Document résumé de synthèse des travaux

De l'Action Spécifique AS 23

"Test avancé de systèmes complexes, test de robustesse"

Co-animateurs :

- Richard Castanet (LaBRI)
- Hélène Waeselynck (LAAS)

Durée : 12 mois (Novembre 2001 - Novembre 2002)

Equipes	**Personnes impliquées**
• Irisa, projets Triskell et Vertecs	Claude Jard, Thierry Jéron, Hervé Marchand
• LaBRI, équipe MVTsi (Modélisation, Vérification et Test de systèmes informatisés)	Richard Castanet, David Janin
• LAAS-CNRS, groupe Tolérance aux fautes et sûreté de fonctionnement informatique	Olfa Abdellatif-Kaddour, Jean Arlat, Hélène Waeselynck
• LRI, équipe Programmation et Génie Logiciel	Marie-Claude Gaudel, Sandrine-Dominique Gouraud, Gregory Lestiennes
• VERIMAG, équipe Systèmes Asynchrones	Jean-Claude Fernandez, Laurent Mounier, Cyril Pachon

1 Introduction

Alors que de nombreux travaux ont traité du test de conformité par rapport à une spécification, un problème encore peu étudié concerne le test de la robustesse d'un système par rapport à des sollicitations quelconques – y compris erronées ou inopportunes – de son environnement.

Ce problème est pourtant appelé à prendre une importance accrue, en particulier dans des systèmes critiques ou embarqués, si l'on considère certaines évolutions récentes tant dans les méthodes d'ingénierie que dans les types d'utilisation des systèmes. Les nouveaux paradigmes de conception à base de composants réutilisables impliquent l'intégration de ces composants dans des contextes applicatifs qui peuvent fortement varier : le test de ces composants doit prendre en compte la diversité des utilisations potentielles, d'une part pour consolider leur aptitude à réagir correctement aux sollicitations, et d'autre part pour permettre l'identification explicite de leurs modes de défaillances, facilitant ainsi la mise en œuvre de parades lors de leur intégration. A un niveau plus macroscopique, le déploiement de systèmes embarqués critiques, devant remplir leur mission de façon autonome en dépit d'aléas, et en particulier de fautes internes ou externes, impose là encore de fortes contraintes de robustesse. Enfin, la tendance croissante à l'interconnexion et à la répartition renforce le caractère inconnu et incertain – voire même malveillant (pirates informatiques,…) – de l'environnement des systèmes informatiques.

L'IEEE définit la robustesse comme "le degré selon lequel un système, ou un composant, peut fonctionner correctement en présence d'entrées invalides ou de conditions environnementales stressantes" (*IEEE Std 610.12-1990*). Partant de là, nous proposons la définition suivante pour le test de robustesse (les modifications par rapport à l'IEEE sont indiquées en italique, et seront justifiées plus bas).

Le **test de robustesse** vise à vérifier la capacité d'un système, ou d'un composant, à fonctionner de *façon acceptable* en présence de *fautes* ou de conditions environnementales stressantes".

Par la suite, nous emploierons le terme générique d'aléa pour désigner à la fois les fautes et les conditions environnementales stressantes.

La robustesse en présence de *fautes* est liée à des propriétés de tolérance aux fautes. Différents points de vue peuvent être adoptés pour classer les fautes [13]. Nous nous intéresserons plus particulièrement à deux d'entre eux. (1) La situation des fautes par rapport aux frontières du système conduit à distinguer les fautes internes et externes. Notons que la définition de la robustesse par l'IEEE, focalisée sur les "entrées invalides", met l'accent sur les fautes externes. Nous pensons qu'il n'y a pas de raison d'écarter a priori les fautes internes. (2) La nature des fautes conduit à distinguer les fautes accidentelles et intentionnelles. En incluant les fautes intentionnelles (notamment, les intrusions et les logiques malignes), on peut considérer les tests de pénétration [24] comme des tests de robustesse ciblant les manipulations non-autorisées de l'information.

Les conditions environnementales stressantes sont liées à des variations du profil d'utilisation, que l'on considère les valeurs ou les caractéristiques temporelles des sollicitations du système. Ces variations peuvent être inhérentes à l'environnement du système en vie opérationnelle (ex. : charge variable d'un réseau), ou être induites par la ré-utilisation du système dans un environnement différent de celui pour lequel il avait été conçu. Le fait de considérer des aléas qui ne sont pas forcément des fautes implique que la notion de robustesse n'est pas limitée à la notion de tolérance aux fautes.

C'est à dessein que nous employons le terme de fonctionnement *acceptable*, plutôt que *correct*. Dans certains cas, l'exigence de robustesse peut se traduire par un renforcement de la spécification, en ajoutant des propriétés liées à la prise en compte d'aléas. Une implémentation robuste doit alors être correcte. Mais dans le cas général, la correction et la robustesse peuvent être des exigences orthogonales. En présence d'aléas, on tolère parfois des violations de la spécification, pourvu que certaines propriétés de robustesse soient satisfaites. En fait, l'existence d'une spécification fonctionnelle n'est même pas un préalable au test de robustesse. Le test peut être conduit dans l'objectif de donner des informations sur le comportement du système confronté à des aléas. Par exemple, plusieurs travaux ont porté sur la caractérisation des modes de défaillance de systèmes exécutifs (micro-noyaux, systèmes d'exploitation ou intergiciels) [2, 12, 16, 17].

Ce document présente dans un premier temps (chapitre 2) la position du problème telle qu'elle a été identifiée par les membres de l'AS et ensuite les prospectives (chapitre 3) proposées par les différentes équipes partenaires (IRISA, LAAS, LaBRI, LRI, Vérimag).

2 Position du problème

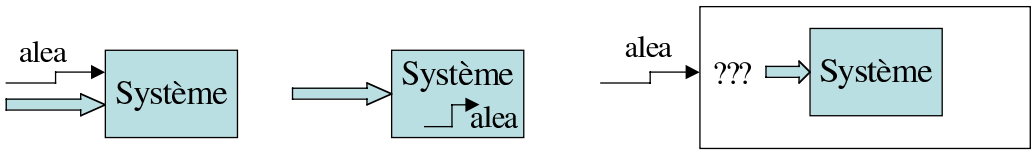
Pour comprendre les spécificités du test de robustesse, nous partons d'une situation de référence, le test de conformité par rapport à un modèle ne prenant pas en compte les aléas, et analysons les écarts induits par la considération d'aléas. La discussion précédente sur la nature des aléas, et sur la notion de comportement acceptable, montre deux exemples d'écarts possibles : 1) le domaine d'entrée n'est pas nécessairement le même ; 2) le comportement du système peut être observé selon un point de vue différent.

2.1 Domaine d'entrée du test

Le domaine d'entrée doit être étendu pour prendre en compte l'occurrence d'aléas. Nous distinguons différents cas liés à la combinaison de deux points de vue :

- la situation de l'aléa par rapport aux frontières du système. Elle peut être externe, interne, ou au-delà des frontières du système (figure 1).
- Aléa lié à un aspect de la réalité représenté/non représenté par le modèle (i.e., l'aléa peut-il être exprimé dans les termes du modèle ?).

La combinaison de ces points de vue donne en fait cinq cas, car on écarte la combinaison *situation au-delà des frontières / aspect de la réalité représenté dans le modèle*.



(a) aléa externe (b) aléa interne (c) aléa au-delà des frontières du système

Figure 1. Situation de l'aléa par rapport aux frontières du système

Aléa externe, lié à un aspect de la réalité représenté dans le modèle

Le modèle précise les types des paramètres d'entrées et les pré-conditions devant être satisfaites. Leur analyse permet d'expliciter des classes d'entrées invalides. Par exemple, pour un paramètre d'entrée $x \in [0..10]$, deux classes invalides pourraient être $x<0$ et $x>10$. La décomposition en classes d'entrées invalides, et la couverture de ces classes au cours du test, est une pratique classique, qui est notamment utilisée pour la caractérisation des modes de défaillance des systèmes exécutifs [12, 17].

Aléa externe, lié à un aspect de la réalité non représenté dans le modèle

A titre d'exemple, dans [23], différents profils de charge sont utilisés pour tester la conformité de sous-systèmes (objets concurrents) par rapport à leur modèle d'interface. Le modèle d'interface définit l'alphabet d'entrée, l'ordre dans lequel les événements d'entrée doivent être traités, et des post-conditions relatives à ces traitements. Le fait que le temps entre deux événements puisse être inférieur au temps de réaction d'un objet est un aspect de la réalité non représenté dans le modèle.

Aléa interne, lié à un aspect de la réalité représenté dans le modèle

Typiquement, on corrompt les données et messages échangés entre composants du système sous test. Le modèle de faute est ainsi lié à la description des interfaces entre composants. Cette approche a été utilisée dans un but de caractérisation des modes de défaillance [2, 22] : on étudie les mécanismes de propagation d'erreur au sein d'un système, suite à la défaillance (simulée) d'un composant. Elle a

également été utilisée pour tester des mécanismes de tolérance aux fautes pour des calculateurs parallèles [5], ou encore pour vérifier des propriétés de systèmes répartis en présence de fautes de communication [7].

Aléa interne, lié à un aspect de la réalité non représenté dans le modèle

Les aspects bas niveau tels que l'utilisation dynamique de la mémoire, des registres, etc. ne sont généralement pas modélisés. A titre d'exemple, pour la caractérisation de modes de défaillance de micronoyaux, un modèle de fautes supporté par l'outil MAFALDA est la corruption bit-flip de mots mémoire [2] . D'un point de vue pratique, la mise en œuvre du test peut s'appuyer sur des outils d'analyse en-ligne de l'activité du système, pour déterminer où et quand injecter les fautes de bas niveau [21].

Aléa au-delà des frontières du système, lié à un aspect de la réalité non représenté dans le modèle

Certains aléas sont liés à un modèle de l'environnement, et ne peuvent être directement exprimés en termes de paramètres d'entrée du système. Par exemple, [3] et [20] étudient des processus de contrôle/commande, et les modèles de fautes considérés sont relatifs à un simulateur de l'environnement physique contrôlé. C'est le simulateur qui traduit l'effet des aléas (ex : force du vent venant affecter le déplacement d'un véhicule, défaillance d'un actionneur, …) sur les entrées du logiciel de contrôle/commande, conformément aux lois physiques qu'il encode. Un autre exemple, issu de travaux sur l'injection de fautes dans le matériel, est le bombardement de circuits par des ions lourds [6] : dans ce cas, les aléas environnementaux sont les radiations, et on ne sait pas a priori comment le circuit sera affecté.

Notons qu'un modèle d'aléas ne se cantonne pas nécessairement à l'un de ces cas. En particulier, les tests de pénétration [24], qui cherchent les vulnérabilités/failles susceptibles d'être exploitées par un attaquant, reposent sur l'explicitation d'hypothèses d'aléas pouvant concerner les cinq cas identifiés ci-dessus.

2.2 Domaine de sortie et oracle

Nous avons mentionné que, dans le cas du test de robustesse, le comportement du système peut être observé selon un point de vue différent de celui adopté pour un test de conformité. Les différences peuvent résider dans ce que l'on observe (sorties de test), et comment (oracle de test). Nous distinguons quatre cas, les deux derniers étant le plus souvent rencontrés dans la littérature.

Sorties observées et oracle inchangés

Le système doit délivrer le même service qu'en l'absence aléas

Sorties inchangées et oracle modifié

Ce cas pourrait correspondre au test de classes d'entrées invalides, en supposant que le domaine de sortie reste le même. L'oracle doit être étendu pour expliciter les cas en dehors de la spécification.

Plus de sorties observables, oracle modifié

L'extension du domaine de sortie peut répondre à deux besoins :

- prendre en compte les modes de défaillances. Par exemple, la levée d'une exception devient une sortie attendue.
- Déterminer si l'état interne du système est erroné ou normal. Dans ce cas, le choix de ce qu'on observe est problématique, car les mécanismes de propagation d'erreur au sein d'un système sont complexes. Le plus souvent ce choix est lié à une analyse ad hoc prenant en compte des informations de bas niveau (par exemple, écriture intempestive dans une zone mémoire particulière, …).

L'oracle se base sur ces observations. Pour des systèmes de taille réaliste, il est souvent lié à un modèle comportemental "rudimentaire", focalisé sur quelques propriétés [2, 7, 22].

Plus de sorties observables, oracle "flou"

Parfois, on ne sait pas déterminer a priori un comportement attendu. Par analyse des résultats de test, on détermine a posteriori si le comportement observé est robuste. Ce cas se rencontre pour beaucoup de travaux relatifs à la caractérisation des modes de défaillance, et pour certains tests de pénétration.

2.3 Conclusion sur les spécificités du test de robustesse

Un test de robustesse ne se ramène pas toujours à un test de conformité. Notre analyse est que la prise en compte d'aléas pose des problèmes de modélisation, qui rejaillissent sur les solutions pouvant être apportées aux problèmes de l'oracle et de la sélection des entrées de test.

Lorsqu'il est possible d'exhiber un modèle du système pour lequel les aléas soient des entrées banalisées, il est envisageable d'exploiter le corpus théorique acquis dans le cadre du test de conformité. Cependant, il reste nécessaire d'inventer de nouvelles méthodes de sélection de test ciblant les propriétés de robustesse.

Dans d'autres cas, on pourra partir d'un modèle nominal, et l'enrichir avec des notions spécifiques à la robustesse, telles que la notion d'état erroné, de mode dégradé, etc. Ceci nécessitera de développer de nouvelles méthodes de modélisation et d'analyse de modèles.

Enfin, il restera toujours des cas où la modélisation du système en présence d'aléas sera problématique, et où la sélection des tests devra être basée sur un modèle des entrées. Dans la littérature, les aléas sont alors vus comme *se superposant* à l'activité nominale du système, et la sélection des tests considère séparément ces deux types d'entrées (par exemple, modèle de type profil opérationnel pour les entrées relatives à l'activité nominale, et classes d'équivalences de fautes pour les seconds). La superposition *activité nominale x aléa* s'effectue souvent selon des procédés ad hoc (ex : interception et corruption aléatoire des messages engendrés par l'activité du système), faute d'hypothèses pertinentes pour guider le test. Eventuellement, elle est mise en œuvre à l'aide d'outils d'analyse de l'activité du système : traçage en-ligne des chemins d'exécution et ressources utilisées, …

3 Prospective issue de l'AS

Ce chapitre décrit les différentes prospectives proposées par les équipes de recherche associées dans le cadre de l'Action Spécifique. Certaines équipes avaient déjà des travaux en cours pouvant se rattacher à la problématique du test de robustesse, tandis que pour d'autres il s'agissait d'un thème entièrement nouveau. Dans tous les cas, les orientations de prospective ont bénéficié des apports conceptuels issus de l'Action. Les approches proposées par l'IRISA, le LaBRI, le LRI et Vérimag sont basées sur la construction d'un modèle formel spécifiant le système à étudier, et mettent l'accent sur les aspects analyse de modèle et sélection de tests à partir des modèles. Le cas de systèmes complexes pour lesquels aucune spécification formelle complète n'est disponible est abordé par le LAAS, qui propose notamment de guider la sélection par des heuristiques et des méthodes issues de travaux sur l'injection de fautes.

3.1 Prospective IRISA

Nous nous intéressons à la génération de tests de robustesse à partir de spécifications comportementales. Nous partons du principe que pour générer des tests de robustesse du système, il faut se baser sur une spécification elle-même robuste. L'approche que nous proposons est donc fondée sur la donnée d'une spécification du système à tester et de ses propriétés de robustesse. Nous ne supposons pas que la spécification satisfait ces propriétés de robustesse, parce qu'elle est issue d'une spécification incomplète par exemple, et nous proposons dans un premier temps de l'assurer par des techniques de synthèse de contrôleur [19]. Ces techniques ont pour but d'empêcher la spécification d'atteindre des comportements non robustes, i.e. ne satisfaisant pas les propriétés requises.

Nous proposons un modèle de spécification comportementale du système, raffinement du modèle des systèmes de transition. Dans ce modèle, on partitionne les événements selon plusieurs aspects: en événements normaux ou aléas (pour l'aspect robustesse), en entrées, sorties et événements internes (pour le test), en événements contrôlables et incontrôlables (pour le problème de contrôle). On propose aussi

de partitionner l'ensemble des états en modes: un mode nominal et des modes dégradés, conséquences d'aléas. Les propriétés attendues du systèmes, en particulier les propriétés de robustesse, doivent également être spécifiées, en logique temporelle par exemple. Typiquement, la robustesse est exprimée par une propriété exprimant que, quelque soit l'hostilité des aléas (parmi des modèles d'aléas réalistes), le système doit avoir un comportement acceptable.

La spécification initiale ne satisfait pas forcément cette propriété. Il est alors nécessaire de restreindre ses comportements par contrôle afin d'assurer les propriétés. L'idée est que le contrôle s'effectue à l'intérieur du système, mais que seuls certains événements sont contrôlables. Par exemple les aléas sont a priori non contrôlables ainsi que les entrées du système, mais certains événements internes ou sorties sont contrôlables.

Après contrôle, on dispose donc d'une spécification qui décrit les comportements attendus du système, et en particulier ses modes, et des propriétés de robustesse satisfaites par la spécification, et que devrait aussi satisfaire le système sous test. Le problème est alors de produire une suite de tests qui puisse détecter qu'une implémentation n'est pas robuste, ou qui puisse certifier la robustesse, moyennant des hypothèses. Les questions à résoudre sont, d'une part la sélection de tests (sélectionner un ensemble significatif mais raisonnable et correct de tests), d'autre part la question de l'oracle (quand est-ce qu'un test doit rejeter ou accepter une implémentation). Pour la sélection, on se fonde à la fois sur la spécification afin de se focaliser sur les passages de modes, et sur ses propriétés de robustesse afin de cibler les états critiques (états à partir desquels un contrôle a été opéré et où les propriétés pourraient être invalidées dans le système). Pour l'oracle, on peut se fonder sur la spécification (pour tester la conformité à une spécification robuste) et/ou ses propriétés (pour tester la robustesse). Si la spécification est l'oracle, on peut réutiliser des techniques déjà développées pour le test de conformité, dans l'outil de génération de tests TGV [11] par exemple. Si l'oracle dépend aussi des propriétés, de nouvelles techniques doivent être envisagées. Par ailleurs, notre approche ouvre de nouvelles pistes de recherche intéressantes. Par exemple il apparaît nécessaire d'étudier la relation entre conformité et satisfaction de propriété. Egalement, nous nous intéressons à l'expression du problème de génération de tests en termes de problème de contrôle, ce qui permettrait de traiter l'ensemble du problème avec les mêmes outils théoriques et logiciels.

3.2 Prospective LAAS

Trois axes de recherche se rattachent à cette AS. Le premier concerne la caractérisation de modes de défaillance de systèmes exécutifs par injection de fautes. Les deux suivants portent sur le test de propriétés de tolérance aux fautes, l'objectif étant de provoquer la mise en défaut de ces propriétés par combinaison judicieuse d'entrées fonctionnelles et de fautes. Dans un cas, on dispose d'une modélisation d'un algorithme de tolérance aux fautes, et on explore une méthode originale permettant de dériver des critères de sélection de test à partir de preuves partielles. Dans le second cas, on ne dispose pas de modèle exploitable pour le test : la sélection se base sur des heuristiques d'optimisation qui évaluent en-ligne la "dangerosité" des combinaisons testées.

Caractérisation de modes de défaillance de systèmes exécutifs

Des travaux antérieurs ont notamment porté sur la caractérisation de micro-noyaux [2] ou d'intergiciels CORBA [15]. On s'intéresse maintenant à la caractérisation de composants particuliers d'un système exécutif que sont les pilotes de périphériques, ou *drivers*. Des études ont montré que les drivers pouvaient représenter jusqu'à 70% du code source d'une installation système, et être responsable de plus de la moitié des problèmes rencontrés en opération. Le test de la robustesse d'un noyau de système d'exploitation vis-à-vis des défaillances de ces composants constitue donc un thème tout à fait pertinent. Il pose cependant des problèmes de modélisation du domaine d'entrée. En effet, outre leur caractère bas niveau, les drivers ont pour caractéristique de ne pas posséder d'interface clairement définie au sein du système. On étudiera la possibilité de modéliser des protocoles d'interaction entre les drivers et les mécanismes du noyau système, pour faciliter la définition et la réalisation du test.

Complémentarité test et preuve pour la vérification de la tolérance aux fautes

On trouve dans la littérature des exemples d'algorithmes de tolérance aux fautes, partiellement prouvés (manuellement, ou avec des assistants de preuve), mais révélés incorrects a posteriori. Notre objectif est d'explorer la conception de jeux de test basés sur l'analyse de l'arbre de preuve, de façon à cibler les lacunes de la preuve. Une première étude de cas a montré qu'une démonstration (manuelle) incorrecte ne permet pas nécessairement d'extraire des informations pertinentes pour le test [14]. Nous poursuivrons ces investigations avec l'exemple d'un algorithme incorrect possédant une démonstration plus aboutie, celle-ci ayant été consolidée par *model-checking* d'instances de l'algorithme. Nous développerons également une modélisation PVS des deux algorithmes précédents. Ceci nous permettra de considérer, pour la conception du test, l'utilisation d'informations issues d'une preuve formelle – mais partielle. Nous élargirons ensuite notre étude à d'autres exemples d'algorithmes de tolérance aux fautes, cette fois en partant d'algorithmes prouvés, et en analysant l'impact de fautes insérées dans ces algorithmes sur les arbres de preuve.

Heuristiques d'optimisation pour le test de robustesse

Nos travaux s'intéressent au couplage de logiciels de contrôle/commande à leur environnement physique (dispositif matériel, lois physiques du processus contrôlé), en prenant en compte l'occurrence de fautes physiques. Une stratégie de test a déjà été définie et expérimentée [3]. Elle consiste en la construction incrémentale de scénarios de test, avec pour objectif de provoquer une défaillance catastrophique. Chaque étape de la stratégie explore les suites possibles de scénarios identifiés à l'étape précédente, cette exploration étant guidée par des heuristiques d'optimisation (ex : recuit simulé, algorithmes génétiques) qui évaluent la « distance » par rapport à des situations dangereuses pour le système. D'autres auteurs avaient par ailleurs montré le potentiel de telles heuristiques pour le test de robustesse [20]. Un problème crucial concerne cependant le calibrage d'une heuristique au problème traité, qui s'effectue généralement de façon empirique et requiert un grand nombre d'essais expérimentaux. Nous étendrons donc les travaux précédents en abordant des aspects plus fondamentaux des heuristiques. Dans la littérature, des études récentes ont porté sur la mise en relation de mesures de problèmes d'optimisation avec les comportements des heuristiques. Il conviendra de déterminer si ces mesures offrent un cadre conceptuel adéquat pour prévoir, ou expliquer, le comportement d'heuristiques appliquées à des instances de problèmes de test de robustesse.

3.3 Prospective LaBRI

Dans l'approche proposée par le LaBRI, on considère le test de robustesse selon une extension du test de conformité. En conséquence, on propose de définir une relation de robustesse, notée "Rob", étendant la relation "Conf" utilisée dans le cadre du test de conformité :

I *Rob* S si $\forall \sigma \in L_\theta^*$, $^\forall A \subseteq L_\theta$, I après σ refuse A $\Rightarrow$ S après σ refuse A;

où I et S sont respectivement l'implantation (ou son modèle) et la spécification, σ est une trace, L* est le langage sur l'alphabet d'entrée (pour prendre en compte les événements inopportuns) et L_θ^* est le langage étendu avec des événements corrompus ou inconnus.

Dans cette approche nous faisons l'hypothèse que la robustesse est plus forte que la conformité : I *Rob* S $\Rightarrow$ I *Conf* S. Cette hypothèse peut être clairement discutée. En particulier elle n'est envisageable que si l'on dispose d'un modèle de la spécification.

Spécification

- S peut être spécifiée par un LTS : S = (Q, A, T, q_0)

où A est l'ensemble des actions (entrées, sorties ou évènements internes), Q l'ensemble des états, T l'ensemble des transitions q_0 l'état initial.

Le langage bâti avec A peut être étendu en L_θ : L $\cup$ {θ}, où θ est un événement inconnu ou corrompu. Pour le modèle de fautes, on ne considère que les événements inopportuns, inconnus ou corrompus. La

propriété de robustesse à tester qui définit un comportement acceptable de l'implantation ne s'exprime que par une séquence d'événements à observer.

Méthodologie

Un testeur de robustesse permet de vérifier que l'implantation d'une part a un comportement correct en accord avec la spécification et d'autre part il détecte des événements corrompus ou inconnus (provenant de l'environnement externe). A partir de ce testeur, il est possible de dériver des tests abstraits. Plusieurs étapes sont considérées pour la construction du testeur de robustesse : construction du graphe de refus de la spécification pour avoir l'ensemble des refus associé à chaque état (détermination des situations de blocage), complétion du graphe de refus avec des traces de fautes sur chaque état et ajout d'un événement générique θ représentant les événements inconnus ou corrompus (le testeur passe dans un état de refus qui représente une situation dégradée et revient soit dans le l'état de départ avec une transition non observable, soit dans un état de reprise identifié) et enfin prise en compte des actions du graphe de refus étendu afin de générer les tests abstraits.

Construction du graphe de refus

Un graphe de refus est défini par RG : (Q, A, Ref, T, q_0) où Q est un ensemble fini d'états, A un ensemble fini d'actions, Ref : S $\rightarrow$ *P(P(L))* l'ensemble des refus , avec L le langage associé aux actions, T : l'ensemble des transitions $\subseteq$ (Q X A X Q), q_0 l'état initial.

Ajout de traces de fautes

Une trace de fautes est une séquence d'actions alternées avec les ensembles de refus : f = A_0. a_0. A_1. a_1… A_n où n est un entier positif, $a_i \in$ L (langage des actions), a_i est une action qui peut être exécutée dans l'état s_i et A_i est l'ensemble de refus de l'état s_i en exécutant a_i .

rf.traces (M) est l'ensemble de toutes les traces de fautes du système.

Construction du testeur de robustesse

Le graphe de refus est complété avec les traces de fautes et pour chaque sous ensemble de l'ensemble de refus de chaque état les événements θ (autres actions de l'alphabet d'entrée) sont ajoutés. Ainsi l'alphabet des entrées sera : L $\cup$ {θ }. Pour continuer le test, une action inobservable est ajoutée pour retourner dans un état du mode nominal.

La dernière étape est l'extraction des tests abstraits du graphe de refus étendu. Un calcul de couverture de tests est possible. Des hypothèses d'uniformité sur les domaines des paramètres doit être utilisée pour réduire le nombre de tests, ainsi que des considérations sur la contrôlabilité et l'observabilité lors de la mise en œuvre des tests.

3.4 Prospective LRI

Les solutions que nous proposons sont applicables dans le cas où l'on sait spécifier les défaillances et les mécanismes de tolérances aux fautes en se basant sur les mêmes modèles que pour le test de conformité.

On pourrait penser que dès l'instant où l'on a spécifié les mécanismes de tolérance aux fautes ou de passage à des modes dégradés et de retour au fonctionnement normal, le test de robustesse peut se ramener à du test de conformité. Dans le cas, très peu fréquent en pratique, où l'on peut être exhaustif, c'est vrai. Cependant le problème n'est pas aussi simple dans le cas réaliste où l'on doit sélectionner un ensemble fini, voire même petit, dans un jeu de test exhaustif de taille prohibitive. Il faut développer des méthodes de sélection adaptées, et étudier leur couplage à des techniques d'injection de fautes.

Il existe dans la littérature plusieurs spécifications formelles de protocoles tolérants aux fautes [1, 9, 10, etc] qui peuvent servir de base à une telle étude. Dans ces exemples, l'application des méthodes de test de conformité classiques pose les problèmes suivants:

- fréquence de cas de défaillances: la couverture systématique de toutes les traces possibles (ou de toutes les transitions possibles) amène à considérer des défaillances à tout moment dans le système ce qui n'est certainement pas intéressant car il y a des ensembles d'états où les conséquences d'une

défaillance sont identiques. Au contraire, il existe des scénarii intéressants dans lesquels on a une succession de défaillances et de récupérations [8] et qui ne sont pas couverts par les stratégies classiques de conformité.

- défaillances non contrôlables par les testeurs: la notion classique du test est basée sur la mise en parallèle du système sous test avec le testeur; cette composition permet le contrôle et l'observation de l'exécution du système en présence de données normales. Mais les défaillances ne peuvent pas être provoquées par des messages du testeur tel qu'il est dérivé dans le cadre du test de conformité. Il faut introduire des mécanismes d'injection de fautes.
- non déterminisme déséquilibré des défaillances: l'occurrence de défaillances est hautement non déterministe et heureusement est de très faible probabilité. Or les méthodes de test de conformité de systèmes non déterministes font toutes l'hypothèse d'un non déterminisme équilibré.

Nous proposons de résoudre ces trois problèmes de la manière suivante:

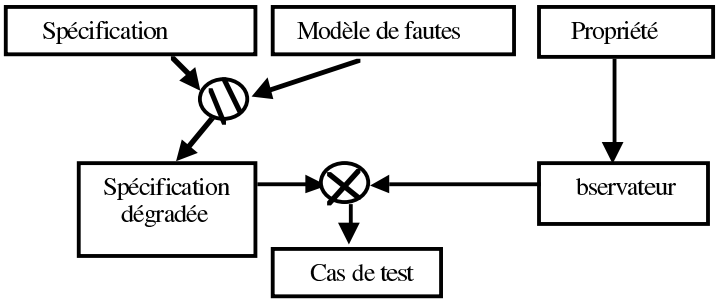
En ce qui concerne la fréquence des cas de défaillances, on peut introduire de nouvelles sortes d'hypothèses d'uniformité pour définir des classes d'équivalence d'états. Dans le test de conformité ces hypothèses définissent des classes d'équivalence de données [4]. L'analyse de la spécification et plus particulièrement des dépendances entre messages peut aider à définir de telles classes d'équivalence. On pourrait aussi spécifier des situations où les défaillances sont les plus plausibles ce qui revient à se donner des objectifs de test spécifiques à la robustesse.

En ce qui concerne le contrôle des défaillances, on pourrait modifier la notion de testeur pour que celui-ci contrôle l'injection de fautes. Cependant, il semble intéressant de conserver une séparation entre fautes et données comme c'est fait classiquement par les environnements d'injection de fautes. Cela mène à un environnement de test où l'implémentation et le testeur se synchronisent avec un injecteur de fautes, et à des méthodes de dérivation de tests qui produisent des couples testeur-injecteur de fautes.

3.5 Prospective Vérimag

Dans cette action Vérimag s'intéresse à la génération automatique de tests de robustesse. Le *critère de robustesse* est vu ici comme une propriété externe P : une implémentation est dite *robuste* si et seulement si elle satisfait P en présence d'aléas. Une différence importante entre cette approche et les techniques "classiques" de test de conformité porte sur la "correction" de l'implémentation qui est établie ici par rapport à une propriété et un modèle de fautes (et non par rapport à la spécification initiale). Elle nécessite donc plusieurs éléments :

- S une *spécification* du comportement attendu de l'implémentation dans son environnement nominal (hors aléas). S peut être décrite par exemple par un ensemble de systèmes de transitions étendus communicants. Cette spécification servira de référence pour la synthèse des cas de test, mais il n'est pas nécessaire que l'implémentation lui soit conforme.
- MF un *modèle de fautes* décrivant l'ensemble des aléas considérés, aussi bien ceux dus à l'environnement (entrées incorrectes, stress, valeurs de paramètres hors domaine, etc.) que ceux dus à des défaillances possibles de composants internes à l'implémentation (pannes, erreurs de communication, etc.). Divers formalismes peuvent être envisagés pour décrire ce modèle de fautes en fonction de l'application considérée, mais il doit pouvoir se ramener à un ensemble de *mutations* de S (modification de valeurs de timer, de la topologie des communications, des valeurs échangées, etc.).
- P une *propriété de robustesse*, définissant le comportement acceptable de l'implémentation en présence d'aléas. En pratique P peut caractériser soit un invariant du système, soit une propriété de progrès ("l'implémentation revient toujours en mode nominal après être entré en mode dégradé"). Nous avons considéré ici P comme une propriété linéaire, décrivant l'ensemble de séquences d'exécutions robustes (finies ou infinies) de l'implémentation.



Concrètement, la technique de génération de tests envisagée peut alors être résumée de la manière suivante (c.f. figure ci-dessus) :

- génération d'une *spécification dégradée* Sd par mutation de S avec le modèle de faute MF.
- génération d'un *observateur* O (exprimé par un automate de Rabin [18]) et décrivant l'ensemble des séquences de ¬P. Cet observateur identifie les séquences non robustes de Sd.
- synthèse des cas de tests à partir de Sd et O : les séquences d'exécution non robustes sont extraites de Sd et transformées en cas de test (modulo les critères de contrôle et d'observation considérés). Un verdict de robustesse leur est alors associé.

4 Conclusion

La présentation du problème du test de robustesse a permis de l'identifier clairement et en particulier de le positionner par rapport au test de conformité. Ce type de test est particulièrement crucial pour tous les systèmes embarqués ou pour tout système devant tolérer des fautes. Les équipes de recherche impliquées dans cette AS ont réussi à définir une problématique scientifique nouvelle et des visions complémentaires, issues de discussions scientifiques riches, pour résoudre le problème du test de robustesse. Un certain nombre d'axes de prospectives ont été identifiés et doivent maintenant être confrontés à des exemples concrets, permettant d'illustrer les problèmes inhérents au test de robustesse : complexité combinatoire, modélisation totale ou partielle, problèmes d'observabilité et de commandabilité lors de la mise en œuvre des tests. Par ailleurs il est indispensable de souligner que les besoins industriels sont importants dans la mesure où la sûreté de fonctionnement d'un système est un enjeu majeur : la problématique du test de robustesse a été intégrée dans plusieurs projets proposés au sixième PCRD auxquels participeraient des membres de cette AS.

Bibliographie

[1] J. Arlat, M. Aguera, Y. Crouzet, J. Fabre, E. Martins and D. Powell, Experimental Evaluation of the Fault Tolerance of an Atomic Multicast Protocol, IEEE Trans. on Reliability, 1990, pp.455-467", vol. 39, n°4

[2] J. Arlat, J.-C. Fabre, M. Rodríguez and F. Salles, "Dependability of COTS Microkernel-Based Systems", *IEEE Transactions on Computers*, 51 (2), pp.138-163, February 2002.

[3] O. Abdellatif-Kaddour, P. Thévenod-Fosse, H. Waeselynck, "A stepwise strategy for property-oriented testing", LAAS report no. 02351, September 2002. To appear in *Proc. 18th ACM Symposium on Applied Computing (SAC'2003),* to be held in Melbourne, Florida, USA, March 2003.

[4] G. Bernot and M.-C. Gaudel and B. Marre, "Software testing based on formal specifications: a theory and a tool", *Software Engineering Journal*, pp.387-405, 1991.

[5] D.M. Blough, T. Torii, "Fault-Injection-Based Testing of Fault-Tolerant Algorithms in Message Passing Parallel Computers", *Proc. 27th Int. Symp. on Fault-Tolerant Computing Systems (FTCS-27)*, pp. 258-267, Seattle, WA, USA, IEEE CS Press, 1997.

[6] P. Cheynet, B. Nicolescu, R. Velazco, M. Rebaudengo, M.S. Reorda, M. Violante, "Experimentally Evaluating an Automatic Approach for Generating Safety-Critical Software with respect to Transient Errors", *IEEE Transactions on Nuclear Science*, vol. 47, no. 6, pp. 2231-2236, 2000.

[7] S. Dawson, F. Jahanian, T. Mitton and T.-L. Tung, "Testing of Fault-Tolerant and Real-Time Distributed Systems via protocol Fault Injection", in *Proc. 26th Int. Symp. on Fault-Tolerant Computing (FTCS-26),* (Sendai, Japan), pp.404-414, IEEE CS Press, 1996.

[8] S.-D. Gouraud, G. Lestiennes, Testing Fault Tolerant Protocols described by Formal Specifications, http://www.lri.fr/~gouraud/papiers/GouLes02.ps.

[9] A. Helmy, D. Estrin and S. Gupta, "Fault-oriented Test Generation for Multicast Routing Protocol Design", *Proceedings of Formal Description Techniques & Protocol Specification, Testing, and Verification (FORTE/PSTV)*, IFIP, Kluwer Academic Publication, Paris, France, pp.93-109, 1998.

[10] P. James, M. Endler and M.-C. Gaudel, "Development of an Atomic-Broadcast Protocol Using LOTOS", *Software Practice & Experience*, pp.699-719, vol. 29 No. 8, 1999.

[11] C. Jard, T. Jéron., "TGV: theory, principles and algorithmes", *Proc. Sixth World Conference on Integrated Design & Process Technology (IDPT'02)*, Pasadena, California, USA, June 2002.

[12] P. Koopman and J. DeVale, "Comparing the Robustness of POSIX Operating Systems", in *Proc. 29th Int. Symp. on Fault-Tolerant Computing (FTCS-29),* (Madison, WI, USA), pp.30-37, IEEE CS Press, 1999.

[13] *Guide de la Sûreté de Fonctionnement*, sous la direction de J-C. Laprie, auteurs : J. Arlat, J-P. Blanquart, A. Costes, Y. Crouzet, Y. Deswarte, J-C. Fabre, H. Guillermain, M. Kaâniche, K. Kanoun, J-C. Laprie, C. Mazet, D. Powell, C. Rabéjac, P. Thévenod, Cépaduès Editions, ISBN 2-85428-382-1, 1996.

[14] G. Lussier, H. Waeselynck, "Informal Proof Analysis towards Testing Enhancement", in *Proc. 13th Int. Symp. on Software Reliability Engineering (ISSRE'2002)*, (Annapolis, USA), pp. 27-38, IEEE CS Press, 2002.

[15] E. Marsden, J.-C. Fabre and J. Arlat, "Dependability of CORBA Systems: Service Characterization by Fault Injection", in *Proc. 21st Int. Symposium on Reliable Distributed Systems (SRDS-2002),* (Osaka, Japan), pp.276-285, IEEE CS Press, 2002.

[16] B.P. Miller, L. Fredriksen and B. So, "An Empirical Study of the Reliability of Unix Utilities", *CACM*, 33 (12), pp. 32-44, 1991.

[17] A. Mukherjee, D.P. Siewiorek, "Measuring Software Dependability by Robustness Benchmarking", *IEEE Trans. on Software Engineering*, vol. 23, no. 6, pp. 366-378, 1997.

[18] M.O. Rabin, " Decidability of Second Order Theories and Automata on Infinite Trees", *Transactions of the American Mathematical Society*, Vol. 141, pp 1-35, 1969.

[19] P. J. Ramadge,W. M. Wonham, "The Control of Discrete Event Systems", *Proceedings of the IEEE*; Special issue on Dynamics of Discrete Event Systems, vol 77, no 1, pp 81-98, 1989.

[20] A.C. Schultz, J.J. Grefenstette and K.A. De Jong, "Learning to Break Things: Adaptive Testing of Intelligent Controllers", in *Handbook on Evolutionary Computation*, Chapter G3.5, IOP Publishing Ltd. and Oxford University Press, 1995.

[21] T. K. Tsai, M.-C. Hsueh, Z. Kalbarczyk and R. K. Iyer, "Stress-Based and Path-Based Fault Injection", *IEEE Transactions on Computers*, 48 (11), pp.1183-1201, November 1999.

[22] J. Voas and G. McGraw, *Software Fault Injection: Inoculating Programs Against Errors*, John Wiley & Sons, 1998.

[23] H. Waeselynck, P. Thévenod-Fosse, "A case study in statistical testing of reusable concurrent objects", *Proc. 3rd European Dependable Computing Conference (EDDC-3),* LNCS no. 1667, Springer Verlag, pp. 401-418, Prague, République Tchèque, septembre 1999.

[24] C. Weissman, "Security Penetration Testing Guidelines - Chapter 10", in *Navy Handbook for the Computer Security Certification of Trusted Systems* Technical Memorandum 5540:082A, Naval Research Laboratory, USA, 1995.