INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Projet VerTeCs

Modèles et techniques de VÉRification appliquées au TEst et au Contrôle de Systèmes réactifs

Rennes

THÈME 1C

Table des matières

1. Composition de l'équipe

Le projet VerTeCs est un nouveau projet créé en décembre 2001. Ses personnels sont issus des anciens projets Pampa et EP-ATR ainsi que des nouveaux recrutements opérés en 2001 et 2002.

Responsable scientifique
Thierry Jéron [CR]

Assistante de projet
Catherine Godest [TR CNRS, jusqu'à novembre 2002]
Maud Bellanger [CDD, de novembre 2001 à mars 2002]
Françoise Legeard [CDD, de mars 2002 à juin 2002]
Lydie Letort [TR INRIA, depuis novembre 2002]

Personnel Inria
Bertrand Jeannet [CR]
Hervé Marchand [CR]
Vlad Rusu [CR]

Invité
Ahmed Khoumsi [Professeur Univ. Sherbrooke (Canada), accueilli sur un poste de spécialiste international INRIA, depuis octobre 2002]

Chercheur post-doctorant
Wendelin Serwe [depuis le 1er septembre 2002, à temps partiel dans le projet]

Ingénieurs experts
Simon Pickin [jusqu'à fin octobre, à 50% dans le projet]
Erwan Demairy [depuis le 1er septembre 2002]

Ingénieur associé
François-Xavier Ponscarme [IA, depuis le 1er octobre 2002]

Chercheurs doctorants
Benoît Gaudin [allocataire MESR]
Valery Tschaen [allocataire MESR]
Elena Zinovieva [boursière INRIA]

2. Présentation et objectifs généraux

L'objectif scientifique de notre projet est d'améliorer la fiabilité des systèmes réactifs et critiques, en fournissant à l'ingénieur logiciel des méthodes et des outils de génération de tests et de synthèse de contrôleurs.

Par systèmes réactifs on entend des logiciels qui interagissent avec leur environnement. Citons par exemple les protocoles de télécommunication, les logiciels embarqués, les systèmes de contrôle/commande, etc. Ces systèmes réactifs sont aussi souvent des systèmes critiques, au sens où l'occurrence d'erreurs au cours du fonctionnement du logiciel peut avoir des conséquences graves du point de vue économique ou de la sécurité des personnes.

La correction fonctionnelle de ce type de logiciel est donc primordiale. Cette correction peut être envisagée dès la conception et les techniques de **vérification** diverses (preuve, model-checking, analyse statique, etc) peuvent être très utiles dans cette phase. Toutes ces techniques ont pour but d'assurer qu'un logiciel ou, plus souvent, une spécification d'un logiciel, satisfait certaines propriétés fondamentales de bon fonctionnement. Si des erreurs sont découvertes, ces techniques permettent en général de les diagnostiquer.

Une solution alternative à la vérification est celle du **contrôle** : plutôt que de vérifier la correction du logiciel, on l'assure en interdisant les comportements qui risqueraient de sortir de l'ensemble des comportements acceptables. Cette restriction des comportements s'effectue en interdisant le tir de certaines actions dites contrôlables, sans toutefois pouvoir interdire les actions incontrôlables. La **synthèse de contrôleurs** consiste

à construire automatiquement un contrôleur à partir d'une spécification du système et des propriétés de correction à assurer.

Enfin, même après un effort conséquent de vérification de la conception, le **test** de l'implémentation réelle est toujours nécessaire. De même le contrôleur produit devra aussi être testé. En effet, la vérification et la synthèse de contrôleurs s'effectuent sur un modèle du système qui, même s'il se veut proche du système réel, n'est jamais identique à celui-ci. L'activité de modélisation impose d'abstraire le comportement du système pour n'en considérer que les aspects essentiels et de faire des hypothèses sur l'environnement d'utilisation. Par conséquent, il se peut que certains comportements, non considérés dans le modèle pour la vérification ou la synthèse de contrôleurs, provoquent des erreurs lors de l'exécution du système réel. La vérification (ou le contrôle), même si elle est exhaustive, n'est donc exhaustive que sur un modèle du système, d'où le besoin de valider le système réel par des tests.

3. Fondements scientifiques

3.1. Concepts et outils primordiaux

Mots clés : *systèmes de transitions étiquetés*, *symboliques*, *à entrées-sorties*, *à événements contrôlables/incontrôlables*, *à fonction de transition implicite*.

La structure mathématique qui fédère nos travaux de recherche en vérification, génération de tests, et synthèse de contrôleurs est celle des systèmes de transitions étiquetés et de ses diverses extensions : symboliques, à entrées-sorties, à événements contrôlables/incontrôlables, ou à fonction de transition implicite. Nous nous appuyons sur des théories bien établies telles que la théorie du test de conformité, la synthèse de contrôleurs, l'interprétation abstraite, et la preuve de théorèmes. Les algorithmes utilisés combinent les algorithmes classiques de parcours de graphes, les calculs dans des domaines abstraits (les polyèdres, par exemple) et les algorithmes de décision logique (e.g., satisfiabilité en arithmétique de Presburger).

Un système de transition (LTS pour *Labelled Transition System*) est un graphe orienté dont les arêtes, appelées transitions, sont étiquetées par une lettre prise dans un alphabet d'événements. Les sommets de ce graphe sont appelés états.

$M = (Q, A, T, q_{init})$

Avec : Q ensemble des états, q_{init} l'état initial, A l'alphabet des événements, $T \subset Q \times A \times Q$ la relation de transition.

Il est usuel de parler d'automate d'états finis pour désigner un système de transitions étiqueté dont l'ensemble des états et celui des événements sont finis. Il s'agit en fait du modèle de machine le plus simple que l'on puisse imaginer. Nous employons les LTS pour modéliser des systèmes réactifs. Dans ce cadre les événements représentent les interactions du système avec son environnement. L'alphabet des événements sera partitionné pour préciser les modèles ; par exemple, en test, on parle de système de transitions *entrées-sorties* ou de IOLTS (*input-output* LTS) en distinguant l'orientation des actions (de et vers l'environnement). Dans le cadre de la synthèse de contrôleurs, on va également distinguer des événements contrôlables et incontrôlables, observables et inobservables.

Ces systèmes de transitions sont souvent obtenus à partir de spécifications décrites dans des modèles de plus haut niveau, qui manipulent symboliquement des données typées : booléens, entiers, files FIFO, etc. Les LTS ainsi obtenus constituent alors la sémantique de ces modèles. Ces derniers conservent les notions d'états (appelés plutôt *localités*) et de transitions, qui sont alors seulement des moyens pour décrire un flot de contrôle. Les transitions sont assorties d'une *garde* (une condition booléenne sur les variables) qui conditionne le tirage de la transition, et une liste d'*affectations* qui définissent les nouvelles valeurs des variables lorsque la transition a été tirée. On parle alors de *systèmes de transitions symboliques* (STS).

Un type particulier de systèmes de transitions symboliques que nous utilisons dans la génération de tests symboliques est celui des IOSTS (*input-output* STS). Outre la distinction entre entrées et sorties, les données

manipulées sont partitionnées en *variables, paramètres, et messages*, chacun jouant un rôle particulier dans le modèle.

Avec les (IO)STS la distinction entre contrôle et données est encore présente ; cette distinction devient artificielle lorsqu'on modélise des applications de type flot de données. Dans ce dernier cas, l'état est un vecteur contenant les valeurs courantes des variables, et la relation de transition est une fonction qui définit l'état suivant en fonction de l'état courant et éventuellement d'une entrée. Nos travaux d'interprétation abstraite visent à retrouver à partir d'une telle description flot de données une structure de contrôle optimale pour la vérification.

Nous appuyons nos recherches sur des théories bien établies : la théorie du test de conformité, la synthèse de contrôleurs, et l'interprétation abstraite. En vérification, nous utilisons également la preuve de théorèmes.

Les algorithmes que nous utilisons proviennent de ces théories :

- les algorithmes de parcours de graphes (en largeur, en profondeur, algorithme de Tarjan pour la détection de composantes fortement connexes...). Nous utilisons ces algorithmes autant en vérification qu'en synthèse de tests et de contrôleurs.
- les algorithmes d'interprétation abstraite, spécifiquement, dans le domaine abstrait des polyèdres (par exemple, l'algorithme de Chernikova pour le calcul des formes duales). Nous utilisons de tels algorithmes en vérification et en synthèse de tests.
- les algorithmes de décision logique, telle la satisfiabilité de formules en arithmétique de Presburger. Nous utilisons ces algorithmes dans la synthèse et l'exécution des tests symboliques.

3.2. Génération automatique de tests

Nous nous intéressons principalement au test dit de *conformité*. Le test de conformité est un type de test qui s'intéresse à la correction fonctionnelle d'une implantation d'un système vis à vis d'une spécification de référence. C'est un test de type « boîte noire », car le code de l'implantation n'est en général pas connu.

Le test de conformité consiste à faire interagir l'implantation sous test (*IUT* pour *implementation under test*) avec son environnement. Cet environnement, qui est constitué par un ou plusieurs processus ou utilisateurs, est simulé par un ou un ensemble de testeurs. Ces testeurs exécutent des tests élémentaires appelés *cas de test* qui consistent en une suite d'interactions. Cette suite d'interactions contrôle l'IUT par des stimuli (des envois de messages, des appels de fonctions, de méthodes, etc) et observe ses réactions (messages reçus, résultats d'appels, etc). Les comportements de l'IUT sont contrôlés et observés à travers des interfaces (appelées *PCO* pour *Points de Contrôle et d'Observation* dans le monde télécom). Généralement, chaque *cas de test* a pour objectif le test d'une fonctionnalité précise, décrite dans ce qu'on appelle un *objectif de test*.

Le test de conformité est particulièrement utilisé dans le domaine des protocoles de télécommunication. Dans ce cadre une norme a été décrite, ISO 9646 [48] qui en décrit les ingrédients principaux. Mais le test de conformité n'est pas confiné à ce cadre, et le même type de test peut être utilisé pour des systèmes réactifs quelconques, et en particulier pour les systèmes critiques.

3.2.1. Techniques de génération.

Essentiellement, deux types de méthodes coexistent. Ces méthodes diffèrent sur le modèle utilisé pour représenter les comportements des spécifications de systèmes et par conséquent sur les techniques mises en œuvre.

La première famille, la plus ancienne, est celle des méthodes fondées sur les automates (voir la description de l'état de l'art [52] par exemple). Elle est issue des méthodes de test de circuits et des problèmes de reconnaissance de machines [55][43]. Sous certaines hypothèses à la fois sur la spécification et l'implantation, ces techniques permettent de tester exhaustivement l'implantation pour toutes les fautes dans un certain modèle de faute (caractérisant le type de fautes observables). Les algorithmes cherchent à construire une unique séquence, la plus courte possible, qui permette de montrer l'équivalence entre les comportements de l'implantation et ceux de la spécification. L'atout de ces méthodes est leur exhaustivité. Mais cette exhaustivité est surtout théorique. En effet, si les hypothèses faites sur la spécification sont parfois acceptables

(contrôlabilité, déterminisme, minimalité) elles sont invérifiables sur l'implantation. De plus les algorithmes de génération sont assez coûteux, ce qui interdit de les utiliser sur des systèmes de taille réelle. Ces techniques semblent donc cantonnées à traiter des spécifications peu complexes, en particulier dans le domaine du matériel.

La deuxième famille est fondée sur les systèmes de transitions (LTS pour *Labelled transition system*) et est issue des problèmes d'équivalence et de préordre de test dans les algèbres de processus [40][30][58][35]. Au départ dédiées à la formalisation du problème du test, ces théories ont ensuite évolué vers des algorithmes et des outils de génération de tests [57][64][41]. Les modèles utilisés permettent de représenter les comportements de systèmes réels, et les hypothèses faites à la fois sur la spécification et sur l'implantation sont réduites au minimum. Au lieu de chercher à montrer l'équivalence de comportements comme dans les techniques évoquées plus haut, on cherche à montrer une inclusion de (certains) comportements (un préordre ou une relation de conformité). Les modèles et techniques permettent aisément de prendre en compte le non-déterminisme et l'observation partielle. Des algorithmes efficaces existent maintenant [2] et ont permis de traiter des études de cas de taille industrielle [42],[1].

3.3. Synthèse de contrôleurs

La théorie du contrôle a été développée dans le cadre des systèmes à espaces d'états et de signaux continus. Le système de contrôle-commande est élaboré en utilisant des méthodes de construction garantissant *a priori* les propriétés attendues. La phase de validation est ainsi restreinte aux seules propriétés non garanties par construction et, expérimentalement, à l'évaluation des écarts dus aux erreurs ou approximations dans la modélisation.

La même approche a été adoptée dans le domaine des systèmes à événements discrets. Ici, deux approches différentes du contrôle coexistent.

- dans la première, le modèle physique est représenté par un langage dont l'alphabet représente les différentes actions possibles ; un mot de ce langage représente alors une séquence d'actions correspondant à une trajectoire souhaitée du système. Dans cette approche, l'action du contrôleur a pour effet d'interdire, à différents moments, un certain nombre de lettres de l'alphabet, appelées événements contrôlables [60]. Cette approche a notamment été mise en œuvre sur des automates [60] et des réseaux de Petri [47][46].
- l'autre approche [31] [5], à notre sens plus proche de la réalité industrielle et qui s'approche en esprit des processus continus, consiste à fournir sur certains des ports d'entrée du système (dits événements contrôlables), les événements induisant un comportement souhaité du système vis à vis des objectifs de contrôle (on parlera alors de contrôle entrées/ sorties).

Dans les deux cas, le contrôle revient à ajouter à un système non contrôlé des contraintes supplémentaires induisant une réduction du comportement du système à un comportement souhaité.

Ce comportement souhaité est obtenu à partir de la spécification d'un objectif de contrôle (dans les situations concrètes, on est amené a composer plusieurs objectifs pour obtenir le comportement souhaité[59]). Là encore, il existe plusieurs formalismes pour représenter ces objectifs. Dans l'approche Ramadge & Wonham, ces objectifs s'exprimaient comme des langages (inclus dans le langage de la spécification). L'idée du contrôle était alors de dégager de ce langage de contrôle, le langage le plus permissif et contrôlable (seuls les événements contrôlables peuvent être interdits par le contrôleur). En d'autres termes, ce langage correspond au comportement le plus permissif que le système peut avoir s'il veut à la fois respecter l'objectif de contrôle et le critère de contrôlabilité. A partir de ce langage, un contrôleur peut alors être vu comme une fonction qui, en fonction de l'histoire du système (i.e., la trajectoire que le système a emprunté jusque là), va envoyer au système l'ensemble des événements contrôlables qui doivent être interdits pour rester dans le cadre du comportement contrôlé. Cette théorie a par la suite été étendue au cadre de l'observation partielle [34] sur le système à contrôler, et utilisée pour la synthèse de contrôleurs décentralisés [65].

Commande optimale : Pour le contrôle, certains objectifs de contrôle (l'atteignabilité par exemple) peuvent se traduire par la question simple : est-il possible de « piloter » le système considéré vers un ensemble donné d'états en garantissant certaines propriétés tout au long de la trajectoire ? La théorie du contrôle permet de synthétiser le contrôleur le plus permissif pilotant le système de l'ensemble de départ vers l'ensemble d'arrivée (ce qui en soi est un critère d'optimalité qui se rapproche de celui décrit dans le cadre du test : maximum de liberté vs minimum de contrôlabilité). A ce niveau, une deuxième question peut alors se poser : quelle est la meilleur façon d'y arriver, la plus rapide, la moins coûteuse, la plus sûre ... ? Ce type de problème s'appelle problème de la commande optimale. Le champ d'action de la commande optimale est extrêmement vaste. Il couvre le contrôle de tous les systèmes dynamiques où une performance optimale est souhaitée. Le problème intrinsèque de la commande optimale est le choix du critère sur lequel l'optimalité va porter. Dans certains cas, ce critère est directement imposé par le système. Dans d'autres, au contraire, le choix du critère est (totalement) arbitraire et dépend fortement de la personne qui pose le problème. Cette notion a notamment été étudiée dans [63] ; [4] et étendue au problème de l'observation partielle dans [12]. Intuitivement, l'idée de la commande optimale est de calculer un contrôleur qui pilote un système de manière à forcer celui-ci à achever une tâche en minimisant un certain critère de performance. Dans cette approche, des coûts sont affectés à chaque événement. À partir de cette fonction de coûts, il est possible de définir le coût d'une trajectoire comme étant la somme des coûts des événements qui interviennent dans cette trajectoire. Le but du contrôleur est alors de restreindre le comportement du système de manière à ce que celui-ci n'emprunte que des trajectoires de coûts minimaux. D'autres critères d'optimalité portent plus sur les moyens à utiliser pour atteindre un objectif logique, plutôt que sur le but à atteindre [6]. Ces objectifs s'expriment en ordonnant les états du système suivant une certaine relation d'ordre. Dans ce cas, le contrôle revient à choisir, pour un état donné, la commande faisant évoluer le système vers l'état considéré comme étant le meilleur pour la relation d'ordre. Nous parlerons alors d'objectif d'optimalité statique pour les objectifs de contrôle exprimés comme une relation d'ordre entre états, et de commande optimale statique. Les objectifs d'optimalité statiques peuvent également être composés avec des objectifs de contrôle logiques de manière à obtenir potentiellement des lois de contrôle (expression des commandes en fonction des entrées non contrôlables et des états).

3.4. Interprétation Abstraite et prise en compte des données

Les techniques énumératives décrites ci-dessus, qui s'appliquent aux LTS, sont déjà relativement matures. Il apparaît donc naturel d'étendre ces techniques aux IOSTS ou aux applications flot de données, qui manipulent des variables à valeur dans des domaines potentiellement infinis.

Les techniques de génération de tests ou de synthèse de contrôleurs que nous développons nécessitent de résoudre des questions d'accessibilité ou de coaccessibilités d'états (on dit d'un état s qu'il est *accessible depuis un état initial* s_i si une exécution démarrant dans s_i peut mener à s, et qu'il est *coaccessible depuis un état final* s_f si une exécution démarrant dans s peut mener à s_f). Ces questions peuvent se décider par des calculs de point fixes (mais aussi par des méthodes déductives). D'un point de vue algorithmique, ces calculs de points fixes représentent le noyau dur des méthodes de génération de test ou de contrôleur développée dans l'équipe.

Le grand changement induit par la prise en compte des données est que les points fixes en question deviennet incalculables. On contourne cet obstacle en faisant des approximations et en se plaçant dans le cadre théorique de l'Interprétation Abstraite [38].

D'une manière très générale, l'Interprétation Abstraite est une théorie du calcul approché de points fixes appliqué à l'analyse de programme. La plupart des analyses, entre autres les analyses d'accessibilité, se réduisent à la résolution d'une équation de point fixe

$$x = F(x), x \in C$$

où C est un domaine ordonné. Dans le cas de l'analyse d'accessibilité, en notant S l'espace d'états du programme considéré, C sera le domaine $\wp(S)$ des ensembles d'états du programme, ordonné par inclusion,

F est la fonction de *« états successeurs »* définie par le programme, et l'on cherche à savoir quels sont les états accessibles au cours d'une exécution quelconque.

La résolution exacte de ce type d'équations n'est généralement pas possible, pour des raisons d'indécidabilité (ou de complexité). Les principes de base de l'interprétation abstraite sont :

1. de substituer au *domaine concret* C un *domaine abstrait* A plus simple (approximation statique), et de transposer l'équation de point fixe dans le domaine abstrait : on se ramène à la résolution d'une équation $y = G(y), y \in A$;
2. d'utiliser un *opérateur d'élargissement* pour faire converger en un nombre fini d'étapes le calcul itératif du (plus petit) point fixe de G (approximation dynamique).

Les approximations sont conservatives, de telle sorte que le résultat obtenu est une sur-approximation du résultat exact. Ces deux principes s'illustrent bien par l'*analyse d'intervalle* [37], qui consiste à associer à chaque variable numérique d'un programme un (sur)ensemble de valeurs accessibles sous la forme d'un intervalle :

1. On substitue au domaine concret $\wp(R^n)$ induit par les variables numériques le domaine abstrait $(I_R)^n$, où I_R dénote l'ensemble des intervalles sur les réels ; un ensemble de valeurs pour une variable sera alors représenté par le plus petit intervalle le contenant ;
2. Un calcul itératif de point fixe sur ce domaine peut ne pas converger : il est en effet facile de générer une suite infinie d'intervalles strictement emboîtés. L'opérateur d'élargissement « standard » consiste ici à extrapoler par $+\infty$ la borne supérieure d'un intervalle si celle-ci ne se stabilise pas en un nombre donné d'étapes (et symétriquement pour la borne inférieure).

Dans ce petit exemple, l'espace d'état à abstraire $\wp(R^n)$ a une structure particulièrement simple. Les choses se compliquent lorsque les données sont de différents types (entre autre, Booléens, numériques, tableaux) et qu'il faut de plus être capable de conserver des *relations* entre des valeurs de différents types.

Les programmes effectuant des allocations dynamiques d'objets ont un espace d'état de structure encore plus complexe. Des travaux cependant existent dans ce domaine pour représenter de manière approchée le tas mémoire et les pointeurs entre cellules par des graphes (*shape analysis* [62][61]), mais les valeurs des cellules mémoires sont généralement ignorées.

De même, les programmes avec appels de procédures récursifs, passage de paramètres et variables locales sont délicats à analyser avec précision (i.e., avec prise en compte des données). La difficulté est entre autre d'abstraire les piles d'exécution, qui ont une structure complexe, en particulier lorsque les paramètres peuvent être passés par référence [32].

3.5. Preuve de théorèmes

Nous utilisons la preuve de théorèmes et plus particulièrement l'assistant de preuves PVS [56] comme support à la vérification. La vérification de propriétés de sûreté (notamment, des propriétés d'invariance) à l'aide de cet outil consiste en une démarche systématique de recherche et de renforcement de prédicats inductifs. Un prédicat est inductif sur un système de transitions étendu si le prédicat est vrai initialement et s'il est préservé par l'exécution de chaque transition du système. Une stratégie PVS permet de vérifier si un prédicat est inductif ; si c'est le cas, alors la propriété d'invariance décrite par le prédicat est prouvée, sinon, PVS suggère des propriétés auxiliaires qui permettent de prouver la propriété initiale. L'utilisateur se contente d'interpréter les informations que PVS retourne en cas d'échec. Une description détaillée de l'approche est donnée dans [8]. Une autre approche [24] consiste à coder l'inductivité d'un prédicat par la satisfiabilité d'une formule d'une certaine logique, l'arithmétique de Presburger avec symboles de fonctions, et de tester la satisfiabilité de la formule à l'aide de procédures de décision, ou d'heuristiques, dans la logique en question.

4. Domaines d'application

4.1. Panorama

Mots clés : *Télécommunications*, *logiciel embarqué*, *transports*, *cartes à puces*.

Le projet VerTeCs étudie des modèles et techniques de vérification, test et contrôle les plus génériques possibles, applicables à des domaines variés. Nos travaux sont cependant appliqués principalement dans trois domaines : les télécommunications, le logiciel embarqué et les systèmes de contrôle commande.

4.2. Télécom

Le domaine des télécommunications est un domaine privilégié pour l'application de nos techniques, en particulier nos techniques de génération de tests initialement développées dans ce domaine. C'est un domaine dans lequel l'aspect normalisation est important, ce qui pour nos techniques est un avantage. Par exemple les protocoles sont normalisés ce qui fournit une base de référence pour le test de conformité. Les langages de description de ces protocoles sont aussi normalisés, puisque l'ISO et l'ITU ont normalisé trois langages de spécification formelle : Estelle, Lotos et SDL. Le processus de test de conformité est lui même normalisé par la norme ISO 9646 [48]. Cette norme définit également le langage de description de tests TTCN. Estelle et Lotos sont peu utilisés, ce qui n'est pas le cas de SDL qui sert souvent de langage de description des protocoles normalisés. L'utilisation de ces langages, en particulier SDL et TTCN, a poussé les éditeurs de logiciels à développer des outils SDL d'édition, compilation, simulation et génération de tests à partir de spécifications. Le domaine des télécoms est aussi très moteur dans le développement de la notation UML. Ce contexte est donc très favorable pour transférer les avancées de nos recherches.

4.3. Logiciel embarqué

De plus en plus de logiciels sont maintenant embarqués dans des systèmes de natures différentes. Le logiciel prend ainsi une place importante dans beaucoup de secteurs d'activité comme le transport, l'industrie, le commerce, mais aussi dans notre vie quotidienne. L'enjeu est de construire des logiciels complexes à moindre coût tout en garantissant un niveau de fiabilité dépendant de sa criticité. Ce logiciel embarqué est souvent aussi réparti pour satisfaire des contraintes d'efficacité et de fiabilité.

Le transport, et en particulier l'automobile, est un domaine où l'informatique prend une place grandissante en remplaçant peu à peu l'électronique, que ce soit au niveau navigation, information de traffic, contrôle du moteur, du freinage, confort. L'enjeu est ici de définir, au niveau européen, un processus de développement des architectures logicielles embarqués commun aux constructeurs et équipementiers, fondé sur des standards au niveau communication et composants. Ceci devrait permettre une intégration plus aisée des composants interopérables, et une meilleure réutilisabilité. Des notations à la UML gagnent aussi du terrain dans la description de telles architectures. Dans ce domaine, nous nous intéressons à la validation par le test de conformité et de robustesse.

A l'opposé en termes de taille de système, les cartes à puces et les terminaux possèdent aussi une partie logicielle de plus en plus conséquente. Bien que ces logiciels soient plus orientés vers les aspects sécurité et confidentialité, la fiabilité de ces logiciels est aussi importante. Ce domaine des cartes à puces est notre domaine privilégié pour l'application de nos techniques de génération de tests symboliques et a servi également de support à la preuve compositionelle. Nous avons beaucoup investi en particulier sur une étude de cas de porte-monnaie électronique.

4.4. Systèmes de contrôle-commande

Outre les logiciels embarqués, les systèmes de contrôle-commande s'avèrent être l'un des domaines d'applications privilégiés pour la synthèse de contrôleur et le test. Les systèmes en question contrôlent en général des installations ou des machines coûteuses (usine de production manufacturière, système robotique, etc), qui sont pour la plupart en interaction avec un environnement (incluant du personnel humain). Ce sont

donc des systèmes à sécurité critique. On a aussi des préoccupations de sûreté de la programmation, et de méthodes de validation et vérification, fondées sur des modèles formels. Dans cette optique, les apports tant du point de vue de la synthèse de contrôleurs (qui permet de dériver des contrôleurs sûrs à partir de spécifications) que du test (qui permet notamment de tester la validité de l'implantation du contrôleur obtenu) sont primordiales pour ce type d'applications.

5. Logiciels

5.1. Tgv

Participants : Thierry Jéron [correspondant], Valéry Tschaen, Erwan Demairy, Simon Pickin.

Mots clés : *test de conformité*, *TGV*, *ObjectGéode*, *CADP*, *SDL*, *Lotos*, *TTCN*, *UML*, *IF*.

TGV (Test Generation with Verification technology) est un prototype de générateur automatique de tests de conformité à partir de spécifications (SDL, LOTOS, UML). Son originalité tient dans son algorithmique à la volée. TGV est issu d'un projet commun avec le laboratoire Verimag Grenoble. Il utilise des bibliothèques de la boîte à outils CADP de Vérimag et de l'Inria Rhône-Alpes.

Les premiers développements de TGV datent de 95 et ont vu le jour lors d'un contrat financé par le STEI, auquel ont contribué Vérilog, Cap Sesa Région Rennes, le Cnet Lannion et le Celar de Bruz, VÉRIMAG et l'ancien projet Pampa de l'Irisa. La première version développée par VÉRIMAG et l'ex projet Pampa était utilisable sur les graphes d'états de spécifications SDL produits par l'outil Géode.

Depuis cette première version, TGV a été constamment amélioré du point de vue algorithmique mais aussi par l'interfaçage avec de nouveaux environnements. TGV génère maintenant des tests à la volée (sans construire complètement le graphe d'état de la spécification), à la fois sur des spécifications SDL grâce à sa connexion à ObjectGéode, sur des spécifications Lotos grâce à sa connexion à l'environnement CADP et sur des spécifications UML grâce à sa connexion à l'outil UMLAUT du projet TRISKELL et au langage IF de Verimag. TGV peut aussi fonctionner sur des graphes explicites décrits au format Aldébaran ou dans le format compressé BCG. En sortie, TGV construit un cas de test dans le format Aldébaran ou BCG ou dans le langage TTCN (*Tree and Tabular Combined Notation*).

TGV a été conçu de façon à être relativement indépendant des langages de spécification. Cette indépendance est obtenue par son architecture en couches par l'intermédiaire d'API (*Application Programming Interface*) fournissant les fonctions de parcours de systèmes de transitions. Ces différentes couches utilisent aussi des bibliothèques de stockage de graphes de CADP. La première couche, la seule qui soit spécifique au langage d'entrée, permet de connecter TGV soit à l'API du simulateur ObjectGéode pour SDL, soit à l'API fournie par le compilateur Lotos Caesar de la boîte à outils CADP, soit à une API similaire fournie par UMLAUT, soit encore aux API de parcours de graphes explicites.

L'architecture de TGV correspond à sa découpe fonctionnelle afin de permettre une grande modularité. TGV est doté de nombreuses options permettant d'affiner la génération de tests. Il utilise largement les expressions régulières dans les objectifs de test, les fichiers de masquage et de renommage. Il incorpore le calcul des temporisateurs et le calcul des postambules. Un module appelé VTS, et pouvant se substituer au module principal de TGV, permet aussi de vérifier et corriger à la volée des cas de test.

TGV est distribué gratuitement à des fins de recherche dans la boîte à outils CADP. Il fonctionne sur les systèmes Solaris, Linux et Windows NT. Il a été et est toujours utilisé par les membres du projet sur plusieurs études de cas et aussi par quelques utilisateurs extérieurs. TGV est aussi intégré à l'outil commercial TestComposer de la boîte à outils ObjectGéode (Telelogic) depuis 99 [45].

L'outil TGV a maintenant atteint un niveau de reconnaissance certain dans la communauté industrielle et académique internationale. Nous l'utilisons dans le cadre de plusieurs contrats, soit tel quel avec ses diverses interfaces langage, soit pour l'adapter à un contexte particulier d'étude. En particulier, dans le cadre du contrat européen Agedis,TGV est en cours d'adaptation pour permettre la génération de tests basée sur des critères de sélection plus puissants (voir 7.1 et 6.2.1). TGV sert également de démonstrateur d'idées pour y développer

de nouveaux algorithmes fondés sur les systèmes de transitions, pour le test et la synthèse de contrôleurs. En particulier, un outil de synthèse de contrôleurs avec observation partielle a été bâti sur la base des algorithmes de TGV.

TGV est déposé à l'APP sous le numéro IDDN.FR.001.310012.00.R.P.1997.000.2090.

5.2. Nbac

Participant : Bertrand Jeannet [correspondant].

Mots clés : *Interprétation abstraite*, *polyèdres*, *programmes réactifs*, *types booléens et numériques*.

NBAC (Numerical and Boolean Automaton Checker) est un outil d'analyse et de vérification de systèmes réactifs mis sous la forme d'équations flot de données synchrones, dans lesquels les variables sont de type booléen ou numérique (entiers, réels). NBAC effectue sur ces système des analyses d'accessibilité, de coaccessibilité, ainsi que leur combinaison, qui permettent soit de vérifier des propriétés de sûreté, soit de sélectionner certains états ou certaines exécutions d'un programme sur des critères d'accessibilité et/ou de coaccessibilité (« semantic slicing », nécessaire au test et à la synthèse). L'originalité de NBAC est de traiter de manière couplée et symbolique les aspects booléens et numériques des programmes analysés, et d'utiliser une notion très générale de structure de contrôle pour ajuster dynamiquement le compromis entre précision élevée des analyses et efficacité algorithmique.

Il a été développé au laboratoire VÉRIMAG à Grenoble, et continue d'évoluer. Il est actuellement connecté à la chaîne de compilation du langage synchrone LUSTRE, et une connexion avec l'outil STG de génération de tests symboliques est en cours. L'outil se décompose en différents modules, en particulier

1. Domaines abstraits A chaque type de données correspond un domaine abstrait. Le domaine abstrait composé, qui combine les différents types de données, est généré automatiquement. Il est ainsi très facile de changer un domaine abstrait par un autre (par ex, remplacement des polyèdres par les octogones).
2. Calcul de fonctions de transfert Les calculs d'accessibilité nécessitent de calculer des préconditions et postconditions d'ensembles d'états par des fonctions de transfert complexes. Deux techniques différentes ont été développées [10][16], qui disposent de plusieurs paramètres pour ajuster de manière très large le compromis précision/efficacité de ces calculs.
3. Analyses de point fixe Les analyses de (co)accessibilités par point fixe sur le domaine abstrait partitionné utilisent différentes techniques parues dans la littérature : itérations chaotiques, élargissement paramétré, rétrécissement, ...
4. Partitionnement dynamique Si le résultat d'une analyse se révèle trop imprécis pour en tirer une conclusion, la structure de contrôle utilisée pour l'analyse, qui correspond à un partitionnement du domaine abstrait, est automatiquement raffinée, en utilisant les informations des analyses précédentes [3][10]. Une analyse sur la nouvelle structure de contrôle permet alors d'obtenir des résultats plus précis.
5. Sortie Selon le cas NBAC retourne soit un verdict à un problème de vérification, soit une structure de contrôle réduite représentant un ensemble d'exécutions sélectionnées.

NBAC tire son originalité des points 2, 4 et 5.

5.3. Sigali

Participant : Hervé Marchand [correspondant].

Mots clés : *Vérification symbolique*, *synthèse de contrôleurs*.

SIGALI est un système de calcul formel interactif spécialisé dans l'analyse symbolique de systèmes de transitions modélisés par une relation de transition implicite. Initialement développé dans le projet EP-ATR, il est maintenant maintenu en collaboration avec le projet ESPRESSO.

SIGALI était initialement destiné à la vérification des propriétés statiques et dynamiques de programmes Signal. Depuis plusieurs années, outre les améliorations techniques (nouveau package BDD, etc), nos efforts ont porté sur l'incorporation des techniques de la synthèse de contrôleurs de systèmes à événements discrets. L'adjonction de primitives de création et de manipulation de fonctions à valeurs entières autorise également les calculs de contrôleurs pour des objectifs quantitatifs (e.g., maximisation/minimisation de fonctions de coûts sur une trajectoire du système, etc).

SIGALI est étroitement couplé avec l'environnement POLYCHRONY permettant ainsi un prototypage rapide du système en SIGNAL, le calcul du contrôleur par l'intermédiaire de SIGALI et un retour en Signal en vue de simuler le système contrôlé permettant ainsi de visualiser rapidement le résultat de la synthèse [5].

Depuis cette année, en collaboration avec Verimag Grenoble, SIGALI a été interfacé avec l'environnement MATOU[53] permettant la modélisation de systèmes réactifs par des automates de modes. De la même manière que pour POLYCHRONY, il existe un simulateur générique lié à MATOU, permettant de visualiser le résultat de la synthèse. Ce nouvel environnement a été testé sur quelques exemples dont une spécification de taille significative d'un codeur MPEG4 fournie par ST-Microelectronics.

SIGALI est un logiciel déposé à l'APP. Il peut être obtenu après signature d'une convention de mise à disposition.

5.4. Stg

Participants : Vlad Rusu [correspondant], Elena Zinovieva, François-Xavier Ponscarme, Thierry Jéron.

Mots clés : *test de conformité*, *test et vérification symboliques*.

STG (Symbolic Test Generation) [7][14] est un prototype de générateur de tests symboliques. Nous l'utilisons également comme outil d'abstraction pour la vérification. Cet outil nous sert pour l'instant de plate-forme pour implémenter les résultats de nos recherches sur le test et la vérification symboliques.

STG est implémenté en C++ et fait appel à l'outil de dessin de graphes DOTTY et, actuellement, au solveur Omega pour les contraintes exprimées en arithmétique de Presburger (http://www.cs.umd.edu/projects/omega). Il prend en entrée des spécifications et des objectifs de test symboliques décrits dans une extension du langage IF de Vérimag. Sous certaines hypothèses sur la spécification, STG permet de calculer un *graphe de test* par manipulation syntaxique, résolution de contraintes locales et extraction de sous-graphe à la façon de TGV.

Sous des hypothèses supplémentaires, les graphes de test représentent des abstractions de la spécification, qu'on peut vérifier à l'aide de PVS [56] par rapport à des propriétés de plus haut niveau, et en déduire la correction de la spécification par rapport à ces propriétés.

STG permet aussi l'exécution des tests sur une implantation. Actuellement, un test exécutable en C++ ou en JavaCard est produit et peut être lié à une implantation du système à tester (développée par ailleurs). Différentes stratégies sont utilisables pour résoudre le non-déterminisme dans le code exécutable du test. Les paramètres de l'implantation et du test doivent être fixés à l'exécution pour les rendre exécutables. Lors de l'exécution, la résolution de contraintes (Oméga) est utilisée pour résoudre les contraintes liées aux choix d'émission du testeur.

Nous avons déjà expérimenté STG sur quelques exemples significatifs dont une spécification partielle d'un porte-monnaie électronique de taille conséquente. Les graphes de test produits par manipulation syntaxique et extraction sont déjà extrêmement intéressants et fournissent déjà une aide certaine à la vérification et à la génération de tests [36], [23].

6. Résultats nouveaux

6.1. Synthèse de contrôleurs

Mots clés : *synthèse de contrôleurs*, *génération d'automates*, *méthodes symboliques*, *commande optimale*, *modèle hiérarchique*, *vérification*, *Sigali*, *Signal*.

Sur la base des modèles d'automates symboliques [5], nous nous sommes intéressés à l'application de techniques de synthèse de contrôleurs discrets, elles-mêmes intégrées dans le cadre d'un environnement de programmation au niveau tâche en robotique dans l'environnement SIGNAL/SIGALI et MATOU. En parallèle, de manière à réduire la complexité des algorithmes de synthèse de contrôleurs, nous regardons actuellement comment conserver la hiérarchie implicite des systèmes lors du calcul d'un contrôleur.

6.1.1. Synthèse de contrôleurs sur des systèmes à événements discrets hiérarchiques

Participants : Benoit Gaudin, Hervé Marchand.

La méthodologie appliquée pour la synthèse de contrôleurs est le plus souvent la suivante : les systèmes à contrôler sont à l'origine spécifiés de manière hiérarchique (analyse descendante en génie logiciel). Ensuite, la synthèse s'applique sur le système mis à plat (toute la modularité verticale a été oubliée).

Sachant que la complexité des algorithmes de calcul croît exponentiellement avec le nombre d'états des systèmes mis en parallèle et imbriqués, il semble très important de chercher à garder cette hiérarchie lors du calcul des contrôleurs. Le modèle hiérarchique (HFSM, pour Hierarchical Finite State machine) que nous considérons peut être caractérisé par une collection de structures imbriquées $\langle K_1, ... K_n \rangle$, où K_1 représente le plus haut niveau de la HFSM. À un niveau intermédiaire, la structure K_i est une HFSM, pour laquelle les états sont : soit des « états ordinaires », soit des « macro-états b » qui sont constitués d'un ensemble de structures $(K_j)_{j \in J_b} \subseteq 2^{\langle K_{i+1}, ..., K_n \rangle}$, évoluant en parallèle. Chaque structure peut avoir plusieurs états initiaux (resp. finaux). Le comportement d'une structure est le suivant : lorsque le système transite dans un macro-état b, toutes les structures associées à b sont activées et initialisées dans un de leurs états initiaux. *A contrario*, la sortie d'un macro-état est synchronisée avec la fin des tâches associées aux différentes structures de b (i.e., chaque structure est dans un état final). Entre deux états (ordinaire ou macro), le comportement de la structure est similaire à celui d'un automate plat.

Sur ce nouveau modèle, nous avons montré qu'il était possible d'utiliser les algorithmes classiques de synthèse sur chaque structure en partant du niveau le plus bas et en remontant au fur et à mesure dans la hiérarchie du modèle. Des algorithmes de synthèse de contrôleur pour des objectifs de contrôle traitant de l'invariance et du non-blocage ont ainsi été développés. De manière similaire, nous avons montré comment généraliser le problème de la commande optimale [63] dans ce nouveau cadre hiérarchique [18]. Cette année, nous avons également regardé comment synthétiser des contrôleurs pour des objectifs de contrôle faisant interagir différentes structures (typiquement, l'interdiction pour des structures du système d'être simultanément dans un ensemble d'états donné). Dans un premier temps, nous avons regardé le problème pour un produit de structures et montré comment générer une collection de contrôleurs (un pour chaque structure) et comment les agencer via un oracle pour obtenir un contrôleur global. Des conditions nécessaires et suffisantes au caractère non-bloquant du résultat ont également été caractérisées. Nous avons par la suite étendu ce résultat au cas de HFSM à deux niveaux.

6.1.2. Génération de contrôleurs pour des systèmes de contrôle-commande multi-tâches

Participant : Hervé Marchand.

La conception des systèmes robotiques et de contrôle-commande est de plus en plus difficile, ainsi que leur programmation et leur opération. Ceci est dû à leur taille et complexité grandissante. C'est pour cela que leurs architectures de programmation et exécution temps-réel requièrent plus d'assistance à l'utilisateur et au concepteur, fondée sur des abstractions utilisables et le support d'outils. Nous avons regardé la programmation au niveau tâche en robotique et contrôle-commande, où les tâches encapsulent les lois de commande et nous avons considéré l'application de techniques de synthèse de contrôleurs discrets, intégrées dans le cadre d'un environnement de programmation au niveau tâche en robotique [26][20].

Nous nous sommes également intéressé au problème du contrôle de systèmes multi-tâches, chacune étant constituée de plusieurs modes, implémentant, par exemple, des fonctionnalités avec différents niveaux de qualité (e.g. approximation de divers calculs) et de coûts (e.g. temps de calculs, énergie dépensée, etc). Ce modèle étant donné, nous avons regardé comment synthétiser des contrôleurs permettant de changer

automatiquement de mode de manière à assurer des propriétés comme borner la dépense d'énergie, maximiser la qualité de service tout en minimisant les temps de calcul, etc [21]

Cette étude est réalisée en collaboration avec Éric Rutten de l'équipe Bip (Inria Rhône-Alpes), Karine Altisen (Vérimag, Grenoble) et avec ST Microelectronics pour l'aspect étude de cas (Cf. Section 7.4).

6.2. Génération de tests de conformité sur des modèles énumérés

Mots clés : *génération de tests*, *conformité*, *systèmes de transition*.

Nous continuons à améliorer les techniques de génération de tests sur des modèles énumérés d'IOLTS. Cette année, nous étudions deux aspects principaux : la compositionalité dans la génération de tests et l'adaptation des algorithmes de génération à des critères de sélection de tests plus puissants.

6.2.1. Algorithmique de la génération de tests

Participants : Thierry Jéron, Valéry Tschaen, Erwan Demairy.

Dans le cadre du projet Européen IST Agedis (voir aussi 7.1), nous nous intéressons à la génération de tests pour des descriptions UML, principalement les diagrammes d'états d'UML. Ces derniers sont compilés dans le langage IF de Vérimag [33] et compilés en code de simulation, utilisé par le générateur de tests. L'outil de génération de tests construit est fondé principalement sur TGV et sur des fonctionnalités de l'outil d'IBM Gotcha. Nous devons donc adapter TGV pour qu'il unifie les deux approches.

En particulier, Gotcha permet au test d'observer des valeurs de variables. Le modèle d'IOLTS utilisé par TGV a donc été étendu pour incorporer ce nouveau type d'événements. Par ailleurs l'interface du simulateur IF offre la possibilité à TGV d'utiliser les valeurs de variables. Nous avons donc adapté les algorithmes de TGV pour qu'il utilise cette information et calcule des tests contenant l'observation des valeurs de variables. Par ailleurs, nous devons adapter TGV pour la prise en compte de critères de sélection de tests plus puissants que les objectifs de test actuels. Parmi les possibilités, ces critères de sélection peuvent spécifier de calculer des tests couvrant toutes les valeurs possibles d'une expression quelconque sur les variables. La prise en compte de ces critères demande de revoir en profondeur certains algorithmes de TGV. On peut voir le problème comme celui de la création dynamique d'objectifs de test, pendant la construction des tests, où les états accepteurs des objectifs sont les états où de nouvelles valeurs d'expression sont atteintes. Nous avons également doté TGV d'une nouvelle option qui permet de produire éventuellement plusieurs tests pour un même objectif en incorporant un choix aléatoire dans le parcours de graphe qu'effectue TGV.

6.2.2. Génération compositionelle de tests

Participants : Thierry Jéron, Valéry Tschaen.

Dans le cadre de la thèse de Valéry Tschaen, nous avons commencé à étudier trois aspects de la compositionalité dans la génération de tests [28].

Le premier problème est de générer des tests pour une spécification composée $S = S_1 \times S_2 \times ... \times S_n$ et un objectif de test TP de manière incrémentale, sans calculer le comportement complet de S. Dans TGV, le calcul d'un cas de test serait fondé sur un parcours à la volée du produit de TP avec S. On imagine ici pouvoir calculer un cas de test en calculant un nouvel objectif $TP1 = f(S_1 \times TP)$, puis $TP_2 = f(S_2 \times TP_1)$, ...et finalement $TC = f'(S_n \times TP_{n-1})$ où f, f' opèrent un masquage des actions du produit qui ne sont plus utiles dans la suite du calcul. Cette étude s'inspire des travaux sur la vérification compositionelle [44].

Un autre aspect de la compositionalité consiste à considérer la composition d'objectifs de test simples par des opérations d'intersection, union, complémentaire, concaténation et de générer les tests en se basant sur cette composition. Cette approche est utile quand le langage de description d'objectifs permet ce type d'opérations, en particulier dans des formalismes comme les diagrammes de séquences UML combinés avec les diagrammes d'activités, comme dans le langage d'objectifs Tela défini dans le cadre du projet COTE. L'idée est donc de définir une algèbre d'objectifs de test et de modifier les algorithmes de génération de tests en conséquences. Le but est d'éviter de calculer a priori l'objectif composé, mais de le calculer au vol lors de la génération de tests.

Le dernier aspect consiste à calculer des tests pour plusieurs objectifs de test simultanément. Le but est de factoriser les calculs communs à plusieurs objectifs, tout en découplant les résultats, i.e. les cas de test pour chaque objectif. L'algorithmique nécessaire est une extension de l'algorithmique actuelle de TGV.

6.3. Interaction test et contrôle

Mots clés : *synthèse de contrôleurs*, *génération de tests*.

La génération de tests et la synthèse de contrôleurs sont deux problèmes extrêmement proches, autant du point de vue des modèles possibles que du point de vue techniques et algorithmes. Cette année, nous avons regardé comment mixer ces deux approches dans plusieurs contextes différents. Nous nous sommes intéressés au contrôle de la conformité d'une implantation par le biais d'un superviseur ; au contrôle pour le test de robustesse ; et au test/contrôle des automates temporisés.

6.3.1. Contrôle de la conformité

Participants : Thierry Jéron, Hervé Marchand, Vlad Rusu, Valéry Tschaen.

Nous nous intéressons ici au contrôle de la conformité d'une implantation par le biais d'un superviseur. En d'autres termes : étant données une implantation, dont seule une partie du comportement est connue, et une spécification de référence, comment synthétiser un superviseur de telle manière que l'implantation contrôlée soir conforme à la spécification de référence. Bien sur, suivant la connaissance que le superviseur a de l'implantation, le problème a résoudre est différent.

- Si l'implantation n'a pas d'actions internes (i.e. des événements non observables de « l'extérieur »), le problème se ramène à un problème de synthèse de contrôleurs non-bloquant classique modulo une redéfinition du non-blocage.
- Dans le cas où l'implantation a des actions internes, mais observables par le superviseur, le problème peut se décomposer en plusieurs problèmes de contrôle dont l'un consiste à rendre acyclique un système de transition ayant plusieurs états initiaux et plusieurs états finaux, tout en conservant un critère d'optimalité du résultat basé sur le nombre de dépliages de cycles d'actions internes.
- Le cas où l'implantation a des actions internes, non observables par le superviseur est actuellement à l'étude.

Une fois, ce superviseur calculé, il nous semble nécessaire de tester l'implantation contrôlée (qui a donc été modifiée). L'idée serait de générer des cas des tests qui mèneraient l'implantation vers les endroits où le contrôle a eu lieu de manière à vérifier que l'implantation contrôlée a un bon fonctionnement sous contrôle.

6.3.2. Contrôle et test de robustesse

Participants : Thierry Jéron, Hervé Marchand.

Dans le cadre de l'AS Test du CNRS avec Claude Jard (projet Triskell) (voir aussi 8.1.2), nous nous intéressons au test de robustesse. Au delà de ce cadre, nous collaboration avec Sophie Pinchinat (projet S4) sur l'expression des propriétés de robustesse et leur contrôle. La robustesse peut se définir comme la capacité d'un système à fonctionner de manière acceptable, malgré la présence d'aléas. Nous considérons des aléas de différente nature : faute d'un composant, entrée inattendue ou erronée, etc. Notre démarche du test de robustesse est basée sur la modélisation du système par des modes et de ses propriétés attendues dans les différents modes, le contrôle de la spécification afin qu'elle satisfasse ces propriétés, et la génération de tests à partir de la spécification contrôlée, afin de détecter la non-conformité d'une implantation vis à vis de la spécification contrôlée ou des propriétés attendues.

Le modèle du système est un système de transition (LTS) dont les états sont partitionnés en modes et dont les actions sont partitionnées suivant différents critères : entrées/sorties/actions internes pour le test, événements contrôlables/incontrôlables pour le contrôle, actions normales/aléas pour la robustesse. Dans le type de système ouvert que nous considérons, il est naturel d'exprimer la robustesse du système vis à vis de

son environnement comme un jeu (ou une famille de jeux) où, quoique fasse l'environnement (les aléas), le système S puisse satisfaire ses propriétés P i.e. $S \models P$.

Étant donné une spécification S, et ses propriétés attendues vis à vis de la robustesse P, le premier problème est d'assurer la satisfaction des propriétés par la spécification. Le problème posé est un problème de contrôle. Il s'agit de calculer un superviseur C tel que $S \times C \models P$.

Maintenant, étant donné la spécification contrôlée $S \times C$, la génération de tests consiste à produire des tests ciblant des erreurs éventuelles dans l'implantation des mécanismes de robustesse. Différentes techniques sont envisagées, dépendant du modèle de référence (spécification ou propriétés) et de la relation de conformité voulue entre l'implantation IUT et la spécification contrôlée $S \times C$ (conformité **ioco**, ou satisfiaction $\models$). Par exemple, pour IUT ioco $S \times C$, on cherche à identifier des objectifs de test sélectionnant des séquences de $S \times C$ incluant des transferts de modes en utilisant TGV sur $S \times C$. On peut aussi cibler les actions contrôlées par C. Pour tester que $IUT \models P$, on peut également utiliser TGV sur $S \times C$, mais le verdict dépendra de P, et non pas de $S \times C$.

6.3.3. Test et contrôle de systèmes temps-réel répartis

Participant : Ahmed Khoumsi.

L'objectif est ici d'étudier le test et le contrôle de systèmes temps-réel répartis. Nous avons conçu une première version d'une transformation d'automates temporisés en un type particulier d'automates finis. Ces derniers contiennent des événements spéciaux appelés Set et Exp. Cette transformation a été appliquée au test et au contrôle de systèmes temps-réel déterministes. Nous étudions également l'application de cette transformation au test de systèmes temps-réel non-déterministes [51][49][50]. Plus particulièrement, on s'intéresse à étendre TGV en y introduisant l'aspect temps-réel. Lorsque l'aspect réparti s'ajoute à celui du temps réel, on s'intéresse essentiellement aux problèmes de coordinations et de synchronisations des testeurs locaux où il faut, non seulement respecter des contraintes d'ordres, mais aussi des contraintes temporelles entre des événements répartis.

6.4. Vérification par Interprétation Abstraite

Participants : Bertrand Jeannet, Wendelin Serwe.

Mots clés : *Vérification formelle*, *Interprétation Abstraite*, *Prise en compte des données*.

Nous nous intéressons ici aux techniques de vérification permettant de prendre en compte la valeur des données des systèmes étudiés. Mentionnons que ces techniques de vérification se retrouvent souvent au coeur des méthodes de génération de tests ou de contrôleurs. Nous nous intéressons d'une part à la combinaison de différents types de données, d'autre part à l'extension de techniques adaptées aux systèmes réactifs à des systèmes plus complexes, effectuant des appels de procédure et des allocations dynamiques.

6.4.1. Combinaison de différents types de donnée en Interprétation Abstraite.

Combiner différents types de données en Interprétation Abstraite, et ce de manière modulaire, est délicat : les techniques standards sont soit peu précises mais économiques, soit très précises mais aussi très coûteuses.

Déjà, pour chaque type de données, il faut trouver un compromis entre une précision élevée, obtenue en utilisant un domaine abstrait précis, et une complexité raisonnable ; il s'agit en effet d'obtenir des analyses pouvant traiter des systèmes de taille si possible élevée. Nous nous concentrons ici sur les domaines abstrait numériques : intervalles, polyèdres [39], octogones [54], et leurs combinaisons. Un sujet de stage a été posé pour étudier l'utilisation des octogones en vérification et en génération de tests, au lieu des classiques polyèdres.

Il faut savoir ensuite comment combiner différents types de données ; pour l'instant nous avons principalement besoin de combiner variables booléennes et numériques, mais nous cherchons à le faire dans le cadre d'une méthode plus générale. Là encore, le compromis précision/efficacité est essentiel. La méthode générale de combinaison, qui date de mon travail de thèse [10], consiste à partitionner un treillis abstrait composite « de base » pour ajuster le compromis efficacité/précision. Il reste cependant à calculer efficacement les fonctions de

transfert dans ce treillis partitionné. L'hiver 2002 a été consacré à la conception d'une nouvelle technique pour ce faire, qui a été publiée dans [16] et implantée dans l'outil NBAC (§5.2). Plus précisément, ce travail propose une méthode flexible pour calculer l'image d'une valeur abstraite composite par un vecteur de fonctions de transition. Dans le cadre de l'analyse de programmes synchrones , ce calcul peut être extrêmement coûteux et est le facteur limitant de notre méthode générale.

Ce travail trouve une application directe au sein de l'équipe dans la simplification et l'instanciation des tests symboliques (§6.5.2).

6.4.2. Vérification de systèmes plus expressifs

Nous nous sommes intéressés pour l'instant aux systèmes réactifs, dont la sémantique est relativement simple : un état est défini par la donnée d'un point de contrôle et d'une valuation pour un ensemble fixé de variables. Dans ce cadre la recherche sur la prise en compte des données porte plus sur les aspects algorithmiques que méthodologiques. En revanche, la prise en compte des données dans des programmes plus complexes, avec appels de procédure et allocation dynamique, reste plus délicate mais mérite d'être étudiée, particulièrement dans le cadre de JAVACARD et des cartes à puces.

Des idées précises commencent à se dégager et un rapport de recherche sur l'analyse de programmes récursifs avec variables locales et globales est en cours de rédaction. Nous y réutilisons de manière essentielle les résultats sur la combinaison des différents types de données. En particulier, l'existence d'une pile d'exécution dans la sémantique opérationnelle de ce type de système nous oblige dans un premier temps à introduire le type de donnée « pile » et un domaine abstrait adapté pour ce type de données, puis dans un second temps à combiner ce type avec les Booléens et les numériques.

6.5. Génération de tests et vérification symboliques

Mots clés : *Génération de tests symboliques*, *Vérification formelle*.

Nous nous intéressons aux interactions possibles entre la vérification formelle et le test de conformité. En 2002 nous avons exploré deux voies. La première est l'intégration de ces deux techniques complémentaires de validation dans une même démarche formelle de développement de systèmes réactifs. La deuxième est l'utilisation de techniques d'interprétation abstraite pour la simplification et l'instanciation « intelligente » de cas de test symboliques.

6.5.1. Combinaison vérification-test pour la validation de systèmes

Participants : Vlad Rusu, Elena Zinovieva.

La vérification formelle et le test sont deux approches complémentaires pour la validation des systèmes informatiques au sens large. La *vérification* consiste à prouver la correction d'une spécification formelle d'un système par rapport à des exigences de plus haut niveau. On écrit $\mathcal{S} \models \mathcal{E}$, où $\mathcal{S}$ dénote la spécification et $\mathcal{E}$ dénote l'ensemble des exigences. D'autre part, le *test* consiste à exécuter des jeux d'essais sur une implantation d'un système, et de confier à un « oracle » (typiquement, le testeur) la mission de décider si des erreurs ont été détectées. En *test de conformité*, l'implantation testée est une boite noire, et les jeux d'essais sont des traces observables de cette implantation ; l'oracle est calculé automatiquement à partir de la spécification formelle, opérationnelle du système. La conformité d'une implantation $\mathcal{I}$ avec sa spécification $\mathcal{S}$ s'écrit $\mathcal{I}$ conf $\mathcal{S}$.

Nous avons donc trois vues d'un même système : exigences, spécification, et implantation ; et deux démarches différentes (vérification et test) pour assurer la cohérence entre celles-ci. Typiquement, l'objectif de la validation du système est d'assurer que l'implantation respecte les exigences. Pour atteindre cet objectif il est essentiel que les étapes de vérification et de test soient parcourues. D'une part, la vérification sans le test ne dit rien sur l'implantation, car cette dernière n'est pas prise en compte dans l'étape de vérification ; d'autre part, le test sans la vérification peut détecter des « faux positifs » (si la spécification ne respecte pas les exigences, une implantation conforme, même identique en comportement avec la spécification, ne les respecte pas non plus) et des « faux négatifs » (si la spécification ne respecte pas les exigences, une implantation peut être déclarée non conforme à la spécification alors même qu'elle respecte les exigences).

Cependant, la vérification et le test (notamment, dans l'étape de génération) sont des activités difficiles et coûteuses. Dans l'article [23] nous proposons de factoriser ces efforts par une approche qui utilise la génération de tests symboliques comme outil d'abstraction. Sous certaines conditions raisonnables, les graphes symboliques obtenus par la génération de test symboliques sont des abstractions correctes, au sens où la correction d'un ou plusieurs de ces sous-graphe permettent de conclure à la correction de la spécification entière. Cette abstraction peut considérablement réduire l'effort de vérification, car les sous-graphes sont typiquement de taille bien plus réduite que la spécification entière. D'autre part, ces sous-graphes sont des cas de test symboliques corrects par rapports aux exigences, dans le sens qu'ils ne donneront pas lieu aux résultats faux (positifs ou négatifs) susmentionnés.

Nous utilisons l'outil STG pour la génération des sous-graphes, et PVS [56] pour leur vérification. Nous avons appliqué cette méthode à un porte-monnaie électronique. Les propriétés vérifiées sont des invariants impliquant des quantificateurs sur des domaines non bornés. La spécification elle-même est un automate étendu avec une centaine de transitions et des types de données infinis tels que des tableaux de taille paramétrique. Cette vérification est hors de portée des techniques actuelles basées sur le model-checking ou la preuve seule.

6.5.1.1. Vers une vérification plus automatique d'invariants

Même si PVS fournit un certain niveau d'automatisation pour la vérification des propriétés d'invariance, cette automatisation pourrait être poussée plus loin, notamment pour des propriétés impliquant des vecteurs ou des tableaux de taille paramétrique. Ces types de données peuvent également modéliser des canaux de communication, des files d'attente.... Nous avons développé une méthode de vérification automatique d'invariants basée sur la notion de prédicat inductif. Un prédicat est inductif s'il est vrai initialement et si, en supposant qu'il est vrai avant une transition, on peut inférer qu'il reste vrai après. Des invariants auxiliaires peuvent fournir des informations supplémentaires pour prouver cette préservation. Nous traduisons la propriété d'inductivité dans un contexte d'invariants auxiliaires en un problème de satisfiabilité dans une logique appelée *arithmétique de Presburger avec symboles de fonction*, que nous décidons à l'aide d'une heuristique qui consiste à instancier les quantificateurs par les termes qui sont sous l'incidence des symboles de fonction [24]. Cette heuristique a permis de traiter de manière plus automatique la partie vérification du porte monnaie électronique, vérifié par ailleurs avec PVS dans [23].

6.5.2. Simplification et instanciation de tests symboliques

Participants : Bertrand Jeannet, Vlad Rusu, Elena Zinovieva.

Les cas de test produits actuellement par l'outil STG sont inutilement redondants. Ceci est dû aux techniques presque exclusivement syntaxiques utilisées dans la génération, qui prennent en compte essentiellement le contrôle et très peu les données : essentiellement, un cas de test symbolique est un sous-graphe de la spécification qui, au vu du contrôle seul, mène à la satisfaction de l'objectif de test. L'approche a été prouvée correcte [7] (au sens des propriétés attendues du cas de test : correction des verdicts et exhaustivité relative à l'objectif) mais elle est redondante au sens suivant : si l'on prend en compte les données

- les cas de test comportent des parties inaccessibles ;
- les cas de test ne sont pas contrôlables de manière optimale, au sens où le choix d'un stimulus à donner à l'implantation peut ne plus jamais mener à la satisfaction de l'objectif.

Ces problèmes ne peuvent pas être résolus dans un cadre général et de manière exacte et automatique pour des systèmes de transitions symboliques, car on se heurte à des problèmes d'indécidabilité. Cependant, à l'aide de méthodes approchées issues de l'interprétation abstraite [38] on peut faire mieux qu'actuellement. L'idée est d'utiliser les analyses en avant et en arrière implantées dans l'outil NBAC (cf. Section 5.2) pour détecter, autant que possible, les parties inaccessibles d'un cas de test ; et de renforcer les gardes des transitions émettrices afin d'éliminer les parties qui ne sont pas coaccessibles (i.e., ne pouvant mener la localité finale *Pass*). La formalisation, les algorithmes, et l'implantation de l'approche à travers une connexion STG- NBAC sont brièvement décrits dans [29] et font partie de la thèse d'Elena Zinovieva.

7. Contrats industriels

7.1. Agedis

Participants : Thierry Jéron, Valery Tschaen, Erwan Demairy.

Mots clés : *test*, *génération de tests*, *logiciel réparti*, *UML*.

Agedis (fin 2000-2003) est un projet européen IST (http://www.agedis.de/). Ce projet vise à fournir des méthodes et outils pour l'automatisation du test de logiciels, et en particulier pour les logiciels répartis, principalement du domaine télécom. Nous y collaborons avec IBM Haifa (Israël) et IBM Hursley (G.-B.), France Télécom R& D (Lannion), IntraSoft (Grèce), Imbus (Allemagne), l'université d'Oxford et Vérimag. Dans Agedis, le langage choisi pour la description des systèmes est un profil UML contenant les diagrammes de classe, d'objets et d'états. Le profil permet également la descriptions de critères de sélection de test, combinant objectifs de test à la TGV et critères de couvertures d'expression, en utilisant les diagrammes d'états UML. Une description UML avec ses critères de sélection est compilée dans le langage IF (Verimag), lui même compilé en une API de simulation qui lui offre les fonctions de parcours et d'observation d'un produit entre modèle et critères de sélection. L'outil de génération de tests que nous réalisons utilise l'API du simulateur IF. Il est basé sur TGV et incorpore des fonctionnalités de l'outil d'IBM Gotcha, en particulier relatifs aux critères de couverture (voir aussi 6.2.1). Les tests produits sont ensuite traduits dans un format XML et utilisés par un outil d'exécution de tests basé sur l'outil TC Beans (IBM). Bien que les développements soient encore en cours, une première démonstration de la chaîne d'outils a été réalisé au printemps sur un exemple jouet. Des expérimentations sur des études de cas plus conséquentes sont en cours chez les partenaires industriels.

7.2. Cote

Participants : Thierry Jéron, Simon Pickin.

Mots clés : *test*, *composants*, *UML*.

Cote (2001-2002) est un projet RNTL pré-compétitif (http://www.irisa.fr/cote/) dont le but est de fournir des méthodes et des outils facilitant le test et la certification de composants logiciels. Nous participons à ce projet aux cotés de France Télécom, Gemplus, Softeam et le LSR/IMAG et les projets de l'Irisa Triskell et Lande. Le but du projet COTE est de démontrer une chaîne complète d'outils UML permettant la spécification des composants, la spécification d'objectifs de test, la spécification de tests ou leur génération à partir d'objectifs et leur exécution. Dans ce projet, VerTeCs a été d'abord impliqué dans la définition du langage Tela fondé sur les diagrammes de séquences et d'activité et des annotations OCL. Ce langage permet la description d'objectifs de test et de cas de test. Il est précurseur dans la définition d'un profil test pour UML par l'OMG. Par ailleurs, l'outil de génération de tests de l'atelier est fondé sur UMLAUT-TGV i.e. TGV connecté à l'API de simulation fournie par UMLAUT (projet Triskell). Les objectifs de test utilisés par TGV peuvent être issus de l'outil Tobias du LSR ou de l'outil UML Casting du projet Lande. Les différents outils ont été utilisés avec succès sur plusieurs études de cas.

7.3. Projet Castor

Participant : Hervé Marchand.

Mots clés : *sécurité des systèmes informatiques*, *modèles hiérarchiques*, *synthèse de contrôleurs*.

Le projet CASTOR a pour but la réalisation d'outils pour l'aide à la conception d'architectures sécurisées d'un système d'information. L'autre partenaire de cette action est le projet Inria Lande.

L'objectif général du projet Castor est de montrer la faisabilité de la modélisation de la sécurité d'un système informatique. En effet la complexité des systèmes d'information fait que la notion de sécurité répartie est difficile à mettre en oeuvre (de même que son maintien au cours du temps). L'objectif des travaux consiste à étudier et à prototyper la modélisation d'architectures bâties par assemblage de composants suivant des

règles d'intégration de la sécurité. Dans cette optique, nous nous sommes occupés principalement du modèle théorique. En particulier, nous nous sommes intéressés, cette année, à l'application des techniques de synthèse de contrôleurs pour la génération automatique de chemins d'attaque d'un système. Deux axes principaux ont ainsi été regardés.

- Dans un premier temps, nous avons étudié diverses approches pour l'analyse exhaustive des propriétés de Sécurité de modèles Castor, par le biais d'une abstraction du modèle, puis par l'utilisation de l'environnement Signal/Sigali pour la modélisation d'un exemple concret en Signal et l'analyse de ce système via Sigali (tant d'un point de vue vérification que génération de chemins d'attaques par des techniques de synthèse).
- Dans un deuxième temps, devant la complexité des modèles que nous devions analyser, nous avons regardé l'utilisation d'un modèle hiérarchique pour la synthèse de contrôleurs [18], dans le but de simplifier la complexité algorithmique. (Cf. Section 6.1.1 pour plus de détails sur cet aspect).

7.4. ST Microelectronics

Participant : Hervé Marchand.

Dans ce projet, en collaboration avec l'équipe BIP de l'Inria Rhône-Alpes, nous avons étudié les potentiels de l'approche synchrone sur le flot de conception chez ST-Microelectronics particulièrement du point de vue de la synthèse de contrôleurs discrets dans le cas de systèmes multi-tâches [26]. Cette coopération a donné lieu à une étude de cas de taille significative consistant en la spécification d'un codeur MPEG4 sous forme d'un système multi-tâches. Le but était de calculer, au moyen de techniques de synthèse de contrôleurs, un superviseur partiel à ce système qui choisirait le meilleur algorithme de codage d'images parmi plusieurs choix en fonction de divers paramètres dynamiques.

7.5. ITEA EAST-EEA, Embedded Electronic Architecture

Participants : Thierry Jéron, Hervé Marchand.

Le projet EAST-EEA (ITEA-Project-No. 00009, http://east-eea.test.k-k.de/) est en quelques sortes la suite européenne de l'action de développement AEE à laquelle nous avons participé précédemment. Ce projet regroupe les constructeurs automobiles européens majeurs, les équipementiers et des laboratoires de recherche. Le projet vise à la définition d'une architecture ouverte permettant une meilleure interopérabilité des composants logiciels et matériels. Notre participation, qui est pour l'instant marginale, consiste à promouvoir les techniques automatiques de génération de tests.

8. Actions régionales, nationales et internationales

8.1. Actions nationales

8.1.1. Action de Recherche Coopérative MODOCOP

Participants : Bertrand Jeannet, Thierry Jéron, Vlad Rusu, Wendelin Serwe, François-Xavier Ponscarme.

Modocop (http://www-sop.inria.fr/lemme/modocop) est une ARC (Action de Recherche Coopérative de l'Inria) démarrée le 1erJanvier 2002 et d'une durée prévue de deux ans. Les buts de cette action sont : le développement d'un langage de spécification pour les langages orientés objet concurrents, avec comme cible le langage Java et les futures versions concurrentes du langage JavaCard, et le développement de techniques d'analyse statique, de vérification, et de test pour ce type de langages. L'action regroupe les projets Lemme et Oasis de l'UR Sophia Antipolis, Vasy de l'UR Rhône-Alpes, le laboratoire Grenoblois Verimag, ainsi que les projets Lande et VerTeCs de l'Irisa. Nous participons dans cette action dans les aspects vérification et test symboliques.

8.1.2. Action Spécifique Test no. 23

Participants : Thierry Jéron, Hervé Marchand.

Nous participons depuis fin 2001 à l'Action Spécifique no. 23 du département STIC du CNRS intitulée « Techniques avancées de tests des systèmes complexes » (http://www.laas.fr/TSF/AS23). Cette action pilotée par le LAAS (Toulouse) et le LaBri (Bordeaux) comprend par ailleurs des équipes du LRI (Orsay), de Verimag (Grenoble) et de l'Irisa (Triskell et VerTeCs).

L'objectif de cette AS est d'aborder le problème du test de robustesse pour les systèmes à logiciel prépondérant, critiques ou embarqués. Nous avons d'abord dû définir de façon plus précise ce que signifie la robustesse pour ce type de systèmes et ce que signifie tester la robustesse. La robustesse est la capacité d'un système à fonctionner de manière acceptable en présence d'aléas divers. A partir des techniques maîtrisées par chacun des groupes nous avons ensuite décrit comment générer des tests de robustesse. L'approche que nous proposons (voir 6.3.2) est basée sur la spécification du système dans un modèle de systèmes de transition avec modes, de ses propriétés attendues en logique, le contrôle de la spécification afin qu'elle satisfasse ses propriétés de robustesse et la génération de tests à partir de la spécification contrôlée et d'objectifs de test ciblant l'occurrence d'aléas.

8.2. Réseaux et groupes de travail internationaux

8.2.1. Réseaux d'excellence

Nous avons participé au cours de l'année 2002 à deux expressions d'intérêt pour des réseaux d'excellence. Il s'agit du réseau Testnet (*Integration of Testing Methodologies*, http://www-lor.int-evry.fr/testnet/) qui vise à unifier les techniques de test de protocole et de logiciel et du réseau DEFINE (Dependability Foundations for Information infrastructures - Network of Excellence, http://www.laas.fr/DeSIRE&DeFINE/index.html).

8.2.2. Collaboration avec le Stanford Research Institute (Californie, USA)

Une convention entre le CNRS (pour l'Irisa) et le Stanford Research Institute (SRI) a été signée. Dans ce cadre, Vlad Rusu effectue régulièrement des séjours au SRI. Le dernier date de l'été 2002, lorsque Elena Zinovieva et Vlad Rusu ont effectué un séjour de deux, respectivement un mois. Ils ont travailé sur une méthode de vérification automatique d'invariants inductifs.

9. Diffusion des résultats

9.1. Enseignement universitaire

Thierry Jéron a enseigné (avec Didier Caucal) en 2001-2002 en DEA un module intitulé « Effectivité et Efficacité ». Depuis 2002 Thierry Jéron est responsable du module de tronc commun du DEA intitulé « Composants et Test » (avec Arnaud Gottlieb de l'équipe Lande et Yves le Traon de l'équipe Triskell).

Thierry Jéron enseigne la validation de protocoles à l'IFSIC en DIIC 3, ainsi qu'à l'ENST Brest (en 2^{e}année) et Rennes (en 3^{e}année).

Vlad Rusu enseigne (avec Thomas Genêt de l'équipe Lande) en DEA un module intitulé « Méthodes Déductives pour la Vérification ».

9.2. Thèses et Stages

Thèses en cours dans le projet :

1. Benoit Gaudin, « Contrôle des Systèmes à Évenements Discrets Hierarchiques »
2. Valéry Tschaen, « Modèles et Techniques de Synthèse de Tests »
3. Elena Zinovieva « Génération et Simplification de Test Symboliques »

Stages effectués dans le projet :

1. François-Xavier Ponscarme, « Génération de Tests Symboliques en Java », stage DESS.

9.3. Participation à des jurys

Thierry Jéron a participé aux jurys de concours TR05 pour le recrutement d'assistantes de projets.

Bertrand Jeannet a été membre (examinateur) du jury de la soutenance de thèse de David Garriou, préparée à l'IRCCyN à Nantes et soutenue le 20 septembre 2002.

9.4. Participation à des colloques, séminaires, invitations

Thierry Jéron a donné un cours sur la génération de tests dans le cadre de l'école d´été Jeunes Chercheurs en Programmation à Rennes, en mai 2002.

Thierry Jéron a co-organisé avec Rob Hierons (Univ. Brunel, UK) le Workshop Fates (Formal Approaches to Testing of Software) [9] satellite de CONCUR'02, à Brno (Rep. Tchèque) en août 2002.

Thierry Jéron a donné un séminaire invité sur la théorie et les algorithmes de TGV au laboratoire IEI à Pise en novembre 2002.

Thierry Jéron organise une Rencontre IrisaTech sur le Test et la Qualité de Service à l'Irisa en décembre 2002.

Hervé Marchand à donné une conférence intitulée « Contrôle des Systèmes à Évenements Discrets Hierarchiques » au LAG (Grenoble).

Vlad Rusu a donné une conférence intitulée « Combining Verification and Testing » au LaBRI (Bordeaux) et au Stanford Research Institute (Californie, USA).

Bertrand Jeannet a donné une conférence intitulée « Representing and Approximating transfer functions in Abstract Interpretation of heterogenous datatypes » à l'ENS de Paris en février 2002 (séminaire invité) et à l'ENS Cachan en mars 2002, dans le cadre des réunions de l'Action Spécifique de Recherche du CNRS « Vérification de Propriétés Quantitatives ».

Elena Zinovieva et Valéry Tschaen ont présenté leurs travaux de thèse à l´ecole d'été *Modelling and Verifying Parallel Processes Summer School (MOVEP'02)* à Nantes.

10. Bibliographie

Bibliographie de référence

[1] M. BOZGA, J.-C. FERNANDEZ, L. GHIRVU, C. JARD, T. JÉRON, A. KERBRAT, P. MOREL, L. MOUNIER. *Verification and test generation for the SSCOP protocol.* in « Journal of Science of Computer Programming, special issue on Formal Methods in Industry », numéro 1, volume 36, Janvier, 2000, pages 27-52.

[2] T. JÉRON, P. MOREL. *Test generation derived from model-checking.* in « CAV'99, Trento, Italy », série LNCS, volume 1633, Springer-Verlag, éditeurs N. HALBWACHS, D. PELED., pages 108-122, Juillet, 1999.

[3] B. JEANNET, N. HALBWACHS, P. RAYMOND. *Dynamic Partitioning in Analyses of Numerical Properties.* in « Static Analysis Symposium, SAS'99 », série LNCS, volume 1694, 1999.

[4] H. MARCHAND, O. BOIVINEAU, S. LAFORTUNE. *On the Synthesis of Optimal Schedulers in Discrete Event Control Problems with Multiple Goals.* in « SIAM Journal on Control and Optimization », numéro 2, volume 39, 2000, pages 512-532.

[5] H. MARCHAND, P. BOURNAI, M. LE BORGNE, P. LE GUERNIC. *Synthesis of Discrete-Event Controllers based on the Signal Environment.* in « Discrete Event Dynamic System : Theory and Applications », numéro 4, volume 10, Octobre, 2000, pages 347-368.

[6] H. MARCHAND, M. SAMAAN. *Incremental Design of a Power Transformer Station Controller using Controller Synthesis Methodology.* in « IEEE Transaction on Software Engineering », numéro 8, volume 26, Août, 2000, pages 729-741, http://www.irisa.fr/vertecs/Publis/Ps/IEEE-TSE2000.ps.gz.

[7] V. RUSU, L. DU BOUSQUET, T. JÉRON. *An approach to symbolic test generation.* in « International Conference on Integrating Formal Methods (IFM'00) », série LNCS 1945, Springer Verlag, pages 338-357, 2000.

[8] V. RUSU. *Verifying a Sliding-Window Protocol using PVS.* in « Formal Techniques for Networked and Distributed Systems (FORTE'01) », Kluwer Academic Publishers, pages 251-266, 2001.

Livres et monographies

[9] *FATES'02, Formal Approaches to Testing of Software, A Satellite Workshop of CONCUR'02.* éditeurs R. HIERONS, T. JÉRON., Inria Report, Août, 2002, http://www.brunel.ac.uk/~csstrmh/concur2002/Fates_proceedings.pdf.

Articles et chapitres de livre

[10] B. JEANNET. *Dynamic Partitioning In Linear Relation Analysis. Application To The Verification Of Reactive Systems.* in « Formal Methods in System Design », 2002, à paraître.

[11] T. JÉRON. *TGV : théorie, principes et algorithmes.* in « Techniques et Sciences Informatiques, numéro spécial Test de Logiciels », volume 21, 2002, pages 1265-1294, à paraître.

[12] H. MARCHAND, O. BOIVINEAU, S. LAFORTUNE. *On Optimal Control of a class of partially-Observed Discrete Event Systems.* in « Journal of Automatica », numéro 11, volume 38, Octobre, 2002, pages 1935-1943.

Communications à des congrès, colloques, etc.

[13] A. BENVENISTE, P. CASPI, P. LE GUERNIC, H. MARCHAND, J.P. TALPIN, S. TRIPAKIS. *A Protocol for Loosely Time-Triggered Architectures.* in « Embedded Software Conference (EMSOFT '02) », Grenoble, France, Octobre, 2002.

[14] D. CLARKE, T. JÉRON, V. RUSU, E. ZINOVIEVA. *STG : a Symbolic Test Generation Tool.* in « Tools and Algorithms for the Construction and Analysis of Systems (TACAS'02) », série LNCS 2280, Kluwer Academic Publishers, 2002, http://www.irisa.fr/vertecs/Equipe/Rusu/tacas02.html, Tool paper.

[15] C. JARD, T. JÉRON. *TGV : theory, principles and algorithms.* in « The Sixth World Conference on Integrated Design & Process Technology (IDPT'02) », Pasadena, California, USA, Juin, 2002, http://www.irisa.fr/vertecs/Publis/Ps/2002-IDPT.ps.gz.

[16] B. JEANNET. *Representing and Approximating Transfer Functions in Abstract Interpretation of Hetereogeneous Datatypes.* in « Static Analysis Symposium, SAS'02 », série LNCS, volume 2477, Madrid (Spain), Septembre, 2002.

[17] B. JEANNET, P. D'ARGENIO, K.G. LARSEN. *RAPTURE : A tool for verifying Markov Decision Processes.* in « Tools Day, International Conference on Concurrency Theory, CONCUR'02 », Brno, Czech Republic, Août,

2002, http://www.irisa.fr/prive/bjeannet/rapture02.ps.gz.

[18] H. MARCHAND, B. GAUDIN. *Supervisory Control Problems of Hierarchical Finite State Machines.* in « 41th IEEE Conference on Decision and Control », Las Vegas, USA, Décembre, 2002, http://www.irisa.fr/vertecs/Publis/Ps/2002-CDC.ps.gz.

[19] H. MARCHAND, L. ROZÉ. *Diagnostic de pannes sur des systèmes à événements discrets : une approche à base de modèles symboliques.* in « 13ème Congrès Francophone AFRIF-AFIA de Reconnaissance des Formes et Intelligence Artificielle », pages 191-200, Angers, France, Janvier, 2002, http://www.irisa.fr/vertecs/Publis/Ps/2002-RFIA.pdf.

[20] H. MARCHAND, E. RUTTEN. *A case study in applying discrete control synthesis to excavator operation.* in « IEEE International Conference on Systems, Man and Cybernetics (IEEE SMC) », Hammamet, Tunisia, octobre, 2002.

[21] H. MARCHAND, E. RUTTEN. *Managing multi-mode tasks with time cost and quality levels using optimal discrete control synthesis.* in « 14th Euromicro Conference on Real-Time Systems (ECRTS'02) », Juin, 2002, http://www.irisa.fr/vertecs/Publis/Ps/2002-ECRTS.ps.gz.

[22] S. PICKIN, C. JARD, Y. LE TRAON, T. JÉRON, J.-M. JEZEQUEL, A. LE GUENNEC. *System Test Synthesis from UML Models of Distributed Software.* in « Forte 2002, 22nd IFIP WG 6.1 International Conference on Formal Techniques for Networked and Distributed Systems, Houston, Texas », série LNCS, Springer, Novembre, 2002, http://www.irisa.fr/vertecs/Publis/Ps/2002-Forte.pdf.gz.

[23] V. RUSU. *Verification Using Test Generation Techniques.* in « Formal Methods Europe (FME'02) », 2002, http://www.irisa.fr/vertecs/Equipe/Rusu/FME02/.

[24] V. RUSU, E. ZINOVIEVA, D. CLARKE. *Verifying Invariants More Automatically.* in « Workshop on Verification and Computationonal Logic, VCL'02 », 2002, http://www.irisa.fr/vertecs/Equipe/Rusu/VCL02/.

[25] P. D'ARGENIO, B. JEANNET, H.E. JENSEN, K.G. LARSEN. *Reduction and Refinement Strategies for Probabilistic Analysis.* in « Process Algebra and Probabilistic Methods - Performance Modelling and Verification, PAPM-PROBMIV 2002 », série LNCS, volume 2399, Copenhagen, Denmark, Juillet, 2002, http://www.irisa.fr/prive/bjeannet/djjl02.ps.gz.

Rapports de recherche et publications internes

[26] E. RUTTEN, H. MARCHAND. *Task-level programming for control systems using discrete control synthesis.* rapport technique, numéro 4389, INRIA, Février, 2002, http://www.inria.fr/rrrt/rr-4389.html.

Divers

[27] E. RUTTEN, A. GIRAULT, H. MARCHAND. *Un panorama de la programmation synchrone pour le flot de conception de ST Microelectronics.* Rapport de contrat d'expertise No 1 01 G070700 71299 21 2, Janvier, 2002.

[28] V. TSCHAEN. *Compositionality issues in test Synthesis.* Proceedings of the Modelling and Verifying Parallel Processes Summer School (MOVEP'02), Juin, 2002.

[29] E. ZINOVIEVA. *Symbolic Test Generation for Reactive Systems.* Proceedings of the Modelling and Verifying Parallel Processes Summer School (MOVEP'02), Juin, 2002.

Bibliographie générale

[30] S. ABRAMSKY. *Observational Equivalence as a Testing Equivalence.* in « Theoretical Computer Science », numéro 3, volume 53, 1987.

[31] S. BALEMI, G. J. HOFFMANN, H. WONG-TOI, G. F. FRANKLIN. *Supervisory Control of a Rapid Thermal Multiprocessor.* in « IEEE Transactions on Automatic Control », numéro 7, volume 38, july, 1993, pages 1040-1059.

[32] F. BOURDONCLE. *Sémantique des langages impératifs d'ordre supérieur et interprétation abstraite.* thèse de doctorat, Ecole Polytechnique, Paris, 1992.

[33] M. BOZGA, S. GRAF, L. MOUNIER. *Automated validation of distributed software using the IF environment.* in « Workshop on Software Model-Checking, associated with CAV'01 (Paris, France) », série ENTCS, volume 55, Elsevier Science Publishers, éditeurs S. C. STOLLER, W. VISSER., jul, 2001.

[34] R. D. BRANDT, V. K. GARG, R. KUMAR, F. LIN, S. I. MARCUS, W. M. WONHAM. *Formulas for Calculating Supremal and Normal Sublanguages.* in « Systems and Control Letters », numéro 8, volume 15, 1990, pages 111-117.

[35] E. BRINSKMA. *A Theory for the Derivation of Tests.* in « Protocol Secification, Testing and Verification VIII », North-Holland, pages 63-74, 1988.

[36] D. CLARKE, T. JÉRON, V. RUSU, E. ZINOVIEVA. *Automated Test and Oracle Generation for Smart-Card Applications.* in « International Conference on Research in Smart Cards (e-Smart'01) », série LNCS 2140, Springer Verlag, pages 58-70, 2001.

[37] P. COUSOT, R. COUSOT. *Static determination of dynamic properties of programs.* in « 2nd Int. Symp. on Programming », Dunod, Paris, 1976.

[38] P. COUSOT, R. COUSOT. *Abstract intrepretation : a unified lattice model for static analysis of programs by construction or approximation of fixpoints.* in « Conference Record of the 4th ACM Symposium on Principles of Programming Languages », pages 238-252, Los Angeles, CA, janvier, 1977.

[39] P. COUSOT, N. HALBWACHS. *Automatic discovery of linear restraints among variables of a program.* in « 5th ACM Symposium on Principles of Programming Languages, POPL'78 », Tucson (Arizona), 1978.

[40] R. DE NICOLA, M. HENESSY. *Testing Equivalences for Processes.* in « Theoretical Computer Science », volume 34, 1984, pages 83-133.

[41] J.-C. FERNANDEZ, C. JARD, T. JÉRON, G. VIHO. *Using on-the-fly Verification Techniques for the Generation of Test Suites.* in « Conference on Computer-Aided Verification (CAV '96), New Brunswick, New Jersey, USA », série LNCS, volume 1102, Springer-Verlag, éditeurs A. ALUR, T. HENZINGER., juillet, 1996, Egalement disponible en rapport de recherche Irisa n° 1036.

[42] J.-C. FERNANDEZ, C. JARD, T. JÉRON, C. VIHO. *An Experiment in Automatic Generation of Test Suites for Protocols with Verification Technology.* in « Science of Computer Programming », numéro 29, 1997, pages 123-146.

[43] A. GILL. *Introduction to the Theory of Finite State Machine.* New-York, McGraw-Hill, 1962.

[44] S. GRAF, B. STEFFEN, G. LÜTTGEN. *Compositional Minimization of Finite State Systems Using Interface Specifications.* in « Formal Aspects of Computation », volume 8, septembre, 1996, A preliminary version appeared in CAV'90, LNCS 531.

[45] R. GROZ, T. JÉRON, A. KERBRAT. *Automated Test Generation from SDL specifications.* in « SDL'99 The Next Millenium, 9th SDL Forum, Montréal, Québec », Elsevier, éditeurs R. DSSOULI, G. VON BOCHMANN, Y. LAHAV., pages 135-152, juin, 1999.

[46] L. HOLLOWAY, B. KROGH, A. GIUA. *A survey of Petri Net Methods for controlled Discrete Event Systems.* in « Discrete Event Dynamic Systems : Theory and Application », volume 7, 1997, pages 151-190.

[47] A. ICHIKAWA, K. IRAISHI. *Analysis and control of discrete-events sytems represented by Petri nets.* in « Discrete Event Sytems : Models and Applications, IIASA conf. », Sopron, Hungary, august, 1987.

[48] ISO. *Information Technology - Open Systems Interconnection Conformance Testing Methodology and Framework.* International Standard ISO/IEC 9646, 1992.

[49] A. KHOUMSI. *A method for testing the conformance of real time systems.* in « Proc. 7th Intern. Symposium on Formal Techniques in Real-Time and Fault Tolerant Systems (FTRTFT) », Oldenburg, Germany, September, 2002, http://www.gel.usherb.ca/khoumsi/Research/Public/FTRTFT02.ps.

[50] A. KHOUMSI. *Supervisory control of dense real-time discrete-event systems with partial observation.* in « Proc. 6th Intern. Workshop on Discrete Event Systems (WODES) », Zaragoza, Spain, October, 2002, http://www.gel.usherb.ca/khoumsi/Research/Public/WODES02.ps.

[51] A. KHOUMSI, M. NOURELFATH. *An efficient method for the supervisory control of dense real-time discrete event systems.* in « Proc. 8th Intern. Conf. on Real-Time Computing Systems (RTCSA) », Tokyo, Japan, March, 2002, http://www.gel.usherb.ca/khoumsi/Research/Public/RTCSA02Cont.ps.

[52] LEE, YANNAKAKIS. *Testing Finite State Machines : State Identification and Verification.* in « IEEE Trans Computer », numéro 3, volume 43, 1994, pages 306-320.

[53] F. MARANINCHI, Y. RÉMOND. *Environnement Matou.*. http://www-verimag.imag.fr/PEOPLE/maraninx/MATOU/index.html.

[54] A. MINÉ. *The Octagon Abstract Domain.* in « AST 2001 in WCRE 2001 », série IEEE, IEEE CS Press, pages 310-319, October, 2001, http://www.di.ens.fr/~mine/publi/article-mine-padoII.pdf.

[55] E. F. MOORE. *Gedanken-experiments on sequential machines.* in « Automata Studies », 1956, pages 129-153.

[56] S. OWRE, J. RUSHBY, N. SHANKAR, F. VON HENKE. *Formal Verification for Fault-Tolerant Architectures : Prolegomena to the Design of PVS.* in « IEEE Transactions on Software Engineering », numéro 2, volume 21, 1995, pages 107-125.

[57] M. PHALIPPOU. *Relations d'Implantations et Hypothèses de Test sur les Automates à Entrées et Sorties.* thèse de doctorat, Université de Bordeaux, 1994.

[58] I. PHILLIPS. *Refusal testing.* in « Theoretical Computer Science », numéro 3, volume 50, 1987, pages 241-284.

[59] P. J. RAMADGE, W. M. WONHAM. *Modular Supervisory Control of Discrete Event Systems.* in « Proc. of 7th Int. Conf. Analysis and Optimization of Syst. », série LNCIS, volume 83, Springer-Verlag , Berlin, Germany, éditeurs A. BENSOUSSAN, J. L. LIONS., pages 202-214, Antibes, France, juin, 1986.

[60] P. J. RAMADGE, W. M. WONHAM. *The Control of Discrete Event Systems.* in « Proceedings of the IEEE ; Special issue on Dynamics of Discrete Event Systems », numéro 1, volume 77, 1989, pages 81-98.

[61] M. SAGIV, T. REPS, R. WILHELM. *Parametric shape analysis via 3-valued logic.* in « ACM Transactions on Programming Languages and Systems », numéro 3, volume 24, 2002.

[62] M. SAGIV, T. REPS, R. WILHELM. *Solving shape-analysis problems in languages with destructive updating.* in « ACM Transactions on Programming Languages and Systems », numéro 1, volume 20, 1998.

[63] R. SENGUPTA, S. LAFORTUNE. *An Optimal Control Theory for Discrete Event Systems.* in « SIAM Journal on Control and Optimization », numéro 2, volume 36, march, 1998.

[64] J. TRETMANS. *Testing Labelled Transition Systems with Inputs and Outputs.* in « 8th International Workshop on Protocols Test Systems », Evry - France, September, 1995.

[65] T. YOO, S. LAFORTUNE. *A general Architecture for decentralized supervisory Control of Discrete-Event Systems.* in « Proc of 5th Workshop on Discrete Event Systems, WODES 2000 », Ghent, Belgium, august, 2000.