les paramètres caractéristiques du système. *(Larousse)*

Système : un système est un ensemble de moyens d'acquisition, de traitement, de stockage et de restitution de données, et de moyens de télécommunication mis en œuvre pour une application ou un ensemble d'applications spécifiées. *(Larousse)*

Test unitaire : cette phase consiste à tester indépendamment les unes des autres toutes les unités élémentaires ou modules qui composent le système. Le test unitaire assure que les modules sont conformes aux exigences.

Test d'intégration : cette phase intervient lors de l'assemblage des modules du système. Elle consiste à tester la cohérence des interfaces des modules entre elles et à détecter des erreurs provenant de l'interaction entre unités et n'ayant pas été détectées au niveau du test unitaire. Cette phase permet de vérifier que les modules assemblés réalisent les fonctionnalités prévues.

Test système : cette phase consiste à tester le système dans sa totalité. Elle permet de valider les fonctionnalités du système avant sa mise en exploitation. L'échec d'un test système est lourd de conséquences et doit par conséquent être exceptionnel.

Bibliographie

[ABD06] ABD200. "*Requirements for Systems and Cabin Items Designers*". AIRBUS, 2006.

[Agr95] Horgan (J. R.), London (S.) and Wong (W. E.). "Fault localization using execution slices and dataflow tests". *In: Proceedings of the 6th International Symposium on Software Reliability Engineering*, 1995, pp. 143–151, Toulouse, France, Oct 1995.

[Amm08] Ammann (P.) and Offutt (J.). "Introduction to Software Testing". *In: Cambridge University Press*, 2008.

[Bal94] Balci (O.). "Validation, verification, and testing techniques throughout the life cycle of a simulation study". *In: Annals of Operations Research*, vol. 53, pp. 121–173. 1994.

[Bal95] Balci (O.). "Principles and Techniques of Simulation Validation, Verification and Testing". *In: Proceedings of the 27th conference on Winter simulation*, pp. 147–154, Arlington, Virginia. 1995.

[Bau04] Baufreton (P.), Granier (H.), Cruz (J.-S.) and Dupont (F.). "Visual notations based on synchronous languages for dynamic validation of gals systems". *In: Proceedings of the International Conference on Communications and Control Technologies*. Austin, Texas, USA, Aug 2004.

[Bei90] Beizer (B.). "Software testing techniques (2nd ed.)". *In: Van Nostrand Reinhold Co.*, New York, NY, USA, 1990.

[Bel02] Bel (G.), Boniol (F.), Durieu (G.), Fraboul (C.), Foisseau (J.) and Wiels (V.). "Modèles comportementaux pour l'avionique modulaire intégrée". In:

Proceeding of the 1er Congrès plurisectoriel "Logiciel Temps Réel Embarqué", Météo France. Toulouse, France, Jan 2002.

[Ben01] Benveniste (A.), Bournai (P.), Gautier (T.), Borgne (M. Le), Gernic (P. Le) and Marchand (H.). "The SIGNAL declarative synchronous language: controller synthesis & systems/Architecture design". *In: Proceedings of the 40th IEEE conference on Decision and Control*, pp. 3284–3289. Oriando, Florida, USA, Dec 2001.

[Ben91] Benveniste (A.), Berry (G.). "The Synchronous Approach to Reactive and Real-Time Systems". *In: Proceedings of the IEEE*, vol. 79, no 9, pp. 1270–1282. Sept 1991.

[Ber91] Bernot (G.), Gaudel (M.C.) and Marre (B.). "Software testing based on formal specifications: a theory and a tool". *In: Software Engineering Journal*, vol. 6, pp. 387–405, Nov 1991.

[Ber96] Bertolino (A.) and Strigini (L.). "Using testability measures for dependability assessment". *In: IEEE Transactions on Software Engineering*, vol. 22, n° 2, pp. 97–108, Feb 1994.

[Bin94] Binder (R. V.). "Design for testability in object-oriented systems". *In: Communications of the ACM*, vol. 37, n°9, pp. 87–101, Sep 1994.

[Bou91] Bournai (P.), Cheron (B.), Houssais (B.) and Gernic (P. Le). *Manuel Signal*. Rapport technique No. 575, IRISA, Feb 1991.

[Bow06] Bowen (J. B.) and Hinchey (M.G.). "Ten commandments of formal methods – ten years later". *In: Computer*, pp. 40–48, Jan 2006.

[Bow95] Bowen (J. B.) and Hinchey (M.G.). "Ten commandments of formal methods". *In: Computer*, vol. 28, n° 4, pp. 40–48, Apr 1995.

[Bro03] Broadfoot (G.H.) and Broadfoot (P.J.), "Academia and industry meet: Some experiences of formal methods in practice". *In: Proceedings of the 10th Asia-Pacific Software Engineering Conference (APSEC'03)*, pp. 49. 2003.

[Chu87] Chusho (T.). "Test Data Selection and Quality Estimation Based on the Concept of Essential Branches for Path Testing". *In: IEEE Transactions on Software Engineering*, vol. 13, pp. 509–517, May 1987.

[Cia94] Ciapessoni (E.), Corsetti (E.), Migliorati (M.) and Ratto (E.). "Specifying industrial real-time systems in a logical famework". *In: ICPL94 – Post Conference Workshop on Logic Programming in Software Engineering*. 1994.

[Clu86] Clure (M. C.) and Martin (J.). "Software Maintenance: the Problem and its Solution". *In: Prentice–Hall*, 1986.

[Cou78] Cousot (P) and Halbwachs (N.). "Automatic discovery of linear restraints among variables of a program". *In: Conference Record of the Fifth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pp. 84–97. Tucson, Arizona, 1978.

[Cou77] Cousot (P.) and Cousot (R.). "Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints". *In: Conference Record of the Sixth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pp. 238–252, Los Angeles, California, Jan 1977.

[Cor02] Cormen (T.H), Leiserson (C. E.), Rivest (R. L.) and Stein (C.). "*Introduction à l'algorithmique*". Dunod, Paris, 2002.

[Dam85] Dammak (A.). "Etudes de Mesures de testabilité des Systèmes Logiques". *Thèse, Université de Paris–Sud, Centre d'Orsay*, July 1985.

[Den91] Denney (R.). " Test-Case Generation from Prolog-Based Specifications". *In: IEEE Software*, vol. 6, pp. 49–57, Mar 1991.

[Do06] Do (H.-V.). "Conception testable et test de logiciels flots de données". *Thèse, Institut National Polytechnique de Grenoble*, Oct 2006.

[DOT06] Langage DOT. *http ://www.graphviz.org/Documentation.php*

[Dou06] Doumbia (F.), Do (H-V.) and Delaunay (M.). "*Manuel de référence du langage MAC*". Jun 2006.

[Dou09a] Doumbia (F.), Laurent (O.), Robach (C.) and Delaunay (M.). (Best Paper Award) "Using the Testability analysis methodology for the Validation of AIRBUS Systems". *In: The Proceedings of the First International Conference on Advances in System Testing and Validation Lifecycle*, pp. 86–91, Porto, Portugal, Sept 2009.

[Dou09b] Doumbia (F.), Laurent (O.), Atger (D.) and Robach (C.). “Using the Multiple-Clue approach for system testing on AIRBUS FAL (Final Assembly Line)”. *In: The Proceedings of the IEEE International Test Conference 2009*, Austin, USA, Nov 2009.

[Edi09] SCADE Suite Editor, 2009. *http://www.esterel-technologies.com/products/scade-suite/editor*

[Est05a] Esterel. *Efficient Development of Safe Avionics Software with DO-178B Objectives Using SCADE Suite*. Esterel Technology, Jul 2005.

[Est05b] Esterel Technologies SA. *SCADE Technical Manual*. 2005

[For] Formal Methods. *http://www.wikipedia.org/wiki/Formal_methods*

[Fre91] Freeman (R. S.). “Testability of Software Components”. *In: IEEE Transactions on Software Engineering*, vol. 17, n° 7, pp. 553–564, Jun 1991.

[Gal91] Gallagher (K. B.) and Lyle (J. R.). “Using program slicing in software maintenance”. *In: IEEE Transactions on Software Engineering*, vol. 17, n°8, pp.751–761, Aug 1991.

[Gir96] Girard (P.), Landrault (C.), Pravossoudovith (S.) and Rodriguez (B.) “A Diagnostic ATFG for Delay Faults Based on Genetic Algorithms”. *In: IEEE International Test Conference*, pp. 286-293, Washington, DC, USA, Oct 1996.

[Gou04] Gouraud (S.-D.). “AuGuSTe : a tool for statistical testing experimental results”. *L.R.I., UMR 8623, CNRS and Université Paris XI, 91400 Orsay*, France, Jul 2004.

[Gra89] Granger (P.). “Static analysis of arithmetical congruences”. *In: International Journal of Computer Mathematics*, vol. 30, pP. 165–190, 1989.

[Gui91] Giuning (T.), Mahlsted (U.), and Koopmeiners (H.). “3DIATEST: a Fast Diagnostic Test Pattern Generator for Combinational Circuits”. *In: Proceedings of the International Conference on Computer Aided design*, pp. 194-197, Santa Clara, California, Nov. 1991.

[Hal91a] Halbwachs (N.), Caspi (P.), Raymond (P.) and Pilaud (D.). “The synchronous data programming language Lustre”. *In: Proceedings of IEEE*, vol. 79, n° 9, pp. 1305–1320. Sep 1991.

[Hal91b] Halbwachs (N.), Caspi (P.), Pilaud (D.) and Raymond (P.). “Programmation et Vérification des Systèmes Réactifs : le langage Lustre”. *In: Techniques et Sciences de l’Informatique*, vol. 10, n° 2, pp. 139–158. 1991.

[Hal92] Halbwachs (N.), Lagnier (F.) and Ratel (C.). “Programming and verifying real-time systems by means of the synchronous data-flow programming language Lustre”. *In: IEEE Transactions on Software Engineering, Special Issue on the Specification and Analysis of Real-Time Systems*, pp. 785–793, Sep 1992.

[Hal02] Halbwachs (N.) and Raymond (P.). “*A Tutorial Of Lustre*”, Jan 2002.

[Har91] Harrold (M.J.). “Selecting and Using Data for Integration Testing”. *In: IEEE Software*, vol. 8, pp. 58–65, Mar 1991.

[Har87] Harel (D.). “Statecharts: A visual formalism for complex systems”. *Science of Computer Programming*, vol. 8, n° 3, pp .231–274. Jun 1987.

[Het88] Hetzel (B.). “The complete guide to software testing (2nd ed.)”. *In: QED Information Sciences, Inc.*, Wellesley, MA, USA, 1988.

[Hor94] Horgan (J. R.). “Achieving Software Quality With Testing Coverage Measures”. *In: IEEE Transaction on Computer*, pp. 60–67, Sep 1994.

[How81] Howden (W.E.). “Errors, design properties and functional program tests”. In: Computer Program Testing (Chandrasekaran (B.), Radicchi (S.), eds.), North-Holland, 1981.

[IEE90] IEEE. “IEEE Standard Glossary of Software Engineering Terminology”. *In: IEEE Computer Soc.*, 1990.

[ISO06] ISO. “ISO/IEC 9126-1:2001 Software engineering -- Product quality -- Part 1: Quality model”. *International Organization for Standardization*. 2006.

[Jun99] Jungmayr (S.). “Reviewing software artifacts for testability”. *In: Proceedings of the EuroSTAR ’99*. Barcelona, Nov 1999.

[Jun02] Jungmayr (S.). “Testability Measurement and Software Dependencies”. *In: Proceedings of the 12th International Workshop on Software Measurement*. Magdeburg, Germany, Oct 2002.

[Kam93] Kamkar (M.). “Interprocedural dynamic slicing with applications to debugging and testing”. *Thèse, Université de Linköping*, Suède, 1993.

[Kam95] Kamkar (M.). “An overview and comparative classification of program slicing techniques”. *In: Journal of Systems and Software*, vol. 31, n° 3, pp. 197–214. 1995.

[Kar76] Karr (M.). “Affine relationships among variables of a program”. *In: Acta Informatica*, vol. 6, pp. 133–151, 1976.

[Kha98] Khalil (M.), Le Traon (Y.) and Robach (C.). “Automated strategies for software diagnosis”. *In: Proceedings of the IEEE International Symposium on Software Reliability Engineering (ISSRE'98)*, Paderborn, Germany, Nov 1998.

[Kha01a] Khalil (M.), Robach (C.) and Novak (F.). “Diagnosis Strategies for Hardware or Software Systems”. *In: Journal of Electronic Testing, Theory and Applications*, vol. 18, pp. 241–251, Apr 2001.

[Kha01b] Khalil (M.), Robach (C.), Novak (F.) and Biasizzo (A.). “System level diagnostic strategies and tools”. *In: Proceedings of the 4th IEEE Design and Diagnostics of Electronic Circuits and Systems WORKSHOP*, Gyor, Hungary, Apr 2001.

[Kin00] King (S.), Hammond (J.), Chapman (R.) and Pryor (A.). “Is Proof More Cost-Effective Than Testing?”. *In: IEEE Transactions on Software Engineering*, vol. 26, n° 8, pp. 675–686, Aug 2000.

[Kit98] Kitchenham (B.) and Linkman (S.). “Validation, Verification, and Testing: Diversity Rules”. *In: IEEE Software*, vol. 15, n° 4, pp. 46-49. 1998.

[Kle93] Kleer (J. D.). “A perspective on assumption-based truth maintenance”. *In : Artificial Intelligence*, vol. 59, n° 1-2, pp. 63-67, 1993.

[Kor97] Korel (B.). “Computation of Dynamic Program Slices for Unstructured Programs”. *In: IEEE Transactions on Software Engineering*, vol. 23, n° 1, pp. 17–34, Jan. 1997.

[Las83] Laski (J.W.). “A Data Flow Oriented Program Testing Strategy”. In: IEEE Transactions on Software Engineering, vol. 9, pp. 347–354, May 1983.

[LeT95a] Le Traon (T.) and Robach (C.). “Towards a Unified Approach to the Testability of Co-designed Systems”. *In: the Proceedings of the IEEE International*

Symposium on Software Reliability Engineering (ISSRE'95), pp. 278–285, Toulouse, France, 1995.

[LeT95b] Le Traon (T.) and Robach (C.). "From Hardware to Software Testability". *In: the Proceedings of the IEEE International Test Conference (ITC'95)*, pp. 710–719, Washington D.C, 1995.

[LeT97] Le Traon (T.). "Analyse conjointe logiciel/matériel de la testabilité de systèmes flot de données". *Thèse, Institut National Polytechnique de Grenoble*, Nov 1997.

[Lin97] Lin (J-C.), Lin (S-W.) and Ian-Ho. "An estimated method for software testability measurement". *In: Proceedings the 8th IEEE International Workshop on Software Technology and Engineering Practice*, pp. 116–123, London, Jul 1997.

[Mar00] Marre (B.) and Arnould (A.). "Test sequences generation from lustre descriptions : Gatel". In : ASE'00, 15th IEEE International Conference on Automated Software Engineering, pp. 229–237, Sep 2000.

[Mar04] Marre (B.) and Blanc (B.). "Test selection strategies for Lustre descriptions in Gatel". *In: Model-Based Testing Workshop, ETAPS'04, to appear in ENTCS*, Barcelona, Apr 2004.

[Mat05] The MathWorks. *Simulink Reference*. 2005.

[Mat09] The MathWorks. *Stateflow and Stateflow Coder*. Sep 2009.

[McC76] McCabe (T. J.). "A complexity measure". *In: IEEE Transactions on Software Engineering*, vol. SE-2, n° 4, pp. 308 – 320, Dec 1976.

[Moh79] Mohanty (S. N.). "Models and Measurements for Quality Assessment of Software". *In: ACM Computing Surveys (CSUR)*, vol. 11 , n° 3, Sep 1979.

[Moi02] Moir (I.), Seabridge (A.). "Civil Avionics Systems". *In: Professionnal Engineering Publishing*, 2002.

[Mun93] Munson (J. C.) and Khoshgoftaar (T. M.). "The Measurement of Data Structure Complexity". *In: Journal of Systems and Software*, vol. 20, n° 3, pp. 217-226, Mar1993.

[Mye77] Myers (G. J.). "An extension to the Cyclomatic Measure of Program Complexity". *SIGPLAN Notices*, vol. 12, n° 10, pp. 61–64, Oct 1977.

[Mye79] Myers (G. J.). “The Art of Software Testing” – *John Wiley & Sons Inc*, 1979

[Mor07] Morschhäuser (I.) and Lindvall (M.). “Model-Based Validation & Verification Integrated with SW Architecture Analysis: A Feasibility Study”. *In: IEEE Aerospace Conference*. Montana, Mar 2007.

[Nej88] Nejmeh (B. A.). “NPATH: A measure of execution path complexity and its applications”. *In: Communications of the ACM*, vol. 31, n°2, pp. 188–200, Feb 1988.

[Nou96] Nikoukhah (R.) and Steer (S.). “SCICOS – a dynamic system builder and simulator”. *In: Proceedings of the IEEE International Symposium on Computer-Aided Control System Design*, pp. 430–435. Dearborn, MI. Sep 1996.

[Ost86] Ostrand (T.J.) and Weyuker (E.J.). “Design for a Tool to Manage Specification-Based Testing”. *In: Workshop on Software Testing*, pp. 41–50, Banff, Canada, Jul 1986.

[Ovi80] Oviedo (E. I.). “Control flow, data flow and program complexity”. *In: Proceedings of the 4th International Computer Software and Applications Conference*, pp. 146-152, Chicago, Illinois, Oct 1980.

[Par93] Parissis (I.). “Test de logiciels synchrones spécifiés en lustre”. *Thèse, Université Joseph Fourier*, Grenoble, France, Sep 1996.

[Par03] Parkinson (P.) “Safety-Critical Software Development for Integrated Modular Avionics”. Win River, 2003.

[Pro] Prover Technology. “Prover Product”. *http://www.prover.com/products/*

[Rap85] Rapps (S.) and Weyuker (E. J.). “Selecting software test data using data flow information”. *In: IEEE Transactions on Software Engineering*, vol. 11, n° 4, pp. 367–375, 1985.

[Ray98] Raymond (P.), Weber (D.), Nicollin (X.) and Halbwachs (N.). “Automatic testing of reactive systems”. *In: 19th IEEE Real-Time Systems Symposium*, Madrid, Spain, Dec 1998.

[Rob79] Robach (C.). “Test et Testabilité de Systèmes Informatiques”. *Thèse Docteur–es–Sciences, Université de Grenoble*, Juin 1979.

[Rob89] Robach (C.) and Wodey (P.). “Linking design and test tools: an implementation”. *In: IEEE Transactions on Industrial Electronics*, vol. 36, n°2, pp. 286–295, May 1989.

[RTC92] RTCA/DO-178B, "*Software Considerations in Airborne Systems and Equipment Certification*", Dec 1992.

[Rum04] Rumbaugh (J.), Jacobson (I.) and Booch (G.). *“*UML 2.0 Guide de reference”. *CampusPress*. 2004.

[SAE96] SAE. “*Arp4754: certification considerations for highly-integrated or complex systems*”. Nov 1996.

[Sar98] Sargent (R.G.). “Verification and validation of simulation models”. *In: Proceedings of the 30th conference on Winter simulation*, pp. 121–130, Washington, D.C., Dec 1998.

[Sch79] Schlesinger (S.). “Terminology for model credibility”. *In: Simulation*, vol. 32, n°3, pp. 103-104. 1979.

[Sha75] Shannon (R. E.). “Systems Simulation: The Art and Science”. *In: Prentice-Hall Inc.*, Englewood Cliffs, New Jersey. 1975.

[She88] Shepperd (M.). “A critique of cyclomatic complexity as a software metric”. *In: Software Engineering Journal*, vol. 3, n° 2, pp. 30–36, Mar 1988.

[She94] Sheppard (J.) and Simpson (W.). “System Test and Diagnosis.” *In: Kluwer Academic Publishers*. 1994.

[Sim09] SCADE Suite Simulator, 2009. *http://www.esterel-technologies.com/products/scade-suite/simulation*

[Tai84] Tai (M.). “A program complexity metric based on data flow information in control graphs”. *In: Proceedings of the 7th international conference on Software engineering*, pp. 239 – 248, Orlando, Florida, Oct 1984.

[Tan98] Tan (W. A.) and Behforooz (A.). “Data Complexity Metrics for Large Software”. *In: Proceedings of the SE'98*, pp. 150–153, Las Vegas, Oct 1998.

[Tel] Telelogic. “A requirements management tool for systems and advanced IT applications : DOORS (Dynamic Object Orientated Requirements System)”. *http://www.telelogic.com/products/doors/doors/index.cfm*

[TNI] TNI. “Software products – Sildex”. *http://www.tni.fr/tni/offre/sildex/index.eng.html*

[Ver09] SCADE Design Verifier, 2009. *http://www.esterel-technologies.com/products/scade-suite/design-verifier*

[Voa91] Voas (J. M.), Morrel (L.) and Miller (K. W.). “Predicting Where Faults Can Hide from Testing”. *In: IEEE Software*, vol. 8, n°2, pp. 41–48, Mar 1991.

[Voa92] Voas (J. M.). “PIE: a dynamic failure-based technique”. *In: IEEE Transactions on Software Engineering*, vol. 10, n° 8, pp. 717 – 727, Aug 1992.

[Voa93a] Voas (J. M.) and Miller (K. W.). “Semantic metrics for software testability”. *In: Journal of Systems and Software*, vol. 20, n° 3, pp. 207 – 216, Mar 1993.

[Voa93b] Voas (J. M.), Miller (K. W.) and Payne (J.). “Software Testability and Its Application to Avionic Software”. *In: Proceedings of Computers in Aerospace*, pp. 507–515, San Diego, CA, Oct, 1993.

[Voa95a] Voas (J. M.) and Miller (K. W.). “Software Testability: The New Verification”. *In: IEEE Software*, vol. 12, n° 3, pp. 17–28, May 1995.

[Voa95b] Voas (J.M.). “Software testability measurement for assertion placement and fault localization.” *In: 2nd International Workshop on Automated and Algorithmic Debugging (AADEBUG'95)*, Saint-Malo, France, May 1995.

[Voa96] Voas (J. M.) and Miller (K. W.). “Substituting Voas's Testability Measure for Musa's Fault Exposure Ratio”. *In: Proceedings of the Int'l. Communications Conference*, Dallas, Texas, Jun, 1996.

[Wat76] Watson (C.E.). “The problems of problem solving”. *In: Business Horizons*, vol. 19, n° 4, pp. 88-94. Aug 1976.

[Wey80] Weyuker (E.J.) and Ostrand (T.J.). “Theories of program testing and the application of revealing sub-domains”. *In: IEEE Transactions on Software Engineering*, vol. 6, pp. 236–246, May 1980.

[Woo80] Woodward (M.R.), Hedley (D.) and Hennell (M.A.). “Experience with Path Analysis and Testing of Programs”. *In: IEEE Transactions on Software Engineering*, vol. 6, pp. 278–286, May 1980.

[Xia86] Xiang Z. and Srihari S. N. “A strategy for diagnosis based on empirical and model knowledge” *In: Days of experts systems*, pp. 835-848, Avignon (France), 1986.

Annexe A

Modélisation de certains opérateurs spécifiques d'une bibliothèque « métier »

A.1 Modélisation des opérateurs de type « bascule »

Il existe deux types d'opérateur de bascule : *BASCR* qui est une bascule avec l'entrée RESET (ER) prioritaire et *BASCS* est une bascule avec l'entrée SET (ES) prioritaire. L'opérateur *BASCR* est utilisé pour décrire la modélisation des opérateurs de type bascule (figure ci-dessous). Les entrées liées à l'initialisation de cet opérateur sont :

- *B _ Init* : l'entrée d'initialisation de type booléen ;
- *Init* : la valeur affectée à la sortie *S*1 à l'initialisation.

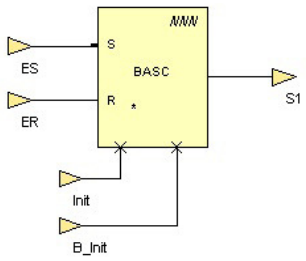


Figure A.1 : Représentation SCADE de BASCR

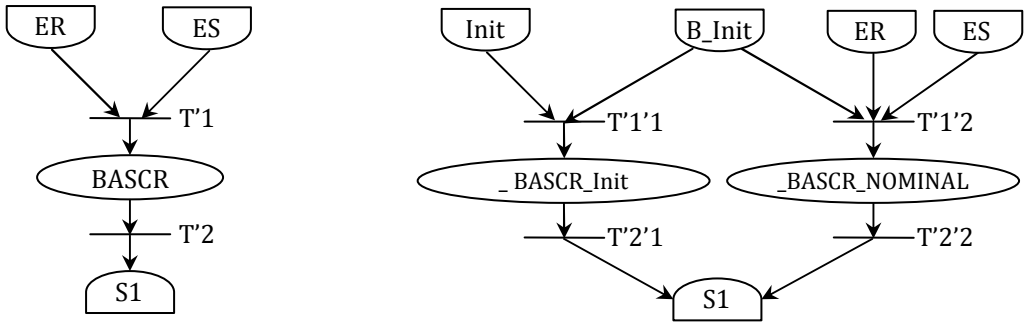
Le comportement de l'opérateur *BASCR* peut être décrit comme suit :

$$Si\ B\_Init(k) = True\ Alors\ S1(k) = Init(k)$$
$$Sinon\ "Calcul\ normal\ de\ l'état\ de\ la\ sortie\ S1"$$

$$Calcul\ normal\ de\ l'état\ de\ la\ sortie\ S1$$
$$Si\ E_R(k) = True\ Alors\ S1(k) = False$$
$$Sinon\ Si\ E_S(k) = True\ Alors\ S1(k) = True$$
$$Sinon\ S1(k) = S1(k-1)$$

A.1.1 Modélisation du MTI élémentaire

Deux différentes modélisations possibles peuvent être décrites pour la représentation du MTI de l'opérateur $BASCR$: sans et avec la prise en compte de l'initialisation de l'opération au cours du fonctionnement du système (**Figure A.1**). La première (a) est constituée d'un module fonctionnel produisant la sortie $S1$ à partir des deux entrées ER et ES. La deuxième (b) comporte deux modules fonctionnels. Le premier module $\_BASCR\_Init$ permet de calculer l'état de la sortie $S1$ à l'occurrence d'une initialisation ; le module $\_BASCR\_NOMINAL$ produit l'état de la sortie $S1$ lorsqu'il n'y a pas d'initialisation.



(a) Sans initialisation **(b) Avec initialisation**

Figure A.2 : Représentation graphique des MTI de BASCR

Les description textuelle associée aux MTI de l'opérateur $BASCR$, avec ou sans la phase d'initialisation, pour l'analyse de testabilité sont décrites ci-dessous.

– Description sans initialisation

```
1 (*
2   Description du MTI de "BASCR" sans initialisation
3 *)
4
5 type BASCR_II_I
6 debut
7       debut_entrees
8               IN1 : 1; (* ES *)
9               IN2 : 2; (* ER *)
10      fin_entrees
```

```
11      debut_sorties
12              OUT1 : 1000; (* S1 *)
13      fin_sorties
14      debut_modele
15              T'1 / IN1, IN2 / BASCR /
16              T'2 / BASCR / OUT1 /
17      fin_modele
18 fin
```

– Description avec initialisation

```
1 (*
2   Description du MTI de "BASCR" avec initialisation
3 *)
4
5 type BASCR_IV_I
6 debut
7       debut_entrees
8               IN1 : 1; (* ES *)
9               IN2 : 2; (* ER *)
10              IN3 : 3; (* B_Init *)
11              IN4 : 4; (* Init *)
12      fin_entrees
13      debut_sorties
14              OUT1 : 1000 ; (* S1 *)
15      fin_sorties
16      debut_modele
17              T'1'1 / IN4, IN3 / _BASCR_Init /
18              T'1'2 / IN1, IN2, IN3 / _BASCR_NOMINAL /
19              T'2'1 / _ BASCR _Init / OUT1 /
20              T'2'2 / _ BASCR _NOMINAL / OUT1 /
21      fin_modele
22 fin
```

A.1.2 Evaluation du CPI de l'opérateur « BASCR »

Nous présentons les méthodes d'évaluation du CPI définies pour l'opérateur $BASCR$ selon les modes de fonctionnement avec ou sans initialisation.

Evaluation du CPI sans initialisation

Afin de mieux de cerner le comportement de l'opérateur $BASCR$, la table ci-dessous fournie les états de la sortie $S1$ en fonction des entrées E_R et E_S.

$E_R(k)$	$E_S(k)$	$S1(k)$
0	0	$S1(k-1)$
1	0	1
0	1	0
1	1	0

Soient $P(S1=1)$ et $P(S1=0)$ respectivement les probabilités d'occurrence de 1 et 0 au niveau de la sortie $S1$ à un cycle d'exécution de l'opérateur *BASCR*.

$$P(S1=1) = P(E_R = 1 \; et \; E_S = 0) \; ou \; P(E_R = 0 \; et \; E_S = 0 \; et \; S1 = 1 \; au \; cycle \; précédent)$$

Sachant que les évènements E_R, E_R et $S1$ sont indépendants d'un cycle à l'autre,

$$P(S1=1) = P(E_R=1)\times P(E_R=0) + P(E_R=0)\times P(E_R=0)\times P(S1=1)$$

$$P(S1=1) = p\times(1-p) + (1-p)\times(1-p)\times P(S1=1)$$

$$\left[1-(1-p)^2\right]\times P(S1=1) = p\times(1-p)$$

$$P(S1=1) = \frac{p\times(1-p)}{\left[1-(1-p)^2\right]}$$

$$\Rightarrow P(S1=1) = \frac{1-p}{2-p}$$

Or, $P(S1=1) + P(S1=0) = 1$. De ce fait, $P(S1=0) = 1-\left(\frac{1-p}{2-p}\right)$

$$\Rightarrow P(S1=0) = \frac{1}{2-p}$$

A partir de ces deux expressions associées aux probabilités d'occurrence des deux états possibles de $S1$, la valeur du CPI, sachant que la quantité d'information théorique au niveau de $S1$ ($Q=1$), en fonction de p est :

$$CPI(BASCR) = -\left(\frac{1-p}{2-p}\log_2\left(\frac{1-p}{2-p}\right) + \frac{1}{2-p}\log_2\left(\frac{1}{2-p}\right)\right)$$

Evaluation du CPI avec initialisation

Calculer le CPI de l'opérateur *BASCR* avec une initialisation possible revient à évaluer celui des modules fonctionnels « *_BASCR_Init* » et « *_BASCR_NOMINAL* ».

a. *Module _BASCR_Init*

Ce module fournit l'information *Null* lorsque $B\_Init = 0$ et calcul l'état de $S1$ lorsque $B\_Init = 1$. De ce fait, trois différents états peuvent être observés au niveau de la sortie de ce module $(Null, 0, 1)$. Par conséquent, la quantité d'information théorique disponible à la sortie du module est : $Q = \log_2(3)$. Les probabilités d'occurrence des différentes valeurs sont notées : $P_{s\_m\_i}(Null)$, $P_{s\_m\_i}(0)$ et $P_{s\_m\_i}(1)$.

Soient P_{1_Init} la probabilité d'occurrence de 1 et P_{0_Init} la probabilité d'occurrence de 0 au niveau de la variable $Init$.

$$P_{s\_m\_i}(Null) = P(B\_Init = 0) = 1 - P_B$$
$$P_{s\_m\_i}(0) = P(B\_Init = 1 \; et \; Init = 0) = P_B \times P_{0\_Init}$$
$$P_{s\_m\_i}(1) = P(B\_Init = 1 \; et \; Init = 1) = P_B \times P_{1\_Init}$$

De ce fait, le CPI du module $\_BASCR\_Init$ s'exprime comme suit :

$$CPI(\_BASCR\_Init) = -\frac{1}{\log_2(3)} \times \sum_{j=\{Null,\,0,\,1\}} P_{s\_m\_n}(j) \log_2\left(P_{s\_m\_n}(j)\right)$$

b. *Module* $\_BASCR\_NOMINAL$

Ce module fournit l'information $Null$ lorsque $B\_Init = 1$ et calcul l'état de $S1$ lorsque $B\_Init = 0$. De ce fait, trois différents états peuvent être observés au niveau de la sortie de ce module $(Null, 0, 1)$. Par conséquent, la quantité d'information théorique disponible à la sortie du module est : $Q = \log_2(3)$. Les probabilités d'occurrence des différentes valeurs sont notées : $P_{s\_m\_n}(Null)$, $P_{s\_m\_n}(0)$ et $P_{s\_m\_n}(1)$.

$$P_{s\_m\_n}(Null) = P(B\_Init = 1) = P_B$$
$$P_{s\_m\_n}(0) = P(B\_Init = 1 \; et \; S1 = 0) = P_B \times \left(\frac{1}{2 - p}\right)$$
$$P_{s\_m\_n}(1) = P(B\_Init = 1 \; et \; S1 = 1) = P_B \times \left(\frac{1 - p}{2 - p}\right)$$

Donc, le CPI du module $\_BASCR\_NOMINAL$ s'exprime comme suit :

$$CPI(\_BASCR\_NOMINAL) = -\frac{1}{\log_2(3)} \times \sum_{j=\{Null,\,0,\,1\}} P_{s\_m\_n}(j) \log_2\left(P_{s\_m\_n}(j)\right)$$

A.2 Modélisation des opérateurs de type « détecteur d'impulsion »

Deux différents opérateurs de type détecteur d'impulsion sont décrits dans la bibliothèque « métier » étudiée : $PULSE1$ (détecteur d'impulsion d'un front montant) et $PULSE2$ (détecteur d'impulsion d'un front descendant). L'opérateur $PULSE1$ est utilisé pour décrire la modélisation de ces types d'opérateur. $PULSE1$ génère une impulsion sur un front du signal en entrée (figure ci-dessous). Les entrées liées à l'initialisation de cet opérateur sont :

- B_Init : l'entrée d'initialisation de type booléen ;
- $Init$: la valeur affectée à la sortie $S1$ à l'initialisation.

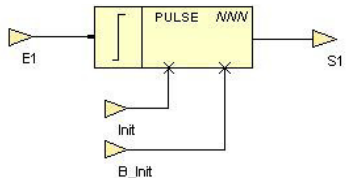


Figure A.3 : Représentation SCADE de PULSE1

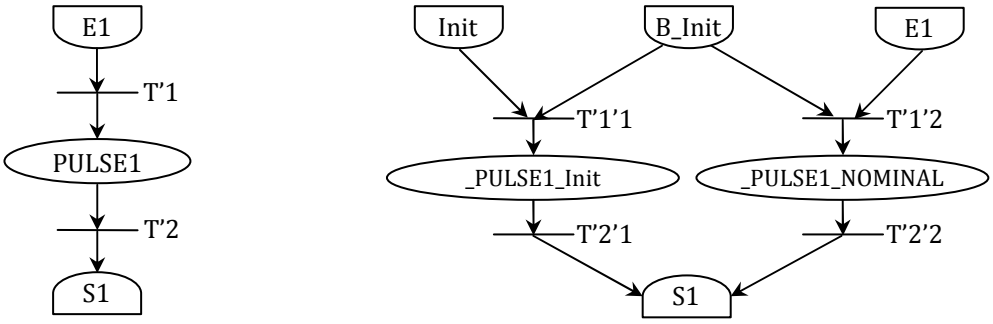
Le comportement de l'opérateur $PULSE1$ peut être décrit comme suit :

$$Si\ B\_Init(k) = True\ Alors\ S1(k) = Init(k)$$
$$Sinon\ "Calcul\ normal\ de\ l'état\ de\ la\ sortie\ S1"$$

$$Calcul\ normal\ de\ l'état\ de\ la\ sortie\ S1$$
$$Si\ E1(k-1) = False\ et\ E1(k) = True\ Alors\ S1(k) = True$$
$$Sinon\ S1(k) = False$$

A.2.1 Modélisation du MTI élémentaire

Deux différentes modélisations possibles peuvent être décrites pour la représentation du MTI de l'opérateur $PULSE1$: sans et avec la prise en compte de l'initialisation de l'opération au cours du fonctionnement du système (figure ci-dessous). La première (a) est constituée d'un module fonctionnel produisant la sortie $S1$ à partir de l'entrée $E1$. La deuxième (b) comporte deux modules fonctionnels. Le premier module $\_PULSE1\_Init$ permet de calculer l'état de la sortie $S1$ à l'occurrence d'une initialisation ; le module $\_PULSE1\_NOMINAL$ produit l'état de la sortie S1 lorsqu'il n'y a pas d'initialisation.



(a) Sans initialisation **(b) Avec initialisation**

Figure A.4 : Représentation graphique des MTI de PULSE1

Les description textuelle associée aux MTI de l'opérateur $PULSE1$, avec ou sans la phase d'initialisation, pour l'analyse de testabilité sont décrites ci-dessous.

– Description sans initialisation

```
1 (*
2   Description du MTI de " PULSE1 " sans initialisation
```

```
3 *)
4
5 type PULSE_I_I
6 debut
7       debut_entrees
8               IN1 : 1; (* E1 *)
9       fin_entrees
10      debut_sorties
11              OUT1 : 1000; (* S1 *)
12      fin_sorties
13      debut_modele
14              T'1 / IN1 / PULSE1 /
15              T'2 / PULSE1 / OUT1 /
16      fin_modele
17 fin
```

– Description avec initialisation

```
1 (*
2   Description du MTI de " PULSE1 " avec initialisation
3 *)
4
5 type PULSE1_III_I
6 debut
7       debut_entrees
8               IN1 : 1; (* E1 *)
9               IN2 : 2; (* B_Init *)
10              IN3 : 3; (* Init *)
11      fin_entrees
12      debut_sorties
13              OUT1 : 1000 ; (* S1 *)
14      fin_sorties
15      debut_modele
16              T'1'1 / IN3, IN2 / _PULSE1_Init /
17              T'1'2 / IN1, IN2 / _PULSE1_NOMINAL /
18              T'2'1 / _ PULSE1 _Init / OUT1 /
19              T'2'2 / _ PULSE1_NOMINAL / OUT1 /
20      fin_modele
21 fin
```

A.2.2 Evaluation du CPI de l'opérateur « PULSE1 »

Dans cette section, nous décrivons les méthodes d'évaluation du CPI définies pour l'opérateur *PULSE*1 selon les modes de fonctionnement avec ou sans initialisation.

Evaluation du CPI sans initialisation

La table de vérité ci-dessous donne l'état de la sortie S1 en fonction de ceux de l'entrée. Cette mémoire correspond à l'état de la sortie E1 au cycle précédent.

$E1(k-1)$	$E1(k)$	$S1(k)$
0	0	0
1	0	1
0	1	1
1	1	0

Soient $P(S1=1)$ et $P(S1=0)$ respectivement les probabilités d'occurrence de 1 et 0 au niveau de la sortie $S1$ à un cycle d'exécution de l'opérateur $PULSE1$.

$$P(S1=1) = P(E(k-1)=0 \; et \; E(k)=1)$$

Sachant que les évènements $E(k-1)$, $E(k)$ sont indépendants d'un cycle à l'autre,

$$P(S1=1) = P(E(k-1)=0) \times P(E(k)=1)$$
$$\Rightarrow P(S1=1) = (1-p) \times p$$

Or, $P(S1=1) + P(S1=0) = 1$. De ce fait, $P(S1=0) = 1-[(1-p) \times p]$

A partir de ces deux expressions associées aux probabilités d'occurrence des deux états possibles de $S1$, la valeur du CPI sachant que la quantité d'information théorique au niveau de $S1, Q=1$ en fonction de p est :

$$CPI(PULSE1) = -\left[\left(p-p^2\right) \times \log_2\left(p-p^2\right) + \left(p^2-p+1\right) \times \log_2\left(p^2-p+1\right)\right]$$

Evaluation du CPI avec initialisation

La modélisation de cet opérateur en tenant compte de l'initialisation comporte deux modules fonctionnels. Une expression de calcul de CPI doit être associée à chaque module pour l'analyse de testabilité.

a. *Module _PULSE1_Init*

Ce module fournit l'information *Null* lorsque $B\_Init = 0$ et calcul l'état de $S1$ lorsque $B\_Init = 1$. Les deux différents états peuvent être observés au niveau de la sortie de ce module sont : *Null* et *Init* (constante 0 ou 1). Par conséquent, la quantité d'information théorique disponible à la sortie du module est : $Q = \log_2(2) = 1$. Les probabilités d'occurrence des différentes valeurs sont notées : $P_{s\_m\_i}(Null)$ et $P_{s\_m\_i}(Init)$.

$$P_{s\_m\_i}(Null) = P(B\_Init = 0) = 1 - P_B$$
$$P_{s\_m\_i}(Init) = P(B\_Init = 1) = P_B$$

De ce fait, le CPI du module $\_PULSE1\_Init$ est calculé à l'aide de l'expression suivante :

$$CPI(\_PULSE1\_Init) = -\left[(1-P_B) \times \log_2(1-P_B) + (P_B) \times \log_2(P_B)\right]$$

b. *Module* $\_PULSE1\_NOMINAL$

Ce module fournit l'information $Null$ lorsque $B\_Init = 1$ et calcul l'état de $S1$ lorsque $B\_Init = 0$. De ce fait, trois différents états peuvent être observés au niveau de la sortie de ce module $(Null, 0, 1)$. Par conséquent, la quantité d'information théorique disponible à la sortie du module est : $Q = \log_2(3)$. Les probabilités d'occurrence des différentes valeurs sont notées : $P_{s\_m\_n}(Null)$, $P_{s\_m\_n}(0)$ et $P_{s\_m\_n}(1)$.

Ces probabilités sont calculées comme suit :

$$P_{s\_m\_n}(Null) = P(B\_Init = 1) = P_B$$
$$P_{s\_m\_n}(0) = P(B\_Init = 1 \ et \ S1 = 0) = P_B \times (p^2 - p + 1)$$
$$P_{s\_m\_n}(1) = P(B\_Init = 1 \ et \ S1 = 1) = P_B \times (p - p^2)$$

Le CPI du module $\_PULSE1\_NOMINAL$ est donné par l'expression suivante :

$$CPI(\_PULSE1\_NOMINAL) = -\frac{1}{\log_2(3)} \times \sum_{j=\{Null, 0, 1\}} P_{s\_m\_n}(j) \log_2 \left(P_{s\_m\_n}(j)\right)$$

A.3 Modélisation des opérateurs de type « stabilisateur de front non rechargeable »

On distingue deux type opérateurs de stabilisateur de front avec une mémoire non rechargeable : $MTRIG1$ (stabilisateur d'un front montant) et $MTRIG2$ (stabilisateur d'un front descendant). L'opérateur $MTRIG1$ est utilisé pour décrire la modélisation de ces types d'opérateur. $MTRIG1$ est un MONOSTABLE sur front montant de l'entrée $E1$ sur Nb_cycles cycles (figure ci-dessous). Les entrées liées à l'initialisation de cet opérateur sont :

- B_Init : l'entrée d'initialisation de type booléen ;
- $Init$: la valeur affectée à la sortie $S1$ à l'initialisation.

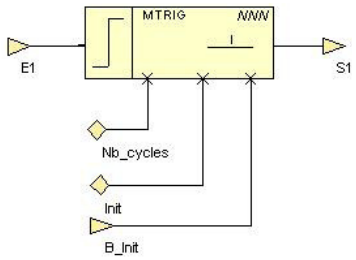


Figure A.5 : Représentation SCADE de MTRIG1

Le comportement de l'opérateur $MTRIG1$ peut être décrit comme suit :

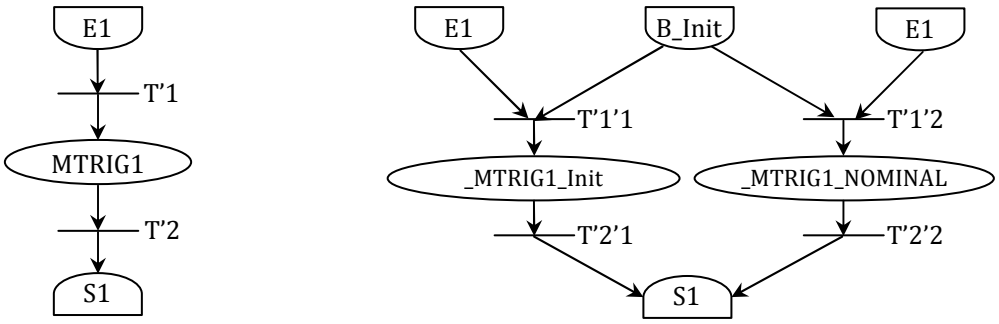
$Si\ B\_Init(k) = True$
$\quad Alors$
$\quad\quad E1(k-i) = E1(k),\ pour\ i\ allant\ de\ 1\ à\ Init$
$\quad\quad Si\ Init \neq Nb\_cycles\ Alors\ E1(k-Init-1) = S1(k-Init-1) = False$
$\quad\quad "Calcul\ normal\ de\ l'état\ de\ la\ sortie\ S1"$
$\quad Sinon\ "Calcul\ normal\ de\ l'état\ de\ la\ sortie\ S1"$

$Calcul\ normal\ de\ l'état\ de\ la\ sortie\ S1$
$Si\ (\exists j \in]k - Nb\_cycles,\ k]\ /\ E1(j-1) = False\ et\ E1(k) = True\ et\ S1(j-1) = False)$
$\quad Alors\ S1(k) = True$
$\quad Sinon\ S1(k) = False$

La sortie $S1$ est à $True$ pendant Nb_cycles lorsque l'entrée $E1$ passe de $False$ à $True$. Si pendant le temps que la sortie $S1$ est $True$, $E1$ passe de $False$ à $True$, ceci n'a pas d'effet sur la sortie $S1$.

A.3.1 Modélisation du MTI élémentaire

Deux différentes modélisations possibles peuvent être décrites pour la représentation du MTI de l'opérateur $MTRIG1$: sans et avec la prise en compte de l'initialisation de l'opération au cours du fonctionnement du système (figure ci-dessous). La première (a) est constituée d'un module fonctionnel produisant la sortie $S1$ à partir de l'entrée $E1$. La deuxième (b) comporte deux modules fonctionnels. Le premier module $\_MTRIG1\_Init$ permet de calculer l'état de la sortie $S1$ à l'occurrence d'une initialisation ; le module $\_MTRIG1\_NOMINAL$ produit l'état de la sortie S1 lorsqu'il n'y a pas d'initialisation. L'entrée de délai ($Init$ et $Nb\_cylcles$) ne sont pas représentées dans le MTI. En effet, elles n'apportent pas plus d'information sur la dépendance des flots contenus dans la spécification.



(a) Sans initialisation **(b) Avec initialisation**

Figure A.6 : Représentation graphique des MTI de MTRIG1

Les description textuelle associée aux MTI de l'opérateur *MTRIG*1, avec ou sans la phase d'initialisation, pour l'analyse de testabilité sont décrites ci-dessous.

– Description sans initialisation

```
1 (*
2   Description du MTI de "MTRIG1" sans initialisation
3 *)
4
5 type MTRIG1_I_I
6 debut
7       debut_entrees
8               IN1 : 1; (* E1 *)
9       fin_entrees
10      debut_sorties
11              OUT1 : 1000; (* S1 *)
12      fin_sorties
13      debut_modele
14              T'1 / IN1 / MTRIG1 /
15              T'2 / MTRIG1 / OUT1 /
16      fin_modele
17 fin
```

– Description avec initialisation

```
1 (*
2   Description du MTI de "MTRIG1" avec initialisation
3 *)
4
5 type MTRIG1_II_I
6 debut
7       debut_entrees
8               IN1 : 1; (* E1 *)
9               IN2 : 2; (* B_Init *)
10      fin_entrees
11      debut_sorties
12              OUT1 : 1000 ; (* S1 *)
13      fin_sorties
14      debut_modele
15              T'1'1 / IN1, IN2 / _ MTRIG1_Init /
16              T'1'2 / IN1, IN2 / _ MTRIG1_NOMINAL /
17              T'2'1 / _ MTRIG1 _Init / OUT1 /
18              T'2'2 / _ MTRIG1_NOMINAL / OUT1 /
19      fin_modele
21 fin
```

A.3.2 Evaluation du CPI de l'opérateur « MTRIG1 »

Nous présentons les méthodes d'évaluation du CPI définies pour l'opérateur *MTRIG*1 selon les modes de fonctionnement avec ou sans initialisation.

Evaluation du CPI sans initialisation

L'expression représentant le comportement de cet opérateur est décrite ci-dessous.

$$Si\ (\exists j \in]k - Nb\_cycles,\ k]\ /\ E1(j-1) = False\ et\ E1(k) = True\ et\ S1(j-1) = False)$$
$$Alors\ S1(k) = True$$
$$Sinon\ S1(k) = False$$

La sortie $S1$ est à $True$ pendant Nb_cycles lorsque l'entrée $E1$ passe de $False$ à $True$. Si pendant le temps que la sortie $S1$ est $True$, $E1$ passe de $False$ à $True$, ceci n'a pas d'effet sur la sortie $S1$.

Notons $cycle(-n)$ le $n^{ième}$ cycle précédent le cycle courant et FM un front montant. La probabilité d'occurrence de $True\ (1)$ au niveau de la sortie $S1$ est définie comme suit :

$$\begin{aligned} P(S=1\ au\ cycle\ courant) = {} & P(FM\ au\ cycle - (Nb\_cycles-1)\ et\ S=0\ au\ cycle - Nb\_cycles) \\ & + P(FM\ au\ cycle - (Nb\_cycles-2)\ et\ S=0\ au\ cycle - (Nb\_cycles-1)) \\ & + \dots \\ & + P(FM\ au\ cycle\ courant\ et\ S=0\ au\ cycle\ précedent) \end{aligned}$$

Pour tout $n \in [1,\ Nb\_cycles]$,
$P(FM\ au\ cycle - (n-1)\ et\ S=0\ au\ cycle - n)$ est notée $P_{FM}(S=1\ au\ cycle - (n-1))$ et
$P_{FM}(S=1\ au\ cycle - (n-1)) = P(FM\ au\ cycle - (n-1)) \times P(S=0\ au\ cycle - n)$ car les évènements sont indépendants les uns des autres.

Par ailleurs, sur deux cycles consécutifs i et $i-1$ de l'intervalle $[1,\ Nb\_cycles]$, les différentes possibilités pour le calcul de $S1$ sont exposées dans le table ci-dessous.

S_{i-1}	E_{i-1}	E_i	S_i	Probabilités
0	0	1	1	$P_{FM}(S=1\ au\ cycle - i)$
1	0	1	0	$P(S=0\ au\ cycle - i)$
0	1	0	0	
1	1	0	0	
0	0	0	0	
1	0	0	0	
0	1	1	0	
1	1	1	0	

D'où l'expression $P_{FM}(S=1\ au\ cycle - i) + P(S=0\ au\ cycle - i) = 1\ \forall i \in [1,\ Nb\_cycles]$.

Comme $P(FM\ à\ un\ cycle) = (1-p) \times p$, nous pouvons écrire l'expression suivante :
$P_{FM}(S=1\ au\ cycle - (n-1)) = ((1-p) \times p) \times (1 - P_{FM}(S=1\ au\ cycle - n))$
Or, $P_{FM}(S=1\ au\ cycle - (n-1)) = P_{FM}(S=1\ au\ cycle - n)$ car les évènements de $E1(k)$ sont indépendants.
$\Rightarrow P_{FM}(S=1\ au\ cycle - n) = ((1-p) \times p) \times (1 - P_{FM}(S=1\ au\ cycle - n))$

$$\Rightarrow P_{FM}(S=1\ au\ cycle - n) = \frac{p - p^2}{1 + p^2 - p} \text{ et } P(S=0\ au\ cycle - n) = \frac{1 + 2p^2 - 2p}{1 + p^2 - p}$$

Donc $\forall n \in [1,\ Nb\_cycles]$:

$$P(FM\ au\ cycle -(Nb\_cycles-1)\ et\ S=0\ au\ cycle - Nb\_cycles)$$
$$= P_{FM}(S=1\ au\ cycle -(Nb\_cycles-1))\times P(S=0\ au\ cycle - Nb\_cycles)$$
$$= P_{FM}(S=1\ au\ cycle -n)\times P(S=0\ au\ cycle -n)$$

$$P(FM\ au\ cycle -(Nb\_cycles-2)\ et\ S=0 -(Nb\_cycles-1))$$
$$= P_{FM}(S=1\ au\ cycle -n)\times P(S=0\ au\ cycle -n)^2$$

$$P(FM\ au\ cycle -(Nb\_cycles-3)\ et\ S=0\ au\ cycle -(Nb\_cycles-2))$$
$$= P_{FM}(S=1\ au\ cycle -n)\times P(S=0\ au\ cycle -n)^3$$

…

$$P(FM\ au\ cycle courant\ et\ S=0\ au\ cycle\ précédent)$$
$$= P_{FM}(S=1\ au\ cycle -n)\times P(S=0\ au\ cycle -n)^{Nb\_cycles}$$

De ce fait,

$$P(S=1\ au\ cycle\ courant) = P_{FM}(S=1\ au\ cycle -n)\times \sum_{i=1}^{Nb\_cycles} P(S=0\ au\ cycle -n)^i$$

$$\Rightarrow P(S=1\ au\ cycle\ courant) = P_{FM}(S=1\ au\ cycle -n)\times \frac{1-P(S=0\ au\ cycle -n)^{Nb\_cycles}}{1-P(S=0\ au\ cycle -n)}$$

$$\Rightarrow P(S=1\ au\ cycle\ courant) = 1-P(S=0\ au\ cycle -n)^{Nb\_cycles} = 1-\left(\frac{1+2p^2-2p}{1+p^2-p}\right)^{Nb\_cycles}$$

$$P(S=0\ au\ cycle\ courant) = \left(\frac{1+2p^2-2p}{1+p^2-p}\right)^{Nb\_cycles}$$

Donc : $CPI(MTRIG1) = -\sum_{j=\{0,1\}} P(j)\log_2(P(j))$

Evaluation du CPI avec initialisation

Le calcul du CPI porte sur les deux modules « $\_MTRIG1\_Init$ » et « $\_MTRIG1\_NOMINAL$ » du MTI de l'opérateur. L'état de la sortie $S1$ est déterminé comme suit :

$Si\ B\_Init(k) = True$
  $Alors$
    $E1(k-i) = E1(k),\ pour\ i\ allant\ de\ 0\ à\ Init$
    $Si\ Init \neq Nb\_cycles\ Alors\ E1(k-Init-1) = S1(k-Init-1) = False$
    $"Calcul\ normal\ de\ l'état\ de\ la\ sortie\ S1"$
  $Sinon\ "Calcul\ normal\ de\ l'état\ de\ la\ sortie\ S1"$

a. *Module* $\_MTRIG1\_Init$

Ce module fournit l'information $Null$ lorsque $B\_Init = 0$ et calcul l'état de $S1$ lorsque $B\_Init = 1$. De ce fait, trois différents états peuvent être observés au niveau de la sortie de ce module $(Null, 0, 1)$. Par conséquent, la quantité d'information théorique disponible à la sortie du module est : $Q = \log_2(3)$.

Soient P_B est la probabilité de l'évènement $B\_Init = 1$, $P_{s\_m\_i}(Null)$ la probabilité d'occurrence de la valeur $Null$, $P_{s\_m\_i}(0)$ la probabilité d'occurrence de la valeur 0 et $P_{s\_m\_i}(1)$ la probabilité d'occurrence de la valeur 1 à la sortie du module $\_MTRIG1\_Init$ représentant la phase initialisation de l'exécution de $MTRIG1$.

$$P_{s\_m\_i}(Null) = P(B\_Init = 0) = 1 - P_B$$

$$P_{s\_m\_i}(1) = P((B\_Init = 1 \; et \; E(k) = 1) \; ou \; (B\_Init = 1 \; et \; S1 = 1 \; sur \; Nb\_cycles - Init \; cycles))$$

$$= P(B\_Init = 1 \; et \; E(k) = 1) + P(B\_Init = 1 \; et \; S1 = 1 \; sur \; Nb\_cycles - Init \; cycles)$$

$$= (P_B \times p) + \left[P_B \times \left(1 - \left(\frac{1 + 2p^2 - 2p}{1 + p^2 - p} \right)^{Nb\_cycles - Init} \right) \right]$$

$$P_{s\_m\_i}(1) = P_B \times \left[p + \left(1 - \left(\frac{1 + 2p^2 - 2p}{1 + p^2 - p} \right)^{Nb\_cycles - Init} \right) \right]$$

$$P_{s\_m\_i}(0) = 1 - \left[P_B \times \left[p + \left(1 - \left(\frac{1 + 2p^2 - 2p}{1 + p^2 - p} \right)^{Nb\_cycles - Init} \right) \right] \right]$$

Donc : $CPI(\_MTRIG1\_Init) = -\frac{1}{\log_2(3)} \times \sum_{j=\{Null,\, 0,\, 1\}} P_{s\_m\_n}(j) \log_2 \left(P_{s\_m\_n}(j) \right)$

b. *Module* $\_MTRIG1\_NOMINAL$

Ce module fournit l'information $Null$ lorsque $B\_Init = 1$ et calcul l'état de $S1$ lorsque $B\_Init = 0$. De ce fait, trois différents états peuvent être observés au niveau de la sortie de ce module $(Null, 0, 1)$. Par conséquent, la quantité d'information théorique disponible à la sortie du module est : $Q = \log_2(3)$.

Soient $P_{s\_m\_n}(Null)$ la probabilité d'occurrence de la valeur $Null$, $P_{s\_m\_n}(0)$ la probabilité d'occurrence de la valeur 0 et $P_{s\_m\_n}(1)$ la probabilité d'occurrence de la valeur 1 à la sortie du module $\_MTRIG1\_NOMINAL$ représentant la phase nominale de l'exécution de $MTRIG1$.

$$P_{s\_m\_i}(Null) = P(B\_Init = 1) = P_B$$

$$P_{s\_m\_i}(1) = P\left(\begin{array}{l}(B\_Init = 0 \ sur \ Nb\_cycles \ et \ S1 = 1) \ ou \\ (B\_Init = 1 \ une \ fois \ sur \ Nb\_cycles - Init \ et \ (S1 = 1 \ en \ ce \ cycle \ ou \ S1 \ après \ ce \ cycle))\end{array}\right)$$

$$= P(B\_Init = 0 \ sur \ Nb\_cycles \ et \ S1 = 1)$$

$$+ P(B\_Init = 1 \ une \ fois \ sur \ Nb\_cycles - Init) \times P(S1 = 1 \ en \ ce \ cycle \ ou \ S1 \ après \ ce \ cycle)$$

$$= (1 - P_B)^{Nb\_cycles+1} \times \left(1 - \left(\frac{1 + 2p^2 - 2p}{1 + p^2 - p}\right)^{Nb\_cycles}\right)$$

$$+ P_B \times \sum_{i=1}^{Nb\_cyles-Init} \left(p + \left(1 - \left(\frac{1 + 2p^2 - 2p}{1 + p^2 - p}\right)^{Nb\_cycles-i}\right)\right) + \left(1 - \left(\frac{1 + 2p^2 - 2p}{1 + p^2 - p}\right)^{i}\right)$$

$$P_{s\_m\_i}(0) = 1 - \left(P_{s\_m\_i}(Null) + P_{s\_m\_i}(1)\right)$$

Donc : $CPI(\_MTRIG1\_NOMINAL) = -\frac{1}{\log_2(3)} \times \sum_{j=\{Null,\,0,\,1\}} P_{s\_m\_n}(j) \log_2\left(P_{s\_m\_n}(j)\right)$

A.4 Modélisation de l'opérateur de type « stabilisateur de front rechargeable »

*MRTRIG*1 et *MRTRIG*2 sont les deux opérateurs de type stabilisateur de front avec une mémoire rechargeable respectivement d'un front montant et d'un front descendant.

*MRTRIG*1 est utilisé pour stabiliser le dernier front montant à l'entrée $E1$ sur Nb_cycles cycles (figure ci-dessous). Les entrées liées à l'initialisation de cet opérateur sont :

- B_Init : l'entrée d'initialisation de type booléen ;
- $Init$: la valeur affectée à la sortie $S1$ à l'initialisation.

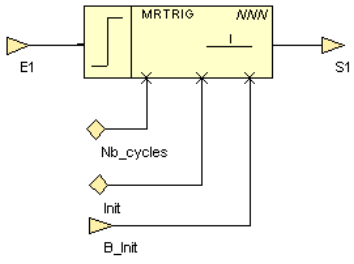


Figure A.7 : Représentation SCADE de MRTRIG1

Le comportement de l'opérateur *MRTRIG*1 peut être décrit comme suit :

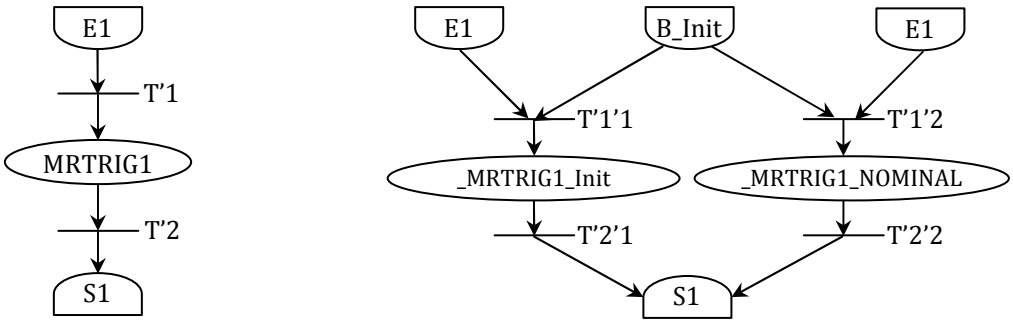
$Si\ B\_Init(k) = True$
$Alors$
$E1(k-i) = E1(k),\ pour\ i\ allant\ de\ 0\ à\ Init$
$Si\ Init \neq Nb\_cycles\ Alors\ E1(k-Init-1) = S1(k-Init-1) = False$
"$Calcul\ normal\ de\ l'état\ de\ la\ sortie\ S1$"
$Sinon$ "$Calcul\ normal\ de\ l'état\ de\ la\ sortie\ S1$"

$Calcul\ normal\ de\ l'état\ de\ la\ sortie\ S1$
$Si\ (\exists j \in\ ]k - Nb\_cycles,\ k]\ /\ E1(j-1) = False\ et\ E1(k) = True)$
$Alors\ S1(k) = True$
$Sinon\ S1(k) = False$

La sortie $S1$ est à $True$ pendant Nb_cycles lorsque l'entrée $E1$ passe de $False$ à $True$. Si pendant le temps que la sortie $S1$ est $True$, $E1$ passe de $False$ à $True$, la sorite $S1$ est maintenue à $True$ pendant à partir de ce cycle.

A.4.1 Modélisation du MTI élémentaire

Deux différentes modélisations possibles peuvent être décrites pour la représentation du MTI de l'opérateur $MRTRIG1$: sans et avec la prise en compte de l'initialisation de l'opérateur au cours du fonctionnement du système (figure ci-dessous). La première (a) est constituée d'un module fonctionnel produisant la sortie $S1$ à partir de l'entrée $E1$. La deuxième (b) comporte deux modules fonctionnels. Le premier module $\_MRTRIG1\_Init$ permet de calculer l'état de la sortie $S1$ à l'occurrence d'une initialisation ; le module $\_MRTRIG1\_NOMINAL$ produit l'état de la sortie S1 lorsqu'il n'y a pas d'initialisation. L'entrée de délai ($Init$ et $Nb\_cylcles$) ne sont pas représentées dans le MTI. En effet, elles n'apportent pas plus d'information sur la dépendance des flots contenus dans la spécification.



(a) Sans initialisation **(b) Avec initialisation**

Figure A.8 : Représentation graphique des MTI de MTRIG1

La description textuelle associée à chacune des deux représentations du MTI de l'opérateur *MRTRIG*1 pour l'analyse de testabilité est décrite ci-dessous.

– Représentation sans initialisation (a)

```
1 (*
2   Description du MTI de "MRTRIG1" sans initialisation
3 *)
4
5 type MRTRIG1_I_I
6 debut
7       debut_entrees
8               IN1 : 1; (* E1 *)
9       fin_entrees
10      debut_sorties
11              OUT1 : 1000 ; (* S1 *)
12      fin_sorties
13      debut_modele
14              T'1 / IN1 / MRTRIG1 /
15              T'2 / MRTRIG1 / OUT1 /
16      fin_modele
17 fin
```

– Représentation avec initialisation (b)

```
1 (*
2   Description du MTI de "MRTRIG1" avec initialisation
3 *)
4
5 type MRTRIG1_II_I
6 debut
7       debut_entrees
8               IN1 : 1; (* E1 *)
9               IN2 : 2; (* B_Init *)
10      fin_entrees
11      debut_sorties
12              OUT1 : 1000 ; (* S1 *)
13      fin_sorties
14      debut_modele
15              T'1'1 / IN1, IN2 / _ MRTRIG1_Init /
16              T'1'2 / IN1, IN2 / _ MRTRIG1_NOMINAL /
17              T'2'1 / _ MRTRIG1_Init / OUT1 /
18              T'2'2 / _ MRTRIG1_NOMINAL / OUT1 /
19      fin_modele
20 fin
```

A.4.2 Evaluation du CPI de l'opérateur « MRTRIG1 »

Nous présentons les méthodes d'évaluation du CPI définies pour l'opérateur *MRTRIG*1 selon les modes de fonctionnement avec ou sans initialisation.

Evaluation du CPI sans initialisation

Notons $cycle(-n)$ le $n^{ième}$ cycle précédent le cycle courant et FM un front montant. La probabilité d'occurrence de $True\ (1)$ au niveau de la sortie $S1$ est définie comme suit :

$$\begin{aligned} P(S=1\ au\ cycle\ courant) &= P(FM\ au\ cycle\ -(Nb\_cycles-1)\ et\ non\ FM\ après) \\ &+ P(FM\ au\ cycle\ -(Nb\_cycles-2)\ et\ non\ FM\ après) \\ &+ \dots \\ &+ P(FM\ au\ cycle\ courant) \end{aligned}$$

Pour tout $n \in [1,\ Nb\_cycles]$,
$P(FM\ au\ cycle\ -(n-1)\ et\ non\ FM\ après)$ est notée $P_{FM}(S=1\ au\ cycle\ -(n-1))$
$P_{FM}(S=1\ au\ cycle\ -(n-1)) = P(FM\ au\ cycle\ -(n-1)) \times P(non\ FM\ sur\ n-2\ cycles)$ car tous les évènements sont indépendants les uns des autres.
Par ailleurs : $P(FM\ à\ un\ cycle) = (1-p)\times p$ et $P_{FM}(non\ FM\ sur\ n-1\ cycles) = (1-P-P^2)^{n-1}$
Donc :

$$\begin{aligned} P(S=1\ au\ cycle\ courant) &= ((1-p)\times p)\times\left[(1-P-P^2)^{Nb\_cycles-1}\right] \\ &+ ((1-p)\times p)\times\left[(1-P-P^2)^{Nb\_cycles-2}\right] \\ &+ \dots \\ &+ ((1-p)\times p)\times\left[(1-P-P^2)^{Nb\_cycles-Nb\_cycles}\right] \end{aligned}$$

$$\Rightarrow P(S=1\ au\ cycle\ courant) = ((1-p)\times p)\times\left[1+(1-p+p^2)+(1-p+p^2)^2+\dots+(1-p+p^2)^{Nb\_cycles-1}\right]$$

$$\begin{aligned} P(S=1\ au\ cycle\ courant) &= ((1-p)\times p)\times \sum_{k=0}^{Nb\_cycles-1}(1-p+p^2)^k \\ &= ((1-p)\times p)\times\left[\frac{1-(1-p+p^2)^{Nb\_cycles}}{1-(1-p+p^2)}\right] \\ &= 1-(1-p+p^2)^{Nb\_cycles} \end{aligned}$$

$$\begin{aligned} P(S=0\ au\ cycle\ courant) &= P(non\ FM\ sur\ Nb\_cycles+1\ cycles) \\ &= P(non\ FM\ au\ cycle\ -(Nb\_cycles-1)) \\ &\quad \times P(non\ FM\ au\ cycle\ -(Nb\_cycles-2))\times\dots \\ &\quad \times P(non\ FM\ au\ cycle\ courant) \\ &= 1-(1-p+p^2)^n \end{aligned}$$

D'où : $CPI(MRTRIG1) = -\sum_{j=\{0,1\}} P(j)\log_2(P(j))$

Evaluation du CPI avec initialisation

Le calcul du CPI porte sur les deux modules « $\_MRTRIG1\_Init$ » et « $\_MRTRIG1\_NOMINAL$ » du MTI de l'opérateur.

a. *Module* $\_MRTRIG1\_Init$

Ce module fournit l'information *Null* lorsque $B\_Init = 0$ et calcul l'état de $S1$ lorsque $B\_Init = 1$. De ce fait, trois différents états peuvent être observés au niveau de la sortie de ce

module $(Null, 0, 1)$. Par conséquent, la quantité d'information théorique disponible à la sortie du module est : $Q = \log_2(3)$.

Soient P_B est la probabilité de l'évènement $B\_Init = 1$, $P_{s\_m\_i}(Null)$ la probabilité d'occurrence de la valeur $Null$, $P_{s\_m\_i}(0)$ la probabilité d'occurrence de la valeur 0 et $P_{s\_m\_i}(1)$ la probabilité d'occurrence de la valeur 1 à la sortie du module $\_MRTRIG1\_Init$ représentant la phase initialisation de l'exécution de $MRTRIG1$.

$$P_{s\_m\_i}(Null) = P(B\_Init = 0) = 1 - P_B$$

$$\begin{aligned} P_{s\_m\_i}(1) &= P((B\_Init = 1 \ et \ E(k) = 1) \ ou \ (B\_Init = 1 \ et \ S1 = 1 \ sur \ Nb\_cycles - Init \ cycles)) \\ &= P(B\_Init = 1 \ et \ E(k) = 1) + P(B\_Init = 1 \ et \ S1 = 1 \ sur \ Nb\_cycles - Init \ cycles) \\ &= (P_B \times p) + \left[P_B \times \left(1 - (1 - p + p^2)^{Nb\_cycles - Init}\right)\right] \end{aligned}$$

$$P_{s\_m\_i}(1) = P_B \times \left[p + \left(1 - (1 - p + p^2)^{Nb\_cycles - Init}\right)\right]$$

$$P_{s\_m\_i}(0) = 1 - \left[P_B \times \left[p + \left(1 - (1 - p + p^2)^{Nb\_cycles - Init}\right)\right]\right]$$

Donc : $CPI(\_MRTRIG1\_Init) = -\frac{1}{\log_2(3)} \times \sum_{j=\{Null, 0, 1\}} P_{s\_m\_n}(j) \log_2\left(P_{s\_m\_n}(j)\right)$

b. *Module* $\_MRTRIG1\_NOMINAL$

Ce module fournit l'information $Null$ lorsque $B\_Init = 1$ et calcul l'état de $S1$ lorsque $B\_Init = 0$. De ce fait, trois différents états peuvent être observés au niveau de la sortie de ce module $(Null, 0, 1)$. Par conséquent, la quantité d'information théorique disponible à la sortie du module est : $Q = \log_2(3)$.

Soient $P_{s\_m\_n}(Null)$ la probabilité d'occurrence de la valeur $Null$, $P_{s\_m\_n}(0)$ la probabilité d'occurrence de la valeur 0 et $P_{s\_m\_n}(1)$ la probabilité d'occurrence de la valeur 1 à la sortie du module $\_MTRIG1\_NOMINAL$ représentant la phase nominale de l'exécution de $MTRIG1$.

$$P_{s\_m\_i}(Null) = P(B\_Init = 1) = P_B$$

$$\begin{aligned} P_{s\_m\_i}(1) &= P\left(\begin{array}{l}(B\_Init = 0 \ sur \ Nb\_cycles \ et \ S1 = 1) \ ou \\ (B\_Init = 1 \ une \ fois \ sur \ Nb\_cycles - Init \ et \ (S1 = 1 \ en \ ce \ cycle \ ou \ S1 \ après \ ce \ cycle))\end{array}\right) \\ &= P(B\_Init = 0 \ sur \ Nb\_cycles \ et \ S1 = 1) \\ &\quad + P(B\_Init = 1 \ une \ fois \ sur \ Nb\_cycles - Init) \times P(S1 = 1 \ en \ ce \ cycle \ ou \ S1 \ après \ ce \ cycle) \\ &= (1 - P_B)^{Nb\_cycles + 1} \times \left(1 - (1 - p + p^2)^{Nb\_cycles}\right) \\ &\quad + P_B \times \sum_{i=1}^{Nb\_cyles - Init} \left(p + \left(1 - (1 - p + p^2)^{Nb\_cycles - i}\right)\right) + \left(1 - (1 - p + p^2)^{i}\right) \end{aligned}$$

$$P_{s\_m\_i}(0) = 1 - \left(P_{s\_m\_i}(Null) + P_{s\_m\_i}(1)\right)$$

Donc : $CPI(\_MRTRIG1\_NOMINAL) = -\frac{1}{\log_2(3)} \times \sum_{j=\{Null,\,0,\,1\}} P_{s\_m\_n}(j) \log_2\left(P_{s\_m\_n}(j)\right)$

A.5 Modélisation des opérateurs de type « retard »

Il existe deux types d'opérateur de retard : $PREV$ qui retarde un flot d'information de type booléen et FBY qui retarde un flot d'information de type booléen ou réel. L'opérateur $PREV$ est utilisé pour décrire la modélisation des opérateurs de retard. $PREV$ permet de retarder les valeur d'un flot d'information en entrée $E1$ sur Nb_cycles cycles au niveau de la sortie $S1$ (figure ci-dessous). Les entrées liées à l'initialisation de cet opérateur sont :

- B_Init : l'entrée d'initialisation de type booléen ;
- $Init$: la valeur affectée à la sortie $S1$ à l'initialisation.

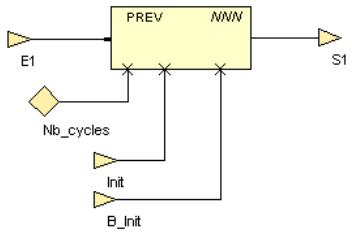


Figure A.9 : Représentation SCADE de l'opérateur PREV

Le comportement de l'opérateur $PREV$ peut être décrit comme suit :

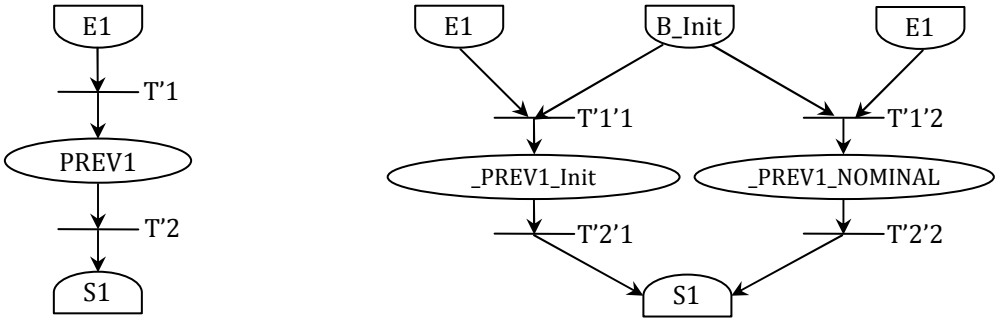
$$Si\ B\_Init(k) = True$$
$$\quad Alors\ S1(k) = Init(k)$$
$$\quad\quad E1(k-i) = Init,\ pour\ i\ allant\ de\ 0\ à\ Nb\_cycles$$
$$\quad Sinon\ "Calcul\ normal\ de\ l'état\ de\ la\ sortie\ S1"$$

$$Calcul\ normal\ de\ l'état\ de\ la\ sortie\ S1$$
$$S1(k) = E1(k - Nb\_cycles)$$

La valeur de la sortie $S1$ au cycle courant correspond à celle de l'entrée $E1$ au $Nb\_cycles^{ième}$ cycle précédent. Lorsque B_Init est $True$, la valeur de $Init$ est affectée à la sortie $S1$ et la mémoire de l'opérateur est initialisée avec cette même valeur sur les Nb_cycles cycles précédents.

A.5.1 Modélisation du MTI élémentaire

Deux différentes modélisations possibles peuvent être décrites pour la représentation du MTI de l'opérateur *PREV* : sans et avec la prise en compte de l'initialisation de l'opération au cours du fonctionnement du système (figure ci-dessous). La première (a) est constituée d'un module fonctionnel produisant la sortie $S1$ à partir de l'entrée $E1$. La deuxième (b) comporte deux modules fonctionnels. Le premier module *_PREV_Init* permet de calculer l'état de la sortie $S1$ à l'occurrence d'une initialisation ; le module *_PREV_NOMINAL* produit l'état de la sortie S1 lorsqu'il n'y a pas d'initialisation. L'entrée de délai (*Init* et *Nb_cylcles*) ne sont pas représentées dans le MTI. En effet, elles n'apportent pas plus d'information sur la dépendance des flots contenus dans la spécification.



(a) Sans initialisation **(b) Avec initialisation**

Figure A.10 : Représentation graphique des MTI de PREV

Description textuelle du MTI de l'opérateur « PREV »

Les description textuelle associée aux MTI de l'opérateur *PREV*, avec ou sans la phase d'initialisation, pour l'analyse de testabilité sont décrites ci-dessous.

– Description sans initialisation

```
1 (*
2   Description du MTI de " PREV " sans initialisation
3 *)
4
5 type PREV_I_I
6 debut
7       debut_entrees
8               IN1 : 1; (* E1 *)
9       fin_entrees
10      debut_sorties
11              OUT1 : 1000; (* S1 *)
12      fin_sorties
13      debut_modele
14              T'1 / IN1 / PREV /
15              T'2 / PREV / OUT1 /
16      fin_modele
```

```
17 fin
```

– Description avec initialisation

```
1 (*
2   Description du MTI de " PREV " avec initialisation
3 *)
4
5 type PREV_II_I
6 debut
7       debut_entrees
8               IN1 : 1; (* E1 *)
9               IN2 : 2; (* B_Init *)
10      fin_entrees
11      debut_sorties
12              OUT1 : 1000 ; (* S1 *)
13      fin_sorties
14      debut_modele
15              T'1'1 / IN1, IN2 / _ PREV_Init /
16              T'1'2 / IN1, IN2 / _ PREV_NOMINAL /
17              T'2'1 / _ PREV_Init / OUT1 /
18              T'2'2 / _ PREV_NOMINAL / OUT1 /
19      fin_modele
21 fin
```

A.5.2 Evaluation du CPI de l'opérateur de retard « PREV »

Nous présentons les méthodes d'évaluation du CPI définies pour l'opérateur $PREV$ selon les modes de fonctionnement avec ou sans initialisation.

Evaluation du CPI sans initialisation

L'expression représentant le comportement de cet opérateur est décrite ci-dessous.

$$S1(k) = E1(k - Nb\_cycles)$$

$PREV$ retarde de Nb_cycles les valeurs du flot en entrée $E1$ au niveau de la sortie $S1$. En se basant sur ce comportement, les valeurs de la sortie $S1$ peuvent être considérées comme celles de l'entrée $E1$ avec un retard de Nb_cycles cycles. De ce fait, la loi distribution l'entrée $E1$ est la même que celle de la sortie $S1$.

Soient $P(S1=1)$ et $P(S1=0)$ respectivement les probabilités d'occurrence de 1 et 0 au niveau de la sortie $S1$ au courant cycle d'exécution de l'opérateur $PREV$.

$$P(S1=1) = P(E(k)=1) = p \text{ et } P(S1=0) = P(E(k)=0) = 1-p$$

A partir de ces deux expressions associées aux probabilités d'occurrence des deux états possibles de $S1$ au courant, la valeur du CPI sachant que la quantité d'information théorique au niveau de $S1, Q = 1$ en fonction de p est :

$$CPI(PREV) = -[p \times \log_2(p) + (1-p) \times \log_2(1-p)]$$

Evaluation du CPI avec initialisation

Le calcul du CPI porte sur les deux modules « $\_PREV\_Init$ » et « $\_PREV\_NOMINAL$ » du MTI de l'opérateur. L'état de la sortie $S1$ est déterminé comme suit :

$$Si\ B\_Init(k) = True$$
$$Alors\ S1(k) = Init(k)$$
$$E1(k-i) = Init,\ pour\ i\ allant\ de\ 0\ à\ Nb\_cycles$$

a. *Module* $\_PREV\_Init$

Ce module fournit l'information $Null$ lorsque $B\_Init = 0$ et calcul l'état de $S1$ lorsque $B\_Init = 1$. De ce fait, trois différents états peuvent être observés au niveau de la sortie de ce module $(Null, 0, 1)$. Par conséquent, la quantité d'information théorique disponible à la sortie du module est : $Q = \log_2(3)$.

Soient P_B est la probabilité de l'évènement $B\_Init = 1$, $P_{s\_m\_i}(Null)$ la probabilité d'occurrence de la valeur $Null$, $P_{s\_m\_i}(0)$ la probabilité d'occurrence de la valeur 0 et $P_{s\_m\_i}(1)$ la probabilité d'occurrence de la valeur 1 à la sortie du module $\_PREV\_Init$ représentant la phase initialisation de l'exécution de $PREV$.

$$P_{s\_m\_i}(Null) = P(B\_Init = 0) = 1 - P_B$$
$$P_{s\_m\_i}(0) = P(B\_Init = 1\ et\ S1 = 0) = P_B \times (1-p)$$
$$P_{s\_m\_i}(1) = P(B\_Init = 1\ et\ S1 = 1) = P_B \times p$$

Donc : $CPI(\_PREV\_Init) = -\frac{1}{\log_2(3)} \times \sum_{j=\{Null,\,0,\,1\}} P_{s\_m\_n}(j) \log_2\left(P_{s\_m\_n}(j)\right)$

b. *Module* $\_PREV\_NOMINAL$

Ce module fournit l'information $Null$ lorsque $B\_Init = 1$ et calcul l'état de $S1$ lorsque $B\_Init = 0$. De ce fait, trois différents états peuvent être observés au niveau de la sortie de ce module $(Null, 0, 1)$. Par conséquent, la quantité d'information théorique disponible à la sortie du module est : $Q = \log_2(3)$.

Soient $P_{s\_m\_n}(Null)$ la probabilité d'occurrence de la valeur $Null$, $P_{s\_m\_n}(0)$ la probabilité d'occurrence de la valeur 0 et $P_{s\_m\_n}(1)$ la probabilité d'occurrence de la valeur 1 à la sortie du module $\_PREV\_NOMINAL$ représentant la phase nominale de l'exécution de $PREV$.

$$P_{s\_m\_i}(Null) = P(B\_Init = 0) = P_B$$
$$P_{s\_m\_i}(0) = P(B\_Init = 1 \; et \; S1 = 0) = (1 - P_B) \times (1 - p)$$
$$P_{s\_m\_i}(1) = P(B\_Init = 1 \; et \; S1 = 1) = (1 - P_B) \times p$$

Donc : $CPI(\_PREV\_NOMINAL) = -\frac{1}{\log_2(3)} \times \sum_{j=\{Null,\,0,\,1\}} P_{s\_m\_n}(j) \log_2\left(P_{s\_m\_n}(j)\right)$

Annexe B

Méthodologie de construction d'un modèle SCADE dans le contexte AIRBUS

B.1 Contexte générale de spécification en SCADE

Généralement, la spécification d'un système à l'aide de l'environnement SCADE correspond à un modèle, construit hiérarchiquement, du nœud principal ou nœud système vers les nœuds sous-systèmes. Dans le contexte AIRBUS, les systèmes de commande de vol spécifiés à l'aide de l'environnement SCADE sont décrits sous la forme d'un ensemble de planches SCADE construites sans considération hiérarchique (pas de notion de système ou sous-système). L'ordonnancement d'exécution de ces planches est défini à partir de l'architecture des systèmes. Afin de reproduire le comportement global d'un de ces systèmes sous SCADE, il est nécessaire de créer un nœud principal regroupant les planches réalisées pour décrire ce système, et de définir l'ordonnancement permettant de respecter l'ordre d'exécution des planches.

L'hypothèse de synchronisme de SCADE est la suivante : à chaque pas d'horloge, pour chaque nœud, toutes les entrées sont lues et toutes les sorties sont calculées sauf ceux utilisés par un opérateur « condact ».

Cependant l'ordre d'exécution des nœuds peut être imposé par le concepteur en insérant des opérateurs de retard d'un cycle.

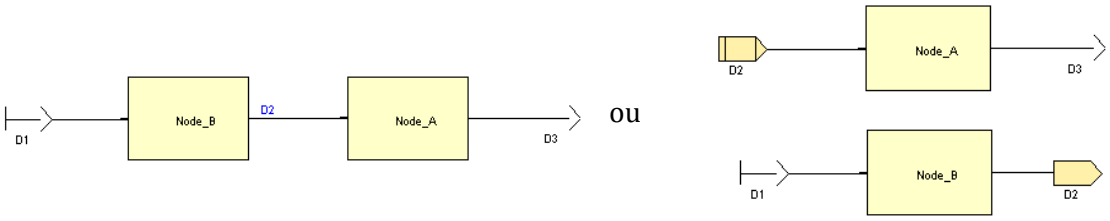


Figure B.1 : Ordre d'exécution des nœuds imposé

En observant la **Figure B.1** ci-dessus, la variable de sortie « D2 » produite par le nœud « Node_B » est consommée par le nœud « Node_A » dans le même pas d'horloge.

Dans le cas de la **Figure B.2** ci-dessous, il n'est pas possible de déterminer lequel des deux nœuds est exécuté en premier.

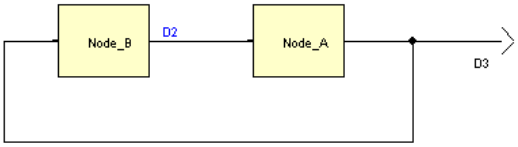
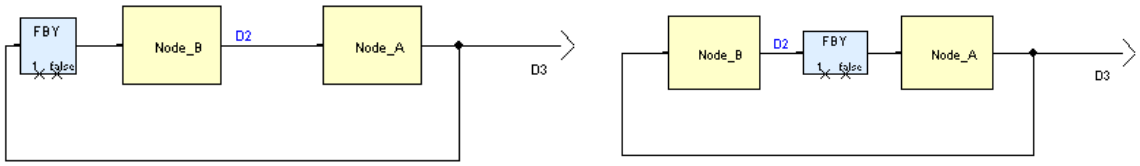


Figure B.2 : Ordre d'exécution des nœuds non déterminé

Ceci montre l'existence d'une boucle de causalité. En insérant un opérateur de retard d'un cycle (FBY avec 1 comme taille de la mémoire), cette boucle de causalité disparaît. Cependant, en fonction de la position de FBY, l'ordre d'exécution n'est pas le même. La **Figure B.3** illustre cette notion de position de l'opérateur de retard.



Node_A est exécuté avant Node_B

Node_B est exécuté avant Node_A

Figure B.3 : Illustration du rôle de l' l'opérateur de retard

B.2 Modélisation de l'aspect multi-cycles de la spécification

Les planches SCADE AIRBUS possèdent des cycles d'exécution différents. La manière classique de représenter ce comportement en SCADE est d'utiliser l'opérateur « condact » qui active le nœud seulement aux cycles correspondant à son cycle d'exécution.

Au niveau des planches SCADE des systèmes AIRBUS, il existe cinq différents cycles d'exécution :

- $C0$: cycle de base (les planches sont exécutées à chaque pas d'horloge) ;
- $C1 : C0 \times 2$ (les planches sont exécutées tous les 2 pas d'horloge) ;
- $C2 : C0 \times 4$ (les planches sont exécutées tous les 4 pas d'horloge) ;
- $C3 : C0 \times 8$ (les planches sont exécutées tous les 8 pas d'horloge) ;
- $C4 : C0 \times 24$ (les planches sont exécutées tous les 24 pas d'horloge) ;

La politique de répartition de charge utilisée chez AIRBUS est de scinder l'exécution de planches de cycle C1, C2, C3 ou C4 en plusieurs sous-cycles :

- C1 : C11 (exécuté tous les 2 pas d'horloge, commençant au pas d'horloge initial) et C12 (exécuté tous les deux cycles, commençant au second pas d'horloge)
- C2 : C21, C22, C23, C24
- C3 : C31, …, C38
- C4 : C41, …, C424

L'ordonnancement général de l'exécution de ces planches est le suivant :

- pas d'horloge 01 : C0, C11, C21, C31, C41
- pas d'horloge 02 : C0, C12, C22 , C32, C42
- pas d'horloge 03 : C0, C11, C23, C33, C43
- pas d'horloge 04 : C0, C12, C24 , C34, C44
- pas d'horloge 05 : C0, C11, C21, C35, C45
- pas d'horloge 06 : C0, C12, C22 , C36, C46
- pas d'horloge 07 : C0, C11, C23, C37, C47
- pas d'horloge 08 : C0, C12, C24 , C38, C48
- pas d'horloge 09 : C0, C11, C21, C31, C49
- pas d'horloge 10 : C0, C12, C22 , C32, C410
- pas d'horloge 11 : C0, C11, C23, C33, C411
- pas d'horloge 12 : C0, C12, C24 , C34, C412
- pas d'horloge 13 : C0, C11, C21, C35, C413
- pas d'horloge 14 : C0, C12, C22 , C36, C414
- pas d'horloge 15 : C0, C11, C23, C37, C415
- pas d'horloge 16 : C0, C12, C24 , C38, C416
- pas d'horloge 17 : C0, C11, C21, C31, C417
- pas d'horloge 18 : C0, C12, C22 , C32, C418
- pas d'horloge 19 : C0, C11, C23, C33, C419

- pas d'horloge 20 : C0, C12, C24 , C34, C420
- pas d'horloge 21 : C0, C11, C21, C35, C421
- pas d'horloge 22 : C0, C12, C22 , C36, C422
- pas d'horloge 23 : C0, C11, C23, C37, C423
- pas d'horloge 24 : C0, C12, C24 , C38, C424

Les règles utilisées pour l'ordonnancement des nœuds exécutés en des sous-cycles différents sont :

- Les nœuds de cycle C0 sont les premiers exécutés, suivis des nœuds de cycles C1, C2, C3 et C4 ;
- Les nœuds du sous-cycle C11 sont exécutés avant ceux du sous-cycle C12 ;
- Les nœuds du sous-cycle C21 sont exécutés avant ceux du sous-cycle C22 qui sont exécutés avant ceux du sous-cycle C23, etc.
- Les nœuds du sous-cycle C31 sont exécutés avant ceux du sous-cycle C32 qui sont exécutés avant ceux du sous-cycle C33, etc.
- Les nœuds du sous-cycle C41 sont exécutés avant ceux du sous-cycle C42 qui sont exécutés avant ceux du sous-cycle C43, etc.
- Avec ces règles, l'ordre d'exécution utilisé pour les sous-cycles est le suivant :
- C0, C11, C21, C31, C41, C12, C22, C32, C42, C23, C33, C43, C24, C34, C44, C35, C45, C36, C46, C37, C47, C38, C48, C49, …, C424

Un opérateur de retard FBY d'un cycle est alors utilisé avant l'utilisation de la valeur d'une variable, lorsque cette valeur est produite par un nœud exécuté dans un sous-cycle ultérieur par rapport à la liste ordonnée des sous-cycles. Au cours de la construction d'un modèle, l'ordonnancement des nœuds est décrit dans un fichier nommé « TEMPO ».

L'illustration de cette technique de simulation de l'aspect multi-cycles se base sur un extrait de la description SCADE des modes longitudinaux du pilote automatique de l'A350XWB. Cet extrait est représenté par la **Figure B.4** ci-dessous. Dans cet extrait, les variables de sortie « *BPITTOENG1* » et « *BPITTOCND* » sont calculées par le nœud « *N450201* ». La valeur d'une variable d'entrée du nœud « *N450201* » correspond à celle de la variable de sortie du nœud « *N455009* » calculée dans un sous-cycle précédent en considérant la liste ordonnée des sous-cycles. Ce retard est modélisé par l'opérateur « *FBY* ».

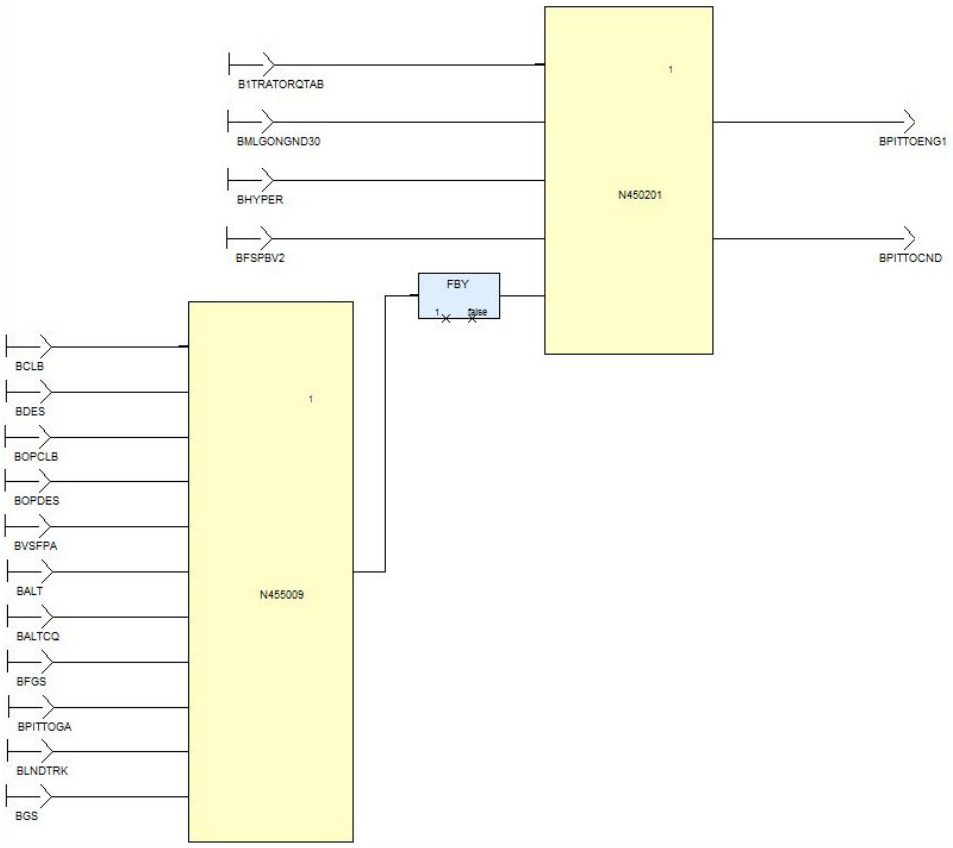


Figure B.4 : Illustration de la technique de simulation de l'aspect multi-cycles

Annexe C

Description MAC d'une planche SCADE

Le langage MAC, reconnu par l'outil d'analyse de testabilité, est un sous-langage du langage DOT. Le langage DOT [DOT06] permet de dessiner des graphes orientés et non orientés sous le logiciel Graphviz. Un graphe écrit du langage DOT est constitué de nœuds, d'arcs (pour un graphe orienté), d'arête (pour un graphe non orienté).

Des attributs peuvent être définis dans le graphe. Ils agissent soit sur l'ensemble du graphe, soit sur un de ses éléments. DOT offre la possibilité de définir des attributs avec une grande souplesse. Ces attributs « non standards » du langage sont ignorés lors de la construction du graphe par Graphviz.

Le langage MAC décrit des graphes orientés représentant la traduction des nœuds SCADE. Le choix du graphe orienté se justifie par la nécessité de connaître le sens de circulation des flots de données entre les différents éléments (entrées, constantes, opérateurs, sorties) du nœud traduit. Des attributs supplémentaires contenant des informations nécessaires à l'analyse de testabilité sont également définis.

La description complète des attributs des nœuds et arcs est donnée dans le manuel de référence du langage MAC [Dou06].

C.1 Exemple de traduction de la planche SCADE N421401

La planche N421401 est extraite du modèle SCADE des modes latéraux du pilote automatique de l'A350XWB (**Figure C.1**).

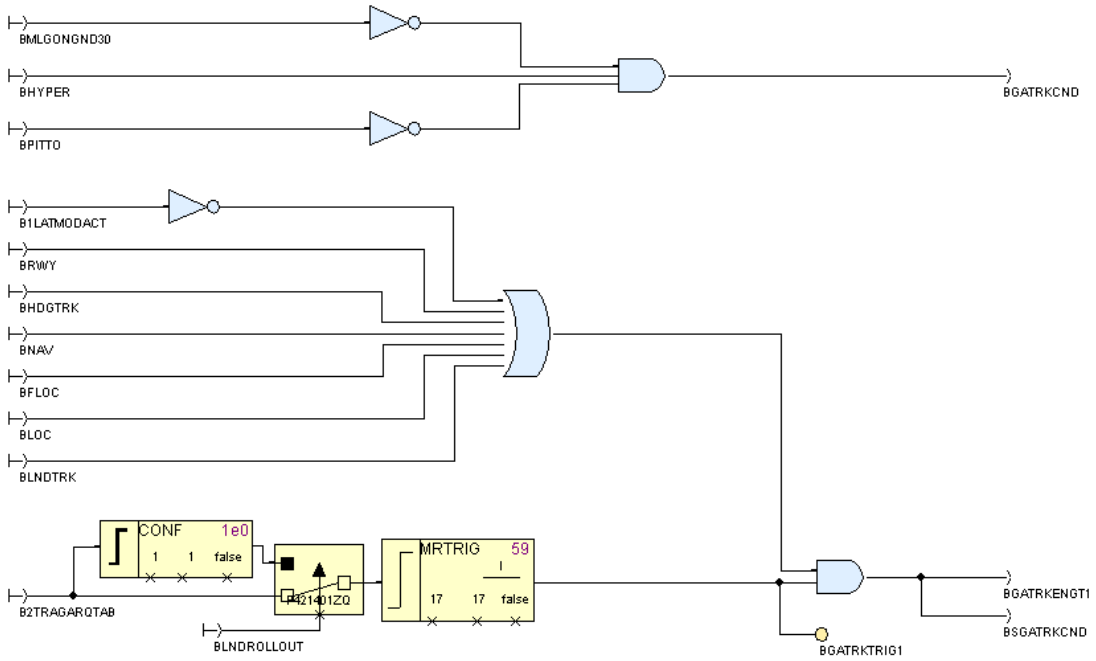


Figure C.1 : Représentation graphique de la planche N421401

La description MAC de cette planche est présentée ci-dessous. Elle contient l'ensemble des informations représentant l'aspect flot de données du digramme d'opérateurs décrit dans la planche.

```
/*
  Traducteur V2
  version du traducteur : light_v0.0.8
  Auteur(s) : F-V-M
  date : Fri 20091204 07h:42
  Parametres du traducteur :
          noeuds_lib = non;
          fichier_types = oui;
          trace = 1;
          initialisation_follow = non;
          toplevel = N421401;
*/
digraph "N421401" {
          software = "SCADE";
          system = "N421401";
          model = "Modes_Lat";
          "BLNDROLLOUT" [
                    system = "N421401",
                    type = "Inport",
                    type_op = "Inport",
                    Hidden = "false",
                    Supplementaire = "false",
                    refblock = "",
                    refblocklib = "",
                    path = "N421401/BLNDROLLOUT",
                    inports = "0",
                    outports = "1",
```

```
                shape = "diamond",
                orientation = "90",
                portnumber = "1",
                inports_types = "",
                inports_names = "",
                outports_types = "Booleen",
                outports_names = "OUT1",
                color = "black",
                fontcolor = "black"
        ];
        "BMLGONGND30" [
                system = "N421401",
                type = "Inport",
                type_op = "Inport",
                Hidden = "false",
                Supplementaire = "false",
                refblock = "",
                refblocklib = "",
                path = "N421401/BMLGONGND30",
                inports = "0",
                outports = "1",
                shape = "diamond",
                orientation = "90",
                portnumber = "2",
                inports_types = "",
                inports_names = "",
                outports_types = "Booleen",
                outports_names = "OUT1",
                color = "black",
                fontcolor = "black"
        ];
        "BPITTO" [
                system = "N421401",
                type = "Inport",
                type_op = "Inport",
                Hidden = "false",
                Supplementaire = "false",
                refblock = "",
                refblocklib = "",
                path = "N421401/BPITTO",
                inports = "0",
                outports = "1",
                shape = "diamond",
                orientation = "90",
                portnumber = "3",
                inports_types = "",
                inports_names = "",
                outports_types = "Booleen",
                outports_names = "OUT1",
                color = "black",
                fontcolor = "black"
        ];
        "B2TRAGARQTAB" [
                system = "N421401",
```

```
            type = "Inport",
            type_op = "Inport",
            Hidden = "false",
            Supplementaire = "false",
            refblock = "",
            refblocklib = "",
            path = "N421401/B2TRAGARQTAB",
            inports = "0",
            outports = "1",
            shape = "diamond",
            orientation = "90",
            portnumber = "4",
            inports_types = "",
            inports_names = "",
            outports_types = "Booleen",
            outports_names = "OUT1",
            color = "black",
            fontcolor = "black"
        ];
        "BHYPER" [
            system = "N421401",
            type = "Inport",
            type_op = "Inport",
            Hidden = "false",
            Supplementaire = "false",
            refblock = "",
            refblocklib = "",
            path = "N421401/BHYPER",
            inports = "0",
            outports = "1",
            shape = "diamond",
            orientation = "90",
            portnumber = "5",
            inports_types = "",
            inports_names = "",
            outports_types = "Booleen",
            outports_names = "OUT1",
            color = "black",
            fontcolor = "black"
        ];
        "BHDGTRK" [
            system = "N421401",
            type = "Inport",
            type_op = "Inport",
            Hidden = "false",
            Supplementaire = "false",
            refblock = "",
            refblocklib = "",
            path = "N421401/BHDGTRK",
            inports = "0",
            outports = "1",
            shape = "diamond",
            orientation = "90",
            portnumber = "6",
```

```
                inports_types = "",
                inports_names = "",
                outports_types = "Booleen",
                outports_names = "OUT1",
                color = "black",
                fontcolor = "black"
        ];
        "BNAV" [
                system = "N421401",
                type = "Inport",
                type_op = "Inport",
                Hidden = "false",
                Supplementaire = "false",
                refblock = "",
                refblocklib = "",
                path = "N421401/BNAV",
                inports = "0",
                outports = "1",
                shape = "diamond",
                orientation = "90",
                portnumber = "7",
                inports_types = "",
                inports_names = "",
                outports_types = "Booleen",
                outports_names = "OUT1",
                color = "black",
                fontcolor = "black"
        ];
        "BFLOC" [
                system = "N421401",
                type = "Inport",
                type_op = "Inport",
                Hidden = "false",
                Supplementaire = "false",
                refblock = "",
                refblocklib = "",
                path = "N421401/BFLOC",
                inports = "0",
                outports = "1",
                shape = "diamond",
                orientation = "90",
                portnumber = "8",
                inports_types = "",
                inports_names = "",
                outports_types = "Booleen",
                outports_names = "OUT1",
                color = "black",
                fontcolor = "black"
        ];
        "BLOC" [
                system = "N421401",
                type = "Inport",
                type_op = "Inport",
                Hidden = "false",
```

```
            Supplementaire = "false",
            refblock = "",
            refblocklib = "",
            path = "N421401/BLOC",
            inports = "0",
            outports = "1",
            shape = "diamond",
            orientation = "90",
            portnumber = "9",
            inports_types = "",
            inports_names = "",
            outports_types = "Booleen",
            outports_names = "OUT1",
            color = "black",
            fontcolor = "black"
        ];
        "BLNDTRK" [
            system = "N421401",
            type = "Inport",
            type_op = "Inport",
            Hidden = "false",
            Supplementaire = "false",
            refblock = "",
            refblocklib = "",
            path = "N421401/BLNDTRK",
            inports = "0",
            outports = "1",
            shape = "diamond",
            orientation = "90",
            portnumber = "10",
            inports_types = "",
            inports_names = "",
            outports_types = "Booleen",
            outports_names = "OUT1",
            color = "black",
            fontcolor = "black"
        ];
        "B1LATMODACT" [
            system = "N421401",
            type = "Inport",
            type_op = "Inport",
            Hidden = "false",
            Supplementaire = "false",
            refblock = "",
            refblocklib = "",
            path = "N421401/B1LATMODACT",
            inports = "0",
            outports = "1",
            shape = "diamond",
            orientation = "90",
            portnumber = "11",
            inports_types = "",
            inports_names = "",
            outports_types = "Booleen",
```

```
                outports_names = "OUT1",
                color = "black",
                fontcolor = "black"
        ];
        "BRWY" [
                system = "N421401",
                type = "Inport",
                type_op = "Inport",
                Hidden = "false",
                Supplementaire = "false",
                refblock = "",
                refblocklib = "",
                path = "N421401/BRWY",
                inports = "0",
                outports = "1",
                shape = "diamond",
                orientation = "90",
                portnumber = "12",
                inports_types = "",
                inports_names = "",
                outports_types = "Booleen",
                outports_names = "OUT1",
                color = "black",
                fontcolor = "black"
        ];
        "CONF1_1" [
                system = "N421401",
                type = "CONF1",
                type_op = "CONF1",
                refblock = "",
                refblocklib = "",
                path = "N421401/CONF1_1",
                inports = "4",
                outports = "1",
                shape = "box",
                orientation = "90",
                inports_types = "Booleen Entier Entier Booleen",
                inports_names = "IN1 IN2 IN3 IN4",
                outports_types = "Booleen",
                outports_names = "OUT1",
                color = "black",
                fontcolor = "black"
        ];
        "CST_1_1" [
                system = "N421401",
                type = "Constant",
                type_op = "Constant",
                refblock = "",
                refblocklib = "",
                path = "N421401/CST_1_1",
                inports = "0",
                outports = "1",
                Constant_Value = "1",
                shape = "diamond",
```

```
        orientation = "90",
        inports_types = "",
        inports_names = "",
        outports_types = "Entier",
        outports_names = "OUT1",
        color = "black",
        fontcolor = "black"
    ];
    "CST_1_2" [
        system = "N421401",
        type = "Constant",
        type_op = "Constant",
        refblock = "",
        refblocklib = "",
        path = "N421401/CST_1_2",
        inports = "0",
        outports = "1",
        Constant_Value = "1",
        shape = "diamond",
        orientation = "90",
        inports_types = "",
        inports_names = "",
        outports_types = "Entier",
        outports_names = "OUT1",
        color = "black",
        fontcolor = "black"
    ];
    "CST_false_1" [
        system = "N421401",
        type = "Constant",
        type_op = "Constant",
        refblock = "",
        refblocklib = "",
        path = "N421401/CST_false_1",
        inports = "0",
        outports = "1",
        Constant_Value = "false",
        shape = "diamond",
        orientation = "90",
        inports_types = "",
        inports_names = "",
        outports_types = "Booleen",
        outports_names = "OUT1",
        color = "black",
        fontcolor = "black"
    ];
    "AND_1" [
        system = "N421401",
        type = "AND",
        type_op = "AND",
        refblock = "",
        refblocklib = "",
        path = "N421401/AND_1",
        inports = "3",
```

```
        outports = "1",
        shape = "",
        orientation = "90",
        inports_types = "Booleen Booleen Booleen",
        inports_names = "IN1 IN2 IN3",
        outports_types = "Booleen",
        outports_names = "OUT1",
        color = "black",
        fontcolor = "black"
    ];
    "NOT_1" [
        system = "N421401",
        type = "NOT",
        type_op = "NOT",
        refblock = "",
        refblocklib = "",
        path = "N421401/NOT_1",
        inports = "1",
        outports = "1",
        shape = "",
        orientation = "90",
        inports_types = "Booleen",
        inports_names = "IN1",
        outports_types = "Booleen",
        outports_names = "OUT1",
        color = "black",
        fontcolor = "black"
    ];
    "SWITCH_B_1" [
        system = "N421401",
        type = "SWITCH_B",
        type_op = "SWITCH_B",
        refblock = "",
        refblocklib = "",
        path = "N421401/SWITCH_B_1",
        inports = "3",
        outports = "1",
        shape = "box",
        orientation = "90",
        inports_types = "Booleen Booleen Booleen",
        inports_names = "IN1 IN2 IN3",
        outports_types = "Booleen",
        outports_names = "OUT1",
        color = "black",
        fontcolor = "black"
    ];
    "MRTRIG1_1" [
        system = "N421401",
        type = "MRTRIG1",
        type_op = "MRTRIG1",
        refblock = "",
        refblocklib = "",
        path = "N421401/MRTRIG1_1",
        inports = "4",
```

```
        outports = "1",
        shape = "box",
        orientation = "90",
        inports_types = "Booleen Entier Entier Booleen",
        inports_names = "IN1 IN2 IN3 IN4",
        outports_types = "Booleen",
        outports_names = "OUT1",
        color = "black",
        fontcolor = "black"
    ];
    "CST_17_1" [
        system = "N421401",
        type = "Constant",
        type_op = "Constant",
        refblock = "",
        refblocklib = "",
        path = "N421401/CST_17_1",
        inports = "0",
        outports = "1",
        Constant_Value = "17",
        shape = "diamond",
        orientation = "90",
        inports_types = "",
        inports_names = "",
        outports_types = "Entier",
        outports_names = "OUT1",
        color = "black",
        fontcolor = "black"
    ];
    "CST_17_2" [
        system = "N421401",
        type = "Constant",
        type_op = "Constant",
        refblock = "",
        refblocklib = "",
        path = "N421401/CST_17_2",
        inports = "0",
        outports = "1",
        Constant_Value = "17",
        shape = "diamond",
        orientation = "90",
        inports_types = "",
        inports_names = "",
        outports_types = "Entier",
        outports_names = "OUT1",
        color = "black",
        fontcolor = "black"
    ];
    "CST_false_2" [
        system = "N421401",
        type = "Constant",
        type_op = "Constant",
        refblock = "",
        refblocklib = "",
```

```
        path = "N421401/CST_false_2",
        inports = "0",
        outports = "1",
        Constant_Value = "false",
        shape = "diamond",
        orientation = "90",
        inports_types = "",
        inports_names = "",
        outports_types = "Booleen",
        outports_names = "OUT1",
        color = "black",
        fontcolor = "black"
    ];
    "NOT_2" [
        system = "N421401",
        type = "NOT",
        type_op = "NOT",
        refblock = "",
        refblocklib = "",
        path = "N421401/NOT_2",
        inports = "1",
        outports = "1",
        shape = "",
        orientation = "90",
        inports_types = "Booleen",
        inports_names = "IN1",
        outports_types = "Booleen",
        outports_names = "OUT1",
        color = "black",
        fontcolor = "black"
    ];
    "AND_2" [
        system = "N421401",
        type = "AND",
        type_op = "AND",
        refblock = "",
        refblocklib = "",
        path = "N421401/AND_2",
        inports = "2",
        outports = "1",
        shape = "",
        orientation = "90",
        inports_types = "Booleen Booleen",
        inports_names = "IN1 IN2",
        outports_types = "Booleen",
        outports_names = "OUT1",
        color = "black",
        fontcolor = "black"
    ];
    "OR_1" [
        system = "N421401",
        type = "OR",
        type_op = "OR",
        refblock = "",
```

```
            refblocklib = "",
            path = "N421401/OR_1",
            inports = "7",
            outports = "1",
            shape = "",
            orientation = "90",
            inports_types = "Booleen Booleen Booleen Booleen Booleen Booleen Booleen",
            inports_names = "IN1 IN2 IN3 IN4 IN5 IN6 IN7",
            outports_types = "Booleen",
            outports_names = "OUT1",
            color = "black",
            fontcolor = "black"
        ];
        "NOT_3" [
            system = "N421401",
            type = "NOT",
            type_op = "NOT",
            refblock = "",
            refblocklib = "",
            path = "N421401/NOT_3",
            inports = "1",
            outports = "1",
            shape = "",
            orientation = "90",
            inports_types = "Booleen",
            inports_names = "IN1",
            outports_types = "Booleen",
            outports_names = "OUT1",
            color = "black",
            fontcolor = "black"
        ];
        "VAR_Isolee_BGATRKTRIG1" [
            system = "N421401",
            type = "DataStoreWrite",
            type_op = "DataStoreWrite",
            refblock = "",
            refblocklib = "",
            path = "N421401/VAR_Isolee_BGATRKTRIG1",
            inports = "1",
            outports = "0",
            shape = "diamond",
            orientation = "90",
            inports_types = "pqb",
            inports_names = "IN1",
            outports_types = "",
            outports_names = "",
            color = "black",
            fontcolor = "black"
        ];
        "BSGATRKCND" [
            system = "N421401",
            type = "Outport",
            type_op = "Outport",
            Supplementaire = "false",
```

```
                refblock = "",
                refblocklib = "",
                path = "N421401/BSGATRKCND",
                inports = "1",
                outports = "0",
                shape = "diamond",
                orientation = "90",
                portnumber = "1",
                inports_types = "Booleen",
                inports_names = "IN1",
                outports_types = "",
                outports_names = "",
                color = "black",
                fontcolor = "black"
        ];
        "BGATRKENGT1" [
                system = "N421401",
                type = "Outport",
                type_op = "Outport",
                Supplementaire = "false",
                refblock = "",
                refblocklib = "",
                path = "N421401/BGATRKENGT1",
                inports = "1",
                outports = "0",
                shape = "diamond",
                orientation = "90",
                portnumber = "2",
                inports_types = "Booleen",
                inports_names = "IN1",
                outports_types = "",
                outports_names = "",
                color = "black",
                fontcolor = "black"
        ];
        "BGATRKCND" [
                system = "N421401",
                type = "Outport",
                type_op = "Outport",
                Supplementaire = "false",
                refblock = "",
                refblocklib = "",
                path = "N421401/BGATRKCND",
                inports = "1",
                outports = "0",
                shape = "diamond",
                orientation = "90",
                portnumber = "3",
                inports_types = "Booleen",
                inports_names = "IN1",
                outports_types = "",
                outports_names = "",
                color = "black",
                fontcolor = "black"
```

```
];
"B2TRAGARQTAB" -> "CONF1_1" [
        signaltype = "Booleen",
        srcport = "1",
        dstport = "1",
        label = "P421401Z6",
];
"CST_1_1" -> "CONF1_1" [
        signaltype = "Entier",
        srcport = "1",
        dstport = "2",
        label = "",
];
"CST_1_2" -> "CONF1_1" [
        signaltype = "Entier",
        srcport = "1",
        dstport = "3",
        label = "",
];
"CST_false_1" -> "CONF1_1" [
        signaltype = "Booleen",
        srcport = "1",
        dstport = "4",
        label = "",
];
"NOT_2" -> "AND_1" [
        signaltype = "Booleen",
        srcport = "1",
        dstport = "1",
        label = "P421401Z9",
];
"BHYPER" -> "AND_1" [
        signaltype = "Booleen",
        srcport = "1",
        dstport = "2",
        label = "P421401ZA",
];
"NOT_1" -> "AND_1" [
        signaltype = "Booleen",
        srcport = "1",
        dstport = "3",
        label = "P421401ZC",
];
"BPITTO" -> "NOT_1" [
        signaltype = "Booleen",
        srcport = "1",
        dstport = "1",
        label = "P421401ZB",
];
"CONF1_1" -> "SWITCH_B_1" [
        signaltype = "Booleen",
        srcport = "1",
        dstport = "1",
        label = "P421401ZO",
```

```
];
"B2TRAGARQTAB" -> "SWITCH_B_1" [
        signaltype = "Booleen",
        srcport = "1",
        dstport = "2",
        label = "P421401Z6",
];
"BLNDROLLOUT" -> "SWITCH_B_1" [
        signaltype = "Booleen",
        srcport = "1",
        dstport = "3",
        label = "P421401ZQ",
];
"SWITCH_B_1" -> "MRTRIG1_1" [
        signaltype = "Booleen",
        srcport = "1",
        dstport = "1",
        label = "P421401ZP",
];
"CST_17_1" -> "MRTRIG1_1" [
        signaltype = "Entier",
        srcport = "1",
        dstport = "2",
        label = "",
];
"CST_17_2" -> "MRTRIG1_1" [
        signaltype = "Entier",
        srcport = "1",
        dstport = "3",
        label = "",
];
"CST_false_2" -> "MRTRIG1_1" [
        signaltype = "Booleen",
        srcport = "1",
        dstport = "4",
        label = "",
];
"BMLGONGND30" -> "NOT_2" [
        signaltype = "Booleen",
        srcport = "1",
        dstport = "1",
        label = "P421401Z8",
];
"AND_2" -> "BSGATRKCND" [
        signaltype = "Booleen",
        srcport = "1",
        dstport = "1",
        label = "P421401ZR",
];
"AND_2" -> "BGATRKENGT1" [
        signaltype = "Booleen",
        srcport = "1",
        dstport = "1",
        label = "P421401ZR",
```

```
];
"OR_1" -> "AND_2" [
        signaltype = "Booleen",
        srcport = "1",
        dstport = "1",
        label = "P421401ZS",
];
"MRTRIG1_1" -> "AND_2" [
        signaltype = "Booleen",
        srcport = "1",
        dstport = "2",
        label = "P421401Z7",
];
"NOT_3" -> "OR_1" [
        signaltype = "Booleen",
        srcport = "1",
        dstport = "1",
        label = "P421401ZZ",
];
"BRWY" -> "OR_1" [
        signaltype = "Booleen",
        srcport = "1",
        dstport = "2",
        label = "P421401Z10",
];
"BHDGTRK" -> "OR_1" [
        signaltype = "Booleen",
        srcport = "1",
        dstport = "3",
        label = "P421401ZT",
];
"BNAV" -> "OR_1" [
        signaltype = "Booleen",
        srcport = "1",
        dstport = "4",
        label = "P421401ZU",
];
"BFLOC" -> "OR_1" [
        signaltype = "Booleen",
        srcport = "1",
        dstport = "5",
        label = "P421401ZV",
];
"BLOC" -> "OR_1" [
        signaltype = "Booleen",
        srcport = "1",
        dstport = "6",
        label = "P421401ZW",
];
"BLNDTRK" -> "OR_1" [
        signaltype = "Booleen",
        srcport = "1",
        dstport = "7",
        label = "P421401ZX",
```

```
        ];
        "B1LATMODACT" -> "NOT_3" [
                signaltype = "Booleen",
                srcport = "1",
                dstport = "1",
                label = "P421401ZY",
        ];
        "AND_1" -> "BGATRKCND" [
                signaltype = "Booleen",
                srcport = "1",
                dstport = "1",
                label = "P421401ZI",
        ];
        "MRTRIG1_1" -> "VAR_Isolee_BGATRKTRIG1" [
                signaltype = "pqb",
                srcport = "1",
                dstport = "1",
                label = "P421401Z7",
        ];
}
```

Annexe D

Résultats détaillées de l'application des approches d'analyse de testabilité

Dans cette section, nous présentons les résultats détaillés de l'analyse de testabilité des systèmes LM1 et LM2 en utilisant les différentes approches (par variable de sortie, par fonction et par composant) définies dans la méthodologie d'aide à la validation des systèmes.

Pour la représentation des résultats ci-dessous, les expressions suivantes seront utilisées :

- **Nb INIT** : Nombre d'écoulements déterminés pour la phase d'initialisation ;
- **Nb INIT STM** : Nombre d'écoulements d'initialisation choisis par Start Small ;
- **Nb NOMI** : Nombre d'écoulements déterminés pour la phase nominale ;
- **Nb NOMI STM** : Nombre d'écoulements de la phase nominale choisis par Start Small.

D.1 Résultats de l'analyse à l'aide de l'approche par variable

Les résultats de l'analyse des systèmes LM1 et LM2 sont présentés respectivement dans les tableaux ci-dessous (**Tableau D.1** et **Tableau D.2**).

Tableau D.1 : Approche par variable – résultats détaillés de l'analyse du système LM1

Liste des sorties	Nb INIT	Nb INIT STM	Nb NOMI	Nb NOMI STM
BFLOC	1	1	240	11
BFLOCCPT	1	1	240	11
BFLOCCPTENGT1	2	2	256	12
BFLOCTRK	1	1	240	11
BFLOCTRKENGT1	2	2	256	12
BGATRK	1	1	240	11

BGATRKENGT1	2	2	256	11
BHDG	1	1	240	11
BHDGTRK	1	1	240	11
BHDGTRKENGT1	1	1	256	11
BHDGTRKENGT2	1	1	240	11
BHDGTRKENGT3	1	1	240	11
BHDGTRKENGT4	1	1	240	11
BHDGTRKENGT5	1	1	240	11
BHDGTRKENGT6	1	1	240	11
BHDGTRKENGT7	1	1	240	11
BHDGTRKENGT8	2	2	240	11
BHDGTRKENGT9	1	1	240	11
BHDGTRKENGT10	1	1	240	11
BHDGTRKENGT11	1	1	240	11
BLNDTRK	1	1	156	11
BLOC	1	1	240	11
BLOCCPT	1	1	240	11
BLOCCPTENGT1	2	2	256	12
BLOCTRK	1	1	240	11
BLOCTRKENGT1	2	2	256	11
BNAV	1	1	240	11
BNAVCND	0	0	2	2
BNAVENGT1	1	1	240	11
BNAVENGT2	1	1	240	11
BNAVHDG	3	3	240	11
BNAVHPATH	3	3	240	11
BNAVTRK	3	3	240	11
BNSFLOCCPT	76	10	240	11
BNSFLOCTRK	76	10	240	11
BNSHDG	56	9	224	11
BNSLNDTRK	5	4	216	11
BNSLOCCPT	18	6	240	11
BNSLOCTRK	10	5	218	11
BNSNAV	76	10	240	11
BNSRWY	3	3	204	11
BNSTRK	76	10	240	11
BRLATMOD	76	10	240	11
BRWY	1	1	240	11
BRWYCND1AB	0	0	2	2
BRWYCND2AB	0	0	1	1
BRWYENGT1	2	2	288	11
BRWYENGT2	1	1	240	11
BRWYLOC	5	4	240	11
BSHDG	4	4	240	11
BSLNDTRK	2	2	190	12

BSNAVHDG	3	3	240	11
BSNAVHPATH	3	3	240	11
BSNAVTRK	3	3	240	11
BSRWYLOC	3	3	240	11
BSRWYTRK	3	3	240	11
BSTRK	4	4	240	11
BTRK	1	1	240	11

Tableau D.2 : Approche par variable – résultats détaillés de l'analyse du système LM2

Liste des sorties	Nb INIT	Nb INIT STM	Nb NOMI	Nb NOMI STM
B1FMSUBRQT	0	0	1	1
B1LNGFMSUB	3	3	10	6
BALT	1	1	10	6
BALTACQ	1	1	10	6
BALTACQCND	0	0	1	1
BALTACQENGT1	1	1	10	6
BALTACQENGT2	1	1	10	6
BALTALT	1	1	10	6
BALTHOLDENGT1	1	1	10	6
BALTHOLDENGT2	1	1	10	6
BALTVS	1	1	10	6
BCLB	1	1	10	6
BCLBDESFPASPD	3	3	10	6
BCLBDESSPDTHR	3	3	10	6
BCLBDESVPASPD	3	3	10	6
BCLBDESVPATHR	3	3	10	6
BCLBDESVSSPD	3	3	10	6
BCLBENGT1	1	1	10	6
BCLBENGT2	1	1	10	6
BCLBENGT3	1	1	10	6
BDES	1	1	10	6
BDESCND	0	0	1	1
BDESENGT1	1	1	10	6
BDESENGT2	1	1	10	6
BDESENGT3	1	1	10	6
BFGS	1	1	10	6
BFGSCPT	1	1	10	6
BFGSCPTENGT1	2	2	12	7
BFGSTRK	1	1	10	6
BFGSTRKENGT1	2	2	12	7
BFPA	1	1	10	6
BFPAFPA	1	1	10	6
BGSCPTENGT1	2	2	12	7

BGSTRKENGT1	2	2	12	6
BNSALT	10	6	10	6
BNSALTACQ	10	6	10	6
BNSCLB	10	6	10	6
BNSCLBDESFPASPD	3	3	10	6
BNSCLBDESVPASPD	3	3	10	6
BNSCLBDESVPATHR	3	3	10	6
BNSCLBDESVSSPD	3	3	10	6
BNSDES	4	4	10	6
BNSFGSCPT	10	6	10	6
BNSFGSTRK	10	6	10	6
BNSFPA	10	6	10	6
BNSGSCPT	3	3	7	6
BNSGSTRK	2	2	10	6
BNSOPCLB	10	6	10	6
BNSOPDES	10	6	10	6
BNSPITTO	2	2	10	6
BNSVS	10	6	10	6
BOPCLB	1	1	10	6
BOPCLBENGT1	1	1	10	6
BOPCLBENGT2	1	1	10	6
BOPCLBENGT3	1	1	10	6
BOPCLBENGT4	1	1	10	6
BOPCLBENGT5	1	1	10	6
BOPCLBENGT6	1	1	10	6
BOPDES	1	1	10	6
BOPDESENGT1	1	1	10	6
BPITGA	1	1	10	6
BPITGAENGT1	1	1	10	6
BPITTO	1	1	10	6
BPITTOENGT1	1	1	10	6
BPITTOGA	1	1	10	6
BRLNGFMSUB	3	3	10	6
BRLNGMOD	10	6	10	6
BSALTALT	3	3	10	6
BSALTVS	3	3	10	6
BSCLBDESFPASPD	3	3	10	6
BSCLBDESSPDTHR	3	3	10	6
BSCLBDESVPASPD	3	3	10	6
BSCLBDESVPATHR	3	3	10	6
BSCLBDESVSSPD	3	3	10	6
BSFPA	3	3	10	6
BSFPAFPA	3	3	10	6
BSIMINITALTACQ	0	0	1	1
BSIMINITCLB	0	0	1	1

BSIMINITDES	0	0	1	1
BSIMINITFGSCPT	0	0	1	1
BSIMINITFGSTRK	0	0	1	1
BSIMINITFPA	0	0	1	1
BSIMINITGSCPT	0	0	1	1
BSIMINITGSTRK	0	0	1	1
BSIMINITOPCLB	0	0	1	1
BSIMINITOPDES	0	0	1	1
BSIMINITPITGA	0	0	1	1
BSIMINITPITTO	0	0	1	1
BSIMINITVS	0	0	1	1
BSOPCLB	3	3	10	6
BSOPDES	3	3	10	6
BSPITGA	2	2	10	6
BSVS	3	3	10	6
BSVSFPAALT	3	3	10	6
BSVSVS	3	3	10	6
BVS	1	1	10	6
BVSFPA	1	1	10	6
BVSFPAALT	1	1	10	6
BVSFPAENGT7	1	1	10	6
BVSFPAENGT8	1	1	10	6
BVSFPAENGT9	1	1	10	6
BVSFPAENGT10	1	1	10	6
BVSFPAENGT11	1	1	10	6
BVSFPAENGT12	1	1	10	6
BVSFPAENGT13	1	1	10	6
BVSFPAENGT14	1	1	10	6
BVSVS	1	1	10	6

D.2 Résultats de l'analyse à l'aide l'approche par fonction

Les résultats de l'analyse des systèmes LM1 et LM2 sont présentés respectivement dans les tableaux ci-dessous (**Tableau D.3** et **Tableau D.4**).

Tableau D.3 : Approche par fonction – résultats détaillés de l'analyse du système LM1

Liste des sorties	Nb NOMI	Nb NOMI STM	Nb INIT	Nb INIT STM
Premier ensemble de variable (première fonction)				
BNSTRK	1	1	0	0
BNSRWY	1	1	0	0
BNSHDG	1	1	0	0

BNSLOCTRK	1	1	0	0
BNAV	1	1	1	1
BNSLOCCPT	1	1	0	0
BTRK	1	1	1	1
BNSLNDTRK	1	1	0	0
BLOC	1	1	1	1
BHDG	1	1	1	1
BFLOC	240	1	1	1
BRWY	2	2	1	1
BLNDTRK	2	2	1	1
BGATRK	1	1	1	1
BLOCTRK	1	1	1	1
BLOCCPT	1	1	1	1
BFLOCTRK	240	8	1	1
BFLOCCPT	240	1	1	1
BRLATMOD	240	1	76	9
BHDGTRK	1	1	1	1
BNSNAV	1	1	0	0
BNSFLOCCPT	18	4	6	4
BNSFLOCTRK	10	4	4	4
Deuxième ensemble de variable (deuxième fonction)				
BNAVCND	2	2	0	0
BNAVENGT1	240	10	1	1
BNAVENGT2	1	1	1	1
Troisième ensemble de variable (troisième fonction)				
BSNAVHPATH	98	2	3	3
BNAVHPATH	98	8	3	3
BSNAVTRK	98	1	3	3
BSNAVHDG	98	1	3	1
BNAVTRK	98	3	3	1
BNAVHDG	240	6	3	1
Quatrième ensemble de variable (quatrième fonction)				
BSTRK	42	9	4	3
BSHDG	4	4	4	3
BHDGTRKENGT11	1	1	1	1
BHDGTRKENGT9	1	1	1	1
BHDGTRKENGT7	1	1	1	1
BHDGTRKENGT8	2	1	2	2
BHDGTRKENGT10	1	1	1	1
BHDGTRKENGT6	1	1	1	1
BHDGTRKENGT5	1	1	1	1
BHDGTRKENGT4	1	1	1	1
BHDGTRKENGT3	1	1	1	1
BHDGTRKENGT1	256	5	1	1
BHDGTRKENGT2	1	1	1	1

Cinquième ensemble de variable (cinquième fonction)				
BFLOCCPTENGT1	256	12	2	2
Sixième ensemble de variable (sixième fonction)				
BFLOCTRKENGT1	256	12	2	2
Septième ensemble de variable (septième fonction)				
BLOCCPTENGT1	256	12	2	2
Huitième ensemble de variable (huitième fonction)				
BLOCTRKENGT1	256	11	2	2
Neuvième ensemble de variable (neuvième fonction)				
BGATRKENGT1	256	11	2	2
Dixième ensemble de variable (dixième fonction)				
BSLNDTRK	190	12	2	2
Onzième ensemble de variable (onzième fonction)				
BRWYCND1AB	2	2	0	0
BRWYENGT1	288	10	2	2
BRWYENGT2	1	1	0	0
BRWYCND2AB	1	1	0	0
Douzième ensemble de variable (douzième fonction)				
BRWYLOC	240	10	5	3
BSRWYLOC	193	1	3	3
BSRWYTRK	144	3	3	1

Tableau D.4 : Approche par fonction – résultats détaillés de l'analyse du système LM2

Liste des sorties	Nb NOMI	Nb NOMI STM	Nb INIT	Nb INIT STM
Premier ensemble de variable (première fonction)				
BPITTOGA	1	1	1	1
BNSPITTO	1	1	0	0
BNSDES	1	1	0	0
BNSFGSCPT	1	1	0	0
BFPA	1	1	1	1
BNSCLB	1	1	0	0
BVS	1	1	1	1
BNSFGSTRK	1	1	0	0
BNSOPDES	1	1	0	0
BALTACQ	1	1	1	1
BNSOPCLB	1	1	0	0
BALT	10	6	1	1
BNSALTACQ	1	1	0	0
BFGSTRK	1	1	1	1
BNSALT	1	1	0	0
BFGSCPT	1	1	1	1
BNSFPA	1	1	0	0
BPITGA	1	1	1	1

BPITTO	1	1	1	1
BDES	1	1	1	1
BNSVS	1	1	0	0
BNSGSCPT	1	1	0	0
BCLB	1	1	1	1
BNSGSTRK	1	1	0	0
BOPDES	1	1	1	1
BOPCLB	1	1	1	1
BRLNGMOD	10	1	10	6
BVSFPA	1	1	1	1
BFGS	1	1	1	1
Deuxième ensemble de variable (deuxième fonction)				
BPITTOENGT1	10	6	1	1
Troisième ensemble de variable (troisième fonction)				
BVSFPAENGT10	3	1	1	1
BVSFPAENGT9	1	1	0	0
BVSFPAENGT7	3	1	1	1
BSFPA	10	6	3	3
BVSFPAENGT13	3	1	1	1
BSVS	3	3	3	3
BVSFPAENGT14	3	1	1	1
BVSFPAENGT12	3	1	1	1
BVSFPAENGT11	3	1	1	1
BVSFPAENGT8	1	1	0	0
Quatrième ensemble de variable (quatrième fonction)				
BSVSFPAALT	6	3	3	3
BVSFPAALT	3	3	1	1
BSFPAFPA	6	2	3	3
BSVSVS	3	1	3	3
BFPAFPA	10	6	1	1
BVSVS	1	1	1	1
Cinquième ensemble de variable (cinquième fonction)				
BOPCLBENGT6	1	1	0	0
BOPCLBENGT5	1	1	0	0
BOPCLBENGT4	1	1	0	0
BOPCLBENGT2	1	1	0	0
BOPCLBENGT3	1	1	0	0
BOPCLBENGT1	10	6	1	1
Sixième ensemble de variable (sixième fonction)				
BOPDESENGT1	10	6	1	1
Septième ensemble de variable (septième fonction)				
BCLBENGT1	10	6	1	1
BCLBCND	1	1	0	0
BCLBENGT2	1	1	1	1
BCLBENGT3	3	3	1	1

Huitième ensemble de variable (huitième fonction)				
BDESENGT1	10	6	1	1
BDESCND	1	1	0	0
BDESENGT2	1	1	1	1
BDESENGT3	1	1	0	0
Neuvième ensemble de variable (neuvième fonction)				
BRLNGFMSUB	10	1	3	1
BNSCLBDESVSSPD	6	4	3	3
BCLBDESVPATHR	10	5	3	3
BCLBDESVPASPD	10	1	3	1
BCLBDESFPASPD	10	1	3	1
BCLBDESVSSPD	10	1	3	1
BCLBDESSPDTHR	10	1	3	1
B1LNGFMSUB	10	1	3	1
B1FMSUBRQT	1	1	0	0
BSCLBDESVPATHR	6	3	3	3
BSCLBDESVPASPD	6	3	3	3
BSCLBDESSPDTHR	6	1	3	1
BSCLBDESFPASPD	6	3	3	3
BSCLBDESVSSPD	6	2	3	1
BNSCLBDESVPATHR	6	1	3	1
BNSCLBDESVPASPD	6	1	3	1
BNSCLBDESFPASPD	6	1	3	1
dixième ensemble de variable (dixième fonction)				
BALTACQENGT2	1	1	1	1
BALTACQCND	1	1	0	0
BALTACQENGT1	10	6	1	1
Onzième ensemble de variable (onzième fonction)				
BALTHOLDENGT2	1	1	0	0
BALTHOLDENGT1	10	6	1	1
Douzième ensemble de variable (douzième fonction)				
BSALTVS	10	1	3	3
BALTALT	10	3	1	1
BALTVS	10	6	1	1
BSALTALT	6	3	3	3
Treizième ensemble de variable (treizième fonction)				
BFGSCPTENGT1	12	7	2	2
Quatorzième ensemble de variable (quatorzième fonction)				
BFGSTRKENGT1	12	7	2	2
Quinzième ensemble de variable (quinzième fonction)				
BGSCPTENGT1	12	7	2	2
Seizième ensemble de variable (seizième fonction)				
BGSTRKENGT1	12	6	2	2
Dix-septième ensemble de variable (dix-septième fonction)				
BPITGAENGT1	10	6	1	1

D.3 Résultats de l'analyse à l'aide l'approche par composant

Les résultats de l'analyse des systèmes LM1 et LM2 sont présentés respectivement dans les tableaux ci-dessous (**Tableau D.5** et **Tableau D.6**).

Tableau D.5 : Approche par composant – résultats détaillés de l'analyse du système LM2

Liste des sorties	Nb NOMI	Nb NOMI STM	Nb INIT	Nb INIT STM
Première composante connexe				
BRLATMOD	240	1	76	4
BNSRWY	1	1	0	0
BNSLNDTRK	1	1	0	0
BNSLOCTRK	1	1	0	0
BNSLOCCPT	1	1	0	0
BNSHDG	1	1	0	0
BNSTRK	1	1	0	0
BNSFLOCTRK	10	1	4	1
BNSFLOCCPT	18	3	6	3
BNSNAV	1	1	0	0
BHDG	1	1	1	1
BHDGTRK	1	1	1	1
BTRK	1	1	1	1
BNAV	1	1	1	1
BFLOCCPT	240	1	1	1
BFLOC	240	1	1	1
BFLOCTRK	240	4	1	1
BLOCCPT	1	1	1	1
BLOC	1	1	1	1
BLOCTRK	1	1	1	1
BGATRK	1	1	1	1
BLNDTRK	2	2	1	1
BRWY	2	1	1	1
BNAVCND	2	2	0	0
BNAVENGT1	2	1	1	1
BNAVENGT2	2	1	1	1
BNAVHDG	3	3	3	1
BNAVTRK	3	3	3	1
BNAVHPATH	3	3	3	3
BSNAVHDG	3	1	3	1
BSNAVTRK	3	1	3	3
BSNAVHPATH	3	1	3	3
BSHDG	10	3	4	3
BSTRK	10	3	4	3
BHDGTRKENGT1	8	1	1	1
BHDGTRKENGT2	2	1	1	1

BHDGTRKENGT3	1	1	1	1
BHDGTRKENGT4	1	1	1	1
BHDGTRKENGT5	1	1	1	1
BHDGTRKENGT6	1	1	1	1
BHDGTRKENGT7	1	1	1	1
BHDGTRKENGT8	2	1	2	1
BHDGTRKENGT9	1	1	1	1
BHDGTRKENGT10	1	1	1	1
BHDGTRKENGT11	1	1	1	1
BFLOCCPTENGT1	2	2	2	2
BFLOCTRKENGT1	2	2	2	2
BLOCCPTENGT1	2	2	2	2
BLOCTRKENGT1	2	2	2	2
BGATRKENGT1	8	2	2	2
BSLNDTRK	4	2	2	2
BRWYCND1AB	2	2	0	0
BRWYCND2AB	1	1	0	0
BRWYENGT1	8	1	2	2
BRWYENGT2	4	1	1	1
BRWYLOC	10	2	5	3
BSRWYLOC	9	2	3	2
BSRWYTRK	6	1	3	1

Tableau D.6 : Approche par composant – résultats détaillés de l'analyse du système LM2

Liste des sorties	Nb NOMI	Nb NOMI STM	Nb INIT	Nb INIT STM
Première composante connexe				
BRLNGMOD	10	1	10	4
BNSPITTO	1	1	0	0
BNSGSTRK	1	1	0	0
BNSGSCPT	1	1	0	0
BNSVS	1	1	0	0
BNSFPA	1	1	0	0
BNSALT	1	1	0	0
BNSALTACQ	1	1	0	0
BNSOPCLB	1	1	0	0
BNSOPDES	1	1	0	0
BNSFGSTRK	1	1	0	0
BNSFGSCPT	1	1	0	0
BNSCLB	4	3	4	3
BNSDES	4	3	4	3
BALT	1	1	1	1
BALTACQ	1	1	1	1
BVS	1	1	1	1

BVSFPA	1	1	1	1
BFPA	1	1	1	1
BOPCLB	1	1	1	1
BOPDES	1	1	1	1
BCLB	10	1	1	1
BDES	10	4	1	1
BPITTO	1	1	1	1
BPITTOGA	1	1	1	1
BPITGA	1	1	1	1
BFGSCPT	1	1	1	1
BFGS	1	1	1	1
BFGSTRK	1	1	1	1
BPITTOENGT1	1	1	1	1
BSVS	3	3	3	3
BSFPA	3	3	3	3
BVSFPAENGT7	1	1	1	1
BVSFPAENGT8	1	1	1	1
BVSFPAENGT9	1	1	1	1
BVSFPAENGT10	1	1	1	1
BVSFPAENGT11	1	1	1	1
BVSFPAENGT12	1	1	1	1
BVSFPAENGT13	1	1	1	1
BVSFPAENGT14	1	1	1	1
BVSVS	1	1	1	1
BFPAFPA	3	3	1	1
BVSFPAALT	3	3	1	1
BSVSVS	3	1	3	3
BSFPAFPA	3	1	3	3
BSVSFPAALT	3	3	3	3
BOPCLBENGT1	1	1	1	1
BOPCLBENGT2	1	1	1	1
BOPCLBENGT3	1	1	1	1
BOPCLBENGT4	1	1	1	1
BOPCLBENGT5	1	1	1	1
BOPCLBENGT6	1	1	1	1
BOPDESENGT1	1	1	1	1
BCLBCND	1	1	0	0
BCLBENGT1	1	1	1	1
BCLBENGT2	1	1	1	1
BCLBENGT3	1	1	1	1
BDESCND	1	1	0	0
BDESENGT1	1	1	1	1
BDESENGT2	1	1	1	1
BDESENGT3	1	1	1	1
BRLNGFMSUB	10	1	3	1

BNSCLBDESVSSPD	6	3	3	3
BNSCLBDESFPASPD	6	1	3	1
BNSCLBDESVPASPD	6	1	3	1
BNSCLBDESVPATHR	6	1	3	1
BCLBDESSPDTHR	10	1	3	1
BCLBDESVSSPD	10	1	3	1
BCLBDESFPASPD	10	1	3	1
BCLBDESVPASPD	10	1	3	1
BCLBDESVPATHR	10	1	3	3
B1FMSUBRQT	1	1	0	0
BSCLBDESSPDTHR	6	1	3	1
B1LNGFMSUB	10	1	3	1
BSCLBDESVSSPD	6	3	3	1
BSCLBDESFPASPD	6	3	3	3
BSCLBDESVPASPD	6	3	3	3
BSCLBDESVPATHR	6	3	3	3
BALTACQCND	1	1	0	0
BALTACQENGT1	1	1	1	1
BALTACQENGT2	1	1	1	1
BALTHOLDENGT1	1	1	1	1
BALTHOLDENGT2	1	1	1	1
BALTVS	1	1	1	1
BALTALT	3	3	1	1
BSALTVS	3	1	3	3
BSALTALT	3	3	3	3
BFGSCPTENGT1	2	2	2	2
BFGSTRKENGT1	2	2	2	2
BGSCPTENGT1	2	2	2	2
BGSTRKENGT1	2	2	2	2
BPITGAENGT1	1	1	1	1

Annexe E

Analyse fonctionnelle du prototype ADIS

Dans cette section, nous proposons une description les fonctions du prototype ADIS en basant sur :

- la méthode d'aide à l'amélioration des résultats de Multiple-Clue dans le contexte FAL ;
- la méthode de recoupement du résultat de l'application de Multiple-Clue sur un objectif de test.

E.1 Définition des notions et des termes spécifiques

Objectif de test

Un objectif de test est constitué d'un ensemble de tests portant sur un ensemble de ressources. A chaque objectif de test est associée une matrice de diagnostic qui indique, pour chaque test, les ressources impliquées. Un exemple matrice est présenté ci-dessous.

	Ressource 1	Ressource 2	Ressource 3	Ressource 4
Test 1	1	1	0	0
Test 2	0	1	0	1

La valeur « 1 » indique que la ressource est activée dans le test. La valeur « 0 » indique que la ressource n'est pas activée dans le test.

Verdict d'un test

L'exécution d'un test peut aboutir à deux verdicts possibles :

- « 1 » : Le verdict du test est un succès (résultat correct). Toutes les ressources qui ont été activées durant le test sont fonctionnelles. Les autres ressources sont suspectées.

– « 0 » : Le verdict du test est un échec (résultat incorrect). Une des ressources activée durant le test est non fonctionnel. De plus, on peut en déduire que toutes les ressources non activées sont fonctionnelles, car on part de l'hypothèse qu'il n'y a qu'une seule ressource défectueuse.

Etats d'une ressource

Une ressource peut être dans trois états différents :

– Etat « 0 » : La ressource est défectueuse.
– Etat « 1 » : La ressource est fonctionnelle.
– Etat « ? » : La ressource fait partie d'un ensemble de ressources indiscernables suspecté.

Sauvegarde des résultats

Nous ferrons référence à 2 types de sauvegardes de résultats :

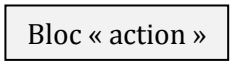
– Local : Chaque résultat obtenu uniquement à partir du diagnostic d'une seule sous matrice (d'un seul objectif de test) est appelé résultat local. En effet, ce résultat ne dépend pas des résultats des autres sous matrices.
– Global : L'état de chaque équipement à partir de l'ensemble des sous matrices est appelé résultat (état) global.

E.2 Architecture du logiciel du prototype « ADIS »

Nous proposons ici d'autre part, une représentation graphique de l'architecture du prototype et d'autre part, une description des principaux traitements.

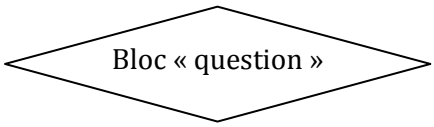
Représentation graphique de l'architecture

Les schémas ci-dessous présentent les différentes étapes du fonctionnement du prototype et la façon dont elles interagissent. Nous utiliserons les blocs ci-dessous pour cette représentation graphique.

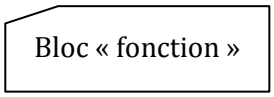


Le bloc « action » permet de décrire une action (non visible par l'utilisateur) à réaliser par le programme. Cela peut être le traitement ou la recherche d'une donnée.

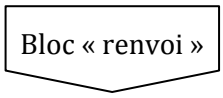
Le bloc « affichage » permet de spécifier les données à afficher. Par exemple, l'affichage d'un menu au début de l'exécution du programme.



Le bloc de question permet de définir les actions appropriées à partir de la réponse à une question.



Le bloc fonction fait référence à un ensemble d'étapes du processus global d'aide à la vérification. Lorsque le bloc fonction se trouve en bas à droit d'un carré entourant plusieurs étapes, alors ces étapes sont représentées par ce bloc.



Le bloc de renvoi indique un renvoi sur une des différentes interfaces. Chaque bloc « renvoi » est identifié par un nom.

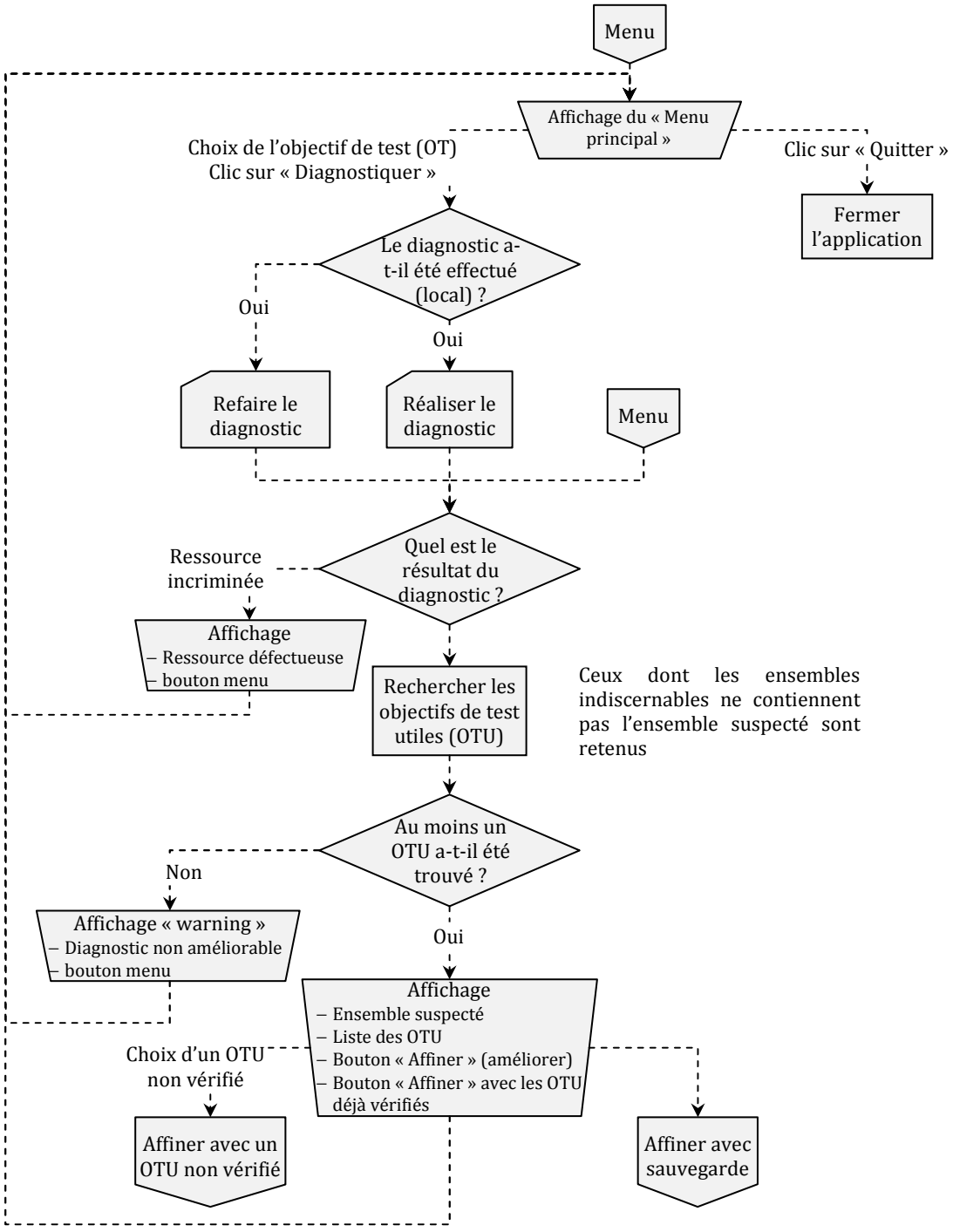


Figure E.1 : Processus global d'aide à l'aide des systèmes sur la FAL

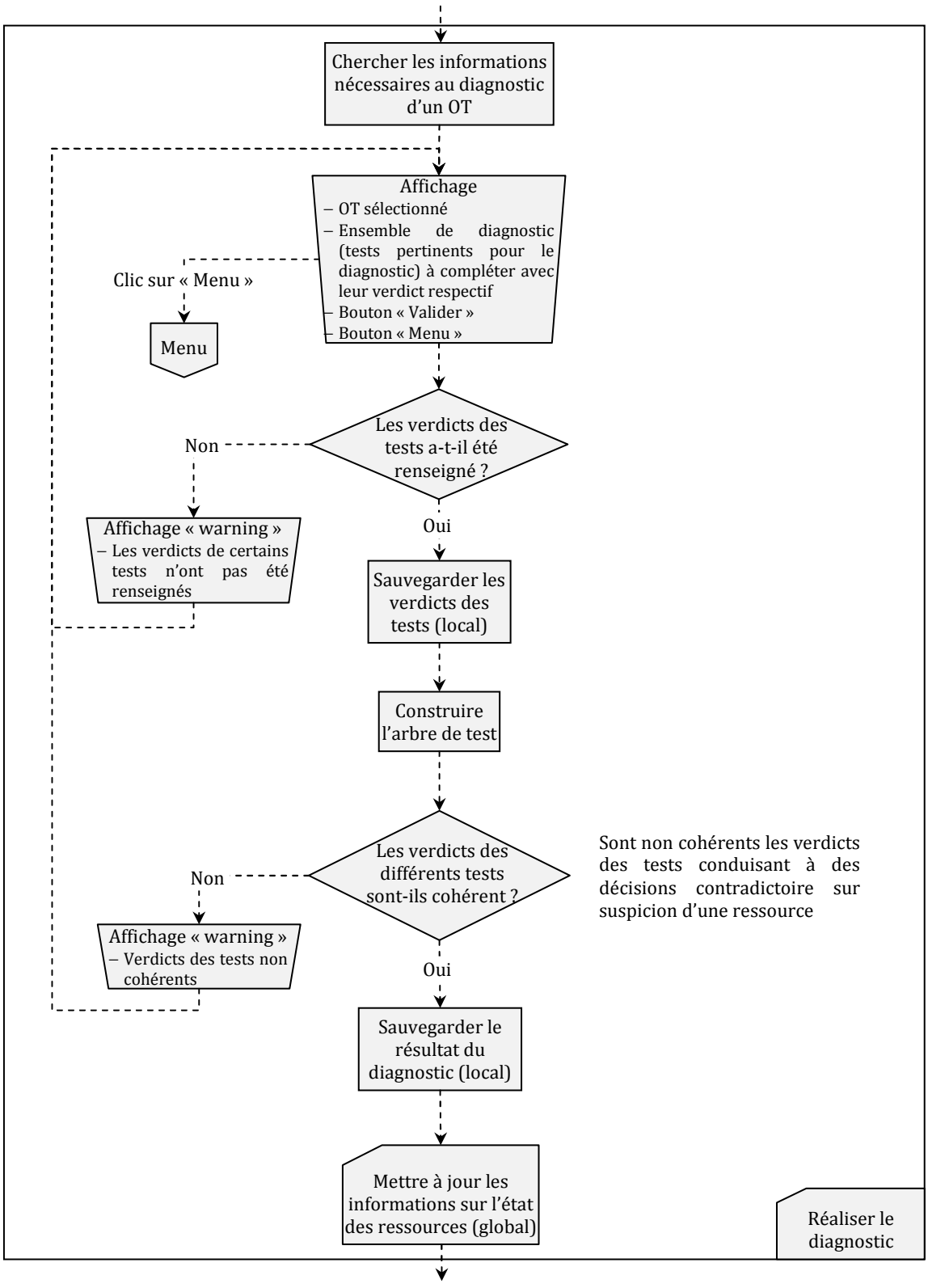


Figure E.2 : Processus de diagnostic d'un objectif de test

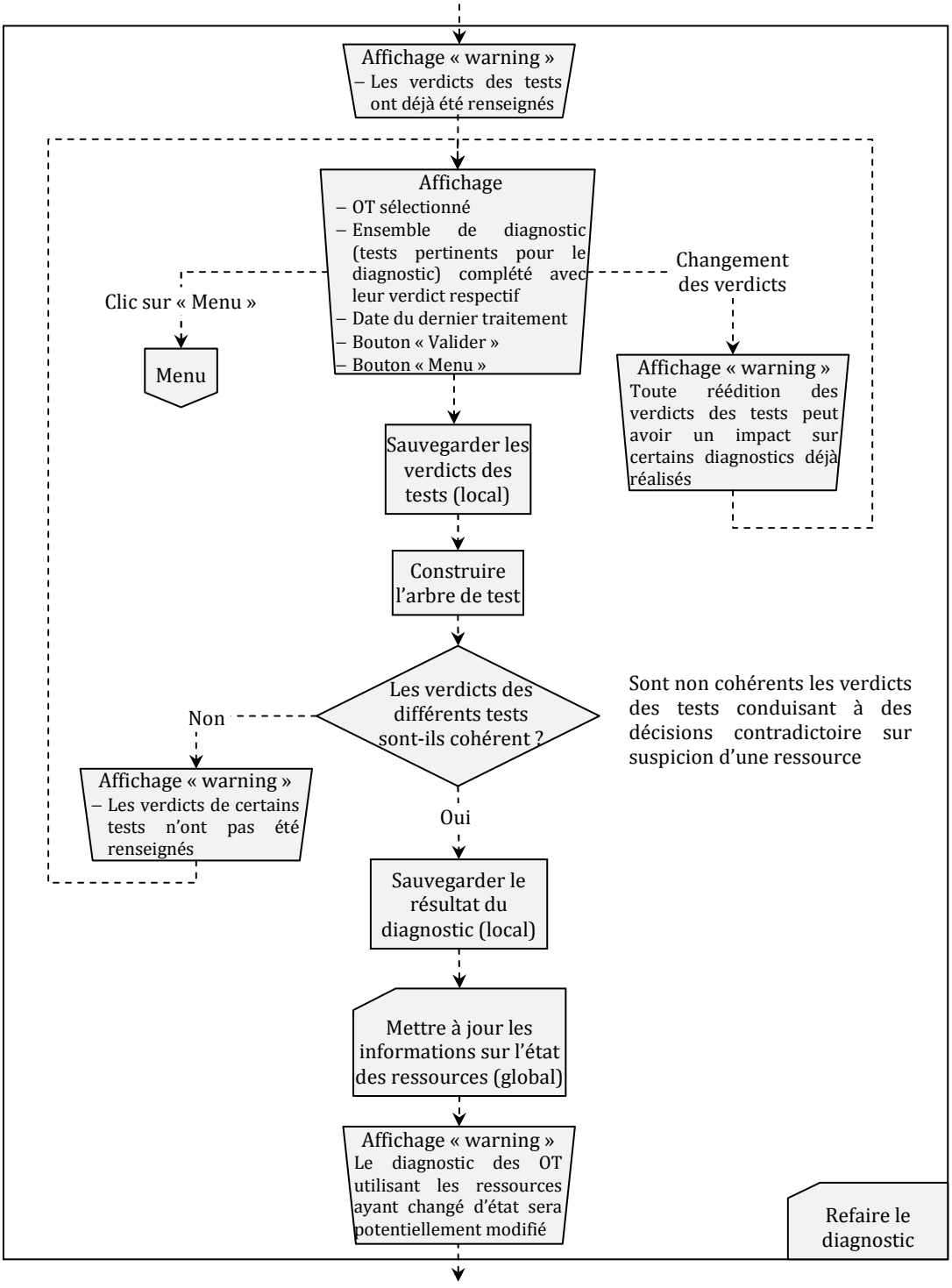


Figure E.3 : Processus de réfection diagnostic d'un objectif

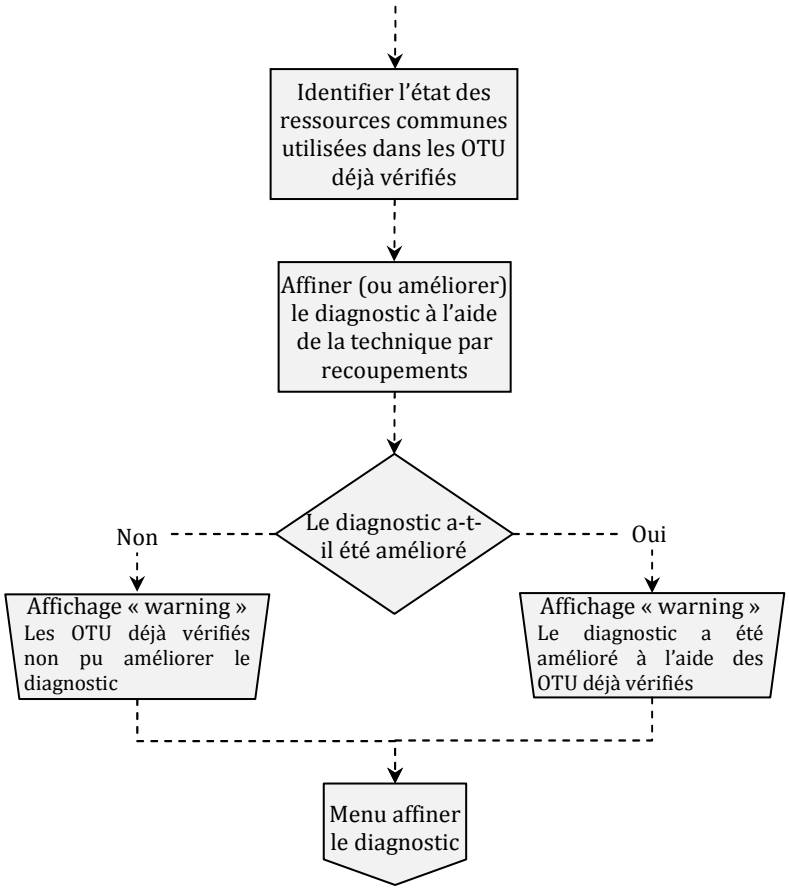


Figure E.4 : Processus d'amélioration du diagnostic à l'aide des informations disponibles

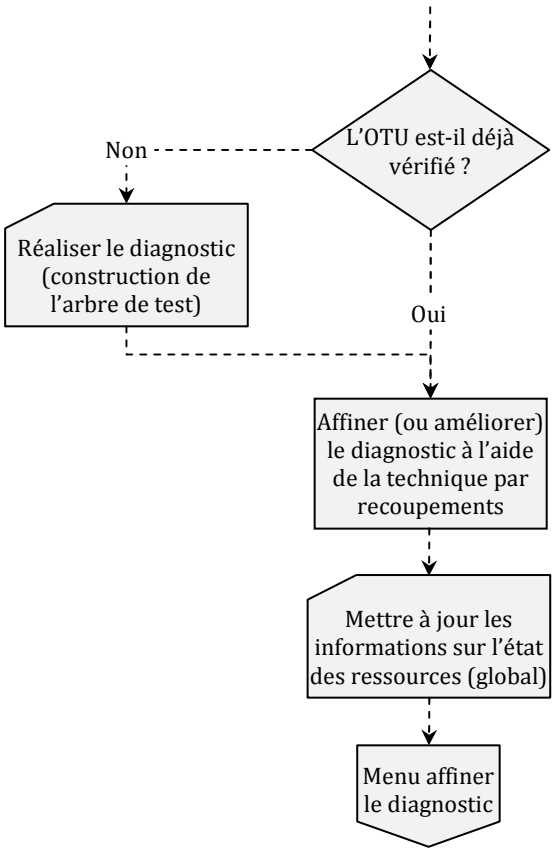


Figure E.5 : Processus d'amélioration du diagnostic à partir d'un objectif de test utile

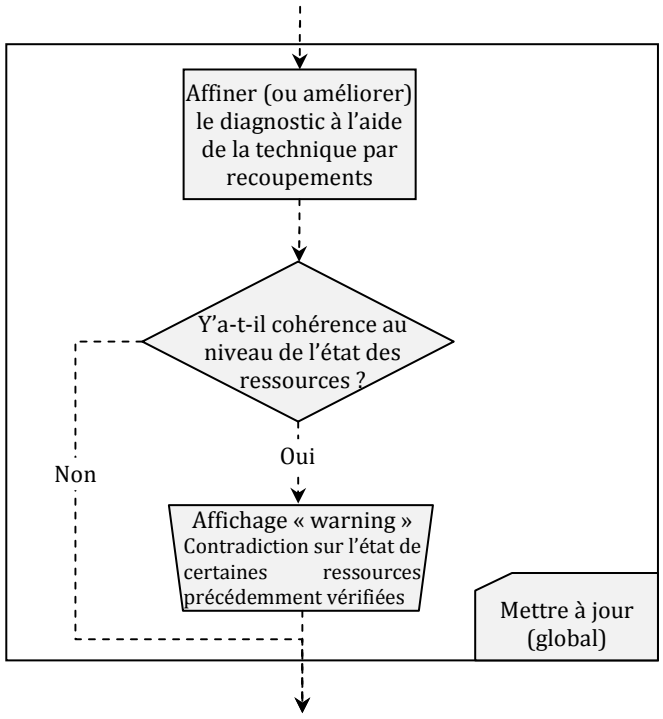


Figure E.6 : Processus de mise à jour des informations sur l'état des ressources activées

E.3 Principaux traitements internes du prototype « ADIS »

Dans cette partie, nous décrivons les deux principaux traitements réalisés par ADIS à savoir la recherche de la ressource défectueuse ou de l'ensemble de ressources indiscernables suspecté et la sauvegarde des informations des diagnostics déjà effectués.

E.3.1 Recherche de l'erreur après la détection d'une faute

Une fois que l'utilisateur a indiqué les verdicts des tests pour un objectif de test donné, le prototype recherche s'il existe une ressource défectueuse ou un ensemble de ressources indiscernables. Notons R_i les ressources et T_i les tests de cet objectif de test. Pour chaque test, une ressource peut être activée ou non (**section E.1**).

Lors du traitement, une ressource peut avoir 2 états :

- Etat « 0 » : La ressource est suspectée ;
- Etat « 1 » : La ressource est fonctionnelle.

A la fin du traitement, toutes les ressources étant à l'état 0 font partie de l'ensemble de ressources indiscernables suspecté. Si une seule ressource est à l'état 0, alors c'est la ressource défectueuse. Toutes les ressources à l'état 1 sont fonctionnelles.

Au départ, nous supposons que toutes les ressources sont suspectées (Etat de toutes les ressources à 0). Ensuite, nous traitons les verdicts des tests T_i l'un après l'autre. Pour chaque test, nous avons 2 choix possibles : Le test indique un défaut ou le test indique un succès. Ceci est le principe de la construction d'un arbre de test.

Test indiquant un défaut (verdict incorrect)

Si le test indique un défaut, alors toutes les ressources activées par ce test sont suspectées. Les autres sont fonctionnelles (hypothèse de l'existence de faute simple). Il suffit donc de faire une opération de « ou binaire » entre l'état antérieur de chaque ressource et :

- 0 : Si la ressource est activée par le test. En effet, si un test précédant l'avait défini comment fonctionnelle, elle sera toujours définie comme fonctionnelle (1 ou 0 = 1). Si elle n'était pas fonctionnelle, ce verdict ne changera pas son état.
- 1 : Si la ressource n'est pas activée par le test. En effet, peu importe l'état des tests précédant, elle est définie comme fonctionnelle.

Le nouvel état des ressources R_i est ainsi défini à partir de ce verdict avant de passer au test suivant.

Test indiquant un succès (verdict correct)

Si le test indique un succès, alors toutes ressources activées par ce test sont considéré comme fonctionnelle. Il suffit donc de faire une opération de « ou binaire » entre l'état antérieur de chaque ressource et :

- 1 : Si la ressource est activée par le test. En effet, peu importe l'état des tests précédant, elle est définie comme fonctionnelle.
- 0 : Si la ressource n'est pas activée pas le test. En effet, ce test n'apporte pas d'informations supplémentaires sur la ressource, elle garde son état antérieur.

Le nouvel état des ressources R_i est ainsi défini à partir de ce verdict avant de passer au test suivant.

Synthèse du traitement

L'algorithme devra donc :

- Initialiser l'état courant des ressources à 0 (suspectées).
- Pour chaque test : Si le test indique un succès, faire un « ou binaire » entre la ligne de la sous matrice correspondant au test et l'état antérieur des ressources ; si le test indique un défaut, faire un « ou binaire » entre la négation binaire de la ligne de la sous matrice correspondant au test et l'état antérieur des ressources.
- Au final, toutes les ressources dont l'état courant est à 1 sont fonctionnelles, les ressources dont l'état courant est à 0 sont suspectées (Si une seule ressource est à 0, alors on peut supposer que seule cette ressource est défectueuse).

E.3.2 Fichiers de sauvegarde

Afin d'éviter à l'utilisateur de renseigner plusieurs fois les mêmes informations, le prototype les sauvegarde automatiquement. Les verdicts des tests d'un objectif de test, ainsi que le résultat du diagnostic effectué sont sauvegardés dans un fichier contenant toutes les sauvegardes dites locales (**section E.1**). L'état de chaque ressource, ainsi que les objectifs de test dont le diagnostic est déjà effectué et qui peuvent avoir un impact sur l'état d'au moins une ressource sont sauvegardés dans un fichier de sauvegarde dit global.

Format de la sauvegarde locale

Les informations de sauvegarde locale sont enregistrées dans un fichier sous le format suivant :

Nom de l'objectif de test	Verdicts des tests	Ressource défectueuse ou bloc suspecté	Date à laquelle le dernier diagnostic a été réalisé
Obj1	1 0 0 1 0 1 1	3	17 Juillet 2009 à 14h29
Obj2	1 0 1 1	1 ;2	17 Juillet 2009 à 14h32
…	…	…	…

Pour les verdicts des tests, « 1 » indique que le test a été un succès, « 0 » un défaut. Le premier résultat indiqué est le résultat du premier test. Le deuxième résultat indiqué est le résultat du deuxième test, et ainsi de suite.

Format de la sauvegarde globale

Les informations de sauvegarde locale sont enregistrées dans un fichier sous le format suivant :

Nom de la ressource	Etat de la ressource	Objectifs de test dont on connaît le diagnostic et qui peuvent avoir un impact sur l'état de la ressource
Ressource1	0	Obj1 ;Obj2
Ressource2	1	Obj2
Ressource3	…	…
…	…	…

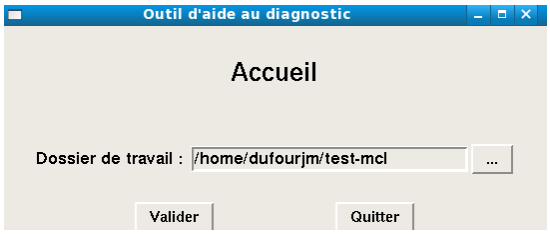
Annexe F

Interfaces du prototype ADIS

ADIS est une application graphique proposant des fonctionnalités permettant de guider le diagnostic des objectifs de test définis pour la vérification d'un système. L'utilisation de ce prototype nécessite la définition ou la spécification d'un « dossier de travail ». Ce dossier ainsi que les principales fonctionnalités proposées par ADIS sont décrits dans suite de cette section.

F.1 Dossier de travail

Ce paramètre est choisi à partir de la fenêtre de lancement qui s'affiche lors du démarrage de l'application. Cette fenêtre demande de choisir un dossier ou répertoire de travail (figure ci-dessous), par défaut, ce dossier correspond au répertoire de lancement de l'application.



Le dossier de travail doit contenir l'ensemble des informations nécessaires permettant d'aider le diagnostic d'un « objectif de test ». Pour chaque objectif de test, ces informations portent sur :

- les ressources utilisées ;
- les tests exécutés ;
- la matrice de diagnostic ;
- les tests équivalents ou redondants pour le diagnostic ;
- l'ensemble de diagnostic.

F.2 Principales fonctionnalités

Dans cette section, les fonctionnalités proposées par le prototype ADIS sont décrites en se basant sur son interface graphique. Cette interface est composée de quatre principales parties : menu principal, détails, diagnostic et rapport.

F.2.1 Menu principal

Le menu principal (figure ci-dessous) est le point d'accès des fonctions « primaires » du prototype. Il affiche et classe l'ensemble des objectifs de test selon qu'ils soient déjà traités ou pas. Un objectif de test est lorsqu'il a été vérifié et le diagnostic déjà réalisé. Ce menu permet de :

- visualiser les informations détaillées sur un objectif de test à l'aide du bouton « Détails » ;
- construire l'arbre de test d'un objectif de test à l'aide du bouton « Diagnostiquer » ;
- supprimer un objectif de test à l'issue d'une probable modification du GTI à partir d'un « clic droit » de la souris;
- gérer les rapports de synthèse des analyses effectuées à l'aide du prototype à partir du bouton « Rapport » ;
- fermer la fenêtre du menu principal à l'aide du bouton « Fermer la fenêtre » ;
- arrêter l'exécution du prototype à l'aide du bouton « Quitter l'application ».

Toutes ses fonctionnalités ne sont accessibles qu'à partir de la sélection de l'objectif de test à l'aide d'un « clic gauche » de la souris.

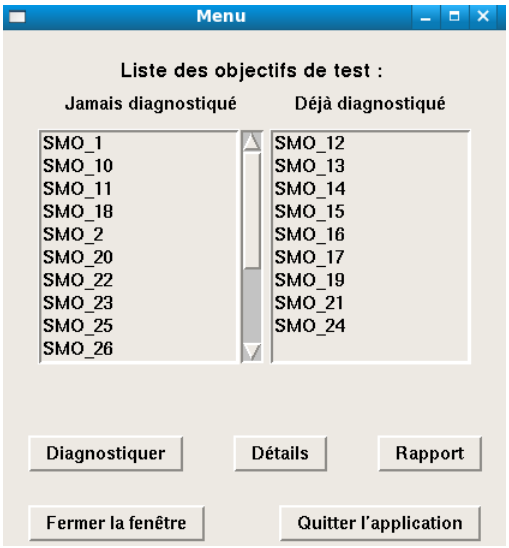


Figure F.1 : Menu principal du prototype ADIS

F.2.2 Détails

Cette partie de l'interface graphique (figure ci-dessous) permet d'afficher l'ensemble des informations disponibles sur un objectif de test. Ces informations concernent :

- la liste des ressources utilisées au cours de la vérification de l'objectif de test ;
- la liste des tests exécutés pour la vérification de l'objectif de test ;
- la liste des tests redondants pour le diagnostic ;
- la liste des ensembles de ressources indiscernables pour le diagnostic ;
- les informations sur le résultat du diagnostic lorsqu'il est déjà réalisé.

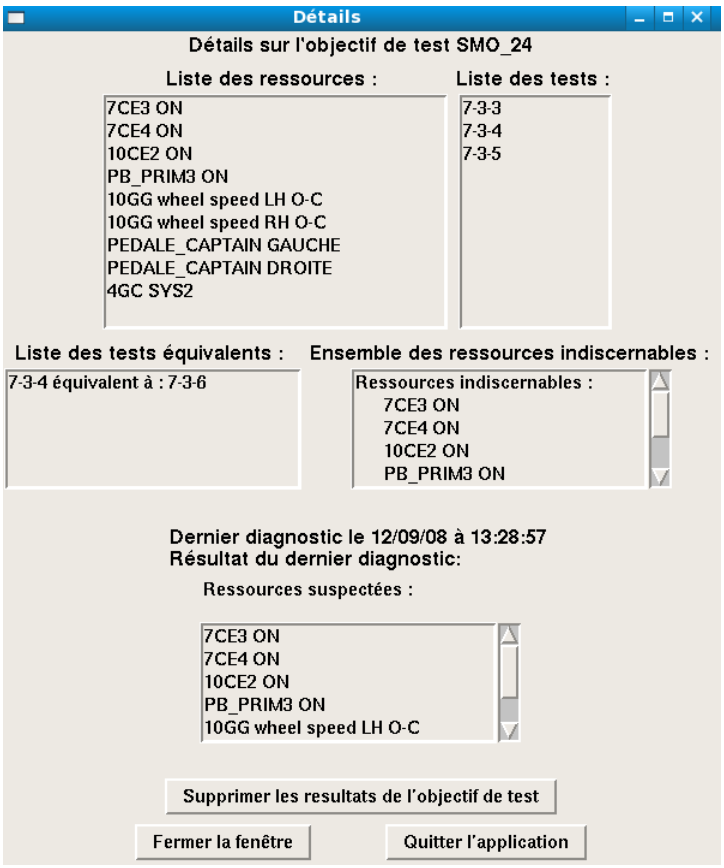


Figure F.2 : Informations détaillées sur un objectif de test

Les trois fonctions accessibles depuis cette interface sont :

- La suppression du résultat du diagnostic à l'aide du bouton « Suppression du résultat du diagnostic ». Ceci permet la reconstruction de l'arbre de test si nécessaire lorsque le verdict de l'ensemble de diagnostic est modifié.
- La fermeture de la fenêtre du menu principal à l'aide du bouton « Fermer la fenêtre » ;
- L'arrêt de l'exécution d'application à l'aide du bouton « Quitter l'application ».

F.2.3 Diagnostic

L'interface de diagnostic (figure ci-dessous) permet de construire automatiquement l'arbre de test à partir du verdict de l'ensemble de diagnostic et de déduire les informations sur le diagnostic d'un objectif de test. Lorsque le diagnostic a déjà été effectué, les informations sur ce diagnostic sont affichées. Le verdict des tests de l'ensemble de diagnostic d'un objectif de test est renseigné par le testeur. Le bouton « Tout correct » permet d'indiquer que tous les tests sont corrects. Cette option est utile lorsque l'ensemble de diagnostic contient le nombre élevé de tests et les verdicts de beaucoup de tests est un succès « correct ». Les verdicts peuvent par ailleurs être modifiés à l'aide d'un clic de la souris (de « correct » à « incorrect » et vice versa). Les boutons « Annuler » et « Valider » permettent respectivement d'arrêter le processus de construction de l'arbre de test et de valider le verdict des tests pour la construction automatique de l'arbre de test.

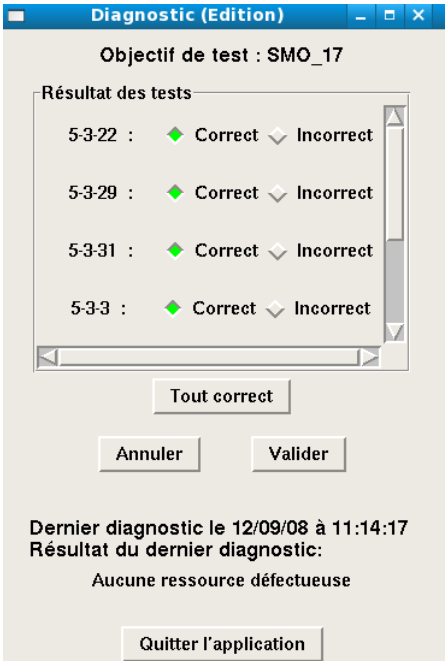


Figure F.3 : Interface de renseignement des verdicts des tests

La réduction des informations sur le diagnostic d'un objectif de test peut aboutir à cinq résultats possibles :

– « *Résultats de tests non cohérents* » : Les résultats de tests exécutés au cours de la vérification sont déclarés « non cohérents » au cours du diagnostic lorsque les verdicts se révèlent contradictoires pour la détection de la ressource défectueuse ou la suspicion d'un ensemble de ressources indiscernables. Cette information est fournie à l'aide de la fenêtre (figure ci-dessous) qui s'affiche après la validation de la saisie des résultats des

tests. Dans cette situation, il faut vérifier que les résultats saisis dans l'étape de diagnostic correspondent bien aux verdicts des tests.

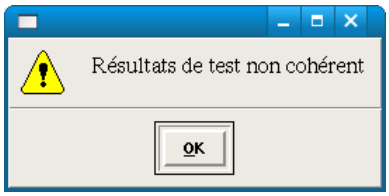


Figure F.4 : Résultat du diagnostic – « Résultats de tests non cohérents »

– « *Aucune ressource défectueuse* » : toutes les ressources sont fonctionnelles (le résultat de chaque test de l'objectif de test est correct). Le diagnostic n'a pas lieu d'être.

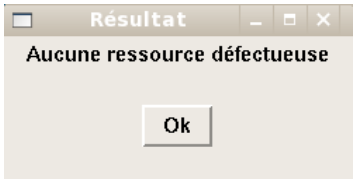


Figure F.5 : Résultat du diagnostic – « Aucune ressource défectueuse »

– « *Ressource défectueuse* » : Cette information est affichée lorsque la lecture de l'arbre de diagnostic a conduit à l'incrimination de la ressource défectueuse. Elle est accompagnée par le nom de la dite ressource (figure ci-dessous).

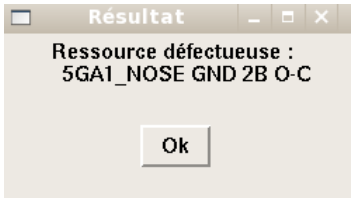


Figure F.6 : Résultat du diagnostic – « Ressource défectueuse »

– « *Ressources suspectées* » : lorsqu'un bloc de ressources indiscernables est suspecté à l'issue de la lecture de l'arbre de test, ce message est produit par le prototype. Deux différentes situations peuvent se présenter à dans ce contexte :

La première correspond au cas où il est possible d'améliorer le degré du diagnostic à l'aide d'objectifs de test utiles. Ces objectifs de test sont proposés au testeur qui pourra choisir selon leur ordre de définition dans le GTI l'objectif de test qui lui permettra d'améliorer le diagnostic ou utiliser le résultat du diagnostic des objectifs de tests utiles déjà vérifié pour améliorer le diagnostic. Cette action d'amélioration du diagnostic est représentée par le verbe « affiner ». La figure ci-dessous illustre ce scénario.

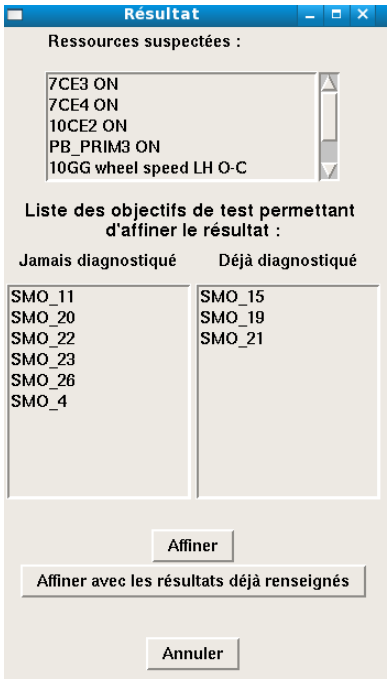


Figure F.7 : Résultat du diagnostic – « Ressources suspectées et amélioration possible »

La deuxième situation correspond à celle où aucun objectif de test défini ne permet d'améliorer le résultat du diagnostic proposé. Dans ce cas, la méthode d'amélioration n'apporte pas d'aide supplémentaire pour l'incrimination de la ressource défectueuse. La figure ci-dessous montre le message affiche dans cette situation.

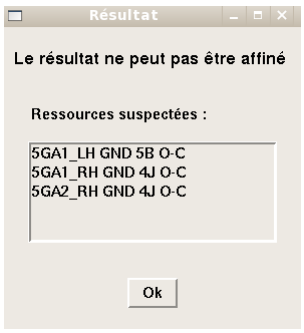


Figure F.8 : Résultat du diagnostic – « Ressources suspectées et pas d'amélioration »

F.2.4 Rapports

Cette partie de l'interface propose des fonctionnalités permettant de gérer les traces des analyses effectuées à l'aide du prototype ADIS. En effet, l'outil enregistre l'ensemble des activités de diagnostic réalisées. Ces traces sont rangées par date et peuvent être : exportées dans un fichier à l'aide du bouton « Enregistrer dans un fichier » ; ou supprimées de la base de l'outil à

l'aide du bouton « Effacer le(s) rapport(s) ». Il existe également deux boutons « Tout cocher » et « Tout décocher » permettant d'aider à la sélection des dates.

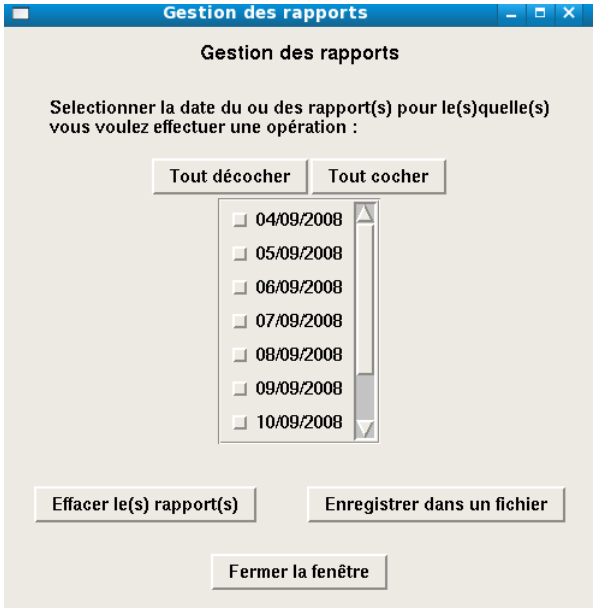


Figure F.9 : Interface de gestion des traces

Annexe G

Résultats détaillés de l'application de méthodologie d'aide à la vérification

Dans cette section, nous présentons les résultats détaillés de l'application de la méthodologie d'aide à la vérification sur le système d'Orientation Roues Avant (ORA).

Pour la représentation des résultats ci-dessous, les expressions suivantes seront utilisées :

- **Nb RU** : Nombre de ressources utilisées par un objectif de test ;
- **Nb TD** : Nombre de tests définis pour la vérification d'un objectif de test ;
- **Nb TR** : Nombre de tests redondants pour le diagnostic ;
- **Nb TC** : Nombre de tests choisis par Multiple-Clue pour le diagnostic ;
- **Nb ERI** : Nombre d'ensembles de ressources de ressources indiscernables identifiés pour un objectif de test par Multiple-Clue ;
- **Nb RI** : Nombre total de ressources indiscernables identifiées pour un objectif de test par Multiple-Clue.

G.1 Résultats de l'application de Multiple-Clue sur « l'ORA »

	Nb RU	Nb TD	Nb TC	Nb TR	Nb ERI	Nb RI
SMO_1	2	2	2	**0**	0	**0**
SMO_2	4	5	3	**2**	0	**0**
SMO_3	11	5	3	**1**	3	**9**
SMO_4	15	12	6	**0**	4	**8**
SMO_5	7	3	3	**0**	3	**7**
SMO_6	2	3	2	**1**	0	**0**
SMO_7	4	6	3	**2**	0	**0**
SMO_8	14	5	3	**1**	4	**12**
SMO_9	8	6	4	**0**	2	**4**
SMO_10	8	3	3	**0**	2	**6**
SMO_11	8	5	4	**0**	2	**4**
SMO_12	8	4	3	**0**	2	**6**
SMO_13	18	5	5	**0**	6	**18**
SMO_14	7	9	4	**4**	0	**0**
SMO_15	7	8	4	**2**	0	**0**
SMO_16	5	3	3	**0**	0	**0**
SMO_17	14	16	13	**3**	0	**0**
SMO_18	11	8	6	**0**	0	**0**
SMO_19	12	13	5	**5**	3	**7**
SMO_20	5	2	2	**0**	1	**4**
SMO_21	6	2	2	**0**	1	**5**
SMO_22	12	18	5	**6**	3	**7**
SMO_23	8	4	2	**1**	1	**6**
SMO_24	9	4	2	**1**	1	**7**
SMO_25	5	8	3	**2**	0	**0**
SMO_26	9	10	4	**3**	2	**5**

G.2 Résultats de l'amélioration du diagnostic de « l'ORA »

Les lignes grisées correspondent aux objectifs de test dont le degré de diagnostic a été amélioré à l'aide la méthode d'amélioration proposée dans ce manuscrit.

	Nb RU	Nb TD	Nb TC	Nb TR	Nb ERI	Nb RI
SMO_1	2	2	2	**0**	0	**0**
SMO_2	4	5	3	**2**	0	**0**
SMO_3	11	5	3	**1**	3	**9**
SMO_4	15	12	6	**0**	2	**4**
SMO_5	7	3	3	**0**	3	**7**
SMO_6	2	3	2	**1**	0	**0**
SMO_7	4	6	3	**2**	0	**0**
SMO_8	14	5	3	**1**	3	**9**
SMO_9	8	6	4	**0**	2	**4**
SMO_10	8	3	3	**0**	0	**0**
SMO_11	8	5	4	**0**	2	**4**
SMO_12	8	4	3	**0**	0	**0**
SMO_13	18	5	5	**0**	5	**13**
SMO_14	7	9	4	**4**	0	**0**
SMO_15	7	8	4	**2**	0	**0**
SMO_16	5	3	3	**0**	0	**0**
SMO_17	14	16	13	**3**	0	**0**
SMO_18	11	8	6	**0**	0	**0**
SMO_19	12	13	5	**5**	3	**6**
SMO_20	5	2	2	**0**	1	**2**
SMO_21	6	2	2	**0**	1	**3**
SMO_22	12	18	5	**6**	3	**6**
SMO_23	8	4	2	**1**	1	**2**
SMO_24	9	4	2	**1**	1	**2 ou 3**
SMO_25	5	8	3	**2**	0	**0**
SMO_26	9	10	4	**3**	1	**3**

Résumé

Les phases de validation et de vérification (V&V) des systèmes réactifs temps réel critique (de plus en plus complexes) sont très importantes en termes de coût et de temps. Dans ce contexte, toute méthode et outil permettant d'aider à la réalisation des activités de V&V est d'une très grande importance au cours du développement. Le test fonctionnel est le moyen le plus utilisé au cours de ces phases de V&V. Or, les méthodes de test présentent des limites : un test exhaustif est quasiment impossible à réaliser en raison de la taille et de la complexité des systèmes considérés. Dans ce contexte, les enjeux de la maîtrise de l'effort de test (complexité et coût) sont majeurs, mais les exigences de qualité pour ces systèmes sont très grandes. L'effort de test caractérise tout autant l'élaboration des jeux de test que le diagnostic. Dans cette optique, nous avons défini deux méthodologies basées sur les concepts d'analyse de testabilité et les stratégies de test. La première méthodologie permet d'aider à la définition de jeux de test pertinents et à l'analyse de couverture des systèmes réactifs spécifiés dans un formalisme flot de données SCADE dans le contexte AIRBUS. La seconde propose des méthodes d'aide à la vérification (identification de tests pertinents et localisation de composants défectueux au cours du diagnostic) de systèmes sur la chaîne d'assemblage finale (FAL) d'un avion AIRBUS.

Mots clé : Systèmes réactifs temps réel, Spécifications SCADE, Analyse de testabilité, Stratégies de test, Validation des exigences, Vérification des exigences, Traçabilité, Analyse de couverture.

Abstract

Reactive real time systems are critical systems. These systems requirements validation and verification (V&V) activities are very important in terms of cost and time. Therefore, methods and tools that can alleviate and efficiently support V&V activities are of great interest for aeronautic domain. Functional testing is the most commonly used technique for these systems requirements V&V. But, testing methods present some limits: exhaustive test data generation is practically impossible because of the size and the complexity of these systems. In this way, controlling testing effort (complexity and cost) is major, but systems quality demands are very huge. Testing effort characterizes as well the tests definition as the diagnosis. In this perspective, we defined two methodologies based on testability analysis concepts and test strategies. The first methodology aims at guiding relevant functional tests definition and facilitating AIRBUS reactive systems SCADE data flow specification coverage analysis during the validation activities. The second methodology proposes methods supporting the verification activities (relevant tests definition for diagnosis and faulty component identification) based on functional testing on AIRBUS Final Assembly Line (FAL).

Keywords: Reactive real time systems, SCADE specification, Testability analysis, Test strategies, Requirements validation, Requirements verification, Traceability, Coverage analysis.