Evaluation de la sûreté de fonctionnement informatique. Fautes physiques, fautes de conception, malveillances

Mohamed Kaâniche

N° d'ordre : 129 Année 1999

HABILITATION À DIRIGER LES RECHERCHES

présentée au
Laboratoire d'Analyse et d'Architecture des Systèmes du CNRS

en vue de l'obtention du
Diplôme de l'Institut National Polytechnique de Toulouse

par

Mohamed Kaâniche
Ingénieur ENAC
Docteur INPT
Chargé de Recherche au CNRS

Évaluation de la sûreté de fonctionnement informatique —

fautes physiques, fautes de conception, malveillances

Soutenue le 12 février 1999 devant le Jury composé de :

M.	Jean-Claude LAPRIE	Président
M.	Gianfranco BALBO	Rapporteur
M.	Paul CASPI	Examinateur
M.	Ravishankar IYER	Rapporteur
Mme.	Karama KANOUN	Rapporteur
M.	Raymond MARIE	Examinateur

Rapport LAAS n° 99062

Les travaux exposés dans ce mémoire ont été effectués au
Laboratoire d'Analyse et d'Architecture des Systèmes du CNRS
7, avenue du Colonel Roche 31077 Toulouse Cedex 4

AVANT-PROPOS

Les travaux présentés dans ce mémoire ont été effectués au Laboratoire d'Analyse et d'Architecture des Systèmes du Centre National de la Recherche Scientifique (LAAS-CNRS). Je tiens tout d'abord à exprimer ma profonde gratitude à Messieurs Alain Costes et Jean-Claude Laprie, Directeurs successifs du LAAS depuis mon entrée en 1988, pour leur accueil et pour la confiance qu'ils m'ont témoignée.

Je remercie tout particulièrement Jean-Claude Laprie pour m'avoir accueilli dans le groupe "Tolérance aux fautes et Sûreté de Fonctionnement informatique" (TSF) alors qu'il en était responsable, et pour avoir dirigé et orienté mes travaux. Ayant travaillé étroitement avec lui, en particulier, sur la modélisation de la croissance de fiabilité et sur la définition du modèle de développement à sûreté de fonctionnement explicite, j'ai pu bénéficier de la vaste étendue de ses compétences. Je lui suis reconnaissant d'avoir su orienter mes travaux, pour les conseils et critiques qu'il a pu me donner, et aussi pour m'avoir offert l'environnement idéal pour mener mes recherches et me confronter à la communauté scientifique tant nationale qu'internationale de très haut niveau. Son dynamisme, sa rigueur, et sa recherche de la perfection ont été pour moi un grand stimulateur pour continuer dans cette voie.

Je réserve une mention amicale et privilégiée à Karama Kanoun, pour avoir dirigé mes travaux durant ma thèse de Doctorat et pour avoir suivi de près la préparation de ce mémoire. Notre étroite collaboration durant ces dix dernières années, en particulier sur les thèmes concernant la modélisation de la croissance de fiabilité et la construction de modèles complexes basés sur les GSPN, nos confrontations de points de vue et ses critiques constructives ont grandement contribué à l'amélioration de la qualité de mes recherches. Je tiens aussi à la remercier pour ses encouragements et son soutien amical, qui ont été pour moi une source d'énergie supplémentaire.

Je tiens à exprimer ma profonde reconnaissance à Alain Costes qui, en tant que membre du groupe TSF, n'a pas hésité à me consacrer une partie de son temps pour faire des critiques constructives sur certains aspects de mes travaux, sans oublier ses encouragements et ses conseils dont j'ai pu bénéficier.

Je remercie Jean-Claude Laprie, Directeur de Recherche au LAAS-CNRS, pour l'honneur qu'il me fait en présidant mon jury de soutenance ainsi que :

- Monsieur Gianfranco Balbo, Professeur à l'Universita di Torino, Italie,
- Monsieur Paul Caspi, Directeur de Recherche au CNRS, VERIMAG,
- Monsieur Ravishankar Iyer, Professeur à l'University of Illinois at Urbana-Champaign, Illinois, Etats-Unis,
- Madame Karama Kanoun, Directeur de Recherche au LAAS-CNRS,
- Monsieur Raymond Marie, Professeur à l'Université de Rennes.

pour avoir accepté de participer à ce Jury et plus particulièrement Karama Kanoun, Gianfranco Balbo et Ravishankar Iyer pour avoir accepté la charge d'être Rapporteurs.

Je tiens à remercier vivement Yves Deswarte pour notre collaboration sur l'évaluation quantitative de la sécurité-confidentialité et dans le cadre du projet SQUALE, ainsi que

pour son ouverture d'esprit et son soutien amical. Notre collaboration et la complémentarité de nos thèmes de recherches ont été pour moi une expérience riche d'enseignements et un atout pour élargir le spectre de mes travaux.

Mes travaux ont grandement bénéficié de la contribution et de la collaboration des doctorants et stagiaires que j'ai eu le plaisir d'encadrer. Je tiens à exprimer ma profonde reconnaissance, en particulier, à Marc Dacier, Nicolae Fota, et Rodolphe Ortalo, avec lesquels je partage une bonne partie des résultats présentés dans ce mémoire. Nos discussions ont été à la fois animées, constructives et amicales.

Mes remerciements s'adressent aussi à l'ensemble des membres du groupe TSF et du *LIS*, qu'ils soient permanents, doctorants ou stagiaires, ainsi qu'à Joëlle Penavayre et Marie-José Fontagne. Ils ont tous, à un moment ou à un autre, su contribuer à leur façon au bon déroulement de mes travaux. Qu'ils me pardonnent de ne pas dévoiler leurs noms car la liste est longue. Je tiens à réserver une mention particulière à Pascale Thévenod, qui, en partageant mon bureau durant ces dix dernières années, a dû supporter mes sautes d'humeur sans être avare d'encouragements et de soutien amical, ainsi qu'à Yves Crouzet et Jean Arlat à qui je tiens à témoigner ma sincère reconnaissance, pour leur grande disponibilité et leur soutien amical et scientifique. J'aurais aussi une pensée à la mémoire de Christian Béounes qui m'a initié aux réseaux de Petri stochastiques et pour sa précieuse et sympathique collaboration à certains de mes travaux sur la croissance de fiabilité. Je voudrais également remercier David Powell, responsable de TSF, pour m'avoir encouragé à préparer cette habilitation.

Je remercie vivement le Professeur Ravishankar Iyer pour m'avoir offert l'opportunité de séjourner pendant une année au *Center for Reliable and High-Performance Computing* (CRHC) de l'Université d'Illinois à Urbana-Champaign et pour notre collaboration dans le cadre du projet mené avec la société StorageTek (Louisville, États-Unis). À ce titre, je remercie Luigi Romano et Zbigniew Kalbarczyk pour leur participation à ce projet. Je réserve aussi une mention spéciale à Michel Cukier, Beth Dennisson et Fran Baker, pour leur accueil chaleureux et pour avoir rendu mon séjour dans l'Illinois agréable et plaisant, en dépit de la rudesse du climat.

Certains aspects des travaux théoriques synthétisés dans ce mémoire, n'auraient pu être mis en pratique, ni étayés sans l'aide de la compagnie brésilienne des télécommunications TELEBRAS CPqD, du Centre d'Etudes de la Navigation Aérienne (CENA) à Toulouse, et de la société StorageTek à Louisville-Colorado aux États-Unis. Je tiens à remercier les membres de ces organismes pour leur coopération.

Le développement, l'utilisation ou la diffusion des outils SoRel et SURF2 ont bénéficié du support du service *"Informatique et Instrumentation"* du LAAS. Je tiens à remercier tous les membres de ce service pour leur assistance et leur disponibilité. Mes remerciements s'adressent aussi aux membres des services *Direction, Documentation-Edition, Entretien, Gestion, Magasin*, et *Réception-Standard* qui m'ont toujours permis de travailler dans d'excellentes conditions.

Je ne saurais finir cet avant-propos, sans penser à tous ceux et toutes celles qui, de près ou de loin, et trop nombreux pour être nommés, m'ont apporté leur soutien moral et amical.

Sommaire

1

1. INTRODUCTION GÉNÉRALE ET CONTEXTE DES TRAVAUX

Les travaux présentés dans ce document ont pour cadre la sûreté de fonctionnement des systèmes informatiques. Indispensables pour la communication et le traitement de l'information, les systèmes informatiques sont devenus le cœur de la plupart des systèmes utilisés dans la société moderne. Durant les dernières décennies, les systèmes informatiques ont connu plusieurs mutations du point de vue des domaines d'application, de la complexité des traitements effectués, des technologies utilisées, de l'architecture interne, etc. Ces mutations se sont traduites en même temps par une évolution à la fois, de la nature des fautes qui sont à l'origine des défaillances et de leur importance relative, et des besoins en sûreté de fonctionnement. De ce fait, les techniques et méthodes à mettre en œuvre pour satisfaire les exigences de sûreté de fonctionnement doivent aussi évoluer en conséquence. Historiquement, les fautes physiques ont été la principale source des défaillances informatiques. Actuellement, les fautes de conception, les fautes d'interaction dues à l'homme, et les malveillances (fautes intentionnelles avec volonté de nuire) constituent la préoccupation majeure des concepteurs et des utilisateurs des systèmes informatiques. Ces fautes ayant leur origine dans le processus de développement, ceci conduit naturellement à focaliser l'attention sur les méthodes permettant de maîtriser la mise en œuvre de la sûreté de fonctionnement pour contrer les différentes classes de fautes, et d'avoir confiance dans la capacité du système à satisfaire ses exigences durant sa vie opérationnelle et jusqu'à son retrait du service.

Les travaux présentés dans ce document s'inscrivent dans cette optique. Dans la suite, nous donnons quelques définitions de base extraites de [132] et nous décrivons les principaux thèmes de nos travaux. Ces thèmes sont ensuite développés dans les Chapitres 2, 3, 4 et 5. Le dernier chapitre résume les principales conclusions et donne quelques orientations pour des travaux futurs.

1.1. Définitions

La *sûreté de fonctionnement* d'un système est définie comme la propriété permettant à ses utilisateurs de placer une confiance justifiée dans le service qu'il leur délivre. Elle peut être vue selon des propriétés différentes mais complémentaires qui permettent de définir ses *attributs* :

- le fait d'être prêt à l'utilisation conduit à la *disponibilité ;*
- la continuité de service conduit à la *fiabilité ;*
- la non-occurrence de conséquences catastrophiques conduit à la *sécurité-innocuité ;*
- la non-occurrence de divulgations non-autorisées de l'information conduit à la *confidentialité ;*
- la non-occurrence d'altérations inappropriées du système conduit à l'*intégrité.*
- l'aptitude aux réparations et aux évolutions conduit à la *maintenabilité.*

L'association à la confidentialité, de l'intégrité et de la disponibilité, vis-à-vis des actions autorisées, conduit à la *sécurité-confidentialité*.

Les entraves à la sûreté de fonctionnement sont les *fautes, erreurs et défaillances*. Une défaillance survient lorsque le service délivré dévie de l'accomplissement de la fonction du système (c'est-à-dire, ce à quoi il est destiné). Une erreur est la partie de l'état du système qui est susceptible d'entraîner une défaillance. La cause adjugée ou supposée d'une erreur est une faute.

Les fautes et leurs sources sont extrêmement diverses. On peut les classer selon cinq points de vue : leur cause phénoménologique (fautes physiques ou fautes dues à l'homme), leur nature (fautes accidentelles, fautes intentionnelles avec ou sans volonté de nuire[1]), leur phase de création ou d'occurrence (fautes de développement, fautes opérationnelles), leur situation par rapport aux frontières du système (fautes internes, fautes externes), et leur persistance (fautes permanentes, fautes temporaires). La considération simultanée de différents points de vue conduit à la notion de fautes combinées. Par exemple : les *fautes de conception* sont des fautes de développement accidentelles ou intentionnelles sans volonté de nuire ; les *intrusions* sont des fautes opérationnelles externes intentionnellement nuisibles

La distinction entre différentes classes de fautes permet d'identifier des moyens appropriés pour se prémunir contre ces fautes. On distingue généralement quatre classes de moyens pour la sûreté de fonctionnement :

- *prévention de fautes* : comment empêcher l'occurrence ou l'introduction de fautes,
- *tolérance aux fautes* : comment fournir un service à même de remplir la fonction du système en dépit des fautes,
- *élimination des fautes* : comment réduire la présence (nombre, sévérité) des fautes,
- *prévision des fautes* : comment estimer la présence, la création et les conséquences des fautes.

---

[1] Une faute intentionnellement nuisible est généralement dénommée *malveillance*.

La prévention de fautes relève de l'ingénierie "générale" des systèmes. Elle concerne le choix et la mise en œuvre d'un ensemble de processus visant à maîtriser la conception, la réalisation et la validation du système, et à assurer le bon fonctionnement du système durant sa vie opérationnelle et jusqu'à son retrait de service. La tolérance aux fautes vise à doter le système de mécanismes pour le traitement d'erreurs (éliminer les erreurs avant qu'une défaillance ne survienne) et le traitement de fautes (éviter qu'une, ou des fautes ne soient activées de nouveau). L'élimination des fautes consiste à vérifier si le système satisfait les propriétés requises ; et si ce n'est pas le cas, à identifier les fautes et les corriger. Enfin, la prévision des fautes est conduite en effectuant des évaluations du comportement du système par rapport à l'occurrence des fautes et à leur activation.

Une part importante de nos travaux porte sur la prévision des fautes. On distingue deux types d'évaluation :

- les *évaluations ordinales* qui consistent à identifier, classer et ordonner les défaillances, ou les méthodes et techniques mises en œuvre pour les éviter
- les *évaluations probabilistes*, qui sont destinées à évaluer en termes de probabilités le degré de satisfaction de certains attributs de la sûreté de fonctionnement.

Les méthodes d'évaluation probabiliste diffèrent selon que le système est considéré comme étant en fiabilité stabilisée ou en croissance de fiabilité. Ces deux notions jouent un rôle important dans nos travaux ; elles peuvent être définies comme suit :

- *fiabilité stabilisée* : l'aptitude du système à délivrer un service correct (c'est-à-dire, qui accomplit la fonction du système) est préservée. Par exemple : 1) dans le cas d'une défaillance du matériel, le composant défaillant est remplacé par un autre, identique, et non défaillant, 2) dans le cas d'une défaillance du logiciel, le système est relancé sur un point d'entrée différent de celui ayant provoqué la défaillance ;
- *croissance de fiabilité* : l'aptitude du système à délivrer un service correct est améliorée. Ceci se produit par exemple quand la faute (ou éventuellement les fautes) dont l'activation a conduit à défaillance est diagnostiquée comme une faute de conception (matérielle ou logicielle) et est éliminée.

Une décroissance de fiabilité est théoriquement et pratiquement possible ; par exemple suite à l'introduction de fautes durant des actions correctives. Dans une telle situation, il est souhaitable que la décroissance soit limitée dans le temps, et que la fiabilité soit globalement croissante sur une longue période d'observation.

1.2. Thèmes des travaux

Nos travaux couvrent plusieurs aspects complémentaires, que nous avons groupés en quatre thèmes développés aux chapitres 2, 3, 4 et 5, respectivement :

1) Définition de méthodes permettant de faciliter la construction de modèles complexes pour l'analyse et l'évaluation de la sûreté de fonctionnement (Chapitre 2).
2) Modélisation de la croissance de fiabilité pour l'évaluation de mesures caractérisant l'évolution de la fiabilité et de la disponibilité de systèmes en tenant compte de l'élimination progressive des fautes de conception (Chapitre 3).

3) Définition d'une approche d'évaluation quantitative de la sécurité-confidentialité permettant de suivre l'évolution de la capacité d'un système à résister à des attaques potentielles en fonction de modifications affectant la configuration opérationnelle, le comportement des utilisateurs, la politique de sécurité, etc. (Chapitre 4)
4) Élaboration d'un modèle de développement permettant d'incorporer, à chaque étape du développement, les activités relatives à la sûreté de fonctionnement, et définition de critères d'évaluation et de certification de systèmes vis-à-vis de leur aptitude à satisfaire leurs exigences de sûreté de fonctionnement durant la vie opérationnelle et jusqu'au retrait du service (Chapitre 5).

Les trois premiers thèmes traitent de la prévision des fautes en utilisant des méthodes d'évaluation probabiliste. La prévision des fautes trouve tout son intérêt quand elle est intégrée durant les différentes phases du cycle de vie des systèmes. Durant la conception, plusieurs architectures sont envisageables. La modélisation de ces architectures et l'évaluation de mesures de sûreté de fonctionnement permet alors de fournir des éléments de décision pour sélectionner celle qui est la plus adaptée pour satisfaire les exigences de sûreté de fonctionnement. Plusieurs techniques de construction de modèles, adaptées à ce contexte, sont proposées dans la littérature. Cependant, ces techniques ont encore des limites quand il s'agit de construire des modèles complexes représentatifs du comportement de systèmes réels. La complexité peut résulter de : 1) la représentation fine des interactions entre composants, tant matériels que logiciels, 2) le nombre élevé de composants à considérer explicitement dans le modèle, et 3) la prise en compte de plusieurs niveaux de dégradation de service, etc. Les travaux résumés dans le Chapitre 2, proposent des solutions pour faciliter la construction de modèles complexes en considérant deux types d'approche : la modélisation par réseaux de Petri stochastiques généralisés et la simulation comportementale en présence de fautes. Dans ces travaux, le système est considéré en fiabilité stabilisée et les modèles sont utilisés pour analyser l'efficacité des mécanismes de tolérance aux fautes, quantifier l'évolution des mesures de sûreté de fonctionnement (fiabilité, disponibilité, sécurité-innocuité), etc. Les méthodes proposées sont illustrées sur deux cas réels : le système informatique du contrôle en route du trafic aérien français (CAUTRA), et un système commercial de stockage de données basé sur une architecture RAID (Redundant Array of Inexpensive Disks).

Le comportement en fiabilité stabilisée correspond au régime stationnaire du système, et ne prend pas en compte l'élimination progressive des fautes de conception qui conduit généralement à un comportement non stationnaire. Nos travaux sur la modélisation et l'évaluation de la croissance de fiabilité, présentés au Chapitre 3, visent à évaluer des mesures caractérisant l'évolution de la fiabilité et de la disponibilité de systèmes en tenant compte de l'élimination progressive des fautes de conception (accidentelles ou intentionnelles sans volonté de nuire). Traditionnellement, l'évaluation de la croissance de fiabilité s'est focalisée sur le logiciel en adoptant une approche boîte noire. Nous avons étendu ces travaux en adoptant une approche boîte blanche pour modéliser des systèmes constitués de composants matériels et logiciels. Le modèle du système en croissance de fiabilité est obtenu à partir de la transformation d'une chaîne de Markov décrivant le comportement du système en fiabilité stabilisée, en utilisant des propriétés

markoviennes du modèle hyperexponentiel développé au LAAS. Cette approche permet, en particulier, de réutiliser l'ensemble des résultats obtenus dans le domaine de la construction et du traitement de chaînes de Markov (en particulier, ceux présentés dans le Chapitre 2). Ces travaux sont complétés, d'une part par la définition d'un modèle de croissance de fiabilité en temps discret qui possède des propriétés intéressantes pour prendre en compte de façon explicite l'influence du profil d'utilisation sur l'évolution des mesures de fiabilité, et d'autre part par la définition d'une méthode permettant de faciliter la mise en œuvre d'une étude de croissance de fiabilité dans un contexte industriel. Nous avons illustré cette méthode sur plusieurs cas réels (en particulier, les logiciels TROPICO développés par la compagnie brésilienne TELEBRAS), dont un résumé est donné à la fin du Chapitre 3.

Historiquement, les méthodes d'évaluation probabiliste de mesures de sûreté de fonctionnement ont été limitées, de façon générale, à la prise en compte des fautes physiques ou des fautes de conception accidentelles ou intentionnelles sans volonté de nuire. Cependant, de plus en plus, on s'intéresse aussi à l'impact des fautes intentionnelles avec volonté de nuire, c'est-à-dire aux malveillances. Cet intérêt n'est plus limité aux systèmes militaires, mais s'étend aussi à des systèmes qui ont des exigences moins fortes en termes de sécurité-confidentialité. Dans le Chapitre 4, nous présentons une approche originale basée sur les chaînes de Markov, qui fournit des mesures quantitatives probabilistes permettant de surveiller l'évolution de la sécurité-confidentialité pendant l'exploitation opérationnelle, en tenant compte des modifications survenant dans le comportement des utilisateurs, la configuration opérationnelle, la politique de sécurité, etc. Pour illustrer le bien fondé de cette approche, nous l'avons expérimentée sur un système opérationnel de grande taille constitué de plusieurs centaines de machines Unix connectées à travers un réseau local. Les résultats sont commentés à la fin du Chapitre 4.

Les travaux présentés dans les Chapitres 2, 3 et 4 sont focalisés sur la prévision de fautes. Néanmoins, les activités relatives à la prévision de fautes doivent s'intégrer dans une démarche globale de développement de systèmes sûrs de fonctionnement qui couvre tout le cycle de vie. Une telle démarche doit aussi prendre en compte, de façon globale et harmonisée, les autres moyens de la sûreté de fonctionnement, c'est-à-dire, la prévention de fautes, la tolérance aux fautes et l'élimination des fautes. Dans le Chapitre 5, nous présentons un modèle de développement à sûreté de fonctionnement explicite, basé sur cette philosophie, qui vise à pallier certaines lacunes des modèles de développement traditionnels. L'autre point qui est traité dans ce chapitre est comment certifier un système pour avoir une confiance justifiée dans sa capacité à satisfaire ses exigences de sûreté de fonctionnement, pendant la vie opérationnelle et jusqu'à son retrait du service ? Pour cela, nous proposons des critères d'évaluation qui permettent de prendre en compte l'ensemble des attributs de la sûreté de fonctionnement et non seulement la sécurité-confidentialité, comme c'est le cas par exemple des critères ITSEC, ou la sécurité-innocuité, comme c'est le cas des normes de l'avionique, du ferroviaire, ou du nucléaire. Pour valider ces critères, nous les avons expérimentés sur le système de commande de la nouvelle ligne de métro automatique à Paris, METEOR.

La figure 1 permet de situer dans le temps mes principales contributions à partir de 1988, date à laquelle j'ai commencé ma Thèse de Doctorat au LAAS-CNRS.

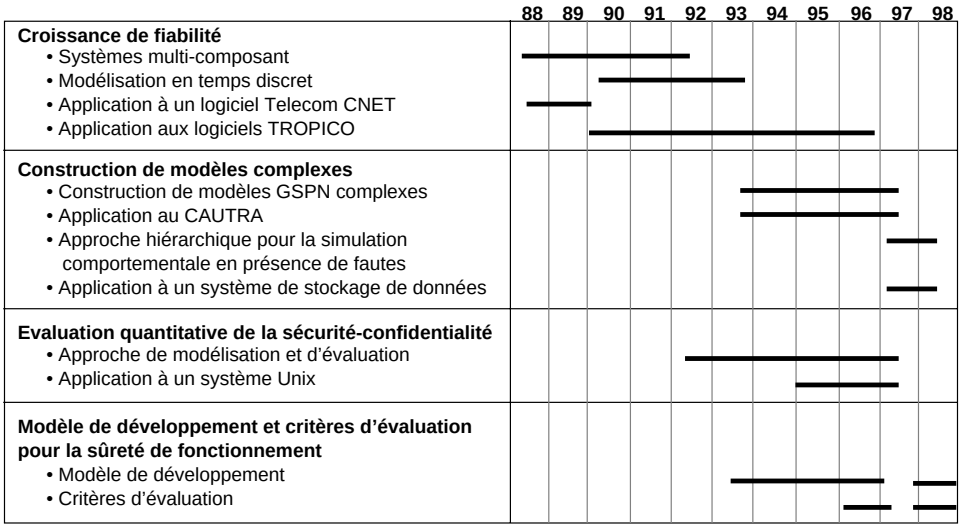


Figure 1.1– Chronologie des travaux

2

2. MODÈLES COMPLEXES POUR LA SÛRETÉ DE FONCTIONNEMENT

Ce chapitre synthétise nos travaux sur la spécification et la construction de modèles complexes, issus de la description d'architectures tolérantes aux fautes, pour l'analyse et l'évaluation de la sûreté de fonctionnement. Nous considérons le cas où le système analysé est en fiabilité stabilisée. Ces modèles sont typiquement utilisés durant la phase de conception pour comparer différentes architectures possibles du système et analyser l'impact des fautes susceptibles d'affecter la sûreté de fonctionnement. Deux approches complémentaires sont proposées et discutées : la première est basée sur les réseaux de Petri stochastiques et la seconde est basée sur la simulation comportementale et hiérarchique en présence de fautes.

Les travaux sur la modélisation par réseaux de Petri ont été effectués dans le cadre de la thèse de Doctorat de Nicolae Fota dont j'ai assuré l'encadrement conjointement avec Karama Kanoun. Les travaux sur la simulation hiérarchique ont été menés durant mon séjour au "Center for Reliable and High Performance Computing" de l'université d'Illinois à Urbana-Champaign, aux Etats-Unis.

2.1. Modélisation par réseaux de Petri stochastiques

Les réseaux de Petri stochastiques (RdPS) [67] et leurs extensions, tout particulièrement les réseaux de Petri stochastiques généralisés (dénommés GSPN pour "Generalized Stochastic Petri Nets") [149, 150], sont bien adaptés pour la construction de modèles d'évaluation de la sûreté de fonctionnement de systèmes en tenant compte des dépendances stochastiques qui peuvent résulter des communications entre les composants, de l'architecture (répartition des composants logiciels sur les composants du matériel), des procédures de tolérance aux fautes et de maintenance, etc. [147, 172]. Les GSPN permettent de générer automatiquement une chaîne de Markov à partir d'une description du comportement du système et des interactions entre ses composants, et offrent des moyens d'analyse et de vérification structurelle des modèles. Il existe

plusieurs outils pour construire des modèles basés sur des RdPS et leurs extensions, par exemple, GreatSPN [37], SURF2 [15], SPNP [41], UltraSan [183].

Un aspect délicat dans la modélisation par GSPN concerne la maîtrise de la construction et du traitement de modèles complexes. La complexité résulte généralement du niveau de détail de la modélisation, du nombre de composants à modéliser explicitement et de leurs interactions. Différentes méthodes ont été proposées pour maîtriser la complexité des modèles [13], dont la plupart sont basées sur le principe de décomposition et d'agrégation qui consiste à ne pas générer le modèle global, mais à construire des sous-modèles et à combiner les mesures obtenues par le traitement des sous-modèles afin de calculer les mesures du système global. On peut citer par exemple, les travaux basés sur les principes de quasi-indépendance [42], les automates stochastiques superposés [58], la décomposition temporelle [2, 17], des modélisations hiérarchiques et hybrides combinant des réseaux de Petri et d'autres formalismes tels que files d'attente [14], arbres de fautes et diagrammes de fiabilité [12, 147], des méthodes exploitant des symétries et des propriétés structurelles du modèle [38, 197], etc. De façon générale, ces travaux imposent des restrictions sur les interactions entre sous-modèles qui ne sont pas toujours satisfaites quand on construit des modèles de sûreté de fonctionnement de systèmes réels induisant de fortes dépendances. Dans ce contexte, il est nécessaire de construire le modèle global et d'évaluer les mesures de sûreté de fonctionnement à partir de ce modèle. Quelques méthodes définissant des règles de construction modulaire de modèles GSPN ont été proposées pour faciliter l'élaboration de modèles complexes (cf. par exemple [20, 54, 111, 157, 175, 182]). Notre expérience dans la modélisation de systèmes réels tolérants aux fautes pour lesquels une modélisation fine des composants et de leurs interactions est requise a montré les limites de certaines de ces méthodes [68, 69]. Dans ce contexte, l'utilisateur doit souvent gérer un ensemble important d'hypothèses de défaillances et de restaurations et différents types de dépendances qu'il faut représenter dans le modèle. Construire directement le modèle sans s'appuyer sur une méthode systématique et structurée est souvent voué à l'échec. Par ailleurs, la prise en compte des dépendances se traduit par un nombre important de places, de transitions et surtout d'arcs qui rendent le modèle illisible, difficile à comprendre, à vérifier et à mettre à jour. Par conséquent, la définition d'un formalisme de spécification permettant une description structurée et de haut niveau des comportements à prendre en compte dans les modèles faciliterait certainement la construction et la validation de ces modèles.

Nos travaux qui sont résumés dans les paragraphes suivants s'inscrivent dans cette optique. Nous avons développé une *méthode de construction et de validation incrémentale* de modèles GSPN qui consiste à élaborer le modèle de façon progressive. La méthode définit un ensemble de règles pour structurer le modèle sous forme de modules représentant le comportement des composants du système et interconnecter ces modules par des mécanismes élémentaires de couplage afin de représenter leurs interactions. Pour faciliter la mise en œuvre de la méthode, nous l'avons complétée par la définition d'un *formalisme de spécification* qui offre d'une part, une *notation* permettant d'avoir une description structurée et de haut niveau des modules et de leurs interactions, et d'autre part, *de règles de transformation* de la spécification permettant d'obtenir de manière quasi directe le modèle GSPN correspondant.

La méthode et le formalisme de spécification sont présentés dans [68-70]. Nous les avons appliqués pour construire des modèles afin d'évaluer la sûreté de fonctionnement du système informatique de contrôle en route du trafic aérien français (le CAUTRA). Dans la suite, nous résumons les grandes lignes de ces travaux.

2.1.1. Construction modulaire et incrémentale de modèles GSPN

Nous proposons de procéder par étapes pour construire et valider un modèle complexe. À l'étape initiale, on choisit un composant du système et, on construit un GSPN décrivant son comportement en supposant que les autres composants sont dans un état de fonctionnement nominal. Les hypothèses de défaillance et de restauration de ces derniers sont incorporées de façon progressive durant les autres étapes de la modélisation. Á chaque nouvelle étape, on ajoute un nouveau composant et, on met à jour le modèle GSPN construit et validé à l'étape précédente. Il s'agit en particulier de modéliser les interactions entre ce composant et ceux déjà pris en compte dans les étapes précédentes. La validation du modèle est effectuée au niveau du GSPN via l'utilisation de techniques d'analyse et de vérification structurelles, et également au niveau de la chaîne de Markov, en analysant les scénarios décrits dans le modèle. Cette étape est itérée jusqu'à l'intégration finale de tous les composants du système. Le modèle final décrit alors le comportement global du système en tenant compte de toutes les interactions entre les composants.

L'approche incrémentale permet de maîtriser la construction et la validation du modèle puisque, à chaque étape, on ajoute uniquement un nombre réduit d'hypothèses. Pour faciliter la construction des modèles GSPN à chaque étape de l'approche incrémentale et la mise à jour du modèle d'une étape à une autre, nous avons défini un ensemble de règles pour décrire le comportement des composants et leurs interactions.

À chaque composant, on associe un sous-modèle GSPN appelé module qui décrit son comportement. Les interactions entre les composants sont représentées par le biais de mécanismes élémentaires de couplage des modules.

Modules. Un module est constitué d'un ensemble de places et de transitions instantanées ou temporisées. Les transitions représentent les événements qui conduisent à l'évolution de l'état du composant. Quand l'occurrence d'un événement conduit à l'évolution simultanée de l'état de plusieurs composants, il peut être représenté par une transition commune aux modules associés à ces composants. La seule condition imposée à chaque module est qu'il doit avoir son invariant de marquage égal à 1. Cette condition facilite la description des interactions entre les modules et permet une formalisation de ces interactions, basée sur la logique booléenne. Une telle condition bien qu'elle exclut à première vue l'exploitation de symétries dans le modèle n'est pas très contraignante, dans la mesure où de telles symétries sont rares dans des modèles de sûreté de fonctionnement décrivant de façon fine le comportement des systèmes.

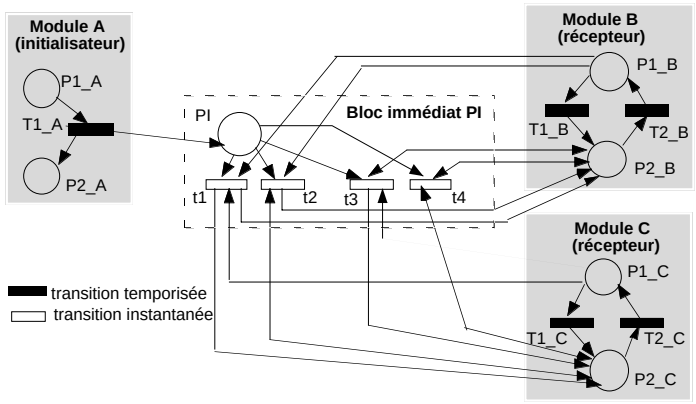
Couplage des modules. Trois mécanismes élémentaires sont utilisés pour le couplage des modules : les *tests de marquages*, qui sont utilisés quand l'occurrence d'un évènement dans un composant est conditionnée par l'état d'autres composants, les

transitions communes, qui modélisent des événements communs à plusieurs modules et les *blocs d'interconnexion*. Ces derniers modélisent les conséquences d'un événement se produisant dans un composant source (appelé initialisateur) sur l'état d'autres composants (appelés récepteurs). Ces conséquences peuvent être conditionnées par l'état d'autres composants du système. Les tests de marquage sont alors utilisés pour traduire ces conditions. Un bloc relie un ou plusieurs modules initialisateurs à un ou plusieurs modules récepteurs. Il est constitué d'une place d'entrée et d'un ensemble de transitions instantanées. Les conséquences d'un événement d'un module initialisateur sur les modules récepteurs modélisées par des blocs, peuvent être de deux types : immédiates (modélisées par des *blocs immédiats*) ou différées jusqu'à ce que les modules récepteurs atteignent une classe de marquages spécifiques (modélisées par des *blocs à effet différé*). Pour chacun de ces deux types de blocs, nous avons défini un ensemble de règles permettant de faciliter leur construction. A titre d'exemple, pour concevoir un bloc immédiat, il faut s'assurer que, quel que soit le marquage du modèle tel que la place d'entrée du bloc est marquée, il existe une et une seule transition du bloc franchissable à partir de ce marquage. Par conséquent, il faut identifier a priori tous les marquages possibles des modules intervenant dans l'interaction modélisée par le bloc et associer une transition instantanée à chacun de ces marquages. Pour identifier le nombre de transitions instantanées nécessaires, il suffit de calculer le produit cartésien des marquages des places du module récepteur et des places des autres modules conditionnant l'évolution du module récepteur suite au franchissement de la transition qui a initialisé le bloc. Une formalisation de cette condition basée sur la logique binaire et des exemples d'illustration sont présentés dans [68].

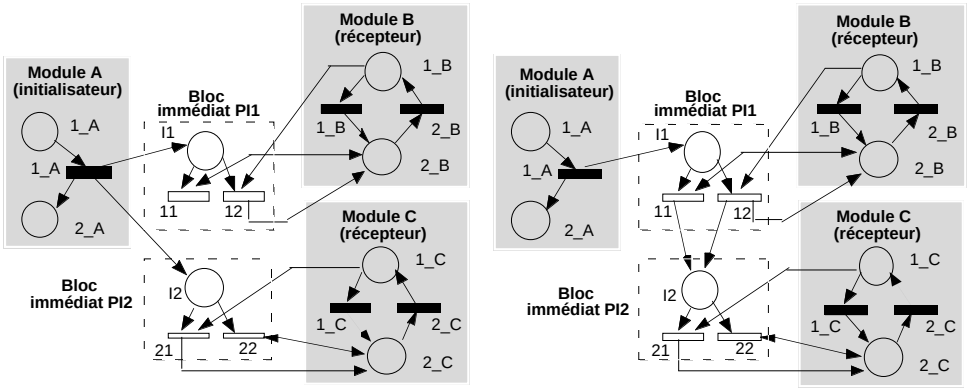
Des blocs d'interconnexion génériques peuvent être définis pour modéliser certains types d'interaction. Par exemple, arrêt des répliques logicielles quand le calculateur défaille, relance automatique du logiciel suite à une détection d'erreur, etc. Dans le cas d'une interaction complexe, il est souvent plus utile et efficace de modéliser cette interaction par plusieurs blocs élémentaires disposés en série ou en parallèle au lieu d'utiliser un bloc unique. Quelques précautions sont nécessaires pour éviter de faire des erreurs lors de la définition et de l'enchaînement de ces blocs. En particulier, des conflits entre les transitions instantanées des différents blocs peuvent apparaître et des priorités de franchissement doivent être définies pour gérer ces conflits. Il est à noter que la décomposition peut être facilitée par l'utilisation des arbres de décision [1, 22]. Un ensemble de règles de construction pour guider la décomposition et des exemples sont présentés dans [68]. La figure 2.1 donne un exemple simple d'illustration qui décrit le cas où la défaillance d'un composant A conduit à l'arrêt des composants B et C. La figure 2.1-a correspond au cas où cette interaction est décrite par un seul bloc, et les figures 2.1-b et 2.1-c donnent le modèle équivalent quand on décompose le bloc en deux blocs élémentaires en parallèle et en série, respectivement.

L'utilisation de blocs d'interconnexion en parallèle ou en série comme alternative à la modélisation par un bloc unique est recommandée dans la mesure où cela favorise la construction de blocs réutilisables pour représenter d'autres types d'interactions, améliorant ainsi la lisibilité et la concision du modèle. Par exemple, le bloc immédiat PI2 de la figure 2.1-c peut être réutilisé quand on a besoin d'arrêter le composant C

uniquement sans modifier l'état du composant B. Il est à noter que la réutilisation est plus facile quand les blocs sont en parallèle que quand ils sont en série. D'autres règles permettant d'optimiser la construction des GSPN et un exemple d'application de ces règles à un système duplex tolérant aux fautes sont présentées dans [68].



a. Représentation de l'interaction par bloc immédiat unique



. Représentation par blocs en parallèle

. Représentation par blocs en série

Figure 2.1-Décomposition série-parallèle : Exemple

2.1.2. Spécification de haut niveau

La mise en œuvre de la méthode proposée au § 2.1.1, avec les outils classiques de génération de GSPN conduit à des modèles dont la description graphique devient rapidement inexploitable, en particulier à cause du nombre important de transitions instantanées et de tests de marquages nécessaires pour décrire les interactions entre les composants. Afin de faciliter la mise en œuvre de la méthode, nous avons défini un *formalisme de spécification* constitué :

- d'une *notation* permettant d'obtenir une description structurée et de haut niveau des modules et de leurs interactions ;
- des *règles de transformation* permettant de générer automatiquement un modèle GSPN par traduction directe de la spécification structurée.

La figure 2.2 présente de façon schématique les principes de notre approche. À chaque module, on associe une représentation graphique appelée *diagramme d'évolution* qui est détaillée par une spécification textuelle basée sur un langage que nous avons défini. Les interactions entre modules sont identifiées dans les spécifications des modules, sous la forme de fonctions logiques basées sur des prédicats, spécifiant les conditions d'activation d'événements ou de leurs conséquences, ou bien d'initialisations de blocs d'interconnexion et d'identification des modules récepteurs correspondant aux interactions modélisées. La spécification détaillée des blocs et de leurs effets est décrite de façon textuelle.

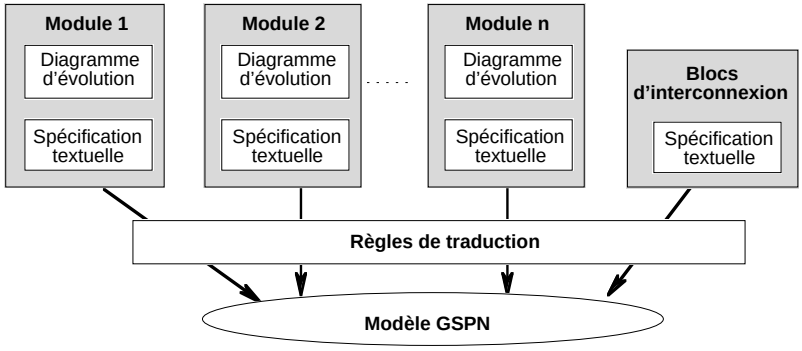


Figure 2.2 – Vue schématique de l'approche de spécification de modèles GSPN

Dans la suite, nous décrivons uniquement les principes du diagramme d'évolution et de spécification textuelle, que nous illustrons par un exemple simple. L'ensemble de l'approche et les règles de traduction sont détaillés dans [68].

Un diagramme d'évolution est un graphe orienté constitué de deux types de nœuds : des *phases* et des *fonctions d'évolution*, interconnectées par des arcs. Une phase représente une classe d'états du composant à partir desquels les mêmes événements ou types d'événements peuvent se produire. Cependant, les changements d'état suite à l'occurrence de ces événements, et parfois les paramètres (taux ou probabilités) associés à ces changements d'état peuvent dépendre du comportement antérieur du système. Pour différencier les états d'une même phase, nous avons introduit des variables spécifiques appelées *mémoires*. L'état du système est alors déterminé par le marquage des phases et des mémoires de ses différents composants.

À chaque phase, on associe une fonction d'évolution qui est spécifiée de façon textuelle en considérant trois niveaux hiérarchiques :

- le **niveau 1** spécifie les *événements* possibles à partir des états de la phase associée, les conditions d'activation spécifiées sous forme d'une fonction logique, et enfin les

paramètres stochastiques caractérisant l'occurrence de l'événement ; les conséquences de chaque événement sont exprimées par les niveaux 2 et 3 ;

- le **niveau 2** spécifie les conséquences de l'événement considéré sur la phase courante et les conditions associées à chacune de ces conséquences exprimées sous la forme d'une fonction logique ;
- le **niveau 3** décrit les conséquences de l'événement sur l'état des autres composants du système sous forme d'initialisation de blocs d'interconnexion.

Chaque diagramme d'évolution possède une *interface* qui spécifie les interactions du module avec les autres modules. L'interface définit : 1) les événements du module qui sont communs à d'autres modules, 2) les blocs d'interconnexion initialisés par d'autres modules qui ont des conséquences sur l'état du module considéré, et enfin, 3) les blocs initialisés par le module et les modules récepteurs associés à ces blocs. La spécification des blocs d'interconnexion, des priorités définissant l'ordre selon lequel les blocs sont initialisés et des conditions d'initialisation des blocs est faite textuellement.

Exemple : La figure 2.3 donne un exemple de diagramme d'évolution. La zone en gris constitue le corps du digramme d'évolution qui représente le comportement du module A résultant de l'occurrence de ses événements propres. P1_A, P2_A et P3_A sont trois phases décrivant les états du composant A et OUT_P1_A et OUT_P2_A, sont les fonctions d'évolution associées à P1_A et P2_A, respectivement. L'interface du diagramme d'évolution est matérialisée graphiquement par :

- la fonction d'évolution **(OUT_Pk_D)** OUT_P2_A qui indique l'existence d'événement(s) commun(s) à partir de P2_A et de la phase Pk_D du module D,
- l'arc B1 **{C}** en sortie de OUT_P1_A qui indique qu'un des événements définis par OUT_P1_A a des conséquences sur le module C qui sont modélisées via un ou plusieurs blocs d'interconnexion en série, dont le premier initialisé est B1,
- la zone hachurée en dessous du corps du diagramme d'évolution, qui indique que des événements activés dans les modules E et F ont des conséquences sur le module A. Pour le module E, ces conséquences sont modélisées par l'enchaînement en série des bloc B2 et B3. Pour le module F, l'interaction est décrite directement par B3.

On peut noter qu'il n'y a pas de fonction d'évolution associée P3_A. Ceci traduit le fait que les évolutions d'état éventuelles à partir de cette phase P3_A ne peuvent résulter que de l'occurrence d'événements dans d'autres modules. Dans l'exemple, ces conséquences proviennent d'événements activés dans les modules E ou F.

La représentation graphique du diagramme d'évolution donne une description concise et de haut niveau du comportement de chacun des modules. La description détaillée des modules et de leurs interactions est fournie par la spécification textuelle. Nous avons défini un langage spécifique pour la description de cette spécification dont une présentation détaillée est donnée dans [68].

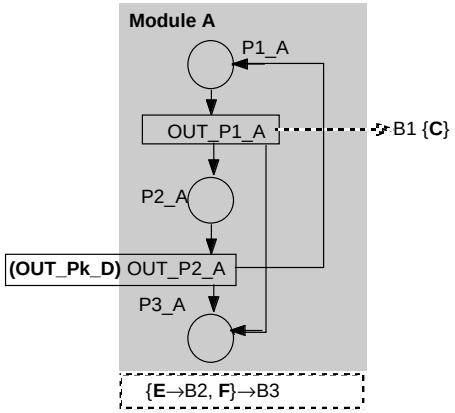


Figure 2.3 – Exemple de diagramme d'évolution

La spécification d'une fonction d'évolution se présente sous la forme suivante :

OUT_P1_A

Ev1 : E_1 (IF fonction logique_1) [p_1, λ_1]

- IF fonction logique_11 DO conséquence_11 sur *P1_A*

  [INIT blocs_11]
- ……
- ELSE DO conséquence_1m sur *P1_A*

  [INIT blocs_1m]

……

Evn : E_n (IF fonction logique_n) [p_n, λ_n]

- ……

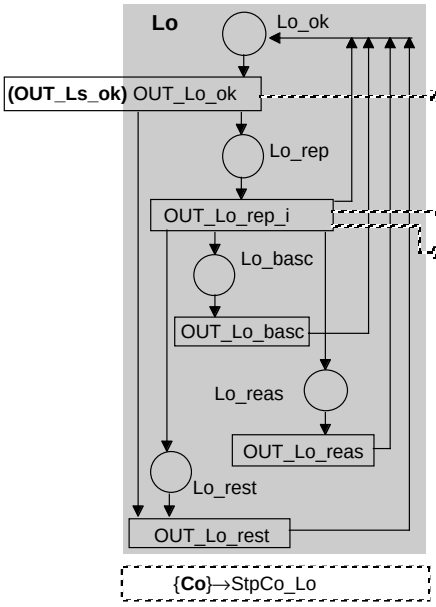
L'interprétation de cette spécification est la suivante :

- OUT_P1_A est la fonction d'évolution associée à la phase P1_A.
- Les symboles "Ev i" identifient le **niveau 1** de structuration des fonctions d'évolution spécifiant les événements possibles à partir de cette phase. E_1 est l'identificateur de l'événement 1 et fonction logique_1 exprime les conditions d'activation de E_1 sous la forme d'une combinaison, avec les opérateurs "$\wedge$" et "$\vee$", de variables booléennes correspondent à l'état des phases et des mémoires du modèle. L'opérateur IF exprime que E_1 n'est activé que si fonction logique_1 prend la valeur "VRAI". Le couple [p_1, λ_1] spécifie les paramètres stochastiques de E_1. : λ_1 est le taux d'occurrence de E_1 et p_1 est la probabilité de sélection de cet événement quand on utilise un sélecteur aléatoire. Par défaut, p_1 prend la valeur 1.
- Les symboles "•" identifient le **niveau 2** de structuration de la fonction d'évolution ; pour l'événement E_1, la spécification (IF fonction logique_11 DO conséquence_11 sur *P1_A*) décrit les conséquences de E_1 sur la phase P1_A en fonction des conditions spécifiées par fonction logique_11.

- Le **niveau 3** de la structuration de la fonction d'évolution, relatif à E_1 et aux conséquence_11 sur la phase courante, décrit les conséquences éventuelles sur les autres modules sous forme d'initialisations de blocs ([INIT blocs_11]).

Exemple : Considérons le diagramme d'évolution de la figure 2.4. Il est extrait de l'exemple traité dans [68] qui correspond à une architecture duplex constituée de deux répliques logicielles (une opérationnelle, notée Lo, et une de secours, notée Ls), deux calculateurs Co et Cs et deux disques Do et Ds. Le diagramme d'évolution décrit le comportement de Lo avec les hypothèses présentées dans [68]. Il faut noter que notre objectif ici est de donner quelques exemples de spécification et non pas d'expliquer le modèle. Á partir de la phase Lo_ok, deux types de défaillances sont distingués : des défaillances locales ou des défaillances de mode commun avec Ls. Quatre procédures de recouvrement de Lo sont distinguées et leur aboutissement dépend de l'état des autres composants et de celui des données de secours stockées sur les disques : reprise automatique (Lo-rep), basculement des répliques Lo—Ls (Lo_basc), restauration locale manuelle (Lo_rest), ou bien relance de Lo avec les données stockées sur Ds (Lo_reas).

Les spécifications textuelles des fonctions d'évolution associées aux phases du module Lo sont données dans la figure 3.4. Tous les noms de fonctions d'évolution, phases et mémoires concernant les autres modules ainsi que les initialisations de blocs incluant des conditions externes sont écrits **en gras.** Les opérateurs "»" et ">" correspondent à des évolutions d'état liées à l'occurrence d'événements temporisés ou instantanés, respectivement. Pour la fonction OUT_Lo_ok, on peut constater que le taux d'occurrence des défaillances locales est spécifié en fonction du marquage de Ls (exprimé par m(Ls_ok)). L'occurrence d'une défaillance de mode commun se traduit par l'évolution de Lo de la phase Lo_ok vers Lo_rest, et simultanément par l'évolution de Ls de la phase Ls_ok vers Ls_hs. Cette défaillance conduit aussi à l'initialisation du bloc +M_di qui a pour rôle de mémoriser que les données de secours sont indisponibles. On peut constater aussi que, seul l'événement Ev1 associé à Lo_rest possède une condition d'activation associée (/Co_d) qui exprime que la restauration manuelle de Lo ne peut pas être effectuée si le calculateur Co est défaillant. Dans tous les autres cas, les événements sont activés à partir du moment où le composant se trouve dans la phase correspondante. Notons enfin que la défaillance du calculateur Co conduit à l'évolution de Lo vers la phase Lo_rest (ceci est modélisé par le bloc StpCo_Lo), et l'échec de la reprise automatique de Lo conduit au basculement des répliques qui nécessite l'arrêt de Ls et ensuite sa relance en mode opérationnel (l'arrêt de Ls est modélisé par le bloc immédiat StpLo-Ls dont la spécification est donnée sur la figure 2.4).



Lo_ok	Lo en cours de fonctionnement
Lo_rep	Lo en cours de reprise automatique
Lo_basc	Basculement de répliques Lo—Ls
Lo_rest	Lo en cours de restauration manuelle
Lo_reas	Commutation sur Ds et relance de Lo
StpLo_Ls	Arrêt de Ls suite à défaillance de Lo
StpCo_Lo	Arrêt de Lo suite à défaillance de Co
Chg_don	Changement état données secours
+M_di	Mémorisation indisponibilité données secours

Fonctions d'évolutions de Lo

OUT_Lo_ok

Ev 1 : Défaillance de Lo en mode local [1, λ_Lo_loc*m(Ls_ok)+(λ _Lo_loc+ λ _Lo_Ls)*(1-m(Ls_ok))]

- DO Lo_ok » Lo_rep

Ev 2 : **(OUT_Ls_ok)** Défaillance mode commun de Lo et Ls [1, λ _Lo_Ls]

- DO (Lo_ok » Lo_rest , **Ls_ok » Ls_hs**)

  [INIT **+M_di**]

OUT_Lo_rep

Ev 1 : Succès reprise automatique de Lo [p_1, rep]

- DO Lo_rep » Lo_ok

  [INIT **Chg_don**]

Ev2 : Echec reprise automatique de Lo [1-p_1, rep]

- IF (**/M_di** ∧ **Cs_ok**) DO Lo_rep » Lo_basc

  [INIT **StpLo_Ls**]

- IF (**/M_di** ∧ **/Cs_ok**) DO Lo_rep » Lo_reas

  [INIT **StpLo_Ls**]

- ELSE DO Lo_rep » Lo_rest

OUT_Lo_basc

Ev1 : Basculement des répliques [1, basc]

- Lo_basc » Lo_ok

OUT_Lo_rest

Ev1 : Restauration de Lo (IF **/Co_d**) [1, rest]

- DO Lo_rest » Lo_ok

OUT_Lo_reas

Ev1 : Commutation Ds et relance Lo [1, reas]

- DO Lo_reas » Lo_ok

Spécification du bloc StpLo_Ls

Imm StpLo_Ls = IF Ls_ok DO (Ls_ok > Ls_hs, INIT +M_di)
ELSE INIT +M_di

Figure 2.4 – Diagramme d'évolution de Lo et spécification des fonctions d'évolution de Lo et du bloc StpLo_Ls

2.1.3. Application au CAUTRA

Nous avons appliqué la méthode présentée aux § 2.1.1 et § 2.1.2 pour modéliser la sûreté de fonctionnement du système informatique de contrôle en route du trafic aérien français (dénommé CAUTRA). Cette étude a été faite dans le cadre d'un contrat de recherche avec le Centre d'Études de la Navigation Aérienne (CENA). Le CAUTRA fournit une assistance automatisée aux contrôleurs et aux régulateurs du trafic afin d'assurer le contrôle du trafic aérien dans des conditions prescrites de sécurité et de régularité. Il est mis en œuvre sur des calculateurs tolérants aux fautes, qui sont répartis géographiquement dans cinq centres de contrôle en route régionaux (CCR) et un centre d'exploitation des systèmes de la navigation aérienne centraux (CESNAC), interconnectés via un réseau de télécommunication dédié [71]. L'objectif de l'étude a été d'évaluer l'impact sur la sécurité du trafic, des défaillances matérielles et logicielles et des procédures de restauration associées. Nous avons défini cinq classes de dégradation des services fournis aux contrôleurs pour assurer la sécurité du trafic. L'objectif de la modélisation a été d'identifier les scénarios conduisant à chacune de ces classes et de chiffrer leurs impacts en considérant deux mesures principales : l'indisponibilité du service et la fréquence d'occurrence en régime stationnaire de la classe correspondante. Nous avons utilisé l'approche incrémentale et le formalisme de spécification pour la construction des modèles de sûreté de fonctionnement des différents sous-systèmes du CAUTRA qui ont une influence significative sur la sécurité du trafic [72]. Il s'agit en particulier du Système de Traitement Initial des Plans de Vols (STIP) [73] du CESNAC et des Systèmes de Traitement Radar et de Traitement des Plans de Vol (STR-STPV) mis en œuvre dans chaque CCR [70]. La méthode utilisée s'est avérée efficace pour maîtriser la complexité de ce système. Par exemple, le modèle STR-STPV a été construit en 18 étapes correspondant à l'intégration progressive de 18 modules décrivant l'architecture de ce système. Ces modules correspondent à trois calculateurs et trois stations Unix en configuration TMR, trois répliques de l'application STR, deux répliques de l'application STPV, trois répliques du système d'exploitation et de supervision, deux platines de commutation de redondances, un réseau Ethernet local, et enfin un composant de gestion des configurations des répliques logicielles sur les calculateurs hôtes. L'architecture, les hypothèses de modélisation et les modèles GSPN sont détaillés dans [68, 70]. La figure 2.5 résume les étapes de la modélisation en donnant à chaque étape le nombre de places et de transitions des modèles et le nombre d'états de la chaîne de Markov correspondante. Les GSPN correspondant aux quatre premières étapes sont présentés dans [70]. À partir de l'étape 11, nous avons appliqué un algorithme de troncature qui consiste à ne considérer que les scénarios où le nombre de défaillances successives est inférieur ou égal au niveau de troncature spécifié. Cet algorithme facilite la génération et le traitement de la chaîne de Markov en négligeant certains états qui n'ont pas une contribution significative aux mesures évaluées. La chaîne de Markov correspondant au modèle GSPN global du STR-STPV, obtenue après réduction, en ne gardant que les états les plus significatifs, comporte 22831 états. Ce modèle nous a permis en particulier, d'évaluer différentes stratégies de tolérance aux fautes, et de comparer plusieurs cohabitations possibles des répliques logicielles sur les calculateurs hôtes (cf. [70]). Les spécifications des modèles STIP et STR_STPV, ainsi que les résultats obtenus pour le CAUTRA global à partir de la combinaison des

résultats fournis par les modèles du STIP, STR-STPV et le réseau de télécommunication sont présentés dans [74]. La construction et le traitement des modèles GSPN ont été effectués avec l'outil SURF2 [15].

Etape	Composant ajouté	Niveau troncature	# Places	#Transitions	#Etats Markov
1	LRp	-	2	2	2
2	LRs	-	6	9	4
3	LRt	-	10	16	10
4	LBt	-	13	20	20
5	LBs	-	17	36	34
6	LBp	-	28	86	42
7	COHAB	-	39	110	252
8	LVp	-	66	198	880
9	LVs	-	72	208	1443
10	PR	-	77	220	5751
11	LAN	4	82	235	5173
12	PV	4	87	249	11974
13	DGt	4	90	255	16155
14	DGs	4	99	310	23218
15	DGp	3	109	394	11670
16	WSt	3	112	414	15295
17	WSs	3	114	440	20357
18	WSp	3	116	501	22831

Figure 2.5 - Etapes de la construction incrémentale du modèle STR-STPV

2.1.4. Conclusion

La construction de modèles GSPN pour l'évaluation de la sûreté de fonctionnement de systèmes réels est une tâche fastidieuse qui nécessite un investissement important de la part des utilisateurs. Pour permettre une meilleure intégration des RSPG dans le processus de conception d'architectures tolérantes aux fautes afin de fournir des évaluations de sûreté de fonctionnement, il est nécessaire de définir des moyens permettant de faciliter leur utilisation. La méthode de construction modulaire et incrémentale et l'approche de spécification que nous avons définies visent à atteindre cet objectif. Le système CAUTRA qui nous a servi de cas d'étude pour expérimenter l'applicabilité de nos travaux est tout à fait représentatif de la complexité des comportements à prendre en compte quand on modélise la sûreté de fonctionnement d'architectures tolérantes aux fautes mises en œuvre dans des systèmes réels. Les résultats de cette expérience ont montré le bien fondé de notre approche pour maîtriser la complexité des modèles. Pour la construction des modèles du CAUTRA, nous avons appliqué de façon manuelle les règles de description structurée des modèles et la traduction en Petri de la spécification. Le développement d'un prototype mettant en œuvre les principes de la méthode fait partie de nos prospectives. Un tel prototype pourrait incorporer des moyens de vérification automatique, au niveau de la spécification textuelle, des règles de construction que nous avons définies. Il est intéressant de

mentionner aussi que le formalisme de spécification peut servir à construire d'autres types de modèles (par exemple, des réseaux d'activités stochastiques SAN [152, 183], ou bien directement une chaîne de Markov), moyennant la définition de règles de transformation spécifiques à chaque type de modèle. Nous avons testé cette possibilité lors du travail mené dans [151] où des modèles SAN ont été générés à partir de la spécification qui nous a servi pour construire un GSPN.

2.2. Simulation comportementale en présence de fautes

L'évaluation de la sûreté de fonctionnement par une approche analytique basée sur une modélisation par chaînes de Markov ou par GSPN repose sur une représentation du comportement du système à un haut niveau d'abstraction de telle sorte que les modèles soient effectivement exploitables. Ces modèles nécessitent la définition d'un ensemble de paramètres pour caractériser les processus qui sont mis en jeu. Certains paramètres, en particulier ceux qui caractérisent l'efficacité des mécanismes de tolérance aux fautes, peuvent être issus de l'analyse du comportement du système à un niveau de détail beaucoup plus fin. L'injection de fautes constitue une approche privilégiée quand il s'agit d'analyser le comportement en présence de fautes, d'évaluer la latence d'erreur et l'efficacité des mécanismes de traitement d'erreurs et de fautes, etc. [5, 90]. Pendant la conception, l'injection de fautes est réalisée dans des modèles de simulation. Différents niveaux d'abstraction peuvent être considérés pour la simulation, allant du niveau physique jusqu'au niveau fonctionnel et comportemental. Plusieurs outils et environnements d'injection de fautes dans des modèles de simulation ont été développés récemment pour analyser des systèmes tolérants aux fautes [6, 44, 93], par exemple FOCUS [39], MEFISTO [94], MEFISTO-L [21], REACT [43], DEPEND [81].

Les travaux présentés dans la suite concernent la simulation comportementale en présence de fautes. Le principal défi quand on fait ce type de simulation est d'arriver à décrire le comportement à un niveau de détail relativement fin, afin d'analyser les effets des fautes au plus près de là où elles sont injectées et d'étudier leurs impacts au niveau système, tout en maîtrisant le temps de la simulation. Le compromis entre complexité du modèle de simulation et temps de la simulation n'est pas facile à trouver et la solution dépend généralement du problème particulier qui est étudié. Différentes techniques ont été développées pour optimiser la simulation, on peut citer par exemple, la simulation hiérarchique de modèle [80], l'utilisation de techniques de réduction de la variance et des facteurs d'importance [83, 139], la simulation parallèle et distribuée [75].

Les travaux résumés dans la suite concernent le développement et l'application d'une approche de simulation hiérarchique pour l'analyse de la sûreté de fonctionnement d'un système de stockage de données intégrant des mécanismes de tolérance aux fautes. L'approche consiste à décrire le comportement du système en présence de fautes à différents niveaux d'abstraction. Un modèle de simulation est associé à chaque niveau et les résultats de la simulation d'un modèle donné sont ensuite utilisés comme paramètres dans le modèle décrivant le comportement à un niveau d'abstraction plus élevé. Dans le cadre de cette étude, nous avons utilisé DEPEND pour la construction et la simulation des modèles [81]. DEPEND a été développé à l'Université d'Illinois à Urbana-

Champaign. Il utilise le langage C++ pour la description de modèle, et offre des fonctions génériques pour modéliser des systèmes tolérants aux fautes, injecter des fautes dans des modèles fonctionnels et comportementaux, analyser la détection et le recouvrement d'erreur, etc. Les résultats de cette étude sont décrits dans [108], nous en donnons les grandes lignes et quelques résultats dans la suite de ce chapitre.

2.2.1. Présentation du système et objectifs de l'étude

Il s'agit d'un système commercial de stockage de données basé sur une architecture RAID intégrant un cache de large capacité. Ce système est utilisé dans des applications critiques et possède des exigences fortes de sûreté de fonctionnement et de performance. Un RAID est constitué d'un ensemble de disques organisés sous forme d'une matrice donnant l'illusion d'un seul disque virtuel de large capacité. Des informations redondantes sont stockées sur les disques afin de restaurer les données perdues suite à des défaillances de disque ou à des fautes affectant des segments de données [33]. L'adjonction d'un cache de large capacité à un RAID permet d'améliorer de façon significative les performances du système. Cependant, le cache étant un composant central dans ce type d'architecture, sa sûreté de fonctionnement est une condition nécessaire pour le bon fonctionnement du système. Dans l'architecture que nous avons étudiée (cf. Figure 2.6), le cache est constitué de cartes mémoire et d'un contrôleur comprenant des interfaces avec la mémoire cache, les systèmes hôtes et les disques, et de deux bus à haute vitesse interconnectant l'ensemble. Pour garantir l'intégrité et la disponibilité des données, le cache est doté de plusieurs mécanismes de détection et de recouvrement d'erreur, en plus de ceux mis en œuvre dans les disques. En particulier, la mémoire cache et les transferts à travers "Bus2" sont protégés par un code détecteur d'erreurs triples et correcteur d'erreurs doubles (EDAC), "Bus1" est protégé par un code de parité et les données transférées entre le cache et les systèmes hôtes ou les disques sont protégées par des codes CRC (cf. [108] pour plus de détails).

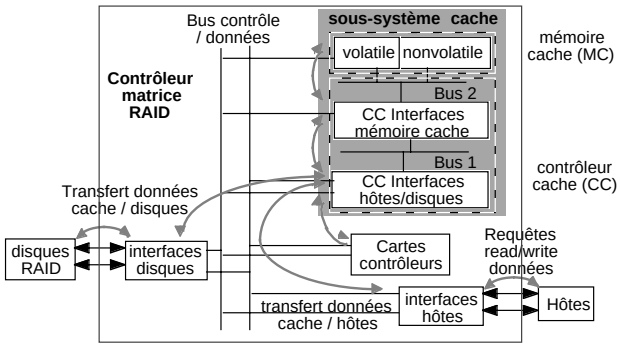


Figure 2.6 – Architecture du système

L'objectif de l'étude a été de simuler le fonctionnement du système en injectant des fautes dans le cache et dans les disques afin d'analyser l'efficacité des mécanismes de détection et de recouvrement d'erreur et d'évaluer la latence d'erreur. La latence d'erreur (temps entre l'occurrence et la détection ou l'élimination de l'erreur) est un paramètre

important pour la conception de systèmes tolérant les fautes dans le sens où si elle est trop élevée, le risque que les erreurs s'accumulent et mettent en défaut la tolérance aux fautes peut être important [7, 8, 35, 36, 63]. Afin d'analyser le système dans des conditions représentatives de son environnement réel, nous avons utilisé des traces de sollicitation (requêtes de transfert de segments de données en lecture ou en écriture), enregistrées en opération, pour simuler les transferts de données entre les entités du système.

2.2.2. Modélisation hiérarchique

La capacité de la mémoire cache et du RAID est de l'ordre de plusieurs Giga-Octets. Pour être en mesure d'évaluer avec précision la couverture des mécanismes de détection et la latence, il est nécessaire de simuler le fonctionnement du système en considérant une granularité des données égale à celle considérée pour le calcul des codes de détection d'erreur (c'est-à-dire, quelques octets). Cependant, la simulation du comportement du système à un tel niveau de détail est infaisable pour deux raisons : 1) la mémoire à allouer pour la simulation dépasse les capacités matérielles des machines et, 2) il est difficile d'obtenir dans un temps raisonnable des données statistiques suffisantes pour estimer les mesures recherchées.

La solution que nous avons adoptée est basée une modélisation hiérarchique du système en considérant trois niveaux d'abstraction (cf. Figure 2.7). Le comportement des composants représentés en gris est détaillé par le modèle du niveau inférieur. Pour chaque modèle, les entrées pour la simulation (décrivant des requêtes de lecture et d'écriture de données) sont soit générées aléatoirement à partir d'une distribution analytique, soit lues directement à partir d'une trace d'exécution réelle. Les modèles de fautes à injecter durant la simulation sont définis par l'injecteur de fautes. Pour chaque composant, on spécifie la nature, permanente ou transitoire, des fautes et les paramètres décrivant leurs instants d'occurrence et leurs effets. Ces paramètres peuvent être spécifiés directement par l'utilisateur ou bien dérivés de la simulation d'un modèle décrivant le comportement à un niveau d'abstraction plus détaillé.

Dans le modèle hiérarchique de la figure 2.7, le comportement, la granularité des données et les mesures évaluées à partir de la simulation sont affinés d'un niveau à un autre. Au niveau 1, les entrées sont des requêtes pour lire ou écrire des segments de données d'un fichier. Ces requêtes sont traduites par des sollicitations au cache pour lire ou écrire chaque segment. Á ce niveau, le cache et les disques sont représentés par une seule entité modélisée en boîte noire. L'injecteur de fautes associé à cette entité définit la probabilité d'échec ou de succès du transfert de chaque segment. Cette distribution de probabilité est évaluée à partir du modèle du niveau 2. Ce modèle détaille les opérations du cache et ses interactions avec les systèmes hôtes et les disques lors du transfert d'un segment. Les données dans les disques et dans la mémoire cache sont modélisées explicitement. Un segment est décrit par une entité atomique dans laquelle des fautes (se traduisant par des inversions de bits) sont injectées durant son transfert à travers le cache, du cache vers les disques ou vers les systèmes hôtes, ou bien quand il est stocké dans la mémoire cache ou dans les disques.

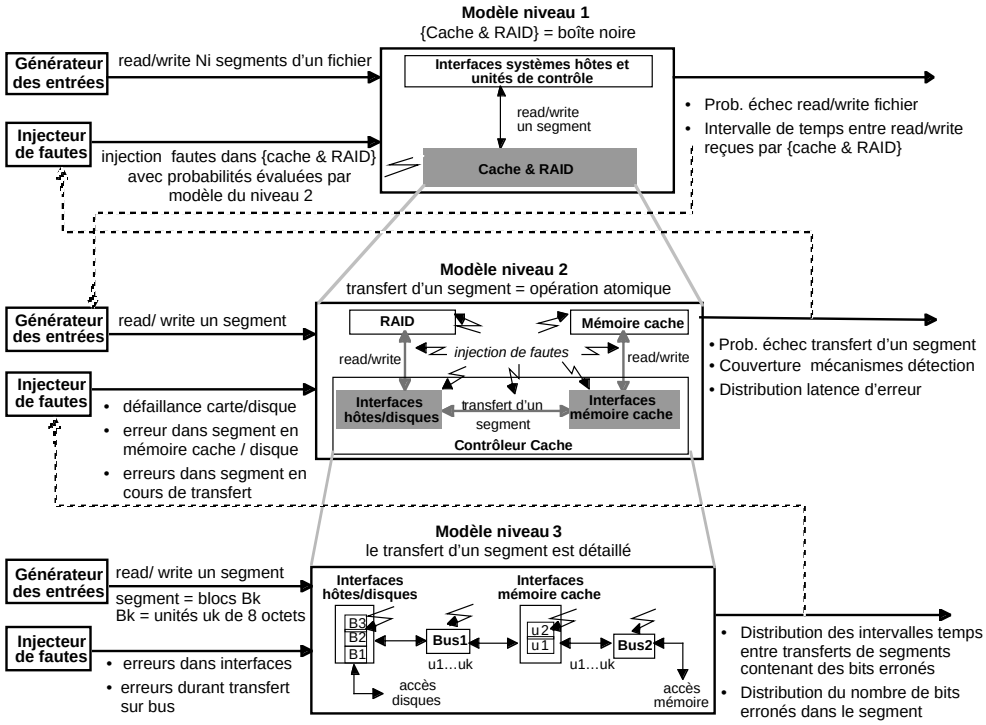


Figure 2.7 - Modélisation hiérarchique du système

D'autres fautes provoquant la défaillance d'un disque ou d'un composant du cache sont aussi simulées. Les opérations de décomposition et d'assemblage du segment durant le transfert ne sont pas détaillées à ce niveau afin de réduire de façon significative le nombre d'événements à simuler et le temps de simulation. Pour être en mesure de simuler les mécanismes de détection d'erreurs, d'évaluer la latence d'erreurs et d'obtenir une distribution de la latence en fonction de l'origine de la faute, l'injection de faute durant le transfert d'un segment à travers le cache est effectuée de la façon suivante :

- dans les interfaces du contrôleur de cache avec les systèmes hôtes et les disques avant le transfert sur le Bus 1 (c'est-à-dire, avant ajout de la parité ou du CRC),
- durant le transfert sur le Bus 1,
- dans les interfaces du contrôleur de cache avec la mémoire cache avant le transfert sur le Bus 2 (c'est-à-dire, avant ajout du code EDAC),
- durant le transfert sur le Bus 2.

Pour chacune de ces injections, on associe une distribution de probabilité définissant l'instant de l'injection et le nombre de bits erronés résultant de l'injection. Ces distributions sont évaluées à partir de la simulation détaillée du transfert d'un segment, qui est effectuée au niveau 3. Dans le modèle du niveau 3, chaque segment est décomposé en blocs structurés sous forme d'unités de données de 8 octets. Les opérations de décomposition et d'assemblage du segment dans les interfaces du

contrôleur du cache et les transmissions sur les bus sont simulées en détail. Durant ces opérations, des fautes transitoires d'une durée prédéfinie sont injectées. Durant la durée de la faute, un ou plusieurs bits sont inversés dans les unités de données qui sont en cours de traitement. Le transfert d'un segment nécessitant plusieurs cycles de traitement, ce modèle de fautes peut conduire alors à l'accumulation d'un nombre important de bits erronés dans un segment. Ceci est illustré sur les courbes présentées sur la figure 2.8 qui donne la densité de probabilité du nombre de bits erronés par segment durant le transfert du segment à travers les interfaces du contrôleur de cache et durant les transmissions sur les bus quand la durée d'une faute transitoire est de 5 microsecondes. Ces courbes ont été obtenues avec un fichier d'entrée correspondant à une trace réelle constituée d'environ 480000 requêtes de lecture ou d'écriture de segments de données.

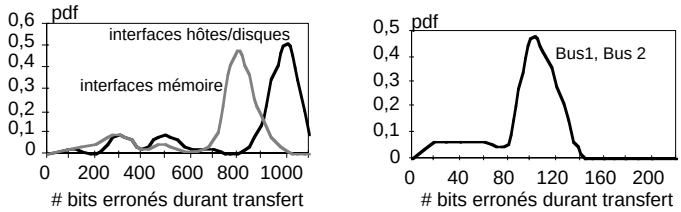


Figure 2.8 - Densité de probabilité du nombre de bits erronés dans un segment résultant de l'injection de fautes transitoires de durée 5 microsecondes

Les figures 2.9 et 2.10 donnent des exemples de résultats obtenus à partir de la simulation du modèle de niveau 2 en utilisant les distributions de la figure 2.8 (d'autres résultats sont présentés dans [108]). La figure 2.9 donne l'évolution durant la simulation de la couverture de détection et de correction d'erreurs du mécanisme EDAC protégeant la mémoire cache et les transmissions sur le Bus2. La figure 2.10 donne la distribution de la latence d'erreur en tenant compte de l'origine de l'erreur (c'est-à-dire, le composant où la première faute affectant le segment a été injectée) et pour toutes classes d'erreurs confondues. En particulier, on peut constater que la distribution de la latence est bi-modale. Le premier mode est dû aux erreurs affectant le Bus 1 qui sont immédiatement détectées par la parité et le second résulte principalement des erreurs qui sont injectées dans la mémoire cache ou dans les disques. Les distributions de la latence pour Bus 1 et Bus 2 sont différentes car les erreurs injectées durant l'écriture d'un segment dans la mémoire à travers le Bus 2 ne sont détectées ou éliminées que beaucoup plus tard quand le segment sollicité en lecture ou en écriture. Cette distribution dépend du profil d'utilisation du système. Pour réduire cette latence, des mécanismes de scrutation périodique de la mémoire cache et des disques ont été mis en œuvre.

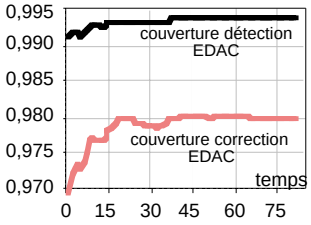


Figure 2.9- Couverture EDAC

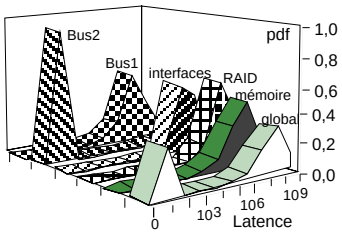


Figure 2.10- Distribution de la latence d'erreur

2.2.3. Conclusion

La modélisation hiérarchique pour la simulation comportementale en présence de fautes constitue une solution prometteuse pour l'analyse de la sûreté de fonctionnement de systèmes tolérants aux fautes pendant la conception. En considérant un niveau de détail relativement fin, on peut simuler des fautes à un niveau bas et propager leurs effets aux niveaux supérieurs. Pour l'étude de cas présentée au § 2.2, nous avons défini uniquement trois niveaux hiérarchiques pour illustrer notre approche de modélisation et le type de résultats qu'on peut obtenir à partir de la simulation. Compte tenu du temps qui a été alloué à l'étude et des informations qui nous ont été fournies sur le système, l'injection des fautes a été effectuée à un niveau d'abstraction qui reste relativement élevé par rapport aux fautes réelles qui, généralement, se produisent à des niveaux beaucoup plus bas (c'est-à-dire, au niveau transistor, circuit, etc.). Pour faire de telles analyses, notre approche de modélisation peut être affinée en définissant des niveaux hiérarchiques supplémentaires. Les modèles de fautes décrivant les niveaux transistor, circuit, registre, … de l'architecture peuvent être obtenus en utilisant des outils de simulation appropriés tels que FOCUS, MEFISTO. Des exemples illustrant l'application de la simulation hiérarchique pour propager l'effet des fautes des niveaux transistor, logique ou circuit, jusqu'au niveau système sont présentés dans [109, 173, 174].

Notons enfin que, outre les avantages que procure la simulation hiérarchique pour la modélisation et l'analyse de la sûreté de fonctionnement, cette approche permet aussi de définir des modèles de fautes hiérarchiques qui peuvent être utilisés pour optimiser la vérification des mécanismes de tolérance par injection de fautes, physique ou par logiciel, sur des prototypes. Cette démarche a été utilisée en particulier dans [196].

2.3. Conclusions sur les modèles complexes

Les deux approches complémentaires présentées dans ce chapitre, concernant la spécification et la construction de modèles GSPN complexes et la modélisation hiérarchique pour la simulation comportementale en présence de fautes, visent à faciliter l'analyse et l'évaluation de la sûreté de fonctionnement de systèmes tolérants aux fautes pendant les phases de conception. Nous avons prouvé le bien fondé et l'efficacité de ces approches en les appliquant à des cas d'étude issus de systèmes réels où la modélisation de la sûreté de fonctionnement a été réalisée à un niveau de détail relativement fin. Notre

démarche qui consiste à associer modélisation conceptuelle et application à des cas concrets, a aussi pour objectif de nous permettre de valider nos approches et d'étudier leurs limites. Nos travaux ont essentiellement porté sur la construction de modèles complexes pour l'analyse et l'évaluation de la sûreté de fonctionnement. Néanmoins, il est aussi important d'apporter des solutions efficaces au problème du traitement des modèles. Plusieurs avancées ont été réalisées dans ce domaine. En plus du cas classique des processus markoviens où les travaux ont été focalisés par exemple sur l'agrégation de modèles [18, 148, 180, 184], le calcul de bornes pour l'estimation de valeurs approchées de mesures de sûreté de fonctionnement ou de performabilité à partir de la génération partielle de l'espace d'état [24, 87, 146, 156, 186], un effort croissant a été récemment consacré à l'étude de processus plus généraux (regénératifs, non markoviens, etc.) [40, 78, 79, 161]. Les travaux sur le calcul de bornes nous semblent être une voie intéressante à approfondir. Au stade actuel, ces travaux ont surtout focalisé sur l'évaluation de mesures de sûreté de fonctionnement en régime asymptotique, en considérant en particulier la disponibilité. L'extension de ces travaux pour évaluer des mesures transitoires permettra de couvrir un spectre d'application plus large.

La conception d'un système tolérant aux fautes nécessite l'utilisation de plusieurs méthodes et formalismes pour des besoins de spécification et de vérification de propriétés fonctionnelles ou de sûreté de fonctionnement, et également pour faire des analyses et des évaluations de performance et de sûreté de fonctionnement. Construire des modèles différents pour satisfaire ces différents objectifs est une solution coûteuse et très fastidieuse. Une solution plus optimale est de définir un paradigme qui soit en mesure de combiner plusieurs formalismes en définissant des interfaces permettant par exemple, de générer des modèles pour faire de l'évaluation à partir des modèles utilisés pour la spécification formelle ou pour la vérification. Les travaux récents basés sur les processus algébriques stochastiques [16, 59] ou le cas d'étude présenté dans [193], qui est basé sur le langage ESTELLE, vont dans cette direction. Cependant, leur champ d'application est pour l'instant orienté vers l'évaluation de la performance. Il nous semble intéressant d'explorer l'applicabilité de ces travaux pour faire des évaluations de sûreté de fonctionnement et surtout d'étudier leurs limites quand il s'agit de traiter des cas concrets d'une complexité équivalente à celle du CAUTRA par exemple.

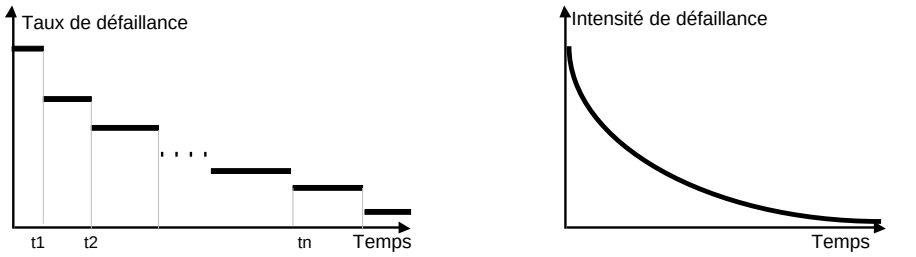
3

3. MODÉLISATION DE LA CROISSANCE DE FIABILITÉ

La croissance de fiabilité se traduit par un accroissement stochastique des temps de fonctionnement entre défaillances. Elle résulte généralement de l'élimination progressive des fautes de conception. De façon générale, la modélisation de la croissance de fiabilité s'est focalisée sur le logiciel ; toutefois, les résultats sont également applicables au matériel. Les travaux résumés dans ce chapitre ont été effectués en collaboration avec Karama Kanoun et Jean-Claude Laprie et s'inscrivent dans la continuité des études menées sur ce thème dans le groupe "Tolérance aux fautes et sûreté de fonctionnement informatique" depuis plus de vingt ans.

3.1. Situation des travaux et principales contributions

L'objectif d'une étude de croissance de fiabilité est d'analyser l'évolution du processus d'occurrences des défaillances au fur et à mesure de l'élimination des fautes de conception et d'évaluer des mesures quantitatives permettant de chiffrer l'impact des fautes sur le fonctionnement du système [110]. Les mesures usuelles considérées sont l'intensité de défaillance (c'est-à-dire, le nombre de défaillances par unité de temps), le taux de défaillance et l'intervalle de temps entre défaillances. Plusieurs modèles, dits "modèles de croissance de fiabilité", ont été proposés dans la littérature pour évaluer ces mesures (voir [145, 158, 194] pour une synthèse de ces modèles). Ces modèles expriment les relations entre les mesures de fiabilité et le temps ou le numéro de la défaillance. La figure 3.1 présente les deux types de représentation du processus de défaillance les plus courants [115] : a) processus de Poisson par morceaux (modèles à taux de défaillance), ou b) processus de Poisson non homogène où l'intensité de défaillance est représentée par une fonction continue (modèles NHPP). Dans les deux cas, l'intensité de défaillance décroît globalement dans le temps et représente ainsi une croissance globale de fiabilité.



a) Processus de Poisson par morceaux b) Processus de Poisson non homogène

Figure 3.1 - Représentations les plus courantes du processus de défaillance dans les modèles de croissance de fiabilité

Historiquement, les travaux effectués sur la croissance de fiabilité ont été centrés sur le logiciel[2] en adoptant une approche boîte noire (le logiciel est vu comme un système mono-composant). Les modèles utilisés sont principalement des modèles en temps continu, qui représentent l'évolution du comportement du logiciel dans un environnement donné en fonction du temps calendaire ou du temps d'exécution et les mesures évaluées à partir de ces modèles concernent essentiellement la fiabilité.

Nos travaux sur la croissance de fiabilité ont été les suivants :

- nous avons contribué à la définition d'une approche de modélisation permettant d'évaluer la *croissance de fiabilité et de disponibilité de systèmes multi-composant —matériel et logiciel —* en tenant compte de la croissance de fiabilité des composants et de leurs interactions,
- nous avons exploré l'apport d'une modélisation de la *croissance de fiabilité en temps discret* (c'est-à-dire, en fonction du nombre d'exécutions), en particulier vis-à-vis de la prise en compte de la *variation de l'environnement d'utilisation*.

Les principales motivations et les résultats de ces travaux sont résumés aux § 3.2 et § 3.3. En plus de ces travaux, qui sont essentiellement de nature théorique, nous avons contribué à l'élaboration d'une méthode globale d'analyse et d'évaluation de la fiabilité du logiciel à partir de statistiques de défaillances et de corrections collectées pendant le développement et la vie opérationnelle, et d'un outil mettant en œuvre les principales étapes de la méthode (l'outil SoRel). Cette méthode vise à guider et à faciliter l'application des modèles dans un contexte industriel. Ces travaux ont été également accompagnés par des applications à des cas réels. Un résumé de la méthode et de ces applications est présenté au § 3.4.

3.2. Croissance de fiabilité de systèmes multi-composant

Les concepteurs et les utilisateurs des systèmes informatiques sont intéressés par des évaluations de la sûreté de fonctionnement du système global incorporant les

---

[2] Il existe quelques travaux consacrés à l'évaluation de la croissance de fiabilité du matériel, en particulier [61, 65, 107]

composants matériels et logiciels en considérant à la fois les fautes physiques et les fautes de conception qui sont susceptibles d'affecter la sûreté de fonctionnement du système. Cependant, il n'existe pas dans la littérature de modèles structurels qui permettent d'obtenir de telles évaluations. Historiquement, les recherches sur l'évaluation du logiciel et sur l'évaluation du matériel ont suivi des orientations différentes [126, 129] et les rares modèles permettant d'évaluer la sûreté de fonctionnement de systèmes —matériel et logiciel— (par exemple [10, 62, 70, 112, 166, 188]) considèrent uniquement le cas où les composants sont en fiabilité stabilisée.

Les recherches sur l'évaluation du logiciel ont été focalisées sur le suivi et la prévision de la fiabilité durant la validation et au début de la vie opérationnelle en se basant sur une modélisation en "boîte noire" de la croissance de fiabilité. De plus, en dépit de l'importance majeure de la disponibilité dans certains domaines d'application, par exemple les télécommunications, cette mesure n'a pas été considérée dans les modèles de croissance de fiabilité. Les modèles prenant en compte la structure du logiciel, que ce soit pour des logiciels non tolérants aux fautes [34, 125, 142] ou pour des logiciels tolérants aux fautes [9, 85, 88, 126] sont moins nombreux et sont applicables uniquement quand les composants sont en fiabilité stabilisée.

Les travaux sur l'évaluation du matériel ont été focalisés sur le développement de modèles structurels permettant de guider la conception et d'estimer le comportement du système en vie opérationnelle. Généralement, ces modèles étudient uniquement l'impact des fautes physiques sur la sûreté de fonctionnement du système. Cependant, les défaillances du matériel dues à des fautes de conception sont loin d'être négligeables (cf. les données d'expérience présentées dans [11, 107]. L'hypothèse d'évolution stabilisée de la disponibilité qui est couramment considérée ne correspond en réalité qu'au régime stationnaire. En effet, les données opérationnelles montrent que la disponibilité peut varier d'une manière significative avant d'atteindre un comportement stabilisé (voir par exemple [122, 192]). Par conséquent, pour effectuer des estimations réalistes de la disponibilité, il est important de tenir compte du phénomène de croissance de fiabilité.

Les travaux présentés dans [47, 128] et plus récemment dans [136, 137] montrent qu'un cadre conceptuel commun peut être considéré pour la modélisation de la sûreté de fonctionnement de systèmes —matériel et logiciel— en tenant compte des fautes physiques et des fautes de conception. Le modèle proposé dans [47] considère uniquement la croissance de fiabilité du logiciel en supposant que les composants matériels sont en fiabilité stabilisée. La théorie présentée dans [128, 136] montre que les résultats obtenus dans le cadre de la modélisation de la croissance de fiabilité du logiciel peuvent être appliqués pour modéliser la croissance de fiabilité des composants matériel résultant de l'élimination des fautes de conception. En particulier, en considérant différentes hypothèses pour les processus de défaillance et de restauration du matériel et du logiciel, il est démontré, en utilisant les processus de renouvellement généralisés, que l'on peut obtenir des mesures décrivant la croissance de fiabilité et la croissance de disponibilité du système global. Cependant, le modèle proposé peut être qualifié de *modèle de connaissance* dans le sens où il a pour principale vocation d'analyser les propriétés des mesures de fiabilité et de disponibilité en fonction des hypothèses considérées, mais il est trop complexe pour être applicable dans un contexte réel.

L'approche de modélisation que nous avons définie dans [95, 133, 138] est basée sur des hypothèses plus simples que celles considérées dans [128, 136]. Elle permet d'évaluer la croissance de fiabilité et de disponibilité de systèmes multi-composant matériel et logiciel tout en respectant les propriétés générales des mesures de fiabilité et de disponibilité établies à partir du modèle de connaissance. Cette approche est basée sur le modèle hyperexponentiel de croissance de fiabilité défini au LAAS [118, 127, 138], et elle est tout à fait applicable pour obtenir des prévisions des mesures de sûreté de fonctionnement à partir des données d'expérience collectées durant l'utilisation du système. Le modèle hyperexponentiel tel qu'il a été défini dans [118, 127, 138] a été initialement appliqué à des systèmes mono-composant. En s'appuyant sur les propriétés markoviennes du modèle, nous avons développé une approche originale permettant d'étendre ces propriétés au cas de systèmes multi-composant. Dans les paragraphes suivants, nous présentons les principales caractéristiques du modèle hyperexponentiel. Nous résumons ensuite les principes de l'approche proposée et nous donnons un exemple d'application.

3.2.1. Modèle hyperexponentiel

Il s'agit d'un modèle NHPP défini par la fonction intensité de défaillance

$$h(t) = \frac{\omega\,\zeta_{sup}\,e^{-\zeta_{sup}t} + \varpi\,\zeta_{inf}\,e^{-\zeta_{inf}t}}{\omega\,e^{-\zeta_{sup}t} + \varpi\,e^{-\zeta_{inf}t}} \qquad \text{avec } 0\le\omega\le 1,\ \omega+\varpi = 1 \text{ et } \zeta_{inf} \le \zeta_{sup}.$$

ω, ζ_{sup} et ζ_{inf} sont les paramètres du modèle.

$h(t)$ est une fonction continue décroissante dans le temps, variant entre $h(0) = \omega\,\zeta_{sup} + \varpi\,\zeta_{inf}$ et $h(\infty) = \zeta_{inf}$. Ce dernier correspond au taux de défaillance résiduel caractérisant le comportement asymptotique du système en fiabilité stabilisée.

Pour les modèles NHPP, l'intensité de défaillance et le taux de défaillance peuvent s'exprimer par la même fonction, seul l'instant d'origine change [159]. Si on note par s_{i-1} l'instant d'occurrence de la défaillance i-1 et τ le temps écoulé depuis cet instant, alors le taux de défaillance $\lambda_i(\tau \mid s_{i-1})$ relatif à l'occurrence de la défaillance i est donné par : $\lambda_i(\tau \mid s_{i-1}) = h(s_{i-1}+\tau)$.

En utilisant cette propriété, on peut considérer que l'intensité de défaillance du modèle hyperexponentiel est dérivée d'une loi de Cox hyperexponentielle à deux étages [48]. Le modèle peut être alors représenté par une chaîne de Markov comportant trois états : un état absorbant D et deux états transitoires U_1 et U_2, avec des probabilités initiales d'occupation ω et ϖ et des taux de transition associés ζ_{sup} et ζ_{inf} (Figure 3.2-a). La modélisation de la croissance de fiabilité peut se ramener alors à la transformation d'une chaîne de Markov classique caractérisant le comportement en fiabilité stabilisée du système (Figure 3.2-a), en une chaîne de Markov à trois états (Figure 3.2-b) permettant de représenter la croissance de fiabilité du système.

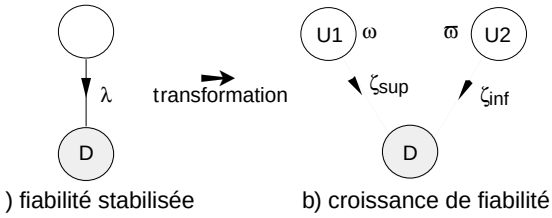


Figure 3.2 - Représentation markovienne du modèle hyperexponentiel

En utilisant la représentation markovienne du modèle, on peut également modéliser la croissance de disponibilité en tenant compte des temps de restauration. Si on note par μ le taux de restauration du système supposé constant, on aboutit au modèle de disponibilité présenté sur la figure 3.3-b. Les différentes étapes qui ont conduit à ce modèle sont détaillées dans [95, 138].

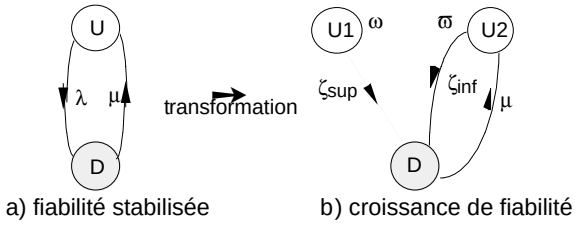


Figure 3.3 - Modèles de disponibilité

La figure 3.4 (courbe *a*) illustre une évolution typique de l'indisponibilité obtenue à partir du traitement du modèle de la figure 3.3-b. La courbe *b* (respectivement, la courbe *c*) correspond à l'indisponibilité obtenue en supposant un comportement en fiabilité stabilisée caractérisé par un taux de restauration μ constant et un taux de défaillance λ constant égal à la valeur maximale (respectivement, minimale) de l'intensité de défaillance associée au modèle hyperexponentiel. L'écart entre la courbe *a* et les courbes *b* et *c*, traduit les erreurs d'estimation effectuées dans le cas où on évalue la disponibilité sans tenir compte de la croissance de fiabilité.

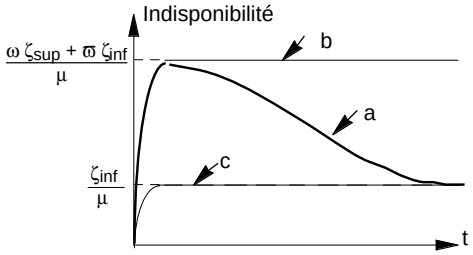


Figure 3.4 - Allure d'évolution de l'indisponibilité

Remarque : La différence fondamentale entre l'approche de transformation illustrée par les figures 3.2 et 3.3 et celle basée sur la méthode des états fictifs [124], réside dans le fait qu'avec la méthode des états fictifs, on simule une fonction de *distribution* non exponentielle, par contre dans notre cas, nous simulons un *processus* de défaillance qui est caractérisé par une fonction intensité de défaillance correspondant au modèle hyperexponentiel. Par conséquent, aucune interprétation physique ne doit être associée aux états des chaînes transformées (figures 3.2-b et 3.2-b). Une telle interprétation conduirait à considérer, à tort, que le système modélisé par la figure 3.3-b atteint un régime stationnaire à partir du moment où il quitte l'état U1.

3.2.2. Modélisation de systèmes multi-composant

En utilisant la représentation markovienne du modèle hyperexponentiel illustrée par les figures 3.2 et 3.3, nous avons étendu la technique de transformation pour modéliser des systèmes multi-composant. La méthode proposée consiste à construire une chaîne de Markov décrivant le comportement en fiabilité stabilisée des composants du système et de leurs interactions et à *transformer* ensuite cette chaîne de Markov pour modéliser le phénomène de croissance de fiabilité. On suppose que l'évolution de la fiabilité de chaque composant peut être décrite par un modèle hyperexponentiel. Des exemples d'application de ce modèle à des données réelles et des comparaisons avec d'autres modèles sont présentés par exemple dans [97, 121, 133]).

L'extension de la transformation au cas multi-composant est décrite dans [95, 138]. La méthode générale que nous avons développée est basée sur les réseaux de Petri stochastiques généralisés (GSPN) qui sont bien adaptés pour représenter le comportement des différents composants du système en présence de défaillances et pour décrire les dépendances stochastiques entre ces composants [154, 160]. Les modèles GSPN décrivant le comportement de chaque composant en fiabilité stabilisée et en croissance de fiabilité sont présentés sur la figure 3.5. Les chaînes de Markov générées à partir de ces GSPN correspondent aux chaînes de Markov de la figure 3.3.

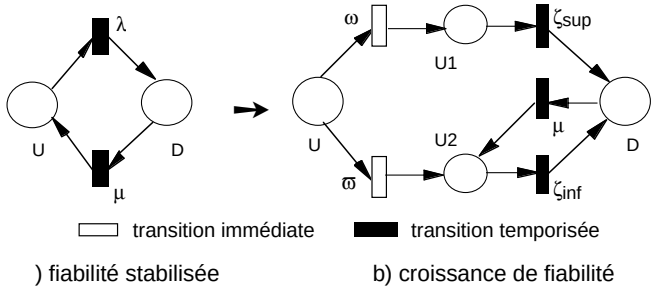


Figure 3.5 - Modèles GSPN relatifs à un composant

Pour la modélisation de systèmes multi-composant, la méthode de transformation se résume en trois étapes :

- construction du GSPN décrivant le comportement du système en fiabilité stabilisée,

- transformation du GSPN conformément au modèle de la figure 3.5,
- traitement de la chaîne de Markov obtenue à partir du graphe des marquages du GSPN transformé pour évaluer les mesures de fiabilité et de disponibilité.

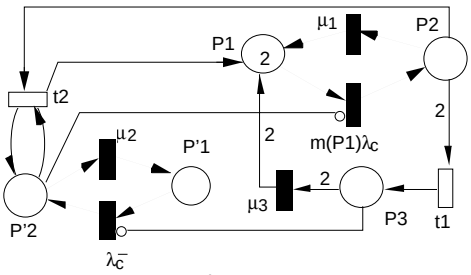
Exemple : Considérons un système tolérant aux fautes, constitué de deux composants redondants intégrant des mécanismes de détection et de recouvrement d'erreurs qui sont supposés imparfaits. On note par :

- λ_c et $\lambda_{\bar{c}}$ les taux de défaillance correspondant respectivement aux erreurs couvertes et aux erreurs non couvertes,
- μ_1, μ_2, et μ_3 respectivement, les taux de restauration après occurrence d'une défaillance couverte, d'une défaillance non couverte, et d'une deuxième défaillance couverte.

Le GSPN et la chaîne de Markov du système en fiabilité stabilisée sont présentés sur la figure 3.6. Les modèles obtenus après transformation sont donnés sur les figures 3.7 et 3.8. Les paramètres $\{\omega_c , \zeta_{c,sup} , \zeta_{c,inf} \}$ et $\{\omega_{\bar{c}} , \zeta_{\bar{c},sup} , \zeta_{\bar{c},inf} \}$ résultent de la transformation de λ_c et de $\lambda_{\bar{c}}$, respectivement, et $m(P_i)$ est le marquage de la place P_i. La figure 3.9 présente un exemple d'évolution de la disponibilité obtenue à partir du traitement de la chaîne de Markov de la figure 3.8 par l'outil SURF2 [15] :

- C1 (respectivement C5) correspond à un comportement en fiabilité stabilisée où λ_c et $\lambda_{\bar{c}}$ prennent la valeur minimale (respectivement maximale) de l'intensité de défaillance correspondante,
- C4 correspond à un comportement en croissance de fiabilité où λ_c et $\lambda_{\bar{c}}$ tendent à décroître dans le temps de leur valeur maximale vers leur valeur minimale,
- C2 (respectivement C3) représente l'évolution de la décroissance de l'indisponibilité du système quand on considère uniquement une croissance de fiabilité liée aux défaillances couvertes (respectivement, non couvertes).

Les résultats de la figure 3.9 montrent qu'il existe une différence significative entre l'allure d'évolution des courbes d'indisponibilité basées sur l'hypothèse d'un comportement en fiabilité stabilisée du système (C1 et C5) et les courbes prenant en compte la croissance de fiabilité (C2, C3 et C4). Par conséquent, le fait de ne pas tenir compte de la croissance de fiabilité des composants risque de conduire à des résultats qui ne sont pas représentatifs du comportement du système modélisé.
D'autres exemples d'application de la méthode de transformation sont présentés dans [95, 113, 138]. En particulier, dans [113] nous avons modélisé trois architectures logicielles tolérantes aux fautes basées sur les blocs de recouvrement [170], la programmation en N-versions [32] et la programmation N-autotestable [131]. Pour chacune de ces architectures, nous avons étudié l'impact des fautes indépendantes et des fautes corrélées sur la fiabilité du système en tenant compte de la croissance de fiabilité. Cette application constitue une étude originale dans le sens où la croissance de fiabilité de logiciels tolérants aux fautes n'avait jamais été abordée auparavant.



Etat	Marquage
1	{2P1,P'1}
2	{P1,P2,P'1}
3	{2P3,P'1}
4	{2P1,P'2}

Figure 3.6 – Modèles en fiabilité stabilisée

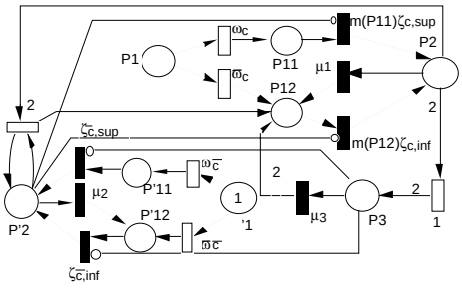


Figure 3.7–GSPN : croissance de fiabilité

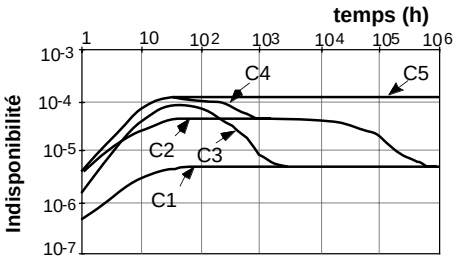
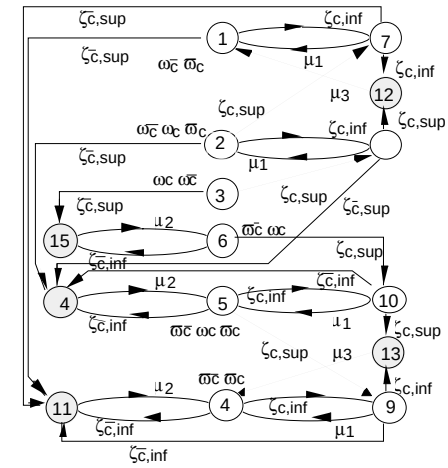


Figure 3.9– Exemples de résultats



tat	arquage
	2P12,P'11}
	P11,P12,P'11}
	2P11,P'11}
	2P12,P'12}
	P11,P12,P'12}

tat	arquage
	2P11,P'12}
	P12,P2,P'11}
	P11,P2,P'11}
	P12,P2,P'12}
0	P11,P2,P'12}

tat	arquage
1	2P12,P'2}
2	2P3,P'11}
3	2P3,P'12}
4	P11,P12,P'2}
5	2P11,P'2}

Figure 3.8- Chaîne de Markov—croissance de fiabilité

3.2.3. Conclusion

La méthode que nous venons de résumer est originale dans le sens où elle a donné naissance au premier modèle structurel permettant de modéliser la croissance de fiabilité et de disponibilité d'un système multi-composant en tenant compte de la croissance de fiabilité des composants. Les résultats sont applicables à des systèmes constitués de composants matériels et logiciels en tenant compte à la fois des fautes de conception et des fautes physiques. La méthode proposée est similaire aux approches classiques de modélisation de systèmes par des processus markoviens homogènes ; seule l'étape de

transformation a été introduite. Par conséquent, on peut bénéficier de tous les résultats existants concernant la construction et le traitement de modèles markoviens. En particulier, nous avons montré dans [134] que dans le cas où les composants sont stochastiquement indépendants, on peut obtenir la chaîne transformée en utilisant l'algèbre de Kronecker [3]. Par ailleurs, l'utilisation des GSPN permet de faciliter la mise en œuvre de la technique de transformation et de prendre en compte les dépendances stochastiques entre les composants. Nous avons montré dans [135] que, de façon plus générale, les GSPN facilitent la modélisation de processus stochastiques non stationnaires. La méthode proposée, satisfait les propriétés des fonctions de fiabilité et de disponibilité établies à partir du modèle de connaissance défini dans [128, 136]. Un exemple d'illustration comparant les résultats obtenus à partir du modèle de connaissance et ceux fournis par le modèle hyperexponentiel est présenté dans [133].

Néanmoins, on peut remarquer à travers l'exemple présenté au § 3.1.2, que la transformation conduit à l'augmentation de l'espace d'état en comparaison avec la chaîne de Markov en fiabilité stabilisée. En effet, si on considère par exemple, un système constitué de n composants stochastiquement indépendants qui sont tels que chaque composant est soit actif, soit défaillant, alors la cardinalité de l'espace des états de la chaîne de Markov caractérisant le comportement en fiabilité stabilisée du système varie entre n+1 (cas où tous les composants sont identiques) et 2^n (cas où tous les composants sont différents). Après l'application de la transformation, la cardinalité de l'espace des états du système varie alors entre (n+1)(n+2)/2 et 3^n. Cette explosion d'état est maîtrisable grâce à la puissance actuelle des techniques et des outils de traitement de chaînes de Markov qui sont capables de traiter des modèles à plusieurs milliers d'états [82, 86, 171, 185, 189].

Enfin, on peut noter que ces travaux ont été effectués en partie dans le cadre du projet européen PDCS et ont donné suite à quatre publications, dont deux dans des revues IEEE [113, 138] et une publication au "Fault-Tolerant Computing Symposium" [133].

3.3. Modélisation en temps discret

La modélisation de la croissance de fiabilité en temps discret (c'est-à-dire, en fonction du nombre d'exécutions) n'a été étudiée que très rarement dans la littérature [60, 66, 169, 195]. Cependant, ce type de modélisation présente plusieurs avantages. En effet, pour certaines applications, il est plus significatif de mesurer la fiabilité en termes de nombre d'exécutions (nombre de transactions bancaires, nombre de lancements réussis, etc.) avant défaillance au lieu de mesurer le temps jusqu'à défaillance. La modélisation en temps discret est aussi bien adaptée pour évaluer des mesures de fiabilité durant les phases de test quand on recueille le nombre et les résultats des jeux de tests, plutôt que d'enregistrer les temps calendaires ou les temps d'exécution avant défaillance. Mais, la raison principale qui nous a amené à nous intéresser à la modélisation de la croissance de fiabilité en temps discret réside dans le fait qu'elle offre la possibilité d'évaluer la fiabilité du logiciel en tenant compte explicitement de son environnement d'utilisation. Il est largement reconnu que les mesures de fiabilité sont fortement corrélées à l'environnement d'utilisation du logiciel [64, 92], cependant, à notre connaissance, il

n'existe pas de solutions satisfaisantes permettant d'estimer la fiabilité du logiciel quand son environnement d'utilisation change. Dans la suite, nous présentons une approche originale qui traite ce problème. Elle est basée sur un nouveau modèle de croissance de fiabilité, le modèle hyperexponentiel en temps discret, dont les principales caractéristiques sont résumées au § 3.3.1. L'approche proposée ainsi que des exemples d'illustration sont présentés au § 3.3.2.

3.3.1. Modèle hyperexponentiel en temps discret

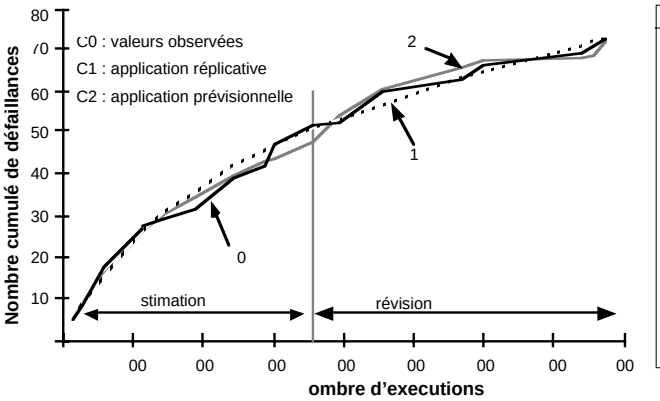
Le modèle hyperexponentiel en temps discret est basé sur des hypothèses équivalentes à celles du modèle hyperexponentiel en temps continu. Il décrit l'évolution de la probabilité de défaillance à l'exécution *P(n)* par une fonction qui décroît en fonction du nombre d'exécutions et atteint asymptotiquement un comportement stabilisé :

$$P(n) = \frac{\theta p_{sup}(1-p_{sup})^{n-1} + \overline{\theta}\, p_{inf}(1-p_{inf})^{n-1}}{\theta(1-p_{sup})^{n-1} + \overline{\theta}(1-p_{inf})^{n-1}} \quad 0 \le \theta \le 1,\ \overline{\theta} = 1-\theta \text{ et } p_{inf} \le p_{sup}$$

θ, p_{sup} et p_{inf} sont les paramètres du modèle.

Pour établir ce modèle, nous avons été amené à étendre les définitions classiques des mesures de fiabilité en temps continu (en particulier, celle du taux de défaillance) pour les appliquer au cas discret. Les propriétés du modèle, les expressions des mesures de fiabilité associées, et les procédures d'estimation des paramètres du modèle à partir de données collectées sous la forme de "nombre d'exécutions entre défaillances" ou bien "nombre de défaillances par séquence d'exécutions" sont détaillées dans [100].

La figure 3.10 présente un exemple d'application du modèle à des données réelles collectées pendant le test de validation d'un logiciel (voir [100]). Pour chaque séquence de test *i*, *si* indique le nombre de jeux de tests (nombre d'exécutions) effectués jusqu'à la séquence *i* et *yi* le nombre cumulé de défaillances. La courbe C0 donne l'évolution du nombre cumulé de défaillances observées en fonction du nombre d'exécutions. C1 donne les valeurs calculées par le modèle en estimant les paramètres avec toutes les données (application replicative), et C2 donne les valeurs obtenues en estimant les paramètres avec les données correspondant aux dix premières séquences d'exécutions et en effectuant ensuite des prévisions pas à pas (application prévisionnelle). Dans les deux cas, le modèle hyperexponentiel en temps discret représente de façon satisfaisante le comportement observé.



i	si	yi
1	14	5
2	28	8
3	57	18
4	71	20
5	114	27
6	143	29
7	186	31
8	243	39
9	286	42
10	300	47
11	358	52
12	393	53
13	457	60
14	571	63
15	600	66
16	743	69
17	758	71
18	773	73

Figure 3.10 – Exemple d'application à des données réelles

Le modèle hyperexponentiel en temps discret peut être interprété comme la version en temps discret du modèle hyperexponentiel en temps continu. De façon analogue à ce dernier modèle, nous avons associé au modèle une interprétation markovienne et nous avons montré que la modélisation de la croissance de fiabilité en temps discret peut se ramener à la transformation d'une chaîne de Markov en temps discret caractérisant le comportement du système en fiabilité stabilisée (Figure 3.11-a) en une chaîne de Markov prenant en compte la croissance de fiabilité (Figure 3.11-b).

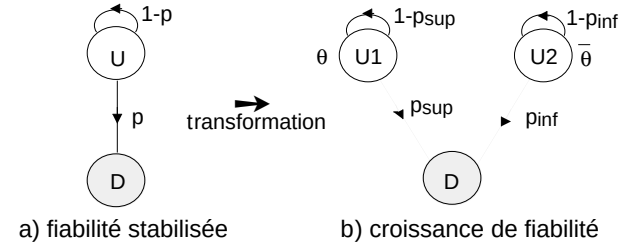


Figure 3.11– Représentation markovienne du modèle en temps discret

La figure 3.12 illustre le lien entre le modèle en temps discret et le modèle en temps continu. En notant par γ le taux d'exécution du logiciel, le taux de défaillance peut être exprimé sous la forme $\lambda = \gamma p$. La figure 3.12-b (respectivement, la figure 3.12-c) montre la chaîne de Markov en temps continu, obtenue après transformation de la probabilité de défaillance à l'exécution notée p (respectivement, la chaîne de Markov en temps continu, obtenue après transformation du taux de défaillance λ). Notons que les chaînes immergées déduites des chaînes de Markov de la figure 3.12 correspondent aux chaînes de Markov de la figure 3.11. L'intérêt de la transformation illustrée sur la figure 3.12-c est qu'elle permet de distinguer explicitement la probabilité de défaillance à l'exécution du taux d'exécution du logiciel. Cette propriété est à la base de l'approche de modélisation de l'environnement d'utilisation du logiciel présentée au § 3.3.2.2

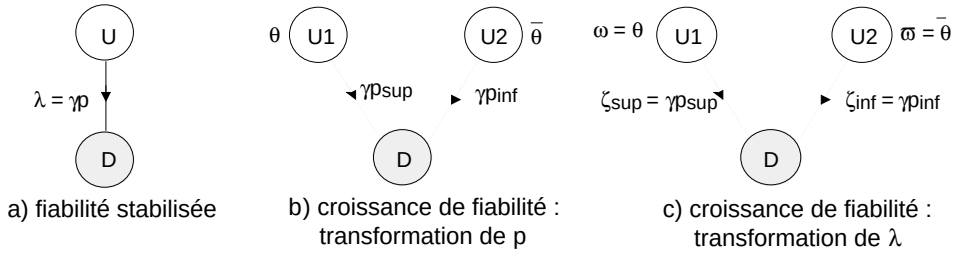


Figure 3.12 –Lien entre les représentations markoviennes des modèles hyperexponentiel en temps discret et en temps continu

3.3.2. Prise en compte de l'environnement d'utilisation

On peut considérer que la fiabilité du logiciel telle qu'elle est perçue par les utilisateurs dans un environnement donné résulte de deux processus fondamentaux : le processus d'exécution qui caractérise l'environnement d'utilisation, et le processus de défaillance qui est conditionné par l'exécution du logiciel et qui dépend essentiellement des caractéristiques internes du logiciel.

Le processus de défaillance à l'exécution est un processus en temps discret qui peut être décrit par le modèle hyperexponentiel en temps discret. Pour prendre en compte la variation des caractéristiques de l'environnement, il est nécessaire de représenter explicitement dans les expressions des mesures de fiabilité les paramètres qui décrivent ces caractéristiques (trajectoire dans l'espace des entrées, variation du taux d'exécution du logiciel , etc.). Dans la suite, nous présentons deux approches permettant de prendre en compte l'environnement d'utilisation du logiciel. La première, appelée *approche par produit de convolution*, permet d'obtenir des mesures de fiabilité en temps discret en tenant compte explicitement de la variation de la trajectoire dans l'espace des entrées. La seconde, appelée *approche markovienne*, consiste à étendre la représentation markovienne présentée sur la figure 3.12-c au cas multi-composant afin d'évaluer des mesures caractérisant la fiabilité du logiciel telle qu'elle est perçue dans le temps par ses utilisateurs dans l'environnement considéré, en tenant compte des probabilités d'activation des composants et de leur taux d'exécution.

3.3.2.1. Approche par produit de convolution

On décompose l'espace des entrées en m sous-domaines disjoints, chacun représentant l'activation d'une classe de service fournie par le logiciel. Le profil d'utilisation du logiciel est décrit par une distribution de probabilité (π_1, …,π_m), où π_i est la probabilité d'activation du sous-domaine i (i= 1…m). On note par Pi(n) l'évolution de la probabilité de défaillance à l'exécution quand il est activé avec des entrées du sous-domaine i. On suppose que Pi(n) peut être décrite par un modèle hyperexponentiel en temps discret.

On se place dans le cas où on a observé le logiciel pendant n_0 exécutions, et on cherche à évaluer la probabilité de défaillance à l'exécution notée P(n), avec $n \geq n_0$ ($n_0 = n_{01} +$

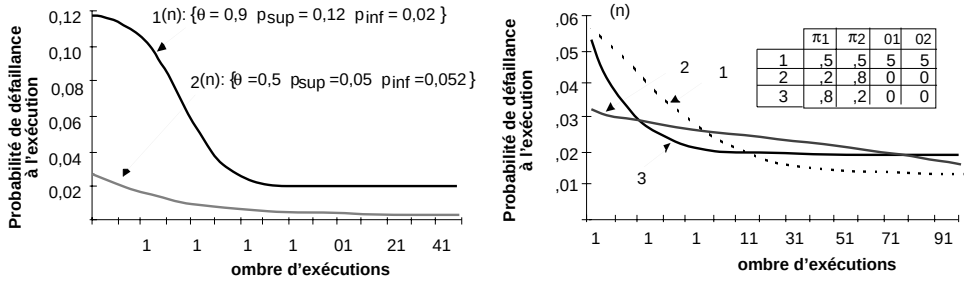
…n_{0m}, n_{0i} étant le nombre d'exécutions par rapport au sous-domaine i). La fonction P(n) permet d'estimer l'évolution future de la probabilité de défaillance à l'exécution à partir de la connaissance du comportement du logiciel par rapport à chacun des sous-domaines d'entrée.

Dans [96], nous donnons l'expression générale de P(n) pour $m \geq 2$ et nous démontrons que P(n) peut être obtenue en calculant le produit de convolution des probabilités Pi(n) et en tenant compte de la distribution (π_1, …, π_m). La convolution résulte du fait que pour un nombre d'exécutions n donné, il faut considérer toutes les combinaisons d'exécutions possibles vis-à-vis de la sélection des entrées par rapport aux sous-domaines d'entrée considérés.

Pour m = 2, l'expression est la suivante :

$$P(n) = P(n/n_0) = \sum_{i=0}^{k-1} \binom{k-1}{i} \pi_1^i \pi_2^{k-1-i} \{\pi_1 P_1(n_{01}+1+i) + \pi_2 P_2(n_{02}+k-i)\} \quad k = n - n_0$$

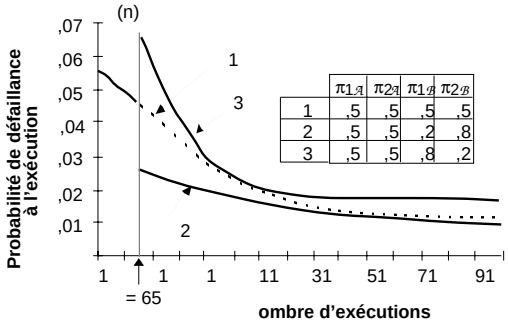
La figure 3.13 donne un exemple d'illustration dans le cas m = 2. On suppose que les probabilités de défaillance à l'exécution P1(n) et P2(n) évoluent conformément aux courbes représentées sur la figure 3.13-a. La figure 3.13-b donne l'évolution de P(n) pour différentes valeurs de π_1 et π_2 sachant que l'on a observé le logiciel durant les 50 premières exécutions ($n_0 = 50$).



a) Évolution de P1(n) et P2(n) b) Évolution de P(n), $n_0 = 50$

Figure 3.13 – Variation de P(n) en fonction du profil d'utilisation

Les courbes C2 et C3 de la figure 3.14 montrent l'évolution de P(n) quand on change d'environnement d'utilisation à partir de l'unité 65. Ces courbes sont à comparer avec C1 qui correspond au cas où on ne change pas d'environnement.



	π_{1A}	π_{2A}	π_{1B}	π_{2B}
1	,5	,5	,5	,5
2	,5	,5	,2	,8
3	,5	,5	,8	,2

Figure 3.14 - Evolution de P(n) après changement du profil d'utilisation

3.3.2.2. Approche markovienne

On considère un système logiciel constitué de k composants ayant chacun une probabilité de défaillance à l'exécution p_i et un taux d'exécution γ_i. Le transfert du contrôle entre les composants est décrit par un processus markovien caractérisé par les paramètres q_{ij} qui définissent la probabilité d'activation du composant j après le composant i, en l'absence de défaillance. Les paramètres γ_i et q_{ij} caractérisent le profil d'utilisation du logiciel dans l'environnement considéré.

Ce modèle structurel a été utilisé dans [125, 126, 142] pour évaluer la fiabilité de logiciels multi-composant en fiabilité stabilisée. Nos travaux ont consisté à étendre ce modèle pour prendre en compte la croissance de fiabilité des composants en utilisant la technique de transformation illustrée sur la figure 3.13-b dans le cas mono-composant.

L'approche que nous avons définie est basée sur les GSPN et se résume par les étapes suivantes [95, 100] :

- Construction d'un GSPN décrivant le comportement en fiabilité stabilisée du système en représentant de façon explicite les probabilités de défaillance p_i, les taux d'exécution γ_i et les probabilités q_{ij} de transfert du contrôle entre les composants.
- Transformation des probabilités p_i conformément au modèle de la figure 3.15.
- Génération du graphe des marquages du GSPN transformé pour obtenir la chaîne de Markov en temps continu du système en croissance de fiabilité.
- Déduction à partir à la chaîne transformée de la chaîne de Markov immergée décrivant la croissance de fiabilité du système en temps discret.

Le traitement de la chaîne de Markov en temps continu permet d'obtenir les mesures de sûreté de fonctionnement caractérisant le comportement du système en tenant compte de façon explicite de son profil d'utilisation. Dans [100], nous décrivons une méthode spécifique permettant de calculer la fonction R(t,t+u) qui représente l'évolution en fonction du temps de la fiabilité pour une durée de mission u courte par rapport au temps d'observation t. La méthode de calcul du temps moyen jusqu'à défaillance (MTTF), de

l'intensité de défaillance et de la disponibilité est présentée dans [101].

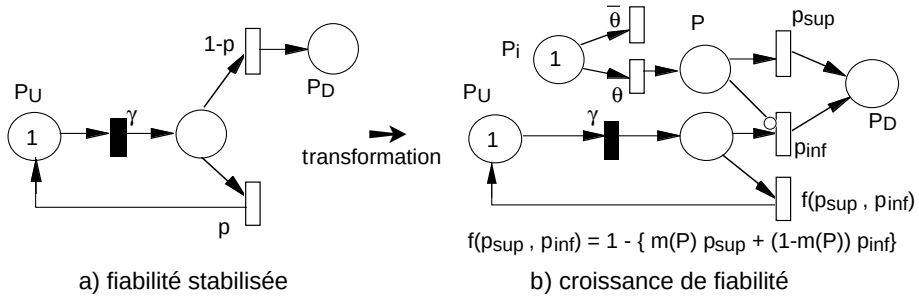


Figure 3.15 –GSPN d'un composant en fiabilité stabilisée et en croissance de fiabilité

L'évaluation de la croissance de fiabilité du système en fonction du nombre d'exécutions est obtenue à partir de la chaîne de Markov immergée. Pour évaluer P(n) et la fonction R(n,n+n0), qui représente l'évolution de la fiabilité en temps discret pour une mission de durée n0, nous avons défini une méthode spécifique utilisant en particulier les techniques d'agrégation d'état définies dans [148].

Afin d'illustrer cette approche de modélisation, nous l'avons appliquée à un logiciel non tolérant aux fautes et à un logiciel tolérant aux fautes constitué d'un bloc de recouvrement. Ces exemples sont détaillés dans [100, 101]. Dans la suite, nous présentons quelques résultats extraits du premier exemple.

Exemple : Considérons un logiciel constitué d'un système d'exploitation (composant 1) et de deux composants constituant le logiciel d'application. Lorsque l'exécution d'un composant d'application est terminée, le système d'exploitation est sollicité afin de déterminer le prochain composant d'application à activer. La chaîne de Markov du système en fiabilité stabilisée est décrite sur la figure 3.16-a. Un état i représente l'activation du composant i et D est l'état de défaillance du système. Pour simplifier la présentation, nous donnons sur la figure 3.16-b la chaîne obtenue après transformation, quand on considère une croissance de fiabilité du composant 2 uniquement en supposant que les autres composants sont en fiabilité stabilisée : θ_2, p_{2sup} et p_{2inf} sont les paramètres du modèle hyperexponentiel en temps discret décrivant cette croissance.

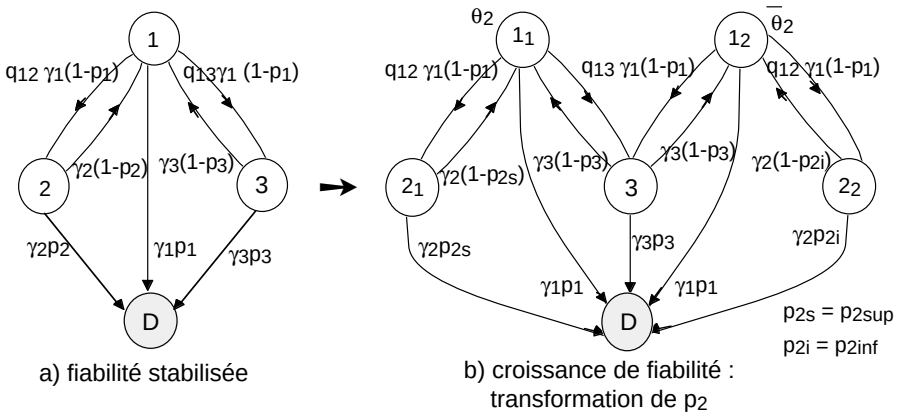


Figure 3.16 – Exemple : illustration de la transformation

Pour cet exemple, et en prenant en compte la croissance de fiabilité des trois composants du système, on démontre que l'intensité de défaillance du système global notée h(t) s'exprime sous la forme suivante :

$$h(t) = a_1\gamma_1\, h_1(a_1 t) + a_2\gamma_2\, h_2(a_2 t) + a_3\gamma_3\, h_3(a_3 t)$$

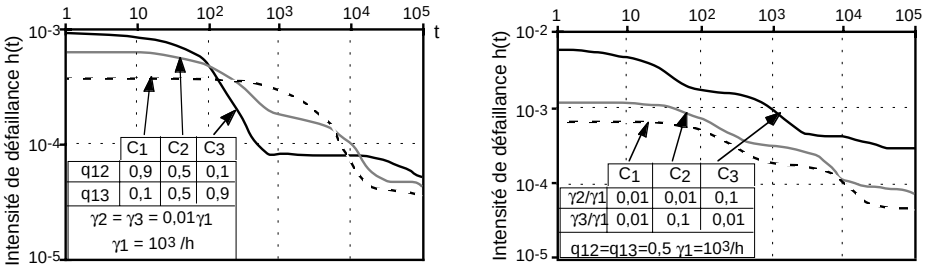
$$a_1 = \frac{1/\gamma_1}{1/\gamma_1 + q_{12}/\gamma_2 + q_{13}/\gamma_3} \quad a_2 = \frac{q_{12}/\gamma_2}{1/\gamma_1 + q_{12}/\gamma_2 + q_{13}/\gamma_3} \quad a_3 = \frac{q_{13}/\gamma_3}{1/\gamma_1 + q_{12}/\gamma_2 + q_{13}/\gamma_3}$$

$$h_i(t) = \frac{\theta_i p_{i\sup} e^{-\gamma_i p_{i\sup} t} + \bar{\theta}_i\, p_{i\inf} e^{-\gamma_i p_{i\inf} t}}{\theta_i e^{-\gamma_i p_{i\sup} t} + \bar{\theta}_i\, e^{-\gamma_i p_{i\inf} t}}$$

Le paramètre a_i représente la proportion du temps pendant lequel le composant i est actif et $h_i(a_i t)$ représente l'évolution de l'intensité de défaillance du composant i en fonction de son temps d'exécution. Cette expression reste valide dans le cas général de systèmes constitués de n composants. Ce résultat constitue une extension de la propriété classique qui relie le taux de défaillance d'un système en fiabilité stabilisée et les taux de défaillance de ses composants, qui a été démontrée et illustrée dans [125, 142].

La figure 3.17 présente des exemples de résultats qui montrent l'impact de la variation des probabilités q_{ij} et des taux d'exécution γ_i sur l'évolution de l'intensité de défaillance. On peut noter que l'évolution de h(t) est plus sensible à la variation des probabilités q_{ij} qu'à celle des taux d'exécution des composants. Ceci est lié au fait que ces paramètres influencent directement la probabilité de défaillance à l'exécution du système.

Ces courbes illustrent bien l'avantage d'une modélisation permettant d'analyser l'impact des caractéristiques de l'environnement sur les mesures de fiabilité du logiciel telles qu'elles sont perçues par les utilisateurs.



a) variation de q_{12} et q_{13} b) variation de γ_1 , γ_2 , γ_3

Figure 3.17 – Evolution de h(t) en fonction des caractéristiques de l'environnement

3.4. Application dans un contexte industriel

La réalité ne correspond pas toujours aux hypothèses considérées dans les développements théoriques et certaines adaptations restent nécessaires pour permettre la mise en œuvre et l'exploitation des modèles dans un contexte industriel. C'est dans cette optique que nous avons complété les travaux présentés dans les paragraphes précédents par la définition et la mise en œuvre d'une méthode globale pour l'analyse et l'évaluation de la fiabilité du logiciel à partir de données de défaillance et de correction collectées durant le développement ou pendant la vie opérationnelle. Cette méthode vise à faciliter l'exploitation des données collectées et à guider l'utilisation des modèles de croissance de fiabilité pour chiffrer l'impact des fautes de conception et assurer le suivi du logiciel. Sa mise en œuvre est facilitée par l'outil SoRel que nous avons développé et qui est actuellement diffusé dans les milieux académique et industriel [114, 117].

3.4.1. Présentation de la méthode

La méthode proposée est décrite dans [115, 116]. La figure 3.18 résume ses principales étapes. L'étape de filtrage vise à analyser la qualité des données collectées et à sélectionner celles qui sont utilisables pour satisfaire les objectifs de l'étude. Cette étape est très importante car l'expérience montre qu'environ 50% uniquement des données collectées sont retenues pour l'étude de fiabilité du logiciel [106, 140]. Le partitionnement des données permet d'effectuer des analyses fines du comportement du logiciel en considérant par exemple, la sévérité des défaillances, la phase du cycle de vie, les composants à l'origine de la défaillance, etc. Trois types d'analyse peuvent être menées sur la base des données retenues : des analyses descriptives, des analyses de tendance ou des évaluations quantitatives basées sur les modèles de croissance de fiabilité. Les analyses descriptives fournissent des statistiques décrivant des relations entre les fautes ou défaillances du logiciel et les caractéristiques du produit (taille, complexité, etc.) ou du processus de développement (types de tests, phase du cycle de vie, etc.). De telles analyses sont très utiles pour améliorer la qualité du logiciel et du processus du développement [84, 176]. Les analyses de tendance visent à : 1) identifier les périodes de croissance et de décroissance de la fiabilité pour assurer un meilleur suivi

de la fiabilité du logiciel, et 2) préparer l'application des modèles de croissance de fiabilité en sélectionnant, sur la base des tendances observées, les modèles et les données utilisables pour effectuer des prévisions en conformité avec les hypothèses de modélisation. Ces analyses sont basées sur des tests statistiques [77, 119] par exemple le test de Laplace, le test de la moyenne arithmétique, etc. Enfin, l'application des modèles de croissance de fiabilité permet d'évaluer des mesures quantitatives telles que, le MTTF, le taux de défaillance, le nombre cumulé de défaillance, etc.

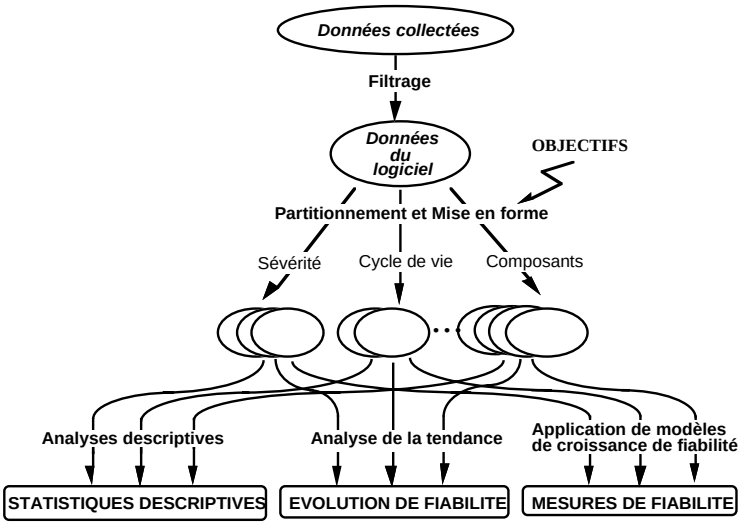


Figure 3.18 - Méthode d'analyse et d'évaluation de la fiabilité du logiciel

3.4.2. Applications

Nous avons appliqué cette méthode à plusieurs cas réels dont un résumé est donné sur la figure 3.19. Les quatre premiers systèmes correspondent à une famille de logiciels d'autocommutateurs téléphoniques développés par la compagnie Brésilienne des télécommunications (TELEBRAS). Le dernier correspond à un logiciel d'un équipement téléphonique pour lequel les données ont été fournies par le CNET-Lannion. La figure 3.19 donne pour chacun de ces systèmes :

- le langage et la taille du logiciel,
- la durée de la période au cours de laquelle les données ont été collectées,
- la phase du cycle de vie au cours de laquelle la collecte a été effectuée (validation (Val), opération (Op)),
- le nombre total de systèmes (#Syst.) en validation ou en opération,
- le nombre de relevés de défaillance et de correction (#RD/RC) qui ont été collectés.

Système	Langage	Taille	Durée	Phases	# Syst.	# RD / RC
TROPICO-R 1500	Assembleur	300 K-Octets	27 mois	Val. /Op.	15	465
TROPICO-R 4096	Assembleur	335 K-Octets	32 mois	Val./Op.	42	210
TROPICO-RS	Assembleur	420 K-Octets	47 mois	Op.	37	212
TROPICO-RA	CHILL	815 K-Lignes	68 mois	Val./Op.	146	3063
Equipement Téléphonique	PLM-86	$5\ 10^5$ Instructions	16 mois	Val.	4	2150

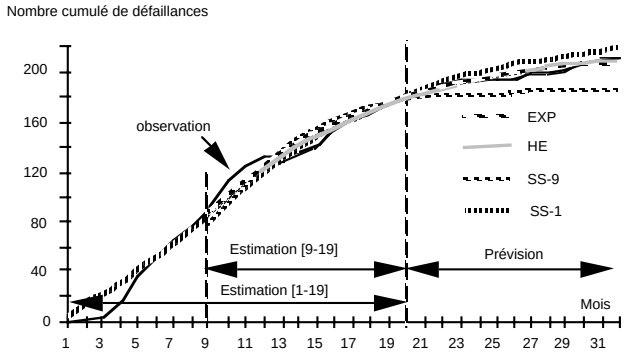
Figure 3.19– Caractéristiques des systèmes étudiés

L'étude effectuée sur le logiciel de l'équipement téléphonique a permis d'illustrer sur la base des données fournies comment mettre en œuvre une étude de croissance de fiabilité du logiciel dans un contexte industriel, de montrer sur un cas concret comment appliquer le modèle hyperexponentiel en temps continu et le type de résultats que l'on peut obtenir, et enfin de comparer ce modèle avec d'autres modèles NHPP tels que le modèle exponentiel de Goel-Okumoto et le modèle en forme de S de Yamada. Les résultats sont détaillés dans [103-106]. En particulier, des recommandations pour la collecte de données sont présentées dans [104].

L'étude sur les autocommutateurs téléphoniques développés par TELEBRAS a été l'occasion d'analyser la fiabilité du logiciel dans un contexte évolutif. En effet, les quatre autocommutateurs appartiennent à la même famille appelée TROPICO. Les deux premiers ont été développés dans un environnement similaire et le passage d'un produit à un autre s'est traduit essentiellement par des évolutions des spécifications du logiciel sans modification majeure du matériel [97]. Le troisième a fait l'objet d'évolution des spécifications et du processus de développement [168]. Enfin, le quatrième a subi des modifications majeures au niveau des spécifications, du langage de programmation, du processus de développement et du matériel [98, 107]. Chacun de ces systèmes a fait l'objet d'une étude portant sur l'évaluation de la fiabilité du logiciel [97, 98, 120, 168]. Pour le système TROPICO-RA, les données collectées ont permis également d'évaluer la fiabilité du matériel en tenant compte des fautes de conception [98].

La figure 3.20 présente un exemple de résultats obtenus dans le cadre de l'étude du TROPICO-R 4096. Les huit premiers mois correspondent à la fin de la validation et la période [9, 32] correspond au début de la vie opérationnelle au cours de laquelle 42 systèmes ont été mis en service progressivement. La figure 3.20 donne les prévisions du nombre cumulé de défaillances obtenues avec le modèle hyperexponentiel [11], le modèle exponentiel (EXP) de Goel-Okumoto et le modèle en forme de S (SS) de Yamada *et al.* EXP, HE, et SS9 donnent les prévisions pour la période [20, 32] en estimant les paramètres avec les données collectées entre les unités [9, 19]. Sur cette période, les tests de tendance ont révélé une croissance monotone de la fiabilité. C'est la raison pour laquelle les résultats de HE et EXP sont conformes au comportement observé. Le modèle SS donne de mauvais résultats car il est appliqué sur une période de croissance de fiabilité uniquement. Les prévisions sont nettement meilleures quand on l'applique avec les données recueillies pendant les 20 premiers mois correspondant à

une décroissance suivie d'une croissance de fiabilité (c'est-à-dire, en accord avec ses hypothèses). Ces derniers sont représentés par la courbe SS-1.



# défaillances	EXP	HE	SS-9	SS-1	Observées
[20 - 32 mois]	33	37	9	40	34

Figure 3.20- Evolution du nombre cumulé de défaillances : observation et estimation

Cet exemple illustre la nécessité de faire des analyses de tendance pour guider l'application des modèles de croissance de fiabilité. Sans de telles analyses, il y a de fortes chances que les prévisions des modèles s'écartent significativement des observations. Malheureusement, c'est ce qui s'est produit dans le passé causant ainsi un scepticisme des industriels vis-à-vis de l'applicabilité des modèles pour l'évaluation de la fiabilité du logiciel.

Une étude comparative de la fiabilité des logiciels TROPICO-R 1500, 4096 et RS est présentée dans [99]. Cette étude nous a permis d'analyser l'impact de certaines caractéristiques du logiciel TROPICO-R et de son processus de développement sur l'évolution de sa fiabilité. En particulier, nous avons pu observer et chiffrer une amélioration de la fiabilité du logiciel TROPICO-R 4096 par rapport à celle du TROPICO-R 1500 en dépit des modifications introduites. Cette amélioration, liée à l'élimination des fautes de conception, est confortée par le fait que les processus de développement de ces deux produits sont similaires. Cependant, ce comportement n'a pas été observé pour le TROPICO-RS dont le processus de développement a été modifié. De telles analyses s'inscrivent dans le cadre des travaux initialisés dans notre groupe de recherche qui visent à incorporer dans les évaluations d'un produit donné, des informations sur le comportement de produits similaires précédents et sur leur processus de développement afin d'améliorer la précision des prévisions et d'obtenir des estimations de la fiabilité assez tôt dans le cycle de vie du logiciel [102, 130].

3.5. Analyse critique et prospectives

La discipline d'évaluation de la croissance de fiabilité en général, et plus particulièrement la fiabilité du logiciel, se trouve de nos jours dans une situation paradoxale. En effet, en

dépit de la reconnaissance générale que les fautes de conception sont la source prédominante de défaillance des systèmes, et des avancées théoriques réalisées dans ce domaine, force est de constater que l'intégration effective des méthodes d'évaluation de la croissance de fiabilité dans les processus de développement industriels reste encore très limitée. Plusieurs raisons peuvent être avancées pour expliquer cette situation. La raison principale est liée au fait que les résultats obtenus jusqu'à présent ne permettent pas de fournir des estimations de la fiabilité du logiciel très tôt dans le cycle de développement afin de réagir sur la conception et le processus de développement avant le début de l'implémentation. En effet, les prévisions effectuées avec les modèles de croissance de fiabilité sont basées exclusivement sur des données de défaillance et de correction collectées durant les phases de tests ou pendant la vie opérationnelle, c'est-à-dire, une fois que le système est développé. Pour obtenir des prévisions plus tôt dans le cycle de vie, il est nécessaire d'incorporer dans les modèles, des informations issues du processus de développement ou bien décrivant le produit durant les phases de spécification et de conception, ou éventuellement des données collectées sur des produits antérieurs. Les études permettant d'établir des corrélations entre le comportement dynamique du logiciel tel qu'il est observé par les utilisateurs, les mesures de complexité relatives au logiciel en tant que produit (dérivées à partir du code par exemple), et les mesures décrivant le processus de développement sont à un stade embryonnaire. Bien que de telles corrélations ont été observées a posteriori sur plusieurs logiciels, il n'existe pas encore de théorie permettant d'obtenir un modèle prévisionnel qui décrit le lien entre la fiabilité du logiciel et ces mesures statiques. L'utilisation de données collectées sur une famille de produits similaires et incorporant des informations sur le processus de développement constitue une voie prometteuse qui a été proposée dans [102, 130]. Une approche appelée produit-processus basée sur les probabilités bayesiennes a été définie et un cas d'étude montrant le bien fondé de cette approche a été analysé dans [102, 130]. Néanmoins, l'aboutissement de ces travaux et la validation des résultats théoriques nécessitent une étroite collaboration entre la recherche et l'industrie pour disposer de données d'expérience servant de support pour ces travaux.

4

4. ÉVALUATION QUANTITATIVE DE LA SÉCURITÉ-CONFIDENTIALITÉ

La part des défaillances informatiques dues à des malveillances (intrusions, virus, …) n'a pas cessé d'augmenter pendant la dernière décennie. Ceci conduit naturellement à se poser la question de l'évaluation de l'aptitude des systèmes à résister à des attaques éventuelles par des intrus. Dans ce chapitre, nous présentons une approche originale basée sur une évaluation quantitative de la sécurité à partir de l'analyse des vulnérabilités révélées pendant l'exploitation opérationnelle. Cette approche vise à fournir aux administrateurs de systèmes informatiques des moyens d'aide à la décision et des outils pour surveiller l'évolution de la sécurité-confidentialité (au sens rappelé au Chapitre 1), dans des environnements caractérisés par des modifications fréquentes de la politique de sécurité, des configurations opérationnelles, du comportement des utilisateurs, etc.

Les travaux résumés dans ce chapitre ont été effectués dans le cadre des thèses de Doctorat de Marc Dacier [49] et Rodolphe Ortalo [163], dont j'ai assuré l'encadrement conjointement avec Yves Deswarte. Ces travaux ont été effectués, en partie, dans le cadre des projets européens PDCS-2 et DeVa. Dans la suite, nous utiliserons le terme "sécurité" pour désigner la sécurité-confidentialité.

4.1. Contexte des travaux et principales contributions

Le développement des réseaux de communication, l'interconnexion à large échelle des systèmes informatiques et l'émergence d'environnements et d'applications distribuées favorisant le partage d'information et le travail coopératif entre les utilisateurs, rend les systèmes informatiques de plus en plus vulnérables face à des attaquants éventuels. En plus des fautes de conception, une part non négligeable de ces vulnérabilités est due à une utilisation laxiste des mécanismes de protection par certains utilisateurs qui ne sont pas prêts, pour améliorer la sécurité, à renoncer à la facilité d'utilisation et de partage d'information. En effet, ces utilisateurs ne réalisent pas que, même s'ils ne se sentent pas directement concernés par la sécurité, leur comportement peut mettre en danger d'autres utilisateurs possédant des informations sensibles ou confidentielles. Dans un tel

contexte, il convient d'obtenir un bon compromis entre sécurité, facilité d'utilisation, partage de l'information et coopération entre les utilisateurs. Une évaluation quantitative du niveau de sécurité courant du système peut permettre aux administrateurs des systèmes informatiques de surveiller l'évolution de la sécurité en fonction des modifications survenant dans l'environnement, le comportement des utilisateurs, etc. et d'identifier les failles qui peuvent être éliminées en apportant une amélioration significative de la sécurité sans avoir d'incidence majeure sur les fonctions du système. De plus, cette évaluation peut fournir des données pragmatiques permettant de sensibiliser les utilisateurs au problème de la sécurité et, si nécessaire, de les convaincre de modifier leur configuration de travail au regard des évolutions observées.

Les méthodes classiques d'évaluation de la sécurité qui sont basées, soit sur des critères normalisés, soit sur l'analyse des risques, ne permettent pas de répondre à ces objectifs. En effet, les évaluations basées sur les critères sont de nature qualitative et consistent à attribuer un niveau de confiance à un système en fonction de ses caractéristiques fonctionnelles (authentification, droits d'accès, etc.) et des méthodes qui ont été utilisées pour son développement et sa validation (spécifications formelles, preuves, tests, etc.). Parmi ces critères, on peut citer les TCSEC développés par le département américain de la défense [57], les critères harmonisés européens ITSEC [25], les critères fédéraux [162] et plus récemment les critères communs [26]. Les méthodes d'analyse des risques sont généralement utilisées pendant le développement et consistent à estimer l'efficacité et le coût des parades à mettre en œuvre pour éliminer ou réduire le risque associé à la réalisation des menaces pesant sur les éléments sensibles du système (voir [49] pour une présentation plus détaillée de ces méthodes).

Les critères normalisés et les méthodes d'analyse des risques sont très utiles pour guider le développement. Cependant, il ne suffit pas de bien concevoir un système, il faut en plus s'assurer qu'il est bien utilisé pendant son exploitation opérationnelle et que les évolutions éventuelles du système et de son environnement ne mettent pas en danger les objectifs de sécurité. Les travaux synthétisés dans ce chapitre visent à pallier ces insuffisances. Nous avons développé une approche originale qui permet d'identifier les vulnérabilités mettant en danger les objectifs de sécurité et d'évaluer leurs impacts sur ces objectifs. L'objectif est de surveiller l'évolution (amélioration, dégradation, ou stabilisation) de la sécurité en fonction des évolutions de l'environnement, des configurations, du comportement des utilisateurs, etc. Pour ceci, nous avons défini des mesures quantitatives permettant d'estimer la capacité d'un système à résister à une attaque en utilisant des techniques probabilistes similaires à celles employées pour l'évaluation de la fiabilité. Il est important de noter que nous ne cherchons pas à évaluer la probabilité qu'un système soit attaqué durant sa vie opérationnelle. En effet, ceci nécessiterait une modélisation de la population d'attaquants et de leur stratégie de choix d'une cible, ce qui nous semble —à l'heure actuelle— irréalisable de façon réaliste. Au contraire, dans nos travaux nous ne nous intéressons qu'au système informatique et à ses utilisateurs, toutes choses pour lesquelles nous disposons de données tangibles.

Dans la suite, nous résumons les principes de notre approche et nous présentons quelques résultats d'une expérimentation que nous avons menée afin de valider cette approche et d'illustrer son applicabilité dans un contexte réel.

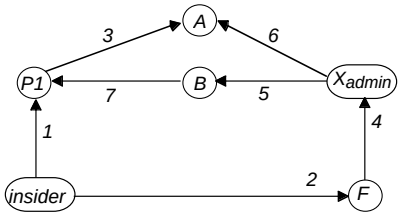
4.2. Présentation de l'approche

Notre approche peut se résumer en trois étapes :

1) Identification des vulnérabilités du système, représentation de ces vulnérabilités sous la forme d'un *graphe des privilèges*, et définition des *objectifs d'évaluation* de la sécurité à partir de ce graphe.
2) Construction du *processus d'intrusion* décrivant les scénarios pour atteindre la cible.
3) Construction d'un *modèle d'évaluation* à partir du processus d'intrusion pour le calcul des mesures de sécurité.

4.2.1. Graphe des privilèges et objectifs d'évaluation

Le graphe des privilèges est un modèle formel qui a été introduit dans [50] pour représenter les vulnérabilités de la configuration opérationnelle d'un système informatique. Dans un graphe des privilèges, un nœud X représente l'ensemble des privilèges (c'est-à-dire, les droits) d'un utilisateur ou d'un ensemble d'utilisateurs (par exemple, un groupe dans Unix). Un arc existe entre un nœud X et un nœud Y, s'il existe une méthode permettant à un utilisateur ayant les privilèges de X d'obtenir ceux représentés par le nœud Y. Les arcs représentent donc des vulnérabilités présentes dans le système. Ces vulnérabilités peuvent correspondre à des fautes qu'il est nécessaire d'éliminer, mais peuvent également correspondre à des méthodes de transfert de privilèges parfaitement licites qui sont utiles au fonctionnement du système ou bien qui ont été conçues pour améliorer la sécurité. Pour le système Unix, on peut citer comme exemples de vulnérabilités : 1) l'absence d'un mot de passe ou la définition d'un mot de passe facile à deviner, 2) la définition de mauvaises protections de fichiers : par exemple si un utilisateur ne protège pas son fichier ".rhost" en écriture, il laisse la possibilité à un attaquant d'acquérir l'ensemble de ses privilèges et éventuellement d'installer un cheval de Troie. Il est important de mentionner que les fichiers ".rhost" sont en principe définis pour faciliter le travail coopératif entre un groupe d'utilisateurs ayant une confiance mutuelle sans que chacun ait besoin de connaître le mot de passe des autres (ceci permet en quelque sorte d'améliorer la sécurité). Cependant, l'utilisation de ces fichiers présente des risques vis-à-vis de la sécurité, même s'ils sont protégés en écriture, en particulier dans le cas où un utilisateur du groupe cède ses privilèges, volontairement ou à son insu, à un utilisateur malveillant n'appartenant pas au groupe et auquel les autres utilisateurs ne font pas confiance. D'autres exemples de méthodes de transfert de privilèges, licites ou illicites, connues pour le système Unix sont décrits dans [51, 76]. Un exemple de graphe de privilèges est présenté sur la Figure 4.1. Les étiquettes sur les arcs identifient des classes de vulnérabilités classiques. Les nœuds (*A*, *B*, *F*) représentent des privilèges d'utilisateurs et (X_{admin}, *P*) des privilèges de groupes d'utilisateurs. Le nœud *insider* représente les privilèges minimaux dont dispose tout utilisateur du système (par exemple, le privilège de se connecter ou de changer son mot de passe).



Classes de vulnérabilités : 1) X peut modifier le .rhost de Y ; 2) X peut deviner le mot de passe de Y ; 3) X peut modifier le .tcshrc de Y ; 4) X est membre de Y ; 5) Y utilise un programme modifiable par X ; 6) X peut modifier l'exécution d'un programme avec le bit setuid positionné, appartenant à Y. 7) X est dans le .rhost de Y.

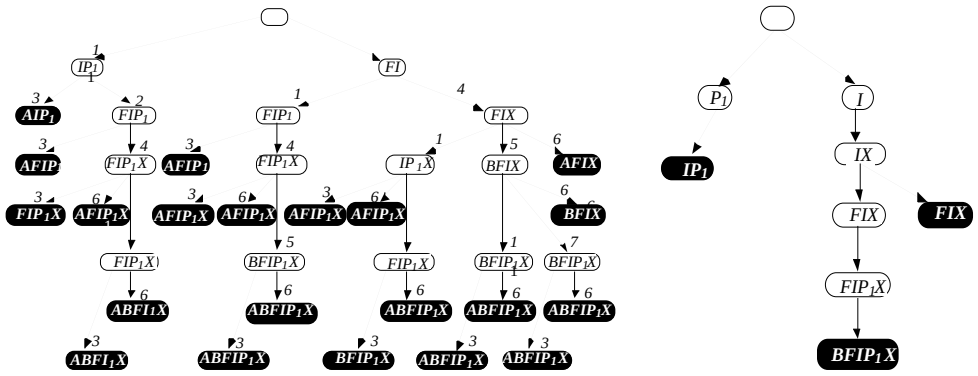
Figure 4.1 – Exemple de graphe de privilèges

Dans un graphe des privilèges, on peut identifier des nœuds que nous appellerons "**cible**" qui correspondent à des privilèges que l'on souhaite protéger (par exemple, les privilèges du *super-utilisateur*). Ces nœuds représentent les objectifs de sécurité du système. Par ailleurs, on peut identifier des nœuds appelés "**attaquant**" qui représentent les privilèges d'attaquants potentiels. Tous les chemins entre un nœud attaquant (par exemple "*insider*") et un nœud cible (par exemple "*A*") sont des possibilités à l'attaquant pour mettre en défaut la politique de sécurité du système. De tels chemins existent dans la plupart des systèmes même s'ils ne sont pas tous facilement exploitables. Par exemple, tous les mots de passe peuvent être devinés : certains sont faciles à trouver par des outils tels que crack [155] parce qu'ils figurent dans un dictionnaire, alors que d'autres nécessitent plus d'effort et de temps. Ceci est vrai pour toutes les classes de vulnérabilités : certaines sont facilement exploitables par un attaquant alors que d'autres nécessitent beaucoup de compétence, ténacité ou chance.

4.2.2. Processus d'intrusion

L'accroissement des privilèges de l'attaquant au fur et à mesure de sa progression vers la cible est caractérisé par un graphe d'état appelé **processus d'intrusion**, où chaque état identifie les privilèges qui ont été obtenus depuis le début du processus d'attaque et les transitions entre états ont lieu quand l'attaquant réussit à exploiter une vulnérabilité lui permettant d'acquérir de nouveaux privilèges. Certaines hypothèses sur le comportement d'un attaquant potentiel sont nécessaires pour identifier les scénarios d'attaque permettant de construire le processus d'intrusion. Tout d'abord, nous supposons que l'attaquant est sensé, et qu'il n'essaiera pas de mettre en œuvre une attaque lui permettant d'obtenir des privilèges qu'il possède déjà. De plus, pour caractériser la progression de l'attaquant vers la cible, nous avons étudié différentes hypothèses dans [53] et plus récemment dans [165], chacune correspondant à un modèle d'attaque particulier. Les figures 4.2-a et 4.2-b donnent les processus d'intrusion obtenus à partir du graphe des privilèges de la figure 4.1 en considérant deux hypothèses différentes du comportement de l'attaquant, notées MT et ML, respectivement. Pour cet exemple, nous rappelons que *A* est la cible et *insider* (noté *I*) est l'attaquant. Avec l'hypothèse MT, à chaque étape du processus d'attaque, l'attaquant peut choisir une

attaque parmi toutes celles qu'il a identifiées durant les étapes précédentes et qui n'ont pas abouti. Par contre, avec l'hypothèse ML, seules les attaques réalisables avec les nouveaux privilèges acquis à partir du nœud du graphe qu'il vient d'atteindre sont considérées (voir [165] pour une discussion détaillée de ces hypothèses).



a) Hypothèse MT b) Hypothèse ML

Figure 4.2 – Processus d'intrusion pour le graphe des privilèges de la figure 4.1

4.2.3. Mesures pour la sécurité et modèle d'évaluation

La sécurité peut être caractérisée par l'effort nécessaire à un attaquant pour atteindre la cible, en considérant tous les chemins entre le nœud "attaquant" et le nœud "cible". Pour évaluer cette mesure, il est nécessaire, dans une première étape, de définir pour chaque arc du graphe des privilèges, une variable qui caractérise l'effort nécessaire à un attaquant pour exploiter la classe de vulnérabilité représentée par l'arc. La notion d'*effort* intègre plusieurs paramètres, par exemple le temps nécessaire pour réussir l'attaque, sa facilité de mise en œuvre, la disponibilité d'outils pour la réaliser, etc. [49, 52, 143]. En fait, il s'agit d'une variable multi-dimensionnelle à laquelle on peut attribuer une valeur caractérisant le taux de succès de l'attaque élémentaire correspondante. L'idéal pour estimer cette valeur serait d'analyser des scénarios d'intrusions connues. Malheureusement, de telles données n'existent pas. Une autre solution consiste à ordonner les classes de vulnérabilités par ordre de difficulté croissante. Un taux de succès élevé correspondrait à une attaque facile et un taux faible à une attaque difficile. Cette classification peut être définie en s'appuyant sur des jugements d'experts et sur des données décrivant le profils des utilisateurs, qui sont collectées par des outils de détection d'intrusions [4, 144]. On peut noter, qu'une méthode d'évaluation similaire est suggérée dans le manuel d'évaluation ITSEM [91] qui accompagne les critères ITSEC.

À partir des valeurs attribuées aux vulnérabilités, le problème de l'évaluation quantitative de la sécurité est d'estimer une valeur globale caractérisant l'effort nécessaire à un attaquant pour mettre en défaut la sécurité en tenant compte de tous les scénarios d'attaque menant à la cible. Dans nos travaux, nous avons considéré, comme mesure de référence, l'effort moyen pour atteindre la cible. Cette mesure est facile à interpréter : plus l'effort moyen est grand, meilleure est la sécurité. Nous avons choisi les chaînes de

Markov comme modèle de base pour évaluer cette mesure. Les chaînes de Markov permettent d'obtenir des expressions analytiques simples des mesures de sécurité tout en satisfaisant un ensemble de propriétés intuitives sur l'évolution de ces mesures en fonction de la structure du graphe des privilèges. Les raisons qui nous ont amené à faire ce choix sont développées dans [49, 52].

Le modèle est basé sur l'hypothèse que la probabilité de succès d'une attaque élémentaire correspondant à une transition dans le processus d'intrusion est décrite par une distribution exponentielle définie par : $P(e) = 1 - \exp(-\lambda e)$, où e est la variable d'effort et λ est le taux de succès associé à l'attaque. Par conséquent, l'effort moyen nécessaire au succès d'une attaque élémentaire est donné par la valeur $1/\lambda$. Cette distribution implique, que si un chemin existe, l'attaquant finira par atteindre la cible s'il dépense un effort suffisant. Par conséquent, nous nous plaçons dans le cas pessimiste où un attaquant persiste jusqu'à atteindre la cible et nous ne considérons pas le cas où le processus d'intrusion est interrompu.

Avec l'hypothèse markovienne, à chaque transition du processus d'intrusion, on attribue le taux de succès associé à la vulnérabilité correspondante. Plusieurs mesures peuvent être calculées à partir de la chaîne de Markov ainsi obtenue, parmi elles, l'effort moyen pour atteindre la cible, noté METF (pour *Mean Effort To security Failure* par analogie avec le *Mean Time To Failure*).

La chaîne de Markov associée au processus d'intrusion est acyclique. Ceci permet d'obtenir des expressions analytiques simples du METF. Si on note par $METF_k$, l'effort moyen calculé à partir d'un état initial k, on obtient l'expression récursive :

$$METF_k = E_k + \sum_{i \in S(k)} P_{ki} \times METF_i \text{ où } E_k = 1 \Big/ \sum_{j \in S(k)} \lambda_{kj} \text{ et } P_{ki} = \lambda_{ki} \times E_k$$

$S(k)$ est l'ensemble des états atteignables à partir de l'état k en une transition, et λ_{kj} est le taux de transition de l'état k vers l'état j.

Selon que l'on considère l'hypothèse MT ou l'hypothèse ML pour caractériser le comportement de l'attaquant, différentes évolutions de la mesure METF (notée $METF_{MT}$ ou $METF_{ML}$) peuvent être observées, à la suite de modifications du graphe des privilèges se traduisant par une augmentation ou une diminution du nombre de chemins vers la cible ou par une variation des taux de transition. Une analyse du comportement attendu de ces mesures est présentée dans [165].

En fonction des variations observées de la mesure et des modifications du graphe, les administrateurs du système sont en mesure de prendre des décisions adéquates pour éventuellement modifier la configuration du système quand une dégradation de la sécurité est observée. Un exemple illustratif est présenté dans [51].

4.3. Validation expérimentale

Afin de valider notre approche et d'analyser la pertinence de nos mesures et leur aptitude à représenter fidèlement l'évolution de la sécurité d'un système, nous avons développé

ESOPE (pour “Évaluation de la Sécurité OPÉrationnelle”) [163], un prototype logiciel qui rassemble un ensemble d’outils permettant d’automatiser les différentes étapes de notre approche, et nous avons mené une expérimentation sur un système réel de grande taille. L’expérience que nous avons menée a été effectuée sur un système Unix distribué constitué de plusieurs centaines de stations de travail connectées à un réseau local (il s’agit d’un sous-ensemble du réseau informatique du LAAS). Ce système est utilisé en moyenne par environ 700 utilisateurs partageant un même système de fichiers (NFS). Nous avons observé le système sur une base quotidienne durant la période juin 1995—mars 1997 pendant laquelle 674 graphes de privilèges ont été construits (un pour chaque jour). Pour chacun de ces graphes, nous avons calculé les valeurs des mesures METF et nous avons analysé tous les événements qui sont à l’origine de modifications dans le graphe des privilèges ou dans les mesures calculées. Une présentation détaillée de l’expérience et des résultats est donnée dans [165]. Nous donnons ici uniquement quelques informations et exemples de résultats.

Pour le système que nous avons étudié, la sécurité n’est pas une préoccupation majeure des utilisateurs. Ceci justifie le nombre important de vulnérabilités que nous avons observé durant l’expérience. Il est important de préciser aussi que, notre objectif principal ayant été de valider nos mesures de sécurité, nous avons observé l’évolution du système sans chercher à éliminer certaines vulnérabilités ou à convaincre les utilisateurs à modifier leur configuration de travail. Durant l’expérience, nous avons étudié 13 classes de vulnérabilités parmi les plus connues pour le système Unix [76], et nous avons évalué les mesures METF en considérant comme attaquant l’utilisateur “insider” et comme cibles : le compte super-utilisateur “root” (objectif 1), et le groupe des administrateurs du système “admin_group” (objectif 2).

Nous avons défini une échelle à quatre niveaux pour attribuer des valeurs aux paramètres λ associés aux vulnérabilités (exprimés sous forme de taux de succès par unité d’effort) : niveau 1 = 10^{-1} ; niveau 2 = 10^{-2} ; niveau 3 = 10^{-3} ; niveau 4 = 10^{-4}. Le niveau 1 correspond aux vulnérabilités qui sont les plus faciles à exploiter et le niveau 4 aux plus difficiles. Au stade actuel de nos travaux, ces valeurs ont été définies de façon raisonnable mais relativement arbitraire. Cependant, ceci ne remet pas en cause la validité des résultats de l’expérience car notre objectif n’est pas d’évaluer une valeur absolue de la sécurité du système mais d’analyser l’évolution des mesures en fonction des modifications de la configuration opérationnelle.

La figure 4.3 donne l’évolution de $METF_{MT}$ et $METF_{ML}$ quand la cible est “root”. Afin de mieux percevoir la pertinence de nos mesures, nous donnons également sur la figure 4.4 l’évolution du nombre de chemins durant l’expérience et la valeur du METF correspondant au plus court chemin menant à la cible (noté $METF_{SP}$).

Figure 4.4 – Évolution de $METF_{SP}$ et du nombre de chemins vers la cible

Les flèches noires sur la figure 4.3 font référence à des événements qui ont conduit à une variation significative des mesures relatives à la cible "root". Par contre, les événements identifiés par les flèches grises n'ont eu d'impact que sur les mesures relatives à la cible admin_group (cf. [165] pour la description de ces événements et pour les résultats concernant la cible admin_group). Ceci illustre bien le fait qu'une mesure de sécurité est

toujours directement liée à l'objectif d'évaluation considéré et n'a pas de sens dans l'absolu. En effet, le système est sûr tant que ses principaux objectifs sont satisfaits, même s'il est relativement facile d'effectuer certaines actions légitimes ou illégitimes qui ne mettent pas en défaut ces objectifs. De plus, différents objectifs d'évaluation peuvent conduire à des perceptions différentes de la sécurité.

À partir de l'analyse des figures 4.3 et 4.4, et des résultats présentés dans [165] on peut retenir les conclusions suivantes sur la pertinence des quatre mesures considérées.

Chemin le plus court. $METF_{SP}$ est moins sensible aux modifications du système que $METF_{MT}$ et $METF_{ML}$. Cette mesure fournit une information importante en identifiant le chemin le plus facile à exploiter par un attaquant, par contre, elle ne prend pas en compte la contribution de chemins ayant une longueur relativement plus élevée et qui peuvent être aussi dangereux que le chemin le plus court. Plus que sa longueur, c'est la nature de ce chemin et les vulnérabilités correspondantes qui sont intéressantes pour améliorer la sécurité. On peut noter aussi que ce chemin fait partie de ceux qui ont un impact majeur sur $METF_{MT}$ et $METF_{ML}$ (cf. l'événement #29 et l'évolution des mesures durant la période P).

Nombre de chemins. Cette mesure est trop sensible aux modifications et ne permet pas d'identifier les chemins qui sont les plus dangereux vis-à-vis de l'objectif de sécurité considéré. En effet, une variation importante du nombre de chemins ne se traduit pas nécessairement par une dégradation ou une amélioration de la sécurité. Par conséquent, l'utilisation de cette mesure comme indicateur pour surveiller l'évolution de la sécurité déclencherait un grand nombre d'alarmes, parmi lesquelles certaines peuvent s'avérer non justifiées car l'impact sur la sécurité est faible. Par conséquent, cette mesure ne semble pas pertinente pour assurer un bon suivi de la sécurité par les administrateurs.

$METF_{MT}$ et $METF_{ML}$. Ces deux mesures présentent un comportement intéressant avec des périodes de stabilité séparées par des variations significatives. Une analyse détaillée des modifications des graphes des privilèges qui se sont produites durant l'expérience a révélé que chaque variation de ces mesures a toujours été liée à un événement pertinent vis-à-vis de l'objectif de sécurité considéré. Une diminution significative de l'une de ces deux mesures est synonyme d'une dégradation de la sécurité, nécessitant ainsi une attention particulière de la part de l'administrateur pour prendre les décisions adéquates afin de faire face aux vulnérabilités qui sont à l'origine de cette évolution. Dans le cas de la mesure $METF_{ML}$, une dégradation de la sécurité n'est pas toujours liée à une augmentation du nombre de chemins menant à la cible, mais peut se produire aussi quand le nombre de chemins diminue. Ceci se produit en particulier quand on supprime un chemin difficile qui est tel que la première vulnérabilité intervenant dans ce chemin est facile à exploiter, conduisant ainsi l'attaquant à choisir ce chemin avec une probabilité élevée (cf. [165] pour une comparaison plus approfondie des hypothèses MT et ML et du comportement des mesures associées à ces hypothèses).

En conclusion, l'expérience que nous avons menée a confirmé que les mesures $METF_{MT}$ et $METF_{ML}$ sont pertinentes pour suivre l'évolution de la sécurité et offrent des informations plus intéressantes que le plus court chemin ou bien le nombre de

chemins. Au stade actuel, aucune donnée tangible ne permet de considérer que l'hypothèse MT est plus réaliste que l'hypothèse ML ou vice-versa pour représenter le comportement d'un attaquant. Cependant, comme nous l'avons montré à travers l'expérience que nous avons menée, cette discussion n'est pas fondamentale dans le sens où les deux hypothèses permettent de fournir des informations pertinentes pour suivre l'évolution de la sécurité. Néanmoins, on peut noter que d'un point de vue pratique, la mesure $METF_{ML}$ est plus facile à calculer que la mesure $METF_{MT}$ car le processus d'intrusion correspondant, est plus simple.

4.4. Conclusion

Les travaux synthétisés dans ce chapitre offrent une voie originale et nouvelle pour aborder le problème de l'évaluation quantitative de la sécurité opérationnelle. En utilisant une approche probabiliste, nous nous démarquons des approches d'évaluation classiques qui sont basées, soit sur une conception binaire de la sécurité, de type tout ou rien, soit sur des critères qualitatifs ou des analyses de risques qui sont plus adaptés pour guider le développement, et moins pour évaluer la sécurité opérationnelle dans des environnements où des modifications fréquentes surviennent dans le comportement des utilisateurs, de la configuration du système, etc. Dans nos travaux, nous nous sommes plus particulièrement intéressé aux systèmes qui sont a priori non sûrs et pour lesquels, un compromis doit être toujours recherché entre le degré de liberté laissé aux utilisateurs, et les exigences de sécurité qu'il faut satisfaire. L'expérimentation que nous avons menée sur un système réel de grande taille nous a permis de montrer que les mesures quantitatives que nous proposons peuvent contribuer à faciliter la tâche des administrateurs des systèmes pour suivre l'évolution de la sécurité opérationnelle et analyser l'impact des vulnérabilités sur les objectifs de sécurité. Nos travaux ont d'abord visé les systèmes informatiques, et plus particulièrement le système Unix. Néanmoins, ils sont applicables à d'autres types de systèmes, et plus généralement à des systèmes d'information [163, 164]. Plusieurs directions sont envisagées pour poursuivre ces travaux. Tout d'abord, il est nécessaire de définir des méthodes permettant de fournir une estimation réaliste des variables d'effort intervenant dans le modèle d'évaluation pour les différentes classes de vulnérabilité considérées. De plus, des études de sensibilité permettraient d'analyser l'impact de la variation des paramètres des modèles sur l'évolution des mesures de sécurité. À plus long terme, il serait intéressant d'étudier comment on pourrait utiliser l'évaluation quantitative de la sécurité, de façon similaire aux méthodes d'évaluation de la fiabilité et la disponibilité, pour guider la conception d'un système. Enfin, il est souhaitable aussi, de définir un cadre de modélisation permettant d'évaluer la sûreté de fonctionnement des systèmes, en tenant compte à la fois des fautes accidentelles ou intentionnelles sans volonté de nuire et des malveillances.

5

5. MODÈLE DE DÉVELOPPEMENT ET CRITÈRES D'ÉVALUATION POUR LA SÛRETÉ DE FONCTIONNEMENT

Les travaux synthétisés dans ce chapitre visent à répondre à deux questions fondamentales : 1) comment prendre en compte les différentes exigences de sûreté de fonctionnement durant le développement, et 2) comment évaluer le niveau de confiance que l'on peut placer dans la capacité d'un système à satisfaire, en opération et jusqu'à son retrait du service, ses exigences de sûreté de fonctionnement ?

Dans la première partie, nous présentons un *modèle de développement à sûreté de fonctionnement explicite* qui définit un cadre permettant d'incorporer les activités relatives à la sûreté de fonctionnement à chaque étape du développement. Ce modèle est le résultat d'une collaboration avec Jean-Claude Laprie et Jean-Paul Blanquart (Matra-Marconi-Space) dans le cadre de travaux effectués au sein du $\mathcal{LIS}$ "$\mathcal{L}$aboratoire d'$\mathcal{I}$ngénierie de la $\mathcal{S}$ûreté de fonctionnement". Dans la deuxième partie, nous proposons des critères d'évaluation pour guider la certification de systèmes ayant des exigences de sûreté de fonctionnement. Ces critères ont été définis dans le cadre du projet SQUALE, en collaboration avec Yves Deswarte et les autres partenaires du projet.

5.1. Modèle de développement à sûreté de fonctionnement explicite

Les données opérationnelles de défaillance des systèmes informatiques font clairement apparaître que les fautes de conception constituent le goulot d'étranglement actuel de la sûreté de fonctionnement de ces systèmes. Les fautes de conception ayant leur origine dans le processus de développement conduit naturellement à focaliser l'attention sur les modèles de développement. L'analyse des modèles de développement de systèmes informatiques proposés dans la littérature ou dans les principales normes utilisées dans le nucléaire, l'avionique, l'espace, le transport, etc., a révélé l'absence d'une démarche d'ingénierie dans laquelle tous les moyens de la sûreté de fonctionnement sont pris en compte de façon explicite à chaque étape du développement. En effet, les modèles de développement de matériels (cf. par exemple [23]) incorporent traditionnellement des

activités de prévision des fautes, mais consacrent moins d'attention aux activités d'élimination des fautes. Les modèles classiques de développement de logiciels (en cascade [177], en spirale [19], etc.) incorporent des activités d'élimination des fautes, mais ne mentionnent ni les activités de prévision des fautes (bien que le contrôle des processus de développement de logiciels ait fait l'objet depuis quelques années d'une attention soutenue), ni les activités de tolérance aux fautes.

Pour maîtriser le développement de systèmes sûrs de fonctionnement, nous soutenons l'idée qu'il est nécessaire de définir un modèle dans lequel des activités de base relatives à la prévention, la tolérance, l'élimination et la prévision de fautes sont incorporées explicitement durant les différentes étapes du développement. Un tel modèle, que l'on qualifie de *modèle de développement à sûreté de fonctionnement explicite*, est brièvement présenté dans ce chapitre. Il est décrit de façon détaillée dans le guide de la sûreté de fonctionnement que nous avons publié chez Cépadues-Editions [132].

5.1.1. Présentation du modèle

Le modèle proposé (Figure 5.1) est basé sur une philosophie similaire à celle du modèle de développement de logiciel proposé dans la norme DO-17B [178], qui consiste à regrouper les activités nécessaires au développement au sein de processus fondamentaux et analyser leurs interactions. Le processus direct de création du système interagit avec quatre processus de sûreté de fonctionnement (prévention des fautes, tolérance aux fautes, élimination des fautes et prévision des fautes). Tous ces processus coexistent avec d'autres processus tels que, assurance qualité, certification, etc. L'axe des ordonnées de la figure 5.1 représente la proportion d'effort alloué à chacun des processus à un instant donné du développement.

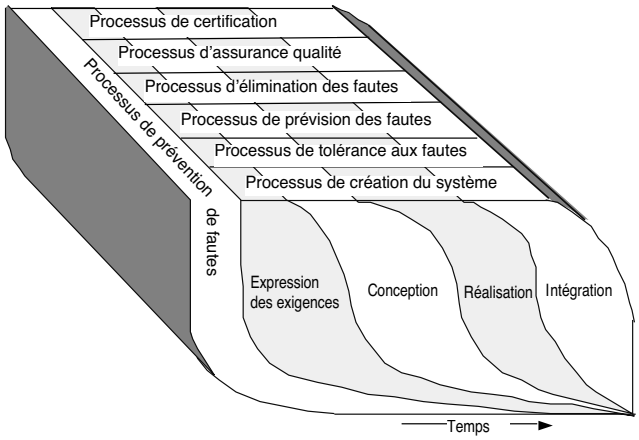


Figure 5.1 - Modèle de développement à sûreté de fonctionnement explicite

La démarche que nous avons suivie pour structurer les processus de création et les processus de sûreté de fonctionnement consiste à identifier pour chaque processus des classes d'activités de base et d'analyser ensuite les interactions entre ces activités au sein d'un même processus et avec les autres processus.

Dans le processus de création, nous distinguons quatre activités de base, correspondant aux étapes classiques du développement, c'est-à-dire, expression des exigences, conception, réalisation et intégration. L'*expression des exigences* consiste en la définition des fonctions et services que le système ou le composant considéré est amené à accomplir ainsi que des contraintes liées à l'environnement, à l'exploitation, etc. Elle recouvre les tâches correspondant à l'expression des besoins et celles correspondant à la description de ces besoins sous forme d'une spécification (formelle ou informelle). Les deux types d'activités sont parfois distingués, faisant apparaître ainsi deux phases distinctes du cycle de vie. Cette distinction se fonde en général sur des différences d'une part en termes de niveau de détail et de formalisme, et d'autre part en termes de responsabilité. Ces deux activités étant de même nature et partageant le même objectif, nous avons choisi de ne pas les distinguer. La *conception* consiste en la définition de l'architecture du système. Cette activité est en général itérée sur les composants du système jusqu'à la définition de composants élémentaires. La *réalisation* consiste à la fois en la réalisation des composants élémentaires matériels et logiciels et l'adaptation des composants éventuellement réutilisés. Enfin, l'*intégration* consiste en l'assemblage des composants et l'intégration du système dans son environnement d'utilisation.

Le processus de prévention de fautes joue un rôle particulier en ce sens qu'il définit et coordonne les activités de création du système avec les autres processus. Trois principales classes d'activités sont distinguées :

- définition des *formalismes et langages* pour les différentes activités intervenant durant le développement ; certains choix peuvent être imposés par les normes ou des exigences de certification ;
- *organisation* de projet, allocation des tâches aux équipes et gestions des ressources ;
- *planification et évaluations des risques* liés au développement.

Le processus de tolérance aux fautes comporte trois classes d'activités :

- description du *comportement en présence de fautes*, qui vise à définir les classes de fautes contre lesquelles le système devra se prémunir ;
- *partitionnement* du système en zones de confinement d'erreur et d'indépendance de fautes ;
- définition d'algorithmes et mécanismes de *traitement des erreurs et des fautes*, sans omettre de protéger ces mécanismes contre les fautes susceptibles de les affecter.

Les hypothèses de fautes établies à partir de l'étude du comportement du système en présence de fautes constituent la base pour le partitionnement du système et la définition des entrées qui seront traitées par les mécanismes de traitements d'erreur et de fautes.

Les processus d'élimination de fautes et de prévision de fautes sont composés respectivement des classes d'activités suivantes :

- *vérification*, *diagnostic* et *modifications* pour le processus d'élimination de fautes,
- énoncé des *objectifs* de sûreté de fonctionnement, *allocation* de ces objectifs aux divers composants du système, et *évaluation* de la sûreté de fonctionnement, pour le processus de prévision de fautes.

Les exigences de sûreté de fonctionnement et les besoins en termes de mécanismes de tolérance aux fautes à mettre en œuvre dans le système résultent d'une analyse globale qui doit prendre en compte tous les processus mis en jeu. Ceci se traduit par de fortes interactions entre les processus de sûreté de fonctionnement et le processus de création et également entre les processus de sûreté de fonctionnement eux-mêmes, en vertu d'une récursion naturelle, qui conduit par exemple à vérifier les résultats des évaluations, et à évaluer la progression des activités d'élimination des fautes. Une analyse plus approfondie de ces interactions est effectuée dans [132].

5.1.2. Hypothèses de fautes

La notion d'hypothèses de fautes revêt une importance capitale dans le développement de systèmes sûrs de fonctionnement. À chaque étape de décomposition du système, il est nécessaire de définir les hypothèses de fautes correspondant aux composants identifiés à ce niveau en tenant compte des hypothèses de fautes établies pour les niveaux de décomposition supérieurs [187]. Ceci conduit à une hiérarchie de modèles de fautes. L'assurance de la cohérence de ces modèles est une tâche difficile qui nécessite une analyse approfondie de la conception et de l'impact des fautes susceptibles d'affecter le comportement du système, en s'appuyant éventuellement sur le retour d'expérience de systèmes similaires. En particulier, à chaque niveau de décomposition, il est important d'étudier comment les fautes activées à ce niveau propagent des erreurs vers les niveaux supérieurs et inférieurs. L'analyse des possibilités de propagation d'erreur est capitale pour la spécification et la conception des zones de confinement d'erreur. Une telle analyse hiérarchique peut servir également à l'optimisation de la vérification des mécanismes de tolérance aux fautes [196], et aussi pour la modélisation en vue de l'évaluation de mesures de sûreté de fonctionnement (voir § 3.3).

Il est à noter que les hypothèses de fautes au titre de la prévention, de la tolérance, de l'élimination et de la prévision ne sont pas nécessairement identiques. En effet, les fautes que l'on va chercher à prévenir ou à tolérer sont par nature différentes de celles que l'on cherche à éliminer (les recouvrements sont représentatifs, dans une certaine mesure, des imperfections de chacun de ces processus). Par conséquent, pour chaque activité d'un processus de sûreté de fonctionnement, il est nécessaire de spécifier clairement les hypothèses de fautes correspondantes. L'analyse des résultats de l'activité doit être ensuite effectuée par rapport à ces hypothèses, sans omettre d'évaluer la validité de ces hypothèses par rapport à la réalité [167].

5.1.3. Directives pour le développement

Afin de guider le développement de systèmes sûrs de fonctionnement, nous avons établi un ensemble de directives sous la forme d'une liste commentée de points clefs qui identifient les principaux aspects, au titre de la prévention, tolérance, élimination et prévision de fautes à prendre en compte durant l'expression des exigences, la conception, la réalisation et l'intégration. La liste complète est décrite dans le guide de la sûreté de fonctionnement [132]. À titre d'exemple, la liste de mots clefs relatifs à l'expression des exigences est donnée sur la Figure 5.3.

Processus de Création

- Spécifications fonctionnelles
  - Définition des fonctions (services attendus en valeur et en temps)
  - Description et enchaînement des phases (systèmes phasés)
  - Répartition préliminaire des tâches entre l'homme et le système
- Description de l'environnement d'utilisation
  - Frontières du système socio-technique, caractéristiques de l'environnement et des profils d'utilisation
  - Modes d'exploitation et de maintenance
- Contraintes de développement, de validation et de maintenance
  - Contraintes physiques (poids, technologie, etc.)
  - Evolutions prévisibles, Réutilisation, Portabilité, Interopérabilité, Testabilité
  - Contraintes d'exploitation et de maintenance

Processus de Tolérance aux Fautes

- Description du comportement en présence de défaillance
  - Identification et dosage des attributs de sûreté de fonctionnement mis en jeu
  - Modes de défaillances et modes dégradés admissibles
  - Durée maximale d'interruption de service pour chaque mode
  - Nombre de défaillances simultanées et consécutives à tolérer dans chaque mode

Processus de Prévision des fautes

- Enoncé des objectifs de sûreté de fonctionnement
  - Choix des mesures à évaluer et assignation d'objectifs quantifiés
- Analyse des modes de défaillance, de leurs effets et de leur criticité
  - Identification des modes de défaillance
  - Classification des défaillances /sévérité
  - Classes de défaillance à prendre en compte
- Hypothèses pour la prévision
  - Hypothèses de modélisation et paramètres
- Allocation des objectifs de sûreté de fonctionnement par fonction
  - Classification des fonctions par niveau de criticité
- Préparation du plan de prévision des fautes
  - Sélection des méthodes et outils d'analyse et d'évaluation ordinale et probabiliste
  - Définition d'un environnement de collecte de données
- Collecte de données et analyse
  - Retour d'expérience de produits similaires, suivi du produit en cours de développement

Processus d'Elimination des Fautes

- Préparation du plan de vérification
  - Stratégies de vérification statique, stratégies de test (critères de test, génération des entrées)
  - Spécification des simulateurs d'environnement et des environnements de test
- Hypothèses de vérification
  - Fonctions, comportements et hypothèses de fautes à analyser
  - Prédicats et invariants à vérifier
- Vérification des exigences
  - Analyses comportementales, revues et inspections des spécifications
  - Agrément de la spécification : prototypage, mise en situation des opérateurs, revues par experts
- Définition de scénarios de vérification fonctionnelle

Processus de Prévention des fautes

- Formalismes et langages
  - Exigences de certification, normes et standards, formalismes et environnements de développement
- Organisation de projet
  - Modèle de cycle de vie, attribution des tâches et organisation des équipes, gestion des ressources
- Planification de projet et évaluation de risques
  - Identification des risques et des moyens de réduction des risques
  - Choix d'une stratégie de développement
  - Planifications des étapes de développement et critères de transitions entre étapes
  - Planification des revues de projet, Planification de la gestion des configurations

Figure 5.2 - Liste de points clefs relative à l'expression des exigences

5.1.4. Conclusion

Le modèle de développement à sûreté de fonctionnement explicite offre un cadre général pour structurer les activités à mettre en œuvre pour maîtriser le développement de systèmes sûrs de fonctionnement. En regroupant les activités relatives à la prévention de fautes, la tolérance aux fautes, l'élimination des fautes et la prévision des fautes au sein de processus fondamentaux interagissant avec le processus direct de création de système, on vise à s'assurer que tous les moyens de la sûreté de fonctionnement sont pris en compte à chaque étape du développement. La démarche proposée est générique et possède un large spectre d'application. Néanmoins, pour un cadre d'application bien défini, un dosage approprié des différentes activités proposées est nécessaire. Certaines activités peuvent être écartées, si par exemple, des composants sont réutilisés ou bien si les objectifs de sûreté de fonctionnement ne justifient pas la mise en œuvre de telles activités. La réutilisation de composants et le développement de composants en vue de leur réutilisation constituent deux objectifs au centre des préoccupations actuelles des constructeurs de systèmes. Les méthodes de développement orientées objets s'avèrent bien adaptées pour répondre à ce besoin. Dans ce contexte, la démarche que nous proposons reste applicable. En effet, nous avons mis l'accent sur la nature des activités qu'il est nécessaire de mener, abstraction faite de la façon selon laquelle ces activités sont agencées entre elles. Les directives et les listes de points clefs proposées pour guider le développement de systèmes sûrs de fonctionnement peuvent ainsi s'appliquer indépendamment de la méthode de développement utilisée. La figure 5.3 donne un exemple qui montre en particulier que, des groupes d'activités peuvent être instanciés plusieurs fois en fonction des approches considérées pour le développement du système et de ses composants (développement en cascade, prototypage, réutilisation, etc.). C'est le modèle du cycle de vie choisi qui doit déterminer l'ordre selon lequel ces activités sont orchestrées.

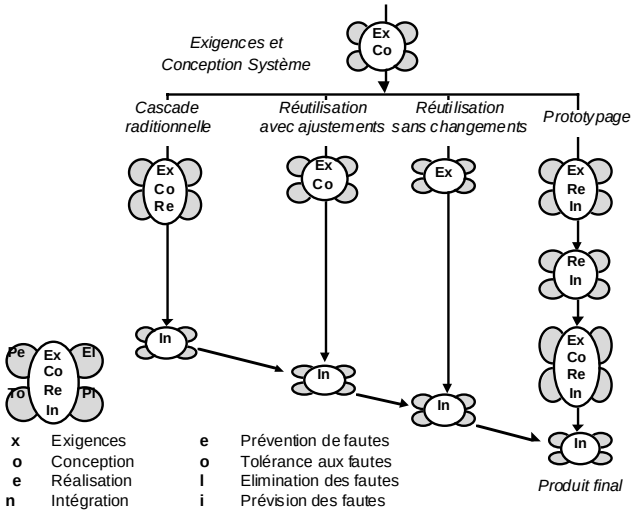


Figure 5.3 - Exemple d'application du modèle

5.2. Critères d'évaluation pour la sûreté de fonctionnement

L'utilisation de systèmes informatiques dans des applications ayant des exigences de sûreté de fonctionnement pose le problème de l'évaluation du niveau de confiance que l'on peut placer dans ces systèmes vis-à-vis de leur aptitude à satisfaire ces exigences, durant la vie opérationnelle et jusqu'à leur démantèlement. On distingue deux classes de standards ou documents normatifs décrivant des critères de certification de systèmes ayant des exigences de sûreté de fonctionnement : ceux, ciblant la sécurité-confidentialité et ceux, focalisant sur la sécurité-innocuité. La première classe a donné naissance aux critères d'évaluation TCSEC, ITSEC, etc. (cf. § 4.1). Ces critères ne sont pas spécifiques d'un secteur d'application donné, mais restent néanmoins focalisés sur les besoins de sécurité des systèmes d'information. En ce qui concerne la deuxième classe, chaque secteur industriel a développé ses propres normes, par exemple, l'avionique [179, 181], le ferroviaire [29-31], le nucléaire [27], la défense [153], etc. Ces normes présentent plusieurs points communs, ce qui a suscité le développement de la norme CEI 1508 [28] qui propose une démarche d'évaluation générique, applicable à divers secteurs industriels. De façon générale, la démarche adoptée dans l'ensemble de ces documents normatifs est de définir une échelle d'évaluation à plusieurs niveaux, en fonction de la criticité du système ou de la confiance que l'on peut placer dans ce système. Les critères à satisfaire pour chacun de ces niveaux varient en termes de degré de formalisation, de rigueur, de détail, …, des processus de développement et de validation, et des preuves apportées en support pour la certification [55, 56].

Cependant, ces documents et les critères correspondants sont focalisés, soit sur la sécurité-innocuité, soit sur la sécurité-confidentialité. Or le plus souvent, il est nécessaire de prendre en compte d'autres attributs comme la disponibilité ou la maintenabilité. Par exemple, dans les transports ferroviaires ou aériens, la sécurité des passagers est primordiale, mais la disponibilité est aussi critique pour garantir la sécurité et également pour des raisons économiques. Il est donc important de définir des critères d'évaluation qui permettent de prendre en compte l'ensemble des attributs de la sûreté de fonctionnement, tout en considérant que leur importance relative peut ne pas être uniforme. Dans le cadre du projet SQUALE, nous avons développé des critères d'évaluation qui visent à atteindre un tel objectif [45, 55, 56]. Ces critères sont génériques dans le sens où ils ne visent pas un secteur d'application donné. De plus, ils sont suffisamment généraux et intègrent certains concepts et activités définis dans les principales normes pour la sécurité-innocuité et la sécurité-confidentialité pour ne pas exiger un travail supplémentaire important en vue d'évaluer et certifier des systèmes qui satisfont les exigences des normes de leur domaine. Dans la suite, nous donnons un bref aperçu de ces critères.

5.2.1. Démarche d'évaluation SQUALE

5.2.1.1. Rôles et schéma d'évaluation

Dans une évaluation utilisant les critères de SQUALE, on distingue trois différents intervenants (Figure 5.4) :

- *Le commanditaire (ou sponsor)* qui demande et finance l'évaluation (il peut s'agir du fournisseur d'un système, du client, d'un organisme de certification, etc.)
- *Le développeur* qui est responsable de la réalisation du système.
- *L'évaluateur* qui est l'organisme responsable de l'évaluation ; il a pour mission de vérifier que les exigences des critères sont bien satisfaites.

Différents schémas d'évaluation peuvent être considérés en fonction du contexte. Par exemple, l'évaluateur peut interagir uniquement avec le sponsor, ou être en contact direct avec le développeur. C'est au sponsor ou au développeur de fournir les informations sur le système soumis à l'évaluation, ses objectifs de sûreté de fonctionnement, une description des activités relatives à la sûreté de fonctionnement, et les justifications et éléments de preuve qui montrent que le système satisfait ses objectifs conformément aux critères d'évaluation SQUALE. Ces informations sont rassemblées dans un document appelé "cible de sûreté de fonctionnement" (similaire à la cible de sécurité définie dans les ITSEC) qui sert de guide de référence pour l'évaluation. Ce document est enrichi au fur et à mesure de la progression du développement. En plus,de ce document, d'autres "délivrables" peuvent être requis pour l'évaluation (par exemple, des documents de conception, le système implémenté, etc.). En principe, le rôle de l'évaluateur est de vérifier, sur la base des données fournies, que les exigences des critères sont bien satisfaites. Néanmoins, il peut réaliser des activités de vérification et de validation complémentaires, directement sur le produit, ou bien indirectement, par exemple en soumettant des jeux de test au développeur qui sera en charge de les mettre en œuvre et de fournir les résultats.

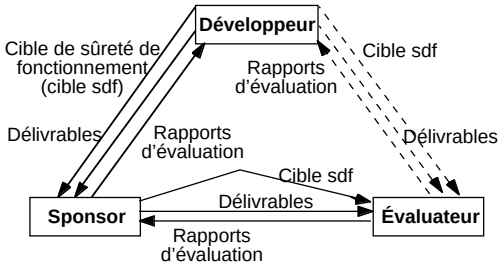


Figure 5.4 – Schéma d'évaluation

5.2.1.2. Cible de sûreté de fonctionnement

Les informations suivantes doivent figurer dans la cible de sûreté de fonctionnement :

1) *description du système et de son environnement* : description des interfaces du système (avec d'autres systèmes, l'environnement physique, les utilisateurs ou opérateurs humains, l'organisation, etc.) ;
2) résultats des *analyses préliminaires de risques* qui identifient ce que le système doit préserver et contre quoi il doit être protégé, une liste de menaces et d'événements indésirables pour le système ou son environnement, et une estimation de leur sévérité;

3) spécification des *objectifs pour chaque attribut de sûreté de fonctionnement* sous la forme d'un profil de sûreté de fonctionnement qui indique l'importance de chaque attribut ;
4) la *politique de sûreté de fonctionnement* qui fixe les principes, règles et procédures à mettre en œuvre pour satisfaire les objectifs (par exemple, politique de sécurité multi-niveaux, "fail-safe", etc.)
5) la spécification des *fonctions de sûreté de fonctionnement* à mettre en œuvre pour satisfaire les objectifs (par exemple, contrôle d'accès, détection d'erreur par duplication et comparaison, etc.)
6) l'allocation de la sûreté de fonctionnement qui définit le rôle de chaque sous-système (matériel, logiciel, humain, etc.) en justifiant ces choix ;
7) le plan des activités et des méthodes à employer pour obtenir une confiance justifiée dans la capacité du système à satisfaire ses objectifs.

Ce document est à définir au début du développement. Il doit être ensuite enrichi, au fur et à mesure de la progression du développement (de l'expression des exigences de haut niveau jusqu'à la réalisation et l'intégration du système, en passant par les différentes étapes de la conception). En particulier, nous avons décrit dans [45] les principales activités à mettre en œuvre, pour la sûreté de fonctionnement, durant le développement, qui permettent d'obtenir les informations nécessaires pour établir la cible d'évaluation.

5.2.1.3. Processus d'assurance

Pour obtenir une confiance justifiée dans la capacité d'un système à satisfaire ses objectifs, il est nécessaire de mettre en œuvre un ensemble d'activités que l'on peut structurer selon quatre processus (appelés *processus d'assurance*) : 1) validation des exigences de sûreté de fonctionnement, 2) vérification de conformité, 3) validation du système implémenté, et 4) gestion de la qualité des processus du cycle de vie.

La *validation des exigences de sûreté de fonctionnement* vise d'une part à s'assurer que les objectifs répondent aux besoins et d'autre part que la politique, les fonctions et les allocations de sûreté de fonctionnement sont pertinentes pour satisfaire les objectifs. Ceci inclut l'analyse de l'efficacité des mécanismes utilisés pour contrer les fautes et risques potentiels, la validité et la représentativité des hypothèses de fautes, l'analyse des vulnérabilités résiduelles du système, etc.

La *vérification de conformité* consiste à s'assurer que les différentes étapes de construction du système sont réalisées correctement, c'est-à-dire, que chaque niveau de décomposition du système satisfait les exigences du niveau supérieur. Ces vérifications doivent être effectuées à chaque étape du cycle de développement (de la définition des exigences de haut niveau jusqu'à la réalisation des composants et l'intégration du système global). Les objectifs de la vérification de conformité sont atteints par la combinaison d'un ensemble d'activités d'inspections, de revues, de tests, par l'emploi de méthodes formelles et d'analyses comportementales.

La *validation du système implémenté* vise à s'assurer que le système tel qu'il est implémenté est conforme à ses objectifs. En particulier, il s'agit de s'assurer que le

système ne possède pas d'effets de bord non spécifiés qui, sans contredire les spécifications du système, lui donnent un comportement qui pourrait avoir des conséquences inacceptables. Ainsi, les activités de validation essayent de s'assurer que les spécifications d'un système ne sont pas incomplètes par rapport à l'ensemble des objectifs et que les hypothèses effectuées pendant le développement ne sont pas erronées. Les activités de validation recommandées comprennent des évaluations expérimentales, des analyses de pénétration et des analyses de canaux cachés

La *gestion de la qualité des processus du cycle de vie* vise à s'assurer que les méthodes, outils et processus utilisés pendant le développement, l'exploitation opérationnelle, la maintenance et éventuellement le démantèlement du système, sont adéquats pour atteindre les objectifs de sûreté de fonctionnement, et que les contrôles prévus dans le plan d'assurance qualité ont été réalisés. Plusieurs aspects doivent être analysés : planification et organisation de projet, définition d'un modèle de cycle de vie, contrôle des modifications, gestion des configurations, documentation, compétence et expérience des équipes, etc. Ces analyses sont effectuées par le biais de procédures d'audit, des revues de projet, des inspections de documents, etc.

5.2.1.4. Niveaux de confiance et critères d'évaluation

Pour un système, les exigences peuvent amener à définir des objectifs portant sur tout ou partie des attributs de la sûreté de fonctionnement et l'importance relative de chacun d'eux peut ne pas être uniforme. À chaque attribut, on associe un niveau de confiance (en d'autres termes, un niveau d'exigence[3]) variant entre (1 à 4) : 1 représente le niveau de confiance le plus bas, et 4 le niveau le plus élevé. Un niveau 0 est aussi défini pour indiquer l'absence d'exigence pour l'attribut considéré. Par exemple, un système pourrait avoir la combinaison de niveaux de confiance : A1, C0, R3, I3, S3, M2, appelée "profil de sûreté de fonctionnement". Ce profil signifie que les niveaux de confiance associés aux attributs disponibilité (A), confidentialité (C), fiabilité (R), Intégrité (I), sécurité-innocuité (S) et maintenabilité (M) sont 1, 0, 3, 3, 3 et 2, respectivement.

Les niveaux de confiance sont spécifiés à partir des résultats des analyses de risques. Au fur et à mesure du raffinement de la spécification et de la conception du système, il est aussi important de faire le lien entre les attributs et les classes de fautes à prendre en compte (malveillances, fautes accidentelles ou intentionnelles sans volonté de nuire). Les niveaux de confiance définissent, d'une part les objectifs de sûreté de fonctionnement à satisfaire par le système, et d'autre part, servent à déterminer comment les activités d'assurance et d'évaluation doivent être employées. En effet, pour chaque activité d'assurance, les critères définissent des niveaux de rigueur (RL), détail (DL) et indépendance (IL) plus ou moins élevés en fonction des niveaux de confiance spécifiés.

---

[3] Nous avons choisi le terme "niveau de confiance" au lieu de "niveau d'exigence" car il peut être utilisé, à la fois pour spécifier le niveau de confiance *attendu* (tel qu'il est défini par le sponsor ou le développeur) et celui *attribué* après l'évaluation (celui-ci pouvant être inférieur à celui attendu). La seconde raison est que la traduction anglaise du terme niveau d'exigence ("Requirement level") ne nous semble pas correspondre à la signification que nous avons voulu lui donner.

Le *niveau de rigueur* détermine la manière avec laquelle les activités doivent être réalisées, le degré de justification qui doit être fourni (par exemple, l'adéquation d'une méthode ou d'un outil, l'efficacité d'une méthode de test…), ou encore le degré de formalisation des méthodes d'analyse ou de vérification utilisées. Le *niveau de détail* détermine les phases du cycle de vie au cours desquelles les activités doivent être réalisées et les exigences sur le contenu des fournitures de sortie de phase délivrées par le développeur aux évaluateurs (par exemple, au niveau 1, l'activité peut être appliquée aux spécifications de haut niveau uniquement, alors qu'au niveau 3 toutes les étapes de raffinement des spécifications jusqu'à l'implémentation doivent être examinées). Enfin, le *niveau d'indépendance* fixe le lien organisationnel et la séparation des responsabilités entre les personnes réalisant l'activité et le développeur. De façon similaire à la norme CEI 1508, nous avons défini trois niveaux d'indépendance : personne indépendante, service indépendant (au sein de la même entreprise), ou bien organisme indépendant (par exemple un laboratoire d'évaluation extérieur à l'entreprise). Les niveaux RL, DL, IL peuvent varier de 1 à 3, où 1 est le niveau le plus faible et 3 le plus élevé.

La figure 5.5 résume, de façon synthétique, les processus et activités d'assurance définis dans les critères SQUALE, les spécifications des niveaux RL, DL et IL en fonction des niveaux de confiance, et les recommandations sur l'utilisation des différentes activités d'assurance.

On distingue trois types de recommandations :

- \- Il n'existe pas de recommandation pour ou contre l'utilisation de l'activité.
- R l'activité est recommandée et elle doit être employée avec les niveaux de rigueur, détail et indépendance, correspondant au niveau de confiance visé. Cependant, il est possible d'appliquer l'activité en utilisant des niveaux RL, DL ou IL plus faibles si on utilise une autre activité du même processus d'assurance avec des niveaux plus élevés. Par exemple, l'utilisation des méthodes et preuves formelles peut compenser le fait qu'on ne fasse pas de tests unitaires, mais ne peut, en aucun cas, remplacer complètement le test.
- HR L'activité est hautement recommandée. Dans le cas où l'activité est appliquée avec des niveaux RL, DL ou IL inférieurs à ceux qui sont requis, le développeur doit apporter des preuves convaincantes pour justifier son choix.

On peut noter que pour certaines activités, par exemple l'assurance qualité, un seul niveau de rigueur, détail, et indépendance est spécifié pour tous les niveaux de confiance.

Remarque. Le concept de niveau de confiance est utilisé dans plusieurs normes et standards (sous différents noms, par exemple niveau d'intégrité [28], niveau d'assurance [25, 181], etc.), et chaque domaine d'application a ses propres règles pour spécifier ces niveaux en fonction du niveau d'acceptabilité des risques ou de la sévérité des défaillances. Les critères SQUALE ont pour objectifs d'être génériques et suffisamment généraux pour être applicables à différents domaines et secteurs d'application. Pour cette raison, nous n'avons imposé aucune règle pour définir ces niveaux de confiance. Notons que pour les normes CEI 1508, DO178B et EN 50128, quatre niveaux sont également définis. Pour les autres normes, telles que ITSEC ou CEI 880, il sera nécessaire de définir une méthode permettant de faire le lien entre les niveaux SQUALE et les niveaux

définis dans ces normes. Une analyse des principales similitudes et différences entre les critères SQUALE et ces normes est présentée dans [45].

Processus/Activités d'assurance	**Niveaux de confiance**			
	1	**2**	**3**	**4**
Validation des exigences				
Analyse préliminaire des risques (HAZOPS, AMDEC, Arbres d'événements, …)	HR un niveau	HR un niveau	HR un niveau	HR un niveau
Évaluation probabiliste	R RL1, DL1, IL1	HR RL2, DL1, IL1	HR RL3, DL2, IL2	HR RL3, DL3, IL3
Analyse de modes/causes communs	R RL1, DL1, IL1	HR RL2, DL1, IL2	HR RL2, DL2, IL2	HR RL2, DL2, IL3
Analyses des risques (AMDEC, arbres de fautes, …)	HR RL1, DL1, IL1	HR RL2, DL1, IL2	HR RL3, DL2, IL2	HR RL3, DL2, IL3
Vérification de conformité				
Analyses statiques (inspections, revues, lectures croisées)	HR RL1, DL1, IL1	HR RL2, DL2, IL1	HR RL2, DL3, IL2	HR RL3, DL3, IL3
Analyses comportementales (Graphes d'états, Petri, tables de décision, …)	R RL1, DL1, IL1	R RL2, DL1, IL1	HR RL3, DL2, IL2	HR RL3, DL3, IL3
Méthodes & preuves formelles	- -	R RL1, DL1, IL1	R RL2, DL1, IL2	HR RL3, DL2, IL3
Test (fonctionnel, structurel, robustesse, …)	HR RL1, DL1, IL1	HR RL1, DL2, IL1	HR RL2, DL3, IL2	HR RL3, DL3, IL3
Analyses de traceabilité	HR RL1, DL1, IL1	HR RL1, DL1, IL2	HR RL2, DL2, IL3	HR RL2, DL2, IL3
Validation système implémenté				
Analyses de pénétration	R RL1, DL1, IL1	HR RL2, DL1, IL2	HR RL3, DL2, IL2	HR RL3, DL3, IL3
Analyse des canaux cachés	- -	HR RL1, DL1, IL1	HR RL2, DL2, IL2	HR RL3, DL3, IL3
Évaluation expérimentale	HR RL1, DL1, IL1	HR RL2, DL1, IL1	HR RL3, DL2, IL2	HR RL3, DL2, IL3
Gestion qualité des processus				
Assurance qualité (audits, revues, inspections, …)	HR un niveau	HR un niveau	HR un niveau	HR un niveau

Figure 5.5 – Exemple d'allocation RL, DL, IL en fonction des niveaux de confiance

5.2.1.5. Décomposition du système et stratégie d'évaluation

Un système est généralement décomposé de façon hiérarchique en plusieurs composants conduisant ainsi à une hiérarchie de profils de sûreté de fonctionnement (un pour chaque composant). L'allocation des exigences de sûreté de fonctionnement aux composants doit faire l'objet d'une analyse approfondie en s'appuyant, en particulier, sur les résultats des analyses de risques et des analyses de modes communs de défaillance. Le

raffinement des profils de sûreté de fonctionnement est utile pour guider la conception, en particulier, dans l'objectif de séparer les fonctions et mécanismes destinés à satisfaire les différentes exigences de sûreté de fonctionnement. Par exemple, on pourrait envisager de distinguer les composants dédiés à la confidentialité, de ceux contribuant à l'accomplissement des objectifs de sécurité-innocuité. Par ailleurs, il est important de noter qu'au fur et à mesure du raffinement du profil de sûreté de fonctionnement, l'importance relative accordée à chaque attribut peut varier, en fonction du rôle de chaque composant dans l'accomplissement des objectifs de sûreté de fonctionnement du système et de la politique de sûreté de fonctionnement adoptée. Ainsi, une exigence de sécurité-innocuité à un niveau de décomposition donné peut se traduire par des exigences de disponibilité ou de fiabilité au niveau inférieur.

La figure 5.6 donne un exemple où l'allocation des exigences de sûreté de fonctionnement définies pour le système (S4, C3,…) conduit à la définition de deux profils de sûreté de fonctionnement : (S4, C0, …) pour le composant U et (S2, C3, …) pour le composant V. Ainsi, les fonctions de confidentialité sont mises en œuvre exclusivement par le composant V, alors que les deux composants contribuent, de façon non uniforme, aux fonctions de sécurité-innocuité. Le composant V est ensuite décomposé en deux composants V1 et V2 tels que V1 ne contribue pas à la confidentialité. Pour chacun de ces composants, les activités d'assurance concernant la validation des exigences de sûreté de fonctionnement, la vérification de conformité et la validation du système implémenté doivent être appliquées avec les niveaux de rigueur, détail et indépendance correspondant au profil associé. Les activités correspondant à la gestion de la qualité des processus doivent être appliquées de façon uniforme indépendamment des profils. Par exemple, le test du composant V2 doit être effectué conformément aux exigences du niveau 3 pour la confidentialité et du niveau 1 pour les fonctions de sécurité innocuité. Cependant, le développeur peut également choisir, pour des raisons d'efficacité, d'appliquer les tests au niveau 3 pour les deux cas.

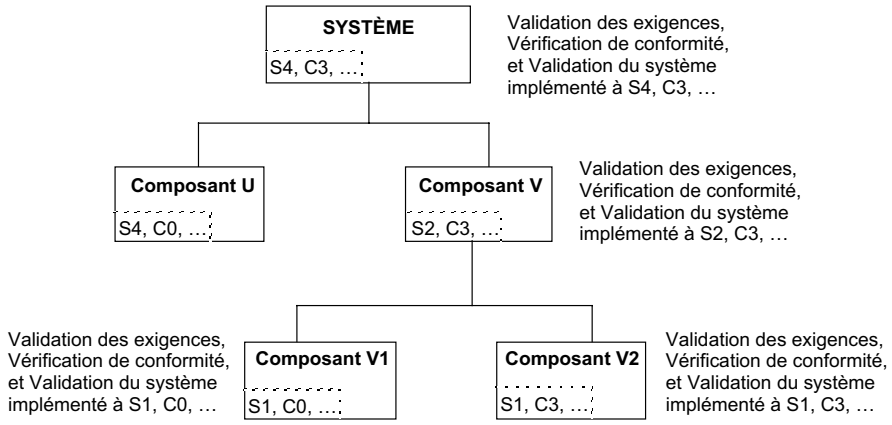


Figure 5.6– Raffinement du profil de sûreté de fonctionnement et évaluation : exemple

À chaque étape de décomposition, il est nécessaire d'étudier la validité des allocations de sûreté de fonctionnement. En particulier, dans l'exemple de la figure 5-6, il faut vérifier

que le composant U ne peut, en aucun cas, avoir des comportements qui peuvent affecter le composant V, vis-à-vis de son aptitude à satisfaire les exigences de confidentialité.

5.2.2. Expérimentation des critères sur METEOR

Afin de valider la première version des critères [46] et d'étudier leurs limites, nous les avons expérimentés sur le système de commande de la nouvelle ligne de métro automatique à Paris, METEOR. Cette ligne est pilotée par un nouveau Système d'Automatisation de l'Exploitation des Trains, le SAET développé par Matra Transport International. Ce système à automatisme intégral autorise la circulation de trains sans conducteur ni accompagnateur. Il est conçu pour répondre à de fortes contraintes permettant d'assurer une très grande qualité de service, une grande souplesse d'exploitation et une facilité d'adaptation à l'environnement (par exemple, trains non équipés transitant parmi des trains en automatisme intégral).

Pour éprouver les critères SQUALE, nous avons choisi comme cible d'évaluation le sous-ensemble du SAET permettant le contrôle et la supervision du système. Il permet la commande et la transmission des ordres opérateur aux trains, les opérations d'exploitation telles que, l'ajout ou le retrait de trains, le changement de programmes d'exploitation et la remontée d'informations du système vers les opérateurs ainsi que les alarmes. Ce système possède de fortes exigences de sûreté de fonctionnement. Les équipements sont redondés pour assurer une haute disponibilité du système. La sécurité-innocuité du pilote automatique est assurée par l'utilisation de la technique du processeur codé déjà éprouvée dans le cadre du projet SACEM [89]. La sécurisation des commandes sécuritaires s'appuie sur le principe de sécurité contrôlée avec une validation de la commande par l'opérateur à partir d'une platine.

L'expérimentation des critères a débuté quand le système METEOR était dans une phase de tests sur site. Par conséquent, nous nous sommes basés sur les documents préparés par les développeurs pour la certification du système conformément aux normes du domaine ferroviaire. Nous avons analysé si les exigences définies dans les critères sont satisfaites. L'expérience a montré que, de façon générale, ils sont applicables et compatibles avec les pratiques industrielles. Cependant, nous avons identifié certaines améliorations afin de les rendre plus souples et de lever des ambiguïtés et des redondances qui existaient dans la version préliminaire des critères que nous avons utilisée dans cette étude [46]. En particulier, nous avons constaté qu'il était nécessaire d'ajuster les critères pour permettre au développeur d'utiliser des activités et techniques d'assurance équivalentes à celles que nous avons préconisées. Ce processus d'ajustement devrait également permettre au développeur de négocier avec l'évaluateur le contenu des fournitures en utilisant au maximum celles qui sont définies dans le cycle de vie du système. L'expérience a également montré la nécessité de préciser clairement certaines définitions, soit pour éviter des interprétations divergentes, soit pour préciser des concepts et notions qui ont été étendus ou restreints par rapport aux définitions couramment admises dans le milieu industriel. Enfin, nous avons constaté qu'il était nécessaire de décrire de façon plus précise et plus flexible les responsabilités des différents intervenants dans l'évaluation, en définissant les rôles de chacun tout en

laissant la possibilité au développeur et à l'évaluateur d'ajuster le schéma proposé. Par exemple, la réalisation d'une activité peut être effectuée soit par un laboratoire spécialisé, soit par le développeur, seule l'analyse des résultats de l'activité restant de la responsabilité de l'évaluateur. Ces améliorations ont été prises en compte et intégrées dans une nouvelle version des critères [45].

5.2.3. Conclusion

Les critères SQUALE ont pour objectif de définir une démarche harmonisée et générique pour certifier des systèmes vis-à-vis de leur aptitude à satisfaire les exigences de sûreté de fonctionnement. Ces critères permettent de tenir compte de l'ensemble des attributs de la sûreté de fonctionnement et de leur importance relative. Dans leur forme actuelle, les critères sont indépendants du domaine d'application. Cependant, certains ajustements restent nécessaires, en particulier, pour les rendre plus facilement applicables dans les différents secteurs industriels (aéronautique, ferroviaire, nucléaire, etc.). Au stade actuel de nos travaux, nous avons analysé les similitudes et différences entre les critères SQUALE et les principales normes utilisées pour l'évaluation de la sécurité-confidentialité, ou la sécurité-innocuité (cf. [45]). La prochaine étape consistera à définir des directives pour montrer comment utiliser les critères en respectant les exigences des normes sectorielles.

L'expérimentation des critères sur le système METEOR a été effectuée alors que le système était déjà développé. Une seconde expérimentation est en cours chez Bouygues Telecom où les critères sont utilisés de façon concourante avec le développement d'un réseau de télécommunication mobile. Cette étude sera l'occasion de tester la dernière version des critères, qui intègre les modifications et améliorations suscitées par l'étude effectuée sur METEOR.

5.3. Analyse critique et défis

La mise en œuvre du modèle de développement à sûreté de fonctionnement explicite et des critères d'évaluation SQUALE engendre un coût élevé, particulièrement pour des systèmes hautement critiques. La question fondamentale est comment réduire les coûts sans dégrader la sûreté de fonctionnement. De plus en plus, on s'oriente vers l'utilisation de composants du commerce (COTS pour "Commercial off-the-shelf Components") dans des systèmes sûrs de fonctionnement. Cependant, s'il est vrai que les COTS offrent plus de flexibilité pour le concepteur et peuvent raccourcir le cycle de développement, ils soulèvent des problèmes sérieux du point de vue de la sûreté de fonctionnement et de la certification [123, 141, 190, 191]. En effet, ces composants n'ayant pas été développés dans l'optique de satisfaire des exigences de sûreté de fonctionnement, ils peuvent avoir des comportements ayant des conséquences inacceptables sur le service fourni par le système dans lequel ils sont utilisés. Pour être en mesure de développer une architecture tolérante aux fautes basée sur des COTS, il est important d'identifier le comportement en présence de fautes de ces composants. Par

conséquent, un effort de test et de validation considérable est nécessaire pour justifier l'utilisation des COTS dans des applications critiques[4]. Théoriquement, cet effort pourrait être réduit si on dispose de suffisamment de données d'expérience provenant de l'utilisation opérationnelle du COTS dans des contextes similaires. Cependant, de telles données ne sont pas disponibles en général, en particulier, à cause de la réticence des fournisseurs et développeurs de les fournir, de l'évolution rapide des technologies (surtout dans le cas du logiciel), etc. Un autre problème lié au COTS concerne la certification. En effet, les normes de certification de systèmes critiques vis-à-vis de la sécurité-innocuité par exemple, imposent l'utilisation d'une approche rigoureuse pour le développement des différents composants du système considéré et la fourniture d'un ensemble de preuves et de justifications détaillées sur l'architecture de ces composants et la façon selon laquelle ils ont été développés et validés. Or de telles informations ne sont pas disponibles pour un COTS. De plus, ces composants ne sont pas généralement développés conformément aux approches préconisées dans les normes.

Par conséquent, une voie de recherche intéressante pour l'avenir concerne le développement de méthodes permettant de faciliter la mise en œuvre et la certification de systèmes basés sur des COTS, en garantissant un niveau de confiance suffisant dans leur aptitude à satisfaire les exigences de sûreté de fonctionnement. En plus des COTS, d'autres défis restent à relever, en particulier la maîtrise de l'évolution et de l'ouverture des systèmes, la coexistence de composants de criticités différentes, etc.

---

[4] En l'absence d'informations suffisantes sur les modes de défaillance des COTS, il est nécessaire de faire des hypothèses faibles sur leur comportement en présence de fautes, ce qui conduit à une architecture complexe dont le coût de développement et de validation peut être très élevé.

6

6. BILAN ET PROSPECTIVE

Nous avons présenté dans ce document les grandes lignes des travaux que nous avons effectués au LAAS depuis 1988 sur : 1) la définition et l'application de méthodes pour faciliter l'analyse et l'évaluation de la sûreté de fonctionnement, 2) la définition d'un modèle de développement à sûreté de fonctionnement explicite, et 3) la définition de critères d'évaluation pour la certification de systèmes sûrs de fonctionnement.

Pour l'ensemble de ces travaux, notre démarche a été guidée par la volonté d'associer modélisation conceptuelle et application à des cas d'étude afin de valider nos résultats théoriques et d'étudier comment les mettre en œuvre en pratique. Dans cette optique, certains de nos travaux ont été mis en œuvre dans de nouveaux outils ou prototypes (par exemple, SoRel pour l'analyse et l'évaluation de la fiabilité du logiciel, et ESOPE, pour l'évaluation quantitative de la sécurité-confidentialité), et d'autres ont contribué à l'amélioration d'outils existants, par exemple l'outil SURF-2.

Durant la période couverte par ce document, nos principales applications ont été les suivantes :

1) le système informatique de contrôle en route du trafic aérien français (le CAUTRA) pour ce qui concerne nos travaux sur la construction de modèles complexes, basés sur les GSPN, pour l'évaluation de la sûreté de fonctionnement ;
2) un système commercial de stockage de données basé sur une architecture RAID (développé par StorageTek aux Etats-Unis) pour illustrer notre approche hiérarchique de simulation comportementale en présence de fautes ;
3) la famille de logiciels d'autocommutateurs téléphoniques TROPICO (développés par la compagnie brésilienne TELEBRAS) pour illustrer certains de nos travaux sur la croissance de fiabilité ;
4) un parc de systèmes informatiques Unix du réseau du LAAS pour expérimenter notre approche d'évaluation quantitative de la sécurité-confidentialité ;
5) le système de commande du métro automatique METEOR (développé par Matra Transport International), pour éprouver les critères d'évaluation de la sûreté de fonctionnement que nous avons définis dans le cadre du projet SQUALE.

6.1. Bilan

Afin de faire un bilan de nos principales contributions, nous reprenons chacun des thèmes développés dans les différents chapitres.

Modèles complexes pour la sûreté de fonctionnement

Notre objectif a été de définir des méthodes permettant de maîtriser la construction de modèles complexes pour l'analyse et l'évaluation de la sûreté de fonctionnement. Nous avons étudié deux types d'approches basées sur les réseaux de Petri stochastiques généralisés (GSPN) et sur la simulation comportementale en présence de fautes, respectivement. Nos principales contributions dans ce domaine sont les suivantes.

Concernant la modélisation par GSPN :

- Nous avons développé une méthode de *construction modulaire et incrémentale* de modèles GSPN complexes. Le modèle est établi en plusieurs étapes, chacune correspondant à la prise en compte des hypothèses de défaillance et de restauration d'un nouveau composant du système en supposant que ceux qui n'ont pas été encore considérés sont dans un état de fonctionnement nominal. Le modèle est mis à jour au fur et à mesure de l'intégration de nouveaux composants. Il est structuré sous la forme de modules (associés aux composants) qui sont interconnectés par des mécanismes de couplage qui permettent de décrire leurs interactions.
- Pour faciliter la mise en œuvre de la méthode, nous l'avons complétée par la définition d'un *formalisme de spécification* qui offre, d'une part, une *notation* permettant d'avoir une description structurée et de haut niveau des modules et de leurs interactions, et d'autre part, des *règles de transformation* de la spécification permettant d'obtenir de manière directe le modèle GSPN correspondant.
- L'application de la méthode incrémentale et du formalisme de spécification pour la modélisation de différentes architectures possibles du CAUTRA et l'évaluation de leur comportement en considérant différents niveaux de dégradation du service, nous a permis de confirmer le bien fondé et l'efficacité de notre démarche.

Concernant la simulation comportementale en présence de fautes :

- Nous avons développé une approche de *simulation hiérarchique* qui permet d'analyser le comportement du système en présence de fautes en considérant différents niveaux d'abstraction. Un modèle de simulation est associé à chaque niveau et les résultats de la simulation d'un modèle donné sont utilisés comme paramètres dans le modèle décrivant le comportement à un niveau d'abstraction plus élevé. Ces paramètres permettent en particulier de propager les effets des fautes d'un modèle à un autre.
- Nous avons illustré cette approche sur un système commercial de stockage de données basé sur une architecture RAID et utilisant un cache de large capacité. Cette étude nous a permis, en utilisant une trace d'exécution réelle, d'évaluer la couverture des mécanismes de détection d'erreurs, mis en œuvre dans le cache et les disques et d'analyser la distribution de la latence d'erreurs.

Pour les deux applications que nous avons étudiées, les analyses et modélisations de la sûreté de fonctionnement ont été effectuées à un niveau de détail relativement fin et la complexité des modèles a été en partie liée au besoin de prendre en compte plusieurs types d'interactions et de dépendances stochastiques. À notre connaissance, il n'existe pas dans la littérature d'exemples où les GSPN ou bien la simulation ont été utilisés pour faire des analyses de sûreté de fonctionnement de complexité comparable.

Croissance de fiabilité

Traditionnellement, les travaux dans ce domaine ont été focalisés sur le logiciel en adoptant une approche boîte noire et les mesures évaluées concernent uniquement la fiabilité. Nos principales contributions sont les suivantes :

- Nous avons contribué au développement d'une approche originale de modélisation permettant d'évaluer la fiabilité et la disponibilité de systèmes multi-composant —matériel et logiciel— en tenant compte de la croissance de fiabilité des composants. Cette approche permet en particulier de bénéficier de l'ensemble des résultats obtenus dans le domaine de la construction et du traitement de modèles markoviens en fiabilité stabilisée.
- Nous avons défini un modèle de croissance de fiabilité en temps discret et nous avons montré que ce type de modélisation offre des possibilités intéressantes, en particulier, pour prendre en compte explicitement le profil d'utilisation dans l'évaluation des mesures de sûreté de fonctionnement.
- Nous avons contribué à la définition d'une méthode d'analyse et d'évaluation de la fiabilité du logiciel et un outil (SoRel) qui permettent, notamment, de guider l'application des modèles de croissance de fiabilité dans un contexte industriel. Nous avons expérimenté cette méthode sur plusieurs cas réels, en particulier, sur la famille de logiciels TROPICO développés par la compagnie brésilienne, TELEBRAS.

Évaluation quantitative de la sécurité-confidentialité

Les méthodes classiques d'évaluation de la sécurité-confidentialité sont basées sur des critères qualitatifs qui prennent en compte la façon selon laquelle le système est mis en œuvre, mais ne sont pas adéquats pour suivre l'évolution de la sécurité opérationnelle. Pour pallier ces insuffisances :

- Nous avons contribué à la définition d'une approche d'évaluation probabiliste qui est basée sur une représentation des vulnérabilités opérationnelles du système sous forme d'un graphe des privilèges. La transformation de ce graphe en une chaîne de Markov permet d'évaluer des mesures caractérisant la capacité d'un système à résister à des attaquants potentiels. Ces mesures donnent aux administrateurs des systèmes des informations pertinentes pour surveiller l'évolution de la sécurité opérationnelle.
- Nous avons contribué au développement d'un outil d'évaluation (ESOPE) et à l'application de notre approche à un parc de systèmes constitué de plusieurs centaines de machines Unix connectées à travers un réseau local et utilisant un même système de fichiers partagé en moyenne par 700 utilisateurs. L'observation, durant une période de 21 mois, de la variation de nos mesures en fonction de l'évolution des

vulnérabilités dans le système, nous a permis de valider notre approche et de montrer la pertinence de nos mesures pour le suivi de la sécurité opérationnelle.

Modèle de développement et critères d'évaluation pour la sûreté de fonctionnement

Un des grand mérites du concept de sûreté de fonctionnement est son aspect d'intégration des techniques destinées à conférer à un système l'aptitude à délivrer un service dans lequel on puisse avoir confiance, et à s'assurer que cette confiance est justifiée. Or, les démarches traditionnelles pour l'ingénierie ou la certification des systèmes informatiques ne prennent pas en compte l'ensemble des moyens ou des attributs de la sûreté de fonctionnement. Pour pallier ces insuffisances :

- Nous avons contribué à la définition d'un modèle de développement qui décrit les principales activités à mettre en œuvre au titre de prévention de fautes, tolérance aux fautes, élimination des fautes, et prévision des fautes, en interaction avec les activités de développement proprement dites (expression des exigences, conception, réalisation et intégration).
- Nous avons contribué à l'élaboration d'une démarche de certification et de critères d'évaluation qui généralisent les démarches d'évaluation proposées par exemple dans les ITSEC pour la sécurité-confidentialité, ou dans les normes de l'avionique, nucléaire, ferroviaire, etc. qui sont focalisées sur la sécurité-innocuité, afin de couvrir l'ensemble des attributs de la sûreté de fonctionnement en tenant compte de leur importance relative pour le système considéré.

6.2. Prospective

L'évolution continue des systèmes informatiques et de leurs domaines d'application fait que les intérêts et enjeux de la sûreté de fonctionnement ne sont pas prêts de s'éteindre. En dépit des avancées réalisées dans ce domaine, plusieurs défis restent à relever durant les prochaines décennies. Dans la conclusion de chacun des chapitres de ce document, nous avons déjà identifié un certain nombre d'extensions possibles de nos travaux. Dans ce qui suit, nous rappelons ces extensions et nous évoquons deux thèmes de recherche supplémentaires qui nous semblent importants et vers lesquels nous souhaitons orienter notre effort. Il s'agit d'une part de l'évaluation expérimentale de systèmes hétérogènes interconnectés et d'autre part de l'analyse et de l'évaluation du comportement en présence de fautes d'applications réparties orientées objets.

Modèles complexes pour la sûreté de fonctionnement

Récemment, un intérêt croissant a été consacré à l'étude des processus algébriques qui présentent l'avantage de pouvoir être utilisés à la fois pour faire des spécifications et vérifications formelles du comportement d'applications réparties et pour générer des modèles stochastiques permettant d'obtenir des évaluations quantitatives. Cependant, leur champ d'application est pour l'instant orienté vers l'évaluation de la performance. Il nous semble intéressant d'explorer l'applicabilité de ces travaux pour faire des évaluations de sûreté de fonctionnement et surtout d'étudier leurs limites quand il s'agit

de traiter des cas concrets d'une complexité équivalente à celle du CAUTRA par exemple. La spécification de haut niveau que nous avons développée pour décrire le comportement du CAUTRA pourra constituer un point de départ pour ce type d'étude.

En plus de la maîtrise de la construction des modèles, il est aussi important d'apporter des solutions efficaces au problème du traitement des modèles. Plusieurs avancées ont été réalisées dans ce domaine, en particulier, en ce qui concerne le calcul de bornes pour l'estimation de valeurs approchées de mesures de sûreté de fonctionnement ou de performabilité à partir de la génération partielle de l'espace d'état [24, 87, 146, 156, 186]. Ces travaux nous semblent être une voie intéressante à approfondir. Jusqu'à maintenant, l'accent a été mis essentiellement sur l'évaluation de mesures de sûreté de fonctionnement en régime asymptotique, en considérant en particulier la disponibilité. L'extension de ces travaux pour évaluer des mesures transitoires permettra de couvrir un spectre d'application plus large. Ces travaux pourront servir à la fois à l'optimisation du traitement de modèles analytiques générés par exemple à partir des GSPN et également de modèles de simulation comportementale en présence de fautes.

Croissance de fiabilité

Les résultats obtenus jusqu'à présent ne permettent pas de fournir des estimations de la fiabilité du logiciel très tôt dans le cycle de développement afin de réagir sur la conception et le processus de développement avant le début de l'implémentation. Pour obtenir des prévisions plus tôt dans le cycle de vie, il nous semble nécessaire d'incorporer dans les modèles, des informations issues du processus de développement ou bien décrivant le produit durant les phases de spécification et de conception, ou éventuellement des données collectées sur des produits antérieurs. Les études permettant d'établir des corrélations entre le comportement dynamique du logiciel tel qu'il est observé par les utilisateurs, les mesures de complexité relatives au logiciel en tant que produit (dérivées à partir du code par exemple), et les mesures décrivant le processus de développement sont à un stade embryonnaire. Bien que de telles corrélations aient été observées a posteriori sur plusieurs logiciels, il n'existe pas encore de théorie permettant d'obtenir un modèle prévisionnel qui décrit le lien entre la fiabilité du logiciel et ces mesures statiques. L'utilisation de données collectées sur une famille de produits similaires et incorporant des informations sur le processus de développement constitue une voie prometteuse qui a été proposée dans [102, 130]. Cette voie mérite d'être approfondie en s'appuyant en particulier sur l'analyse de données issues de cas réels.

Évaluation quantitative de la sécurité-confidentialité

Plusieurs directions peuvent être envisagées pour poursuivre nos travaux dans ce domaine. Tout d'abord, il est nécessaire de définir des méthodes permettant de fournir une estimation réaliste des variables d'effort intervenant dans le modèle d'évaluation pour les différentes classes de vulnérabilité considérées. De plus, des études de sensibilité permettraient d'analyser l'impact de la variation des paramètres des modèles sur l'évolution des mesures de sécurité. À plus long terme, il serait intéressant d'étudier comment on pourrait utiliser l'évaluation quantitative de la sécurité, de façon similaire aux méthodes d'évaluation de la fiabilité et la disponibilité, pour guider la conception

d'un système. Enfin, il est souhaitable aussi, de définir un cadre de modélisation permettant d'évaluer la sûreté de fonctionnement des systèmes, en tenant compte à la fois des fautes accidentelles ou intentionnelles sans volonté de nuire et des malveillances.

Modèle de développement et critères d'évaluation pour la sûreté de fonctionnement

La mise en œuvre du modèle de développement à sûreté de fonctionnement explicite et des critères d'évaluation SQUALE engendre un coût élevé, particulièrement pour des systèmes hautement critiques. La question fondamentale est comment réduire les coûts sans dégrader la sûreté de fonctionnement. De plus en plus, on s'oriente vers l'utilisation de composants du commerce (COTS pour "Commercial off-the-shelf Components") dans des systèmes sûrs de fonctionnement. Cependant, s'il est vrai que les COTS offrent plus de flexibilité pour le concepteur et peuvent raccourcir le cycle de développement, ils soulèvent des problèmes sérieux du point de vue de la sûreté de fonctionnement et de la certification (cf. § 5.3). Par conséquent, une voie de recherche intéressante pour l'avenir concerne le développement de méthodes permettant de faciliter la mise en œuvre et la certification de systèmes basés sur des COTS, en garantissant un niveau de confiance suffisant dans leur aptitude à satisfaire les exigences de sûreté de fonctionnement. En plus des COTS, d'autres défis restent à relever, en particulier la maîtrise de l'évolution et de l'ouverture des systèmes.

Évaluation expérimentale de systèmes hétérogènes interconnectés

L'interconnexion à large échelle de systèmes informatiques hétérogènes favorise le partage de ressources et le travail coopératif entre un grand nombre d'utilisateurs. Cependant, ces avantages peuvent être compromis en cas de dégradation de la sûreté de fonctionnement du réseau de communication ou des systèmes interconnectés. Dans ce contexte, il est nécessaire de disposer de moyens permettant la surveillance opérationnelle du système global, l'identification des éléments défaillants du réseau global, et l'analyse et l'évaluation de la sûreté de fonctionnement du système. La surveillance opérationnelle d'un parc important de machines hétérogènes interconnectées n'est pas une tâche aisée, à cause du nombre important de systèmes à observer et de l'hétérogénéité des systèmes d'exploitation, des applications partagées, des protocoles de communication, etc. L'objectif est de définir une méthode pour la gestion et l'administration de la sûreté de fonctionnement de systèmes matériels et logiciels hétérogènes interconnectés via des réseaux de communication offrant différentes qualités de service et de développer un environnement expérimental pour la mise en œuvre de cette méthode. Il s'agit plus précisément de définir un environnement de collecte de données permettant d'enregistrer automatiquement des informations sur l'état des différents éléments du réseau et des applications mises en œuvre. Ces données serviront à l'identification des circonstances indésirables susceptibles d'affecter la sûreté de fonctionnement et à la mise en œuvre de stratégies de reconfiguration. Ces données serviront également à l'évaluation de la sûreté de fonctionnement opérationnelle du réseau, l'analyse de l'occurrence et de la propagation d'erreurs en tenant compte de la charge du réseau, etc. Il est clair que des choix devront être faits, d'une part, vis-à-vis des objets du système à observer, du type de données à collecter et de la granularité de ces

données, et d'autre part, vis-à-vis des outils d'observation à mettre en œuvre. De plus, l'environnement à mettre en place devra engendrer un minimum de perturbation au niveau du fonctionnement du réseau. À cause du volume important de données qui sont collectées à partir de cet environnement, il est nécessaire de développer des techniques et des outils permettant d'automatiser le plus possible le traitement de ces données. Plusieurs aspects doivent être pris en compte, en particulier : 1) le filtrage et la classification des données qui nécessitent la définition de critères facilitant l'identification précise de la source de chaque événement et l'analyse de la causalité entre les événements, et 2) le traitement statistique des données en vue d'évaluer des mesures représentatives de la sûreté de fonctionnement du système.

Analyse et évaluation du comportement en présence de fautes d'applications distribuées orientées objets

La technologie objet est de plus en plus utilisée pour le développement d'applications distribuées tolérantes aux fautes. Cette technologie offre plusieurs avantages pour faciliter la conception des applications, mais pose en même temps plusieurs problèmes vis-à-vis de la validation. Du point de vue de l'évaluation de la sûreté de fonctionnement, il est important de développer des moyens permettant d'analyser, dès la phase de conception, le comportement en présence de fautes de ces architectures et d'évaluer dans quelle mesure les objectifs de sûreté de fonctionnement peuvent être atteints. Pour être en mesure de tenir compte des spécificités des langages objet (encapsulation, polymorphisme, héritage, invocation dynamique d'objet, etc.), et de représenter les interactions entre les objets distribués, leur comportement en présence de fautes et l'impact de ces fautes sur les propriétés attendues du système, la modélisation doit être faite à un niveau relativement fin. Plusieurs questions peuvent être soulevées : 1) Quel est le niveau de détail le plus adéquat et quel est le formalisme le plus adapté pour la modélisation (modélisation analytique, simulation, etc.) ? 2) Comment maîtriser la complexité des modèles au niveau de la construction et du traitement ? Quel modèle de fautes considérer ? Pour répondre à ces questions, il est nécessaire de se baser sur des études de cas concrets. Les travaux en cours au sein de notre groupe de recherche sur la mise en œuvre d'un environnement de développement orienté objet pour des applications tolérantes aux fautes pourront servir de point de départ pour nos études.

7. RÉFÉRENCES

[1] S. Akers, “Binary Decision Diagrams,” *IEEE Transactions on Computers*, vol. C-27, pp. 509-516, 1978.

[2] H. H. Ammar, S. M. R. Islam, “Time Scale Decomposition of a Class of Generalized Stochastic Petri Net Models,” *IEEE Transactions on Software Engineering*, vol. 15, pp. 809-820, 1989.

[3] V. Amoia, G. De Micheli, M. Santomauro, “Computer-Oriented Formulation of Transition-Rate Matrices via Kronecker Algebra,” *IEEE Transactions on Reliability*, vol. R-30, pp. 123-132, 1981.

[4] D. Anderson, T. F. Lunt, H. Javitz, A. Tamaru, A. Valdes, “Detecting Unusual Program Behavior Using the Statistical Component of the Next-generation Intrusion Detection Expert System (NIDES),” SRI SRI-CSL-95-06, 1995.

[5] J. Arlat, “Validation de la sûreté de fonctionnement par injection de fautes : méthode—mise en œuvre— application,” Thèse de Doctorat d'Etat, Institut National Polytechnique de Toulouse, N° 163, Rapport LAAS N° 90399, 1990.

[6] J. Arlat, “Fault Injection for the Experimental Validation of Fault-Tolerant Systems,” *Workshop Fault-Tolerant Systems*, (Kyoto, Japon), pp. 33-40, IEICE, 1992.

[7] J. Arlat, M. Aguera, Y. Crouzet, J. Fabre, E. Martins, D. Powell, “Experimental Evaluation of the Fault Tolerance of an Atomic Multicast Protocol,” *IEEE Transactions on Reliability*, vol. 39, pp. 455-467, 1990.

[8] J. Arlat, A. Costes, Y. Crouzet, J.-C. Laprie, D. Powell, “Fault Injection and Dependability Evaluation of Fault-Tolerant Systems,” *IEEE Transactions on Computers*, vol. 42, pp. 913-923, 1993.

[9] J. Arlat, K. Kanoun, J. C. Laprie, “Dependability Evaluation of Software Fault Tolerance,” *18th IEEE Int Symp. on Fault-Tolerant Computing (FTCS-18)*, (Tokyo, Japon), pp. 142-147, IEEE Computer Society Press, 1988.

[10] R. L. Aveyard, F. T. Man, “A Study on the Reliability of the Circuit Maintenance System 1-B,” *Bell System Technical Journal*, vol. 59, pp. 1317-1332, 1980.

[11] A. Avizienis, Y. He, "The Taxonomy of Design Faults in COTS Microprocessors," *7th Working Conference on Dependable Computing for Critical Applications (DCCA-7)*, (San Jose, CA, USA), pp. 1-17, IEEE Computer Society Press, 1999.

[12] M. Balakrishnan, K. S. Trivedi, "Componentwise Decomposition for an Efficient Reliability Computation of Systems with Repairable Components," *25th Int. Symposium on Fault-Tolerant Computing (FTCS-25)*, (Pasadena, CA, USA), pp. 259-268, IEEE Computer Society Press, 1995.

[13] G. Balbo, "Stochastic Petri Nets: Accomplishments and Open Problems," *IEEE International Computer Performance and Dependability Symposium (IPDS'95)*, (Erlangen, Allemagne), pp. 51-60, IEEE Computer Society Press, 1995.

[14] G. Balbo, S. Bruell, S. Ghanta, "Combining Queueing Networks and Generalized Stochastic Petri Nets for the Solution of Complex Models of System Behavior," *IEEE Transactions on Computers*, vol. 37, pp. 1251-1267, 1988.

[15] C. Béounes, M. Aguéra, J. Arlat, C. Bourdeau, J.-E. Doucet, K. Kanoun, J.-C. Laprie, S. Metge, J. Moreira de Souza, D. Powell, P. Spiesser, "SURF-2: A Program for Dependability Evaluation of Complex Hardware and Software Systems," *23rd IEEE Int. Symposium on Fault-Tolerant Computing (FTCS-23)*, (Toulouse, France), pp. 668-673, IEEE Computer Society Press, 1993.

[16] M. Bernardo, L. Donatiello, R. Gorrieri, "Integrating Performance and Functional Analysis of Concurrent Systems with EMPA," Dept. of Computer Science, Technical Report UBLCS-95-14, 1995.

[17] A. Blakemore, S. K. Tripathi, "Automated Time Scale Decomposition and Analysis of Stochastic Petri Nets," *5th Int. Workshop on Petri Nets and Performance Models (PNPM'93)*, (Toulouse, France), pp. 248-257, IEEE Computer Society Press, 1993.

[18] A. Bobbio, K. Trivedi, "An Aggregation Technique for the Transient Analysis of Stiff Markov Chains," *IEEE Transactions on Computers*, vol. C-35, pp. 803-814, 1986.

[19] B. W. Boehm, "A Spiral Model of Software Development and Enhancement," *IEEE Computer*, vol. 21, pp. 61-72, 1988.

[20] M. Borrel, "Interactions entre composants matériel et logiciel de systèmes tolérants aux fautes - caractérisation - formalisation - modélisation ; Application à la sûreté de fonctionnement du CAUTRA," , Institut National Polytechnique de Toulouse, N° 1123, Rapport LAAS N° 96001, 1996.

[21] J. Boué, P. Pétillon, Y. Crouzet, J. Arlat, "Early Experimental Verification of Fault Tolerance: The VHDL-base Fault Injection Tool MEFISTO-L," *28th IEEE Int. Symposium on Fault-Tolerant Computing (FTCS-28)*, (Munich, Allemagne), pp. 168-173, IEEE Computer Society Press, 1998.

[22] R. Bryant, "Symbolic Boolean Manipulation with Ordered Binary Decision Diagrams," *ACM Computing Surveys*, vol. 24, pp. 293-318, 1992.

[23] BSI, “Reliability of Constructed or Manufactured Products, Systems, Equipment and Components, Part 1. Guide to Reliability and Maintainability Programme Management,” British Standard Institution, Rapport BS 5760, 1985.

[24] J. A. Carrasco, J. Escriba, A. Calderon, “Efficient Exploration of Availability Models Guided by Failure Distances,” *Performance Evaluation Review*, vol. 24, pp. 242-251, 1996.

[25] CCE, “Critères d'évaluation de la sécurité des systèmes informatiques,” Commission des Communautés Européennes, 1991.

[26] CCIB, “Common Criteria for Information Technology Security Evaluation - Version 2.0,” *Part 1: Introduction and General Model (CCIB-98-026); Part 2: Security Functional Requirements (CCIB-98-027); Part 3: Security Assurance Requirements (CCIB-98-028)*: Common Criteria Implementation Board, 1998.

[27] CEI, “Software for Computers in the Safety Systems of Nuclear Power Stations,” : Commission Electronique Internationale (CEI), 1986.

[28] CEI, “Sûreté fonctionnelle : systèmes relatifs à la sûreté,” in *Partie 1 : Prescriptions générales (CEI 1508-1) ; Partie 2 : Prescriptions pour les systèmes électroniques programmables (CEI 1508-2) ; Partie 3 : Prescriptions concernant les logiciels (CEI 1508-3) ; Partie 4 : Définitions et abréviations (CEI 1508-4) ; Partie 5 : Lignes directrices pour la mise en œuvre de la Partie 1 (CEI 1508-5) ; Partie 6 : Lignes directrices pour l'application des Parties 2 et 3 (CEI 1508-6) ; Partie 7 : Bibliographie des techniques et des mesures (CEI 1508-7)*: Commission Électronique Internationale (CEI), 1995.

[29] CENELEC, “Applications aux chemins de fer : Logiciels pour systèmes de commande et de protection ferroviaire,” Comité Européen de Normalisation Electrotechnique (CENELEC) EN 50128, 1997.

[30] CENELEC, “Applications aux chemins de fer : Systèmes électroniques de sécurité pour la signalisation,” Comité Européen de Normalisation Electrotechnique (CENELEC) ENV 50129, 1997.

[31] CENELEC, “Applications ferroviaires : Spécification et démonstration de la fiabilité, de la maintenabilité et de la sécurité (FDMS),” Comité Européen de Normalisation Electrotechnique (CENELEC), 1997.

[32] L. Chen, A. Avizienis, “N-Version Programming: a Fault Tolerance Approach to Reliability of Software Systems,” *8th IEEE Int. Fault Tolerance Computing Symposium (FTCS-8)*, (Toulouse, France), pp. 3-9, IEEE Computer Society Press, 1978.

[33] P. M. Chen, E. K. Lee, G. A. Gibson, R. H. Katz, D. A. Patterson, “RAID: High-Performance, Reliable Secondary Storage,” *ACM Computing Surveys*, vol. 26, pp. 145-185, 1994.

[34] R. C. Cheung, “A User-Oriented Software Reliability Model,” *IEEE Trans. on Software engineering*, vol. SE-6, pp. 118-125, 1980.

[35] R. Chillarege, R. K. Iyer, “Measurement-Based Analysis of Error Latency,” *IEEE Transactions on Computers*, vol. C-36, pp. 529-537, 1987.

[36] R. Chillarege, R. K. Iyer, “An Experimental Study of Memory Fault Latency,” *IEEE Transactions on Computers*, vol. 38, pp. 869-874, 1989.

[37] G. Chiola, “A Software Package for the Analysis of Generalized Stochastic Petri Nets,” *International Workshop on Timed Petri Nets*, (Torino, Italie), pp. 136-143, IEEE Computer Society Press, 1985.

[38] G. Chiola, C. Dutheillet, G. Franceschinis, S. Haddad, “Stochastic Well-Formed Coloured Nets for Symmetric Modelling Applications,” *IEEE Transactions on Computers*, vol. 42, pp. 1343-1360, 1993.

[39] G. S. Choi, R. K. Iyer, “FOCUS: An Experimental Environment for Fault Sensitivity Analysis,” *IEEE Transactions on Computers*, vol. 41, pp. 1515-1526, 1992.

[40] G. Ciardo, R. Marie, B. Sericola, K. S. Trivedi, “Performability Analysis Using Semi-Markov Reward Processes,” *IEEE Transactions on Computers*, vol. 39, pp. 1251-1264, 1990.

[41] G. Ciardo, J. Muppala, K. Trivedi, “SPNP: Stochastic Petri Net Package,” *International Workshop on Petri Nets and Performance Models (PNPM89)*, (Kyoto, Japon), pp. 142-151, IEEE Computer Society Press, 1989.

[42] G. Ciardo, K. S. Trivedi, “A Decomposition Approach for Stochastic Reward Net Models,” *Performance Evaluation*, pp. 37-59, 1993.

[43] J. A. Clark, D. K. Pradhan, “Reliability Analysis of Unidirectional Voting TMR Systems Through Simulated Fault-Injection,” *IEEE Workshop on Fault Tolerant Parallel and Distributed Systems*, (Amherst, MA, USA), pp. 72-81, IEEE Computer Society Press, 1992.

[44] J. A. Clark, D. K. Pradhan, “Fault Injection — A Method for Validating Computer-System Dependability,” *Computer*, vol. 28, pp. 47-56, 1995.

[45] P. Corneillie, Y. Deswarte, J. Goodson, A. Hawes, M. Kaâniche, H. Kurth, G. Liebisch, T. Manning, S. Moreau, A. Steinacker, C. Valentin, “SQUALE Dependability Assessment Criteria (3rd Draft),” Rapport LAAS N° 98456, 1998.

[46] P. Corneillie, Y. Deswarte, A. Hawes, M. Kaâniche, H. Kurth, T. Manning, S. Moreau, A. Steinacker, “SQUALE—Definition of draft Criteria for the Assessment of Dependable Systems,” Rapport LAAS N° 97166, 1997.

[47] A. Costes, C. Landrault, J.-C. Laprie, “Reliability and Availability Models for Maintained Systems Featuring Hardware Failures and Design Faults,” *IEEE Transactions on Computers*, vol. C-27, pp. 548-560, 1978.

[48] D. R. Cox, H. D. Miller, *The Theory of Stochastic Processes*. London: Chapman and Hall, 1968.

[49] M. Dacier, “Vers une évaluation quantitative de la sécurité informatique,” Thèse de Doctorat, Institut National Polytechnique de Toulouse, N° 971, Rapport LAAS N° 94488, 1994.

[50] M. Dacier, Y. Deswarte, “The Privilege Graph: an Extension to the Typed Access Matrix Model,” *European Symposium in Computer Security (ESORICS'94)*, (Brighton, Royaume-Uni), pp. 319-334, Springer-Verlag, 1994.

[51] M. Dacier, Y. Deswarte, M. Kaâniche, "Models and Tools for Quantitative Assessment of Operational Security," *12th International Information Security Conference (IFIP/SEC'96)*, (Samos, Grèce), pp. 177-186, Chapman & Hall, 1996.

[52] M. Dacier, Y. Deswarte, M. Kaâniche, "Quantitative Assessment of Operational Security: Models and Tools," , Rapport LAAS N° 96493, 1996.

[53] M. Dacier, M. Kaâniche, Y. Deswarte, "A Framework for Security Assessment of Insecure Systems," Rapport LAAS N° 92434, 1993.

[54] O. Daniel, "Les réseaux de Petri stochastiques pour l'évaluation des attributs de la sûreté de fonctionnement de systèmes manufacturiers," Institut National Polytechnique de Grenoble, 1995.

[55] Y. Deswarte, M. Kaâniche, P. Corneillie, P. Benoit, "Dependable Computing System Evaluation Criteria: A Proposal," , Rapport LAAS N° 98499, 1998.

[56] Y. Deswarte, M. Kaâniche, P. Corneillie, P. Benoit, "SQUALE: Critères d'évaluation de la sûreté de fonctionnement," *11ème Colloque de Fiabilité & Maintenabilité*, (Arcachon, France), pp. 367-376, , 1998.

[57] DoD, "Trusted Computer System Evaluation Criteria (TCSEC)," Department of Defense, USA DoD- 5200.28-STD, 1985.

[58] S. Donatelli, "Superposed Generalized Stochastic Petri Nets: Definition and Efficient Solution," *15th Int. Conference on Applications and Theory of Petri Nets*, (Zaragoza, Espagne), pp. 258-277, Springer-Verlag, 1994.

[59] S. Donatelli, M. Ribaudo, J. Hillston, "A Comparison of Performance Evaluation Process Algebra and Generalized Stochastic Petri Nets," *6th Int. Workshop on Petri Nets and Performance Models (PNPM'95)*, (Durham, NC, USA), pp. 158-168, IEEE Computer Society Press, 1995.

[60] T. Downs, "An Approach to the Modeling of Software Testing with Statistical Quality Control," *IEEE Transactions on Software Engineering*, vol. SE-11, pp. 375-386, 1985.

[61] J. T. Duane, "Learning Curve Approach to Reliability Monitoring," *IEEE Transactions on Aerospace*, vol. 2, pp. 563-566, 1964.

[62] J. B. Dugan, M. R. Lyu, "System-Level Reliability and Sensitivity Analyses for Three Fault-Tolerant System Architectures," *4th Int. Working Conference on Dependable Computing for Critical Applications (DCCA-4)*, (San Diego, CA, USA), pp. 295-307, 1994.

[63] J. B. Dugan, K. S. Trivedi, "Coverage Modeling for Dependability Analysis of Fault-Tolerant Systems," *IEEE Transactions on Computers*, vol. 38, pp. 775-787, 1989.

[64] W. K. Ehrlich, J. P. Stampfel, J. R. Wui, "Application of Software Reliability Modeling to Product Quality and Test Process," *Int. Conf. on Software Engineering*, (Nice, France), pp. 108-116, , 1990.

[65] J. M. Finkelstein, "Starting and Limiting Values for Reliability Growth," *IEEE Transactions on Reliability*, vol. R-28, pp. 111-113, 1979.

[66] J. M. Finkelstein, “A Logarithmic Reliability Growth Model for Single-Mission Systems,” *IEEE Trans. on Reliability*, vol. R-32, pp. 508-511, 1983.

[67] G. Florin, S. Natkin, “Les réseaux de Petri Stochastiques,” *Technique et Science Informatiques*, vol. 4, pp. 143-160, 1985.

[68] N. Fota, “Spécification et construction incrémentale de modèles de sûreté de fonctionnement—Application au CAUTRA,” Thèse de Doctorat, Institut National Polytechnique de Toulouse, N° 1293, Rapport LAAS N° 97151, 1997.

[69] N. Fota, M. Kaâniche, K. Kanoun, “A Modular and Incremental Approach for Building Complex Stochastic Petri Net Models,” *1st Int. Conference on Mathematical Methods in Reliability (MMR'97)*, (Bucharest, Roumanie), pp. 151-158, 1997.

[70] N. Fota, M. Kaâniche, K. Kanoun, “Dependability Evaluation of an Air Traffic Control Computing System,” *3rd IEEE International Computer Performance & Dependability Symposium (IPDS-98)*, (Durham, NC, USA), pp. 206-215, IEEE Computer Society Press, 1998.

[71] N. Fota, M. Kaâniche, K. Kanoun, A. Peytavin, “Description du système informatique global du contrôle en route français du trafic aérien,” , Rapport de contrat Convention ADERMIP-LAAS/CENA 94/C0010 (Rapport LAAS N° 94472), 1994.

[72] N. Fota, M. Kaâniche, K. Kanoun, A. Peytavin, “Analyse du système informatique global de l'ATC en route en vue de l'évaluation de sa sûreté de fonctionnement,” Rapport de contrat Convention ADERMIP-LAAS/CENA 94/C0010 (Rapport LAAS N° 95280), 1995.

[73] N. Fota, M. Kaâniche, K. Kanoun, A. Peytavin, “Safety Analysis and Evaluation of an Air Traffic Control Computing System,” *15th Int. Conf. on Computer Safety, Reliability and Security (SAFECOMP'96)*, (Vienna, Autriche), pp. 219-229, Springer-Verlag, 1996.

[74] N. Fota, M. Kaâniche, K. Kanoun, A. Peytavin, “Modélisation et évaluation globale de la sûreté de fonctionnement du CAUTRA,” , Rapport LAAS N° 97172, 1997.

[75] R. M. Fujimoto, “Parallel Discrete Event Simulation,” *Communications of the ACM*, vol. 33, pp. 30-53, 1990.

[76] S. Garfinkel, E. Spafford, *Practical Unix & Internet Security*, 2nd Edition: O'Reilly & Associates (Inc.), 1996.

[77] O. Gaudoin, “Outils statistiques pour l'évaluation de la fiabilité des logiciels,” Thèse de Doctorat, Université Joseph Fourier, Grenoble 1, 1990.

[78] R. German, C. Lindemann, “Analysis of Stochastic Petri Nets by the Method of Supplementary Variables,” *Performance Evaluation*, vol. 20, pp. 317-335, 1994.

[79] R. German, D. Logothetis, K. S. Trivedi, “Transient Analysis of Markov Regenerative Stochastic Petri Nets: A comparison of Approaches,” *6th Int. Workshop on Petri Nets and Performance Models (PNPM'95)*, (Durham, NC, USA), pp. 103-112, IEEE Computer Society Press, 1995.

[80] K. K. Goswami, "Design for Dependability: A simulation-Based Approach," PhD., University of Illinois at Urbana-Champaign, N° UILU-ENG-94-2204, CRHC-94-03, 1994.

[81] K. K. Goswami, R. K. Iyer, L. Young, "DEPEND: A Simulation-Based Environment for System Level Dependability Analysis," *IEEE Transactions on Computers*, vol. 46, pp. 60-74, 1997.

[82] A. Goyal, W. C. Carter, E. de Souza e Silva, S. S. Lavenberg, "The System Availability Estimator," *16th IEEE Int. Symp. on Fault-Tolerant Computing (FTCS-16)*, (Vienna, Austria), pp. 84-89, IEEE Computer Society Press, 1986.

[83] A. Goyal, P. Shabuddin, P. Heidelberger, V. F. Nicola, P. W. Glynn, "A Unified Framework for Simulating Markovian Models of Highly Dependable Systems," *IEEE Transactions on Computers*, vol. 41, pp. 36-51, 1992.

[84] R. B. Grady, *Practical Software Metrics for Project Management and Process Improvement*. Englewood Cliffs, NJ: Prentice Hall, 1992.

[85] A. Grnarov, J. Arlat, A. Avizienis, "On the Performance of Software Fault Tolerance Strategies," *10th IEEE Int. Symp. Fault-Tolerant Computing (FTCS-10)*, (Kyoto, Japon), pp. 251-253, IEEE Computer Society Press, 1980.

[86] D. Gross, D. R. Miller, "The Randomization Technique as a Modeling Tool and Solution Procedure for Transient Markov Processes," *Operation Research*, vol. 32, pp. 343-361, 1984.

[87] B. R. Haverkort, "Approximate Performability and Dependability Analysis using Generalized Stochastic Petri Nets," *Performance Evaluation*, pp. 61-78, 1993.

[88] H. Hecht, "Fault-Tolerant Software," *IEEE Transactions on Reliability*, vol. R-28, pp. 227-232, 1979.

[89] C. Hennebert, G. Guiho, "SACEM: A Fault-Tolerant System for Train Speed Control," *23rd IEEE Int. Symposium on Fault-Tolerant Computing (FTCS-23)*, (Toulouse, France), pp. 624-628, IEEE Computer Society Press, 1993.

[90] M.-C. Hsueh, T. K. Tsai, R. K. Iyer, "Fault-Injection Techniques and Tools," *IEEE Computer*, pp. 75-82, 1997.

[91] ITSEM, *Information Security Evaluation Manual*: Office for Official Publications of the European Communities, Luxembourg, 1993.

[92] R. K. Iyer, D. J. Rossetti, "Effect of System Workload on Operating System Reliability: A Study on IBM 3081," *IEEE Transactions on Software engineering*, vol. SE-11, pp. 1438-1448, 1985.

[93] E. Jenn, "Sur la validation des systèmes tolérant les fautes : injection de fautes dans des modèles de simulation VHDL," Thèse de Doctorat, Institut National Polytechnique de Toulouse, N° 895, Rapport LAAS N° 94361, 1994.

[94] E. Jenn, J. Arlat, M. Rimén, J. Ohlsson, J. Karlsson, "Fault Injection into VHDL Models: The MEFISTO Tool," *24th IEEE Int. Symp. on Fault-Tolerant Computing (FTCS-24)*, (Austin, TX, USA), pp. 66-75, IEEE Computer Society Press, 1994.

[95] M. Kaâniche, “Modèle hyperexponentiel en temps continu et en temps discret pour l'évaluation de la croissance de la sûreté de fonctionnement,” Thèse de Doctorat, Institut National Polytechnique de Toulouse, N° 519, Rapport LAAS N° 92002, 1992.

[96] M. Kaâniche, K. Kanoun, “The Discrete-Time Hyperexponential Model for Software Reliability Growth Evaluation,” *3rd IEEE Int. Symp. on Software Reliability Engineering (ISSRE'92)*, (Raleigh, NC, USA), pp. 64-75, IEEE Computer Society Press, 1992.

[97] M. Kaâniche, K. Kanoun, “Software Failure Data Analysis of two Successive Generations of a Switching System,” *12th Int. Conf. on Computer Safety, Reliability and Security (SAFECOMP'93)*, (Poznan-Kiekrz, Poland), pp. 230-239, Springer-Verlag, 1993.

[98] M. Kaâniche, K. Kanoun, “Reliability of a Commercial Telecommunications System,” *7th IEEE Int. Symp. on Software Reliability Engineering (ISSRE'96)*, (White Plains, NY, USA), pp. 207-212, IEEE Computer Society Press, 1996.

[99] M. Kaâniche, K. Kanoun, M. Cukier, M. Bastos Martini, “Software Reliability Analysis of Three Successive Generations of a Switching System,” *First European Conference on Dependable Computing (EDCC-1)*, (Berlin, Allemagne), pp. 473-490, Springer-Verlag, 1994.

[100] M. Kaâniche, K. Kanoun, J.-C. Laprie, “Discrete-Time Reliability Growth Modeling of Single and Multicomponent Software Systems,” , Rapport LAAS N° 92046, 1992.

[101] M. Kaâniche, K. Kanoun, J.-C. Laprie, “Reliability Growth of Software Systems: From Discrete to Continuous-Time Modeling,” , Rapport LAAS N° 92480, 1992.

[102] M. Kaâniche, K. Kanoun, J.-C. Laprie, “Fiabilité du logiciel: spécification et estimation,” , Rapport LAAS 94380, 1994.

[103] M. Kaâniche, K. Kanoun, S. Metge, “Analyse et évaluation de la croissance de fiabilité du logiciel d'un équipement de télécommunications,” Rapport de contrat Convention CNET n° 89-1B 007909245 (Rapport LAAS N° 89328), 1989.

[104] M. Kaâniche, K. Kanoun, S. Metge, “Proposition d'une procédure de collecte de données pour l'évaluation de la sûreté de fonctionnement d'un système informatique,” Rapport de contrat Convention CNET n° 89-1B 007909245 (Rapport LAAS N° 89329), 1989.

[105] M. Kaâniche, K. Kanoun, S. Metge, “Analyse des défaillances et suivi de la validation du logiciel d'un équipement de télécommunication,” *Annales des télécommunications*, vol. 45, pp. 657-670, 1990.

[106] M. Kaâniche, K. Kanoun, S. Metge, “Utilisation du modèle hyperexponentiel pour le suivi de la validation d'un équipement de télécommunications,” *7ème Colloque International de Fiabilité et de Maintenabilité*, (Brest, France), pp. 332-339, , 1990.

[107] M. Kaâniche, J. Nowak, K. Kanoun, M. R. de Bastos Martini, “The TROPICO-RA Switching System: Data Analyses for Software & Hardware Reliability Evaluation,” Rapport LAAS 94511, 1994.

[108] M. Kaâniche, L. Romano, Z. Kalbarczyk, R. Iyer, R. Karcich, "A Hierarchical Approach for Dependability Analysis of a Commercial Cache-based RAID Storage Architecture," *28th IEEE Int. Symposium on Fault-Tolerant Computing (FTCS-28)*, (Munich, Allemagne), pp. 6-15, IEEE Computer Society Press, 1998.

[109] Z. Kalbarczyk, G. Ries, M. S. Lee, Y. Xiao, J. Patel, R. K. Iyer, "Hierarchical Approach to Accurate Fault Modeling for System Evaluation," *3rd IEEE Int. Computer Performance & Dependability Symposium (IPDS'98)*, (Durham, NC, USA), pp. 249-258, IEEE Computer Society Press, 1998.

[110] K. Kanoun, "Croissance de la sûreté de fonctionnement des logiciels: caractérisation, modélisation et évaluation," Doctorat d'Etat, Institut National Polytechnique de Toulouse, Rapport LAAS N° 89320, 1989.

[111] K. Kanoun, M. Borrel, "Dependability of Fault-Tolerant Systems - Explicit Modeling of the Interactions between Hardware and Software Components," *IEEE International Computer Performance & Dependability Symposium (IPDS'96)*, (Urbana-Champaign, IL, USA), pp. 252-261, IEEE Computer Society Press, 1996.

[112] K. Kanoun, M. Borrel, T. Morteveille, A. Peytavin, "Modeling the Dependability of CAUTRA, a Subset of the French Air Traffic Control System," *26th IEEE Int. Symposium on Fault-Tolerant Computing (FTCS-26)*, (Sendai, Japon), pp. 106-115, IEEE Computer Society Press, 1996.

[113] K. Kanoun, M. Kaâniche, C. Béounes, J.-C. Laprie, J. Arlat, "Reliability Growth of Fault-Tolerant Software," *IEEE Transactions on Reliability*, vol. 42, pp. 205-219, 1993.

[114] K. Kanoun, M. Kaâniche, J.-C. Laprie, "Experience in Software Reliability: From Data Collection to Quantitative Evaluation," *4th Int. Symp. on Software Reliability Engineering (ISSRE'93)*, (Denver, CO, USA), pp. 234-245, IEEE Computer Society Press, 1993.

[115] K. Kanoun, M. Kaâniche, J.-C. Laprie, "Fiabilité du logiciel: de la collecte des données à l'évaluation probabiliste," *Technique et Science Informatiques*, vol. 16, pp. 865-895, 1997.

[116] K. Kanoun, M. Kaâniche, J.-C. Laprie, "Qualitative and Quantitative Reliability Assessment," *IEEE Software*, vol. 14, pp. 77-86, 1997.

[117] K. Kanoun, M. Kaâniche, J.-C. Laprie, S. Metge, "SoRel: A Tool for Reliability Growth Analysis and Prediction from Statistical Failure Data," *23rd IEEE Int. Symp. Fault-Tolerant Computing (FTCS-23)*, (Toulouse, France), pp. 654-659, IEEE Computer Society Press, 1993.

[118] K. Kanoun, J.-C. Laprie, "Modeling Software Reliability and Availability from Development Validation up to Operation," Rapport LAAS N° 85042, 1985.

[119] K. Kanoun, J.-C. Laprie, "Software Reliability Trend Analysis: From Theoretical to Practical Considerations," *IEEE Transactions on Software Engineering*, vol. 9, pp. 740-777, 1994.

[120] K. Kanoun, J.-C. Laprie, T. Sabourin, "A Method for Software Reliability Growth Analysis and Assessment," *1ere Conference sur le Génie Logiciel & ses Applications*, (Toulouse, France), pp. 859-878, 1988.

[121] K. Kanoun, T. Sabourin, "Software Dependability of a Telephone Switching System," *17th IEEE Int Symp. on Fault-Tolerant Computing (FTCS-17)*, (Pittsburgh, PA, USA), pp. 236-241, IEEE Computer Society Press, 1987.

[122] G. Q. Kenney, M. A. Vouk, "Measuring the Field Quality of Wide-Distribution Commercial Software," *3rd IEEE Int. Symp. on Software Reliability Engineering (ISSRE'92)*, (Raleigh, NC, USA), pp. 351-357, IEEE Computer Society Press, 1992.

[123] P. Koopman, J. Sung, D. Siewiorek, T. Marz, "Comparing Operating Systems Using Robustness Benchmarks," *Symp. on Reliable and Distributed Systems (SRDS'97)*, (Durham, NC, USA), pp. 72-79, IEEE Computer Society Press, 1997.

[124] J.-C. Laprie, "Prévision de la sûreté de fonctionnement et architectures de structures numériques temps réel réparables," Doctorat d'Etat, Université Paul Sabatier, Toulouse, France, N° 669, , 1975.

[125] J.-C. Laprie, "Evaluation de la sûreté de fonctionnement des logiciels en opération," *Technique et Science Informatiques*, vol. 2, pp. 233-247, 1983.

[126] J.-C. Laprie, "Dependability Evaluation of Software Systems in Operation," *IEEE Transactions on Software Engineering*, vol. SE-10, pp. 701-714, 1984.

[127] J.-C. Laprie, "Models for Software Availability Evaluation," *2nd National Workshop on Fault-Tolerant Computing Systems (NWFTCS II)*, (Melbourne, Australie), 1984.

[128] J.-C. Laprie, "Vers une théorie de la fiabilité du X-iel," *Technique et Science Informatiques*, vol. 7, pp. 315-330, 1988.

[129] J.-C. Laprie, "Hardware-and-Software Dependability Evaluation," *IFIP 11th World Computer Congress*, (San Francisco, CA, USA), pp. 109-114, North-Holland, 1989.

[130] J.-C. Laprie, "For a Product-in-a Process Approach to Software Reliability Evaluation," *3rd Int. Symp. on Software Reliability Engineering (ISSRE'92)*, (Raleigh, NC, USA), pp. 134-139, IEEE Computer Society Press, 1992.

[131] J.-C. Laprie, J. Arlat, C. Beounes, K. Kanoun, "Definition and Analysis of Hardware-and-Software Fault-Tolerant Architectures," *IEEE Computer*, vol. 23, pp. 39-51, 1990.

[132] J.-C. Laprie, J. Arlat, J. P. Blanquart, A. Costes, Y. Crouzet, Y. Deswarte, J. C. Fabre, H. Guillermain, M. Kaâniche, K. Kanoun, C. Mazet, D. Powell, C. Rabéjac, P. Thévenod, *Guide de la sûreté de fonctionnement*: Cépadues Editions, 1995-1996.

[133] J.-C. Laprie, C. Béounes, M. Kaâniche, K. Kanoun, "The Transformation Approach to the Modeling and Evaluation of the Reliability and Availability Growth," *20th IEEE Int. Symposium on Fault-Tolerant Computing (FTCS-20)*, (Newcastle, UK), pp. 364-371, IEEE Computer Society Press, 1990.

[134] J.-C. Laprie, C. Béounes, M. Kaâniche, K. Kanoun, "The Transformation Approach to the Modeling and Evaluation of the Reliability and Availability Growth of Systems in Operation," *Predictably Dependable Computing Systems*, B. Randell, J.-C. Laprie, H. Kopetz, B. Littlewood, Eds.: Springer-Verlag, 1995, pp. 389-406.

[135] J.-C. Laprie, M. Kaâniche, K. Kanoun, "Modeling Computer Systems Evolutions: Non-Stationary Processes and Stochastic Petri Nets—Application to Dependability Growth," *6th IEEE Int. Workshop on Petri Nets and Performance Models (PNPM'95)*, (Durham, NC, USA), pp. 221-230, IEEE Computer Society Press, 1995.

[136] J.-C. Laprie, K. Kanoun, "X-ware Reliability and Availability Modeling," *IEEE Transactions on Software engineering*, vol. SE-18, pp. 130-147, 1992.

[137] J.-C. Laprie, K. Kanoun, "Software Reliability and System Reliability," in *Handbook of Software Reliability Engineering*, M. R. Lyu, Ed.: McGraw-Hill, 1996, pp. 27-69.

[138] J.-C. Laprie, K. Kanoun, C. Béounes, M. Kaâniche, "The KAT (Knowledge-Action-Transformation) Approach to the Modeling and Evaluation of Reliability and Availability Growth," *IEEE Transactions on Software Engineering*, vol. SE-17, pp. 370-382, 1991.

[139] A. M. Law, D. W. Kelton, *Simulation Modeling & Analysis*: McGraw Hill, Inc., 1991.

[140] Y. Levendel, "Reliability Analysis of Large Software Systems: Defects Data Modeling," *IEEE Transactions on Software Engineering*, vol. SE-16, pp. 141-152, 1990.

[141] U. Lindqvist, E. Jonsson, "A Map of Security risks Associated with Using COTS," *IEEE Computer*, juin, pp. 60-66, 1998.

[142] B. Littlewood, "Software Reliability Model for Modular Program Structure," *IEEE Transactions on Reliability*, vol. R-28, pp. 241-246, 1979.

[143] B. Littlewood, S. Brocklehurst, N. Fenton, P. Mellor, S. Page, D. Wright, J. Dobson, J. McDermid, D. Gollmann, "Towards Operational Security Measures of Computer Security," *Journal of Computer Security*, vol. 2, pp. 211-229, 1993.

[144] T. Lunt, "A Survey of Intrusion Detection Techniques," *Computers & Security*, vol. 12, pp. 405-418, 1993.

[145] M. R. Lyu, "Handbook of Software Reliability Engineering," McGraw-Hill, 1995.

[146] S. Mahévas, G. Rubino, "Bounding Asymptotic Dependability and Performance Measures," *2nd IEEE Int. Computer Performance and Dependability Symposium (IPDS'96)*, (Urbana-Champaign, IL, USA), pp. 176-186, IEEE Computer Society Press, 1996.

[147] M. Malhotra, K. S. Trivedi, "Power-Hierarchy of Dependability-Model Types," *IEEE Transactions on Reliability*, vol. 43, pp. 493-502, 1994.

[148] R. Marie, B. Sericola, “Distribution du temps total de séjour dans un sous-ensemble d'états transitoires d'un processus markovien homogène à espace d'état fini,” IRISA-INRIA, Rapport technique 585, 1986.

[149] A. Marsan, G. Balbo, G. Conte, “A Class of Generalized Stochastic Petri Nets for the Performance Analysis of Multiprocessor Systems,” *ACM Transactions on Computers*, vol. 2, pp. 93-122, 1984.

[150] M. A. Marsan, G. Balbo, G. Franceschinis, S. Donatelli, *Modelling with Generalized Stochastic Petri Nets*: John Wiley & Sons, 1995.

[151] E. Mathieu, “Modélisation du système informatique de contrôle en route du trafic aérien français par réseaux d'activités stochastiques.,” Rapport LAAS N° 96530, 1996.

[152] J. F. Meyer, A. Movaghar, W. H. Sanders, “Stochastic Activity Networks: Structure, Behaviour and Application,” *IEEE International Workshop on Timed Petri Nets*, (Torino, Italie), pp. 139-156, IEEE Computer Society Press, 1985.

[153] MoD, “The Procurement of Safety Critical Software in Defense Equipment-Parts 1 & 2,” UK Ministry of Defence, Interim Defence Standard 00-55, 1991.

[154] M. Molloy, “Performance Analysis Using Stochastic Petri Nets,” *IEEE Transactions on Computers*, vol. C-39, pp. 913-917, 1982.

[155] A. D. E. Muffet, “Crack Version 4.1 — A Sensible Password Checker for Unix,” *disponible par ftp à l'adresse ftp.cert.org*, 1992.

[156] R. R. Muntz, E. de Souza e Silva, A. Goyal, “Bounding Availability of Repairable Computer Systems,” *IEEE Transactions on Computers*, vol. 38, pp. 1714-1723, 1989.

[157] J. K. Muppala, A. Sathay, R. Howe, K. S. Trivedi, “Dependability Modeling of a Heteregeneous VAX-cluster System Using Stochastic Reward Nets,” *Hardware and Software Fault Tolerance in Parallel Computing Systems*, D. R. Avresky, Ed., 1992, pp. 33-59.

[158] J. Musa, A. Iannino, K. Okumoto, *Software Reliability: Measurement, Prediction, Application*. New-York: McGraw-Hill, 1987.

[159] J. Musa, K. Okumoto, “A Logarithmic Poisson Execution Time Model for Software Reliability Measurement,” *COMPSAC'84*, (Chicago, IL, USA), pp. 230-238, , 1984.

[160] S. Natkin, “Les réseaux de Petri stochastiques,” Thèse de Docteur Ingénieur, CNAM, 1980.

[161] V. F. Nicola, M. K. Nakayama, P. Heidelberger, A. Goyal, “Fast Simulation of Highly Dependable Systems with General Failure and Repair Processes,” *IEEE Transactions on Computers*, vol. 42, pp. 1440-1452, 1993.

[162] NIST-NSA, “Federal Criteria for Information Technology Security,” *Version 1.0*: National Institute of Standards and Technology (NIST) and National Security Agency (NSA), 1992.

[163] R. Ortalo, "Evaluation quantitative de la sécurité des systèmes d'information," Thèse de Doctorat, Institut National Polytechnique de Toulouse, N° 1418, Rapport LAAS N° 98164, 1998.

[164] R. Ortalo, Y. Deswarte, "Quantitative Evaluation of Information System Security," *14th IFIP Int. Information Security Conference (IFIP/SEC'98)*, (Vienna-Budapest, Autriche-Hongrie), pp. 321-332, Chapman & Hall, 1998.

[165] R. Ortalo, Y. Deswarte, M. Kaâniche, "Experimenting with Quantitative Evaluation Tools for Monitoring Operational Security," *Dependable Computing for Critical Applications 6 (6th IFIP Int. Working Conference on Dependable Computing for Critical Applications: DCCA-6, Garmish, Allemagne, 1997)*, *Dependable Computing and Fault-Tolerant Systems*, 11 ed: IEEE Computer Society Press, 1998, pp. 307-328.

[166] P. I. Pignal, "An Analysis of Hardware and Software Availability Exemplified on the IBM-3725 Communication Controller," *IBM Journal of Research and Development*, vol. 32, pp. 268-278, 1988.

[167] D. Powell, "Failure Mode Assumptions and Assumption Coverage," *22nd IEEE Int. Symp. on Fault-Tolerant Computing (FTCS-22)*, (Boston, MA, USA), pp. 386-395, IEEE Computer Society Press, 1992.

[168] G. Profos, "Analyse et évaluation de la fiabilité du logiciel de l'autocommutateur TROPICO-RS," Rapport LAAS N° 93091, 1993.

[169] C. V. Ramamoorthy, F. B. Bastani, "Modeling of the Software Reliability Growth Process," *COMPSAC 80*, (Chicago, IL, USA), pp. 161-169, , 1980.

[170] B. Randell, "System Structure for Software Fault Tolerance," *IEEE Transactions on Software Engineering*, vol. SE-1, pp. 220-232, 1975.

[171] A. Reibman, R. Smith, K. Trivedi, "Markov and Markov Reward Model Transient Analysis: An Overview of Numerical Approaches," *European Journal of Operation Research*, vol. 40, pp. 257-267, 1989.

[172] A. L. Reibman, M. Veeraraghavan, "Reliability Modeling: An Overview for System Designers," *IEEE Computer*, vol. 24, pp. 49-57, 1991.

[173] G. Ries, R. K. Iyer, "Evaluating the Impact of Transient Faults on Software Behavior: Case Study of a Commercial High-Speed Network," in *Dependable Computing for Critical Applications 6 (6th IFIP Int. Working Conference on Dependable Computing for Critical Applications: DCCA-6, Garmish, Allemagne, 1997)*, *Dependable Computing and Fault-Tolerant Systems*, 11 ed: IEEE Computer Society, 1997, pp. 271-287.

[174] G. L. Ries, G. S. Choi, R. K. Iyer, "Device-Level Transient Fault Modeling," *24th IEEE International Symposium on Fault-Tolerant Computing (FTCS-24)*, (Austin TX, USA), pp. 86-94, IEEE Computer Society Press, 1994.

[175] I. Rojas, "Compositional Construction of SWN Models," Dept. of Computer Science, University of Edinburgh ECS-CSG-21-96, 1996.

[176] N. Ross, "The Collection and Use of Data for Monitoring Software Projects," in *Measurement for Software Control and Assurance*, B. A. K. a. B. Littlewood, Ed. London and New York: Elsevier Applied Science, 1989, pp. 125-154.

[177] W. W. Royce, "Managing the Development of Large Software Systems: Concepts and Techniques," *WESCON*, 1970.

[178] RTCA, "Software Considerations in Airborne Systems and Equipment Certification," RTCA/EUROCAE, Paper no. 591-91/SC167-164, DO-178 B.5, 1991.

[179] RTCA/EUROCAE, "Considérations sur le logiciel en vue de la certification des systèmes et équipements de bord," Radio Technical Commission for Aeronautics (RTCA), European Organization for Civil Aviation Electronics (EUROCAE), 1992.

[180] G. Rubino, B. Sericola, "A Finite Characterization of Weak Lumpable Markov Processes. Part II: The continuous-time Case," *Stochastic Processes and Their Applications*, vol. 45, 1993.

[181] SAE, "Certification Considerations for Highly-Integrated or Complex Aircraft Systems," in *Doc. ARP 4754, Systems Integration Requirements Task Group AS-1C, ASD*: Society of Automotive Engineers, Inc., 1995.

[182] W. H. Sanders, J. F. Meyer, "A Unified Approach for Specifying Measures of Performance, Dependability, and Performability," in *Dependable Computing and Fault Tolerant Systems*, vol. 4, J.-C. L. A. Avizienis, Ed.: Springer-Verlag, 1991, pp. 215-237.

[183] W. H. Sanders, W. D. Obal II, M. A. Qureshi, F. K. Widjanarko, "The *UltraSAN* Modeling Environment," *Performance Evaluation*, vol. 21, 1995.

[184] P. J. Schweitzer, "A Survey of Aggregation-Disaggregation in Large Markov Chains," in *Numerical Solutions of Markov Chains*, W. J. Stewart, Ed.: Marcel Dekker, 1991, pp. 63-87.

[185] P. Semal, "Analysis of Large Markov Models," Thèse de Doctorat, Faculté des Sciences Appliquées, Université Catholique de Louvain, Belgique, 1992.

[186] P. Semal, "Refinable bounds for Large Markov Cahins," *IEEE Transactions on Computers*, vol. 44, pp. 1216-1222, 1995.

[187] D. P. Siewiorek, R. S. Swarz, *Reliable Computer Systems - Design and Evaluation*. Bedford, MA, USA: Digital Press, 1992.

[188] G. E. Stark, "Dependability Evaluation of Integrated Hardware/Software Systems," *IEEE Transactions on Reliability*, vol. R-36, pp. 440-444, 1987.

[189] W. J. Stewart, A. Goyal, "Matrix Methods in Large Dependability Models," IBM Res. Rep. RC-11485, 1985.

[190] N. Talbert, "The Cost of COTS," *IEEE Computer*, vol. June, pp. 46-52, 1998.

[191] J. M. Voas, "Certifying Off-the-Shelf Software Components," *Computer*, vol. 31, pp. 53-59, 1998.

[192] J. J. Wallace, W. W. Barnes, "Designing for Ultrahigh Availability: The Unix RTR Operating System," *IEEE Computer*, pp. 31-39, 1984.

[193] C.-Y. Wang, K. Trivedi, "Integration of Specification for Modeling and Specification for System Design," *14th Int. Conf. on Application and Theory of Petri Nets*, (Chicago, IL, USA), pp. 473-492, Springer-Verlag, 1993.

[194] M. Xie, *Software Reliability Modeling*. Singapore: World-Scientific, 1991.

[195] S. Yamada, S. Osaki, H. Narihisa, "Software Reliability Growth Modeling with Number of Test Runs," *Trans. IECE Japan*, vol. E-67, pp. 79-83, 1984.

[196] C. R. Yount, D. P. Siewiorek, "A Methodology for the Rapid Injection of Transient Hardware Errors," *IEEE Transactions on Computers*, vol. 45, pp. 881-891, 1996.

[197] P. Ziegler, H. Szczerbicka, "A Structure Based Decomposition Approach for GSPN," *6th Int. Workshop on Petri Nets and Performance Models*, (Durham, NC, USA), pp. 261-270, IEEE Computer Society Press, 1995.

8. TABLE DES MATIÈRES

Habilitation à diriger les recherches de Mohamed Kaâniche

"Évaluation de la sûreté de fonctionnement informatique — fautes physiques, fautes de conception, malveillances"

Résumé

Les travaux résumés dans ce mémoire ont pour cadre la sûreté de fonctionnement des systèmes informatiques. Ils couvrent plusieurs aspects complémentaires, à la fois théoriques et expérimentaux, que nous avons groupés en quatre thèmes. Le premier thème traite de la définition de méthodes permettant de faciliter la construction et la validation de modèles complexes pour l'analyse et l'évaluation de la sûreté de fonctionnement. Deux approches sont considérées : les réseaux de Petri stochastiques généralisés et la simulation comportementale en présence de fautes. Le deuxième thème traite de la modélisation de la croissance de fiabilité pour évaluer l'évolution de la fiabilité et de la disponibilité des systèmes en tenant compte de l'élimination progressive des fautes de conception. Ces travaux sont complétés par la définition d'une méthode permettant de faciliter la mise en œuvre d'une étude de fiabilité de logiciel dans un contexte industriel. Le troisième thème concerne la définition et l'expérimentation d'une approche pour l'évaluation quantitative de la sécurité-confidentialité. Cette approche permet aux administrateurs des systèmes de suivre l'évolution de la sécurité opérationnelle quand des modifications, susceptibles d'introduire de nouvelles vulnérabilités, surviennent dans la configuration opérationnelle, les applications, le comportement des utilisateurs, etc. Enfin, le quatrième thème porte d'une part, sur l'élaboration d'un modèle de développement destiné à la production de systèmes sûrs de fonctionnement, et d'autre part, sur la définition de critères d'évaluation visant à obtenir une confiance justifiée dans l'aptitude des systèmes à satisfaire leurs exigences de sûreté de fonctionnement, en opération et jusqu'au retrait du service.

Mots clés : sûreté de fonctionnement, évaluation, modèles complexes, réseaux de Petri stochastiques généralisés, simulation hiérarchique, injection de fautes, croissance de fiabilité, sécurité-confidentialité, modèles de développement.

"Dependability evaluation of computing systems—phyiscal faults, design faults, malicious faults"

Abstract

The research summarized in this report focuses on the dependability of computer systems. It addresses several complementary, theoretical as well as experimental, issues that are grouped into four topics.

The first topic concerns the definition of efficient methods that aim to assist the users in the construction and validation of complex dependability analysis and evaluation models. The second topic deals with the modeling of reliability and availability growth, that mainly result from the progressive removal of design faults. A method is also defined to support the application of software reliability evaluation studies in an industrial context. The third topic deals with the development and experimentation of a new approach for the quantitative evaluation of operational security. This approach aims to assist the system administrators in the monitoring of operational security, when modifications, that are likely to introduce new vulnerabilities, occur in the system configuration, the applications, the user behavior, etc. Finally, the fourth topic addresses: a) the definition of a development model focussed at the production of dependable systems, and b) the development of assessment criteria to obtain justified confidence that a system will achieve, during its operation and up to its decommissioning, its dependability objectives.

Keywords: dependability, evaluation, complex models, Generalized Stochastic Petri Nets, hierarchical simulation, fault injection, reliability growth, security, development process models.