INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Projet VASY

Validation de Systèmes

Rhône-Alpes

THÈME 1C

Table des matières

1 Composition de l'équipe

Responsable scientifique
Hubert Garavel [CR1 INRIA]

Assistantes de projet
Valérie Gardès [à partir du 1er janvier 2001]
Myriam Vergote [du 1er mai au 12 juillet 2001]

Personnel Inria
Radu Mateescu [CR2 INRIA]
Frédéric Lang [CR2 INRIA, à partir du 1er septembre 2001]

Personnel Bull
Solofo Ramangalahy [responsable d'action BULL, à partir du 1er juin 2001]
Nicolas Zuanon [responsable d'action BULL, jusqu'au 30 avril 2001]

Ingénieurs experts
Frédéric Lang [contrat FORMALCARD (BULL-SCHLUMBERGER), jusqu'au 31 août 2001]
Stéphane Martin [contrat REUTEL-2000 (ALCATEL), jusqu'au 31 janvier 2001]
Bruno Ondet [contrat FORMALFAME (BULL), à partir du 1er septembre 2001]
Frédéric Perret [contrat FORMALFAME (BULL)]
Frédéric Tronel [contrat RNTL PARFUMS, à partir du 1er octobre 2001]

Chercheur post-doctorant
Gordon Pace [boursier ERCIM, à partir du 15 octobre 2001]

Stagiaires
Adrian Curic [stagiaire DEA Université Joseph Fourier, jusqu'au 30 septembre 2001]
Christophe Joubert [stagiaire DEA Université Joseph Fourier, à partir du 1er octobre 2001]
Bruno Ondet [stagiaire DEA Université de Savoie, du 1er mars au 15 juillet 2001]
Gilles Stragier [stagiaire Université Libre de Bruxelles, à partir du 15 septembre 2001]

2 Présentation et objectifs généraux

2.1 Introduction

Créé le 1er janvier 2000 et faisant suite à l'action VASY "Recherche et Applications" (1er janvier 1997 – 31 décembre 1999), le projet VASY s'inscrit dans la problématique de la conception de systèmes sûrs par l'utilisation de méthodes formelles.

Plus précisément, nous nous intéressons à tout système (matériel, logiciel, télécommunications) faisant intervenir du parallélisme *asynchrone*, c'est-à-dire tout système dont on peut modéliser le comportement parallèle par une sémantique d'entrelacement des événements (*interleaving semantics*).

Pour la conception de systèmes sûrs, nous préconisons l'utilisation de techniques de des-

cription formelle, complétées par des outils informatiques adaptés, offrant des fonctionnalités de simulation, prototypage rapide, vérification et génération de tests.

Parmi les différentes approches existant pour la vérification, nous concentrons nos efforts sur la vérification "basée sur les modèles" (*model checking*) qui recouvre un grand nombre de techniques spécialisées (vérification énumérative, à la volée, symbolique, etc.). Ces techniques, bien que moins générales que les approches par preuves (*theorem proving*), possèdent pourtant l'avantage de permettre une détection automatique, rapide et économique des erreurs de conception dans des systèmes complexes.

Nos travaux se situent au confluent de deux grandes approches en méthodes formelles : l'approche basée sur des *modèles* (très répandue en Amérique du Nord) et l'approche basée sur des *langages* (plus développée en Europe) :

- Sous le terme de *modèles*, on désigne diverses représentations de programmes parallèles (automates, réseaux d'automates communicants, réseaux de Petri, diagrammes de décision binaire, etc.) ainsi que les algorithmes de vérification qui s'y appliquent. D'un point de vue théorique, il importe de rechercher des résultats généraux, donc indépendants de tout langage de description particulier, ce qui incite à la recherche de modèles mathématiques simples et expressifs.
- En pratique, ces modèles sont souvent trop rudimentaires pour servir à la description directe d'un système complexe (une telle approche est fastidieuse et comporte un fort risque d'erreur). C'est pourquoi il est indispensable de s'appuyer sur des formalismes de plus haut niveau (c'est-à-dire des *langages*) permettant de décrire des problèmes réels et complexes sous forme de programmes. Ces programmes sont ensuite analysés et traduits automatiquement vers des modèles sur lesquels opèrent les algorithmes de vérification.

Pour mener à bien la vérification de systèmes complexes (de taille "industrielle"), il nous semble nécessaire de maîtriser simultanément la technologie des modèles et celle des langages.

2.2 Technologie des modèles – vérification

Par vérification, on entend la comparaison — à un certain niveau d'abstraction — d'un système avec les *propriétés* qui décrivent le fonctionnement attendu du système (c'est-à-dire les services que celui-ci doit fournir).

Les techniques de vérification que nous mettons en œuvre reposent en grande partie sur le modèle des *systèmes de transitions étiquetées* (ou, plus simplement, *automates*, ou encore *graphes*) composés d'un ensemble d'états, d'un état initial et d'une relation de transition entre ces états. Ces techniques consistent à engendrer automatiquement, à partir de la description du système à vérifier, un graphe fini qui en modélise le comportement, puis à vérifier les propriétés sur le graphe grâce à une procédure de décision.

Selon le formalisme utilisé pour exprimer les propriétés, on distingue deux approches :

- *Propriétés comportementales :* elles décrivent le fonctionnement du système sous forme d'automates (ou bien en utilisant des descriptions de plus haut niveau que l'on traduit ensuite en automates). Etant donné que le système à vérifier et ses propriétés comporte-

mentales peuvent tous deux être représentés par des automates, la vérification consiste à les comparer au moyen de *relations d'équivalence ou de préordre*.
Concernant la vérification de propriétés comportementales, nous développons un outil de minimisation pour la bisimulation forte et la bisimulation de branchement, qui permet aussi la minimisation de systèmes stochastiques et probabilistes. En outre, nous collaborons avec d'autres équipes qui développent des outils basés sur les relations d'équivalence et de préordre.

– *Propriétés logiques :* elles caractérisent des propriétés essentielles du système, telles que l'absence de blocage, l'exclusion mutuelle ou l'équité. Parmi les formalismes utilisés, les *logiques temporelles* et le *μ-calcul modal* s'avèrent bien adaptés pour décrire l'évolution du système dans le temps. Dans ce cas, la vérification consiste à s'assurer que l'automate modélisant le système à vérifier satisfait un ensemble de propriétés logiques.
Concernant la vérification de propriétés logiques, nos travaux dans ce domaine portent sur l'évaluation efficace du μ-calcul modal et sur son extension par des variables typées, afin de prendre en compte les données contenues dans les états et les transitions du graphe. Cette extension (dont nous avons mis en évidence l'utilité sur de nombreux exemples, notamment industriels) permet d'exprimer des propriétés qu'il n'est pas possible d'écrire en μ-calcul standard comme, par exemple, le fait qu'une variable donnée soit toujours croissante sur un chemin d'exécution. Nous travaillons aussi à la conception et à l'implémentation d'algorithmes d'évaluation efficaces pour cette extension du μ-calcul.

Bien que ces techniques soient efficaces et complètement automatisables, leur principale limitation réside dans le problème de l'*explosion d'états*, qui survient lorsque le nombre d'états du système à vérifier dépasse les capacités en mémoire de la machine. C'est pourquoi nous fournissons des technologies logicielles (voir § 4.1) permettant de manipuler ces graphes de deux manières :

– soit sous forme *explicite*, en gardant en mémoire l'ensemble des états et des transitions (vérification *énumérative*) ;
– soit sous forme *implicite*, en explorant dynamiquement les parties du graphe en fonction des besoins (vérification *à la volée*).

2.3 Technologie des langages – compilation

En ce qui concerne les langages, il nous semble essentiel de s'appuyer sur des langages possédant simultanément un *caractère exécutable* et une *sémantique formelle*, ceci pour plusieurs raisons :

– Les techniques de *model checking* nécessitent de pouvoir exécuter efficacement les programmes à vérifier.
– La modélisation de systèmes critiques ne saurait reposer sur des langages dont la sémantique ne serait pas rigoureusement définie, car cela conduit bien souvent à des ambiguïtés et des divergences d'interprétation (notamment entre concepteurs et implémenteurs).

– En général, les techniques de *model checking* ne peuvent garantir la correction d'un système infini, puisqu'elles ne peuvent vérifier que des abstractions finies de ce système. C'est pourquoi on doit utiliser aussi des techniques de preuve, lesquelles ne s'appliquent qu'aux langages ayant une sémantique formelle.

Dans ce contexte, nos travaux actuels portent sur plusieurs langages :

– Nous nous intéressons depuis longtemps au langage LOTOS, le seul langage de description de protocoles ayant le statut de norme internationale [ISO88] et possédant les propriétés ci-dessus. Il s'agit d'un langage basé sur les concepts des algèbres de processus (notamment CCS [Mil89] et CSP [Hoa85]) pour la description du contrôle et les types abstraits algébriques [EM85] pour la description des données. LOTOS autorise à la fois la description du parallélisme asynchrone (aspects liés à la répartition, la synchronisation et la communication entre tâches) et celle des structures de données complexes manipulées dans les protocoles et les systèmes distribués.
Nous utilisons LOTOS pour diverses études de cas industrielles et nous développons des outils logiciels pour ce langage dans le cadre de la boîte à outils CADP (voir § 4.1).
– Les algèbres de processus sont des formalismes particulièrement bien adaptés à la spécification des protocoles de télécommunication et des systèmes répartis. Cependant, en dépit d'une base mathématique rigoureuse, d'efforts de normalisation (notamment ceux concernant le langage LOTOS) et d'un nombre croissant d'études de cas traitées avec succès, les algèbres de processus ne sont pas encore pleinement acceptées en milieu industriel. Elles se voient parfois supplantées par des langages dont l'apparence plus conviviale (syntaxe graphique ou proche des langages algorithmiques classiques) masque une absence de sémantique formelle assez préoccupante lorsqu'il s'agit de modéliser et de valider des systèmes critiques.
Les besoins en méthodes formelles et vérification allant en croissant, il est nécessaire de réfléchir à de nouveaux langages qui combineraient les fondements théoriques rigoureux et l'expressivité des algèbres de processus avec une simplicité d'utilisation permettant d'assurer une meilleure diffusion industrielle.
Cette réflexion est également guidée par l'apparition de protocoles à contraintes temporelles fortes — protocoles utilisés dans les réseaux à haut débit — pour lesquels les aspects temporels doivent être pris en compte de manière quantitative, et non plus seulement qualitative.
Nous travaillons sur ces questions, notamment dans le cadre de la révision de la norme LOTOS entreprise à l'ISO depuis 1992 ; cette révision a donné naissance en 2001 à la norme internationale E-LOTOS (*Enhanced*-LOTOS) [ISO01] qui vise à conjuguer une expressivité

[ISO88] ISO/IEC, « LOTOS — A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour », *International Standard n° 8807*, International Organization for Standardization — Information Processing Systems — Open Systems Interconnection, Genève, septembre 1988.

[Mil89] R. MILNER, *Communication and Concurrency*, Prentice-Hall, 1989.

[Hoa85] C. A. R. HOARE, *Communicating Sequential Processes*, Prentice-Hall, 1985.

[EM85] H. EHRIG, B. MAHR, *Fundamentals of Algebraic Specification 1 — Equations and Initial Semantics*, *EATCS Monographs on Theoretical Computer Science*, *6*, Springer Verlag, 1985.

[ISO01] ISO/IEC, « Enhancements to LOTOS (E-LOTOS) », *International Standard n° 15437:2001*, In-

sémantique accrue (par exemple, avec l'introduction du temps quantifié) et une facilité d'apprentissage pour des non-experts. L'historique de nos contributions à l'ISO est disponible sur notre serveur Web, à l'adresse http://www.inrialpes.fr/vasy/elotos.
En parallèle, nous étudions une variante simplifiée du langage E-LOTOS, appelée LOTOS NT (LOTOS *Nouvelle Technologie*) [4], dans laquelle nous avons introduit certains concepts qui nous semblent pertinents (ce qui n'est pas toujours chose aisée dans une norme internationale).
Comme E-LOTOS, LOTOS NT se compose de trois parties : une *partie données*, qui permet une description naturelle des types de données et des fonctions tout en étant facilement analysable et implémentable, une *partie contrôle*, qui étend l'algèbre de processus de LOTOS par des constructions plus expressives et la prise en compte du temps quantitatif, et des *modules*, qui autorisent la structuration et la réutilisation des descriptions LOTOS NT.
La différence essentielle entre les deux langages réside dans le fait que LOTOS NT est un langage impératif alors que E-LOTOS s'inscrit dans un cadre fonctionnel. De plus, LOTOS NT se distingue d'E-LOTOS sur certains aspects (typage statique, surcharge d'opérateurs, tableaux) qui en font un langage plus facile à utiliser et plus simple à implémenter.
Nous travaillons sur l'implémentation de LOTOS NT pour lequel nous développons le compilateur TRAIAN (voir § 4.2).

2.4 Implémentation et expérimentation

Dans la mesure du possible, nous essayons de valider nos propositions par le développement d'outils et l'application de ces outils à des études de cas complexes, souvent industrielles. Cette confrontation systématique avec les problèmes d'implémentation et d'expérimentation est un aspect essentiel de notre approche.

3 Domaines d'applications

Les modèles théoriques que nous utilisons (automates, algèbres de processus, bisimulations, logiques temporelles) et les logiciels que nous développons sont suffisamment généraux pour ne pas dépendre trop étroitement d'un seul secteur applicatif.

Nos méthodes peuvent s'appliquer à tout système ou protocole composé d'agents distribués communiquant par messages. Ce cadre conceptuel trouve de nombreuses incarnations dans le domaine du logiciel, du matériel et des télécommunications. Les études de cas conduites ces dernières années avec la boîte à outils CADP (voir notamment § 5.3.3) illustrent bien cette diversité applicative :

– *architectures matérielles :* circuits asynchrones, arbitrage de bus, cohérence de caches, conception conjointe matériel-logiciel ;

ternational Organization for Standardization — Information Technology, Genève, septembre 2001.

- *bases de données :* protocoles transactionnels, bases de connaissances distribuées, gestion de stocks ;
- *électronique grand public :* télécommandes audiovisuelles, vidéo à la demande, bus FIREWIRE, réseaux locaux domestiques ;
- *protocoles de sécurité :* authentification, commerce électronique, distribution de clés cryptographiques ;
- *systèmes embarqués :* applications sur cartes à puce, contrôle de trafic aérien ;
- *systèmes répartis :* mémoire virtuelle, systèmes de fichiers répartis, algorithmes d'élection, de reconfiguration dynamique et de tolérance aux pannes ;
- *télécommunications :* réseaux à haut débit, administration de réseaux, téléphonie mobile, interactions de services téléphoniques ;
- *interactions homme-machine :* interfaces graphiques, visualisation de données biomédicales, etc.

4 Logiciels

4.1 La boîte à outils CADP

Participants : Hubert Garavel [correspondant], Frédéric Lang, Stéphane Martin, Radu Mateescu, Frédéric Perret.

Mots clés : application critique, application répartie, compilation, concurrence, génération de code, génie logiciel, logique temporelle, méthodes formelles, modélisation, mu-calcul, parallélisme asynchrone, protocole de communication, spécification formelle, synchronisation, système distribué, vérification de programme.

En collaboration avec le laboratoire VERIMAG, nous développons la boîte à outils CADP (CÆSAR/ALDÉBARAN *Development Package*) pour l'ingénierie des protocoles et des systèmes distribués [1, 7, 2] (voir `http://www.inrialpes.fr/vasy/cadp`). Au sein de cette boîte à outils, nous avons en charge les logiciels suivants :

- CÆSAR [5, 3] est un compilateur qui produit, à partir d'un programme LOTOS, du code exécutable ou des modèles sur lesquels différentes méthodes de vérification peuvent être appliquées. Le programme source LOTOS est traduit successivement en une algèbre de processus simplifiée, un réseau de Petri étendu avec des variables et des transitions atomiques, et, finalement, un système de transitions étiquetées obtenu par simulation exhaustive du réseau de Petri.
- CÆSAR.ADT [6] est un compilateur qui traduit les définitions de types abstraits LOTOS vers des bibliothèques de types et de fonctions en langage C. La traduction met en œuvre un algorithme de compilation par filtrage et des techniques pour la reconnaissance des classes de types usuels (nombres entiers, énumérations, tuples, listes, etc.) qui sont identifiées automatiquement et implémentées de manière optimale.
- OPEN/CÆSAR [8] est un environnement logiciel extensible permettant de construire des

outils de simulation, de vérification et de génération de tests sur des graphes représentés sous forme implicite. Ces outils peuvent être développés de manière simple, modulaire et indépendante du langage utilisé pour décrire les systèmes à valider. De ce point de vue, l'environnement OPEN/CÆSAR est l'un des constituants essentiels de la boîte à outils CADP, puisqu'il effectue la jonction entre les outils dédiés aux langages et les outils opérant sur les modèles. L'environnement OPEN/CÆSAR comprend un ensemble de bibliothèques avec leurs interfaces de programmation, ainsi que divers outils parmi lesquels :

- EVALUATOR [10, 12], qui évalue à la volée des formules de μ-calcul régulier d'alternance 1,
- EXECUTOR, qui permet l'exécution aléatoire,
- EXHIBITOR, qui recherche à la volée des séquences d'exécution caractérisées par une expression régulière,
- GENERATOR et REDUCTOR, qui construisent le graphe des états accessibles,
- SIMULATOR, XSIMULATOR et OCIS, qui permettent la simulation interactive, et
- TERMINATOR, qui recherche les états de blocage.

– BCG (*Binary Coded Graphs*) est un format qui utilise des techniques efficaces de compression permettant de stocker des graphes (représentés sous forme explicite) sur disque de manière très compacte. Ce format joue un rôle central dans la boîte à outils CADP. Il est indépendant du langage source et des outils de vérification. En outre, il contient suffisamment d'informations pour que les outils qui l'exploitent puissent fournir à l'utilisateur des diagnostics précis dans les termes du programme source. Pour exploiter le format BCG, nous développons un environnement logiciel qui se compose de bibliothèques avec leurs interfaces de programmation et de plusieurs outils, notamment :

- BCG_DRAW, qui construit et affiche une représentation 2D en POSTSCRIPT d'un graphe,
- BCG_EDIT, qui permet de modifier interactivement la représentation graphique produite par BCG_DRAW,
- BCG_INFO, qui affiche diverses informations statistiques concernant un graphe,
- BCG_IO, qui effectue des conversions entre BCG et divers autres formats de graphes,
- BCG_LABELS, qui permet de masquer et/ou de renommer par des expressions régulières les étiquettes d'un graphe,
- BCG_MIN, qui permet de minimiser un graphe selon la bisimulation forte ou la bisimulation de branchement (éventuellement étendue au cas des systèmes probabilistes ou stochastiques), et
- BCG_OPEN, qui permet d'appliquer à tout graphe BCG les outils disponibles dans l'environnement OPEN/CÆSAR.

– XTL (*eXecutable Temporal Language*) [11, 9] est un langage adapté à l'expression des algorithmes d'évaluation et de diagnostic pour les formules de logiques temporelles telles que HML [HM85], CTL [CES86], ACTL [NV90], etc. D'inspiration fonctionnelle, ce langage

[HM85] M. HENNESSY, R. MILNER, « Algebraic Laws for Nondeterminism and Concurrency », *Journal of the ACM 32*, 1985, p. 137–161.

[CES86] E. M. CLARKE, E. A. EMERSON, A. P. SISTLA, « Automatic Verification of Finite-State Concurrent

offre des primitives d'accès à toutes les informations contenues dans les graphes BCG : états, étiquettes des transitions, fonctions *successeurs* et *prédécesseurs*, ainsi qu'aux types et fonctions du programme source. Il permet la définition de fonctions récursives qui servent à calculer des prédicats de base et des modalités temporelles caractérisant des ensembles d'états et de transitions.
- SVL (*Script Verification Language*) est un langage permettant d'exprimer de manière simple des scénarios de vérification élaborés, lesquels seront exécutés en appelant, dans un ordre approprié, les différents outils de CADP avec les paramètres adéquats.

A ces outils s'ajoutent ceux développés par le laboratoire VERIMAG et l'ex-projet PAMPA (Rennes) :
- ALDÉBARAN effectue la comparaison et la minimisation de graphes selon diverses relations d'équivalence ou de préordre,
- EXP.OPEN et PROJECTOR calculent des produits et des abstractions d'automates communicants,
- TGV (*Test Generation based on Verification*) engendre automatiquement des tests de conformité en fonction d'objectifs de tests définis par l'utilisateur.

Tous ces outils — ainsi que d'autres développés par l'ex-projet MEIJE (Sophia-Antipolis) et les Universités de Liège et d'Ottawa — sont intégrés au sein de l'interface graphique EUCALYPTUS (développée en TCL/TK) qui offre un accès facile et uniforme aux différents outils, en masquant à l'utilisateur les conventions d'appel et les formats spécifiques à chaque outil.

Dans la compétition actuelle entre les différents langages, méthodologies et outils proposés pour la spécification et la vérification formelle, la boîte à outils CADP possède plusieurs avantages : elle s'appuie sur un langage normalisé, comporte des outils robustes (bien que perfectibles) et regroupe une communauté importante d'utilisateurs.

4.2 Le compilateur TRAIAN

Participants : Hubert Garavel [correspondant], Frédéric Lang.

Mots clés : application critique, application répartie, compilation, concurrence, génération de code, génie logiciel, méthodes formelles, modélisation, parallélisme asynchrone, protocole de communication, spécification formelle, synchronisation, système distribué, vérification de programme.

Pour le langage LOTOS NT, nous développons un compilateur, appelé TRAIAN, dont le but est de traduire automatiquement une description LOTOS NT vers un programme C pouvant ensuite être utilisé à des fins de simulation, de prototypage rapide, de vérification et de test.

La version actuelle de TRAIAN effectue l'analyse lexicale et syntaxique, la construction des

Systems using Temporal Logic Specifications », *ACM Transactions on Programming Languages and Systems 8*, 2, avril 1986, p. 244–263.

[NV90] R. D. NICOLA, F. W. VAANDRAGER, *Action versus State Based Logics for Transition Systems*, *Lecture Notes in Computer Science*, *469*, Springer Verlag, 1990, p. 407–419.

arbres de syntaxe abstraite, les vérifications de sémantique statique et la génération de code C pour les définitions de types et de fonctions contenues dans les descriptions LOTOS NT.

Bien que cette version de TRAIAN soit encore incomplète (elle ne traite pas les définitions de processus LOTOS NT), elle a d'ores et déjà trouvé un champ d'application original et utile : la construction de compilateurs [16]. L'approche suivie consiste à utiliser conjointement l'outil SYNTAX développé à l'INRIA Rocquencourt (pour les aspects lexicaux et syntaxiques) et le langage LOTOS NT (pour les aspects sémantiques, notamment la définition, la construction et le parcours des arbres abstraits). Il est également possible de programmer directement en langage C certaines parties du compilateur, ce qui offre une souplesse appréciable pour certains traitements spécifiques.

C'est ainsi que tous les compilateurs récents de l'équipe VASY — notamment EVALUATOR 4.0 (voir § 5.1.3), SVL 2.0 (voir § 5.1.4) et NTIF (voir § 5.2.1) — comportent une forte proportion de code LOTOS NT qui est ensuite traduit en code C par le compilateur TRAIAN. L'utilisation combinée des technologies SYNTAX, LOTOS NT et TRAIAN, donne entière satisfaction, tant pour la qualité des compilateurs produits que pour la facilité et la rapidité du développement lui-même.

Le compilateur TRAIAN est diffusé sur Internet : il existe une page Web qui lui est consacrée (voir `http://www.inrialpes.fr/vasy/traian`) à partir de laquelle on peut télécharger librement le compilateur.

5 Résultats nouveaux

5.1 Technologie des modèles – vérification

Mots clés : automate, bisimulation, compilation, concurrence, génération de code, génie logiciel, logique temporelle, méthodes formelles, modélisation, mu-calcul, parallélisme asynchrone, protocole de communication, spécification formelle, synchronisation, système distribué, vérification de programme.

Résumé : *En 2001, nos travaux sur la vérification ont porté sur l'extension et l'amélioration d'outils existants, ainsi que sur le développement de nouveaux outils visant à faciliter l'utilisation des techniques formelles en milieu industriel.*

5.1.1 Développement de l'outil BCG_MIN

Participants : Hubert Garavel, Frédéric Perret.

En étroite collaboration avec Holger Hermanns (Université de Twente, Pays-Bas), nous avons poursuivi le développement, entrepris en 1999, de l'outil de minimisation de graphes BCG_MIN. Cet outil traite trois sortes de graphes (tous encodés dans le format BCG) :

- des systèmes de transitions "ordinaires", tels que ceux produits à partir de descriptions LOTOS ;
- des systèmes de transitions "probabilistes" dont chaque transition peut être étiquetée, soit par une action a "ordinaire", soit par une probabilité $p \in [0,1]$, soit par un couple (a,p) ;
- des systèmes de transitions "stochastiques" dont chaque transition peut être étiquetée, soit par une action a "ordinaire", soit par un paramètre réel λ qui détermine une loi de distribution exponentielle donnée par $prob(x > t) = e^{-\lambda t}$, soit par un couple (a,λ).

De par l'existence de ces différents types de transitions, les modèles acceptés par BCG_MIN sont suffisamment généraux pour couvrir de nombreux modèles probabilistes[1] et stochastiques[2].

Pour minimiser les systèmes de transitions ordinaires, BCG_MIN implémente l'algorithme de Kanellakis et Smolka [KS90] pour la bisimulation forte et l'algorithme de Groote et Vaandrager [GV90] pour la bisimulation de branchement. Pour les systèmes de transitions probabilistes et stochastiques, BCG_MIN implémente l'algorithme de Hermanns et Siegle [HS99].

BCG_MIN fait partie de la boîte à outils CADP et ses performances sont reconnues : ainsi, il est cité dans [GvdP00] comme "*the best implementation of the standard algorithm for the branching bisimulation*". Concrètement, BCG_MIN permet de minimiser efficacement des graphes de grande taille que d'autres outils tels qu'ALDÉBARAN et FC2MIN ne parviennent pas à traiter par manque de mémoire. Par ailleurs, l'utilisation du format très compact BCG entraîne des gains à la fois en place disque et en rapidité.

En 2001, les travaux relatifs à BCG_MIN ont porté sur plusieurs points :

- Nous avons entrepris d'optimiser BCG_MIN afin de traiter des graphes de taille plus importante. Le plus grand graphe minimisé par BCG_MIN en 2000 comportait 6 millions

[1] *Discrete Time Markov Chains, Discrete Time Markov Reward Models, Alternating Probabilistic LTS, Discrete Time Markov Decision Processes, Generative Probabilistic LTS, Reactive Probabilistic LTS, Stratified probabilistic LTS.*

[2] *Continuous Time Markov Chains, Continuous Time Markov Reward Models, Continuous Time Markov Decision Processes, Interactive Markov Chains, Timed Processes for Performance (TIPP) Models, Performance Evaluation Process Algebra (PEPA) Models, Extended Markovian Process Algebra (EMPA) Models.*

[KS90] P. C. KANELLAKIS, S. A. SMOLKA, « CCS expressions, finite state processes, and three problems of equivalence », *Information and Computation 86*, 1, mai 1990, p. 43–68.

[GV90] J. GROOTE, F. VAANDRAGER, « An Efficient Algorithm for Branching Bisimulation and Stuttering Equivalence », *in : Proceedings of the 17th ICALP (Warwick)*, M. S. Patterson (éditeur), *Lecture Notes in Computer Science*, *443*, Springer Verlag, p. 626–638, 1990.

[HS99] H. HERMANNS, M. SIEGLE, « Bisimulation Algorithms for Stochastic Process Algebras and their BDD-based Implementation », *in : Proceedings of the 5th International AMAST Workshop ARTS'99 (Bamberg, Germany)*, J.-P. Katoen (éditeur), *Lecture Notes in Computer Science*, *1601*, Springer Verlag, p. 244–265, mai 1999.

[GvdP00] J. GROOTE, J. VAN DE POL, « State space reduction using partial τ-confluence », *in : Proceedings of the 25th International Symposium on Mathematical Foundations of Computer Science MFCS'2000 (Bratislava, Slovakia)*, M. Nielsen, B. Rovan (éditeurs), *Lecture Notes in Computer Science*, *1893*, Springer Verlag, p. 383–393, Berlin, août 2000. Paru également comme rapport technique CWI SEN-R0008, Amsterdam, mars 2000.

d'états et 11 millions de transitions, mais des limitations en mémoire empêchaient d'aller au-delà.
Après un examen attentif des algorithmes et des structures de données utilisés dans BCG_MIN, nous avons mis en œuvre diverses optimisations destinées à réduire la consommation en mémoire.
Ainsi, la taille mémoire allouée pour une transition est passée de 20 à 16 octets. De même, la taille mémoire allouée pour un état est passée de 36 à 20 octets dans le cas des systèmes de transitions ordinaires, et de 44 à 28 octets dans le cas des systèmes de transitions probabilistes et stochastiques.
Ces progrès ont permis d'augmenter la taille des graphes traités par BCG_MIN. C'est ainsi que la nouvelle version de BCG_MIN a permis de minimiser un graphe de 8 millions d'états et 43 millions de transitions fourni par N. Zuanon dans le cadre de FORMALFAME (voir § 5.3.1 et 6.2).

– La base de tests développée pour BCG_MIN en 1999 et 2000 a été améliorée afin de permettre le passage automatique des tests de non-régression à chaque modification du code de BCG_MIN.
– Nous avons utilisé BCG_MIN pour la vérification fonctionnelle et l'évaluation de performances du protocole d'arbitrage du bus SCSI-2. Un article a été écrit et soumis à une conférence internationale.

5.1.2 Développement de l'outil BISIMULATOR

Participant : Radu Mateescu.

L'utilisation de relations de bisimulation (équivalence forte, équivalence de branchement, équivalence observationnelle, etc.) pour la vérification comporte deux aspects complémentaires : la *minimisation* d'un graphe — telle qu'effectuée notamment par l'outil BCG_MIN (voir § 5.1.1) — et la *comparaison* de deux graphes selon une certaine relation d'équivalence.

C'est ce second aspect que nous avons abordé en 2001, en développant un nouvel outil appelé BISIMULATOR. Cet outil prend en entrée deux graphes à comparer, l'un représenté implicitement au moyen de l'environnement OPEN/CÆSAR [8], l'autre représenté explicitement sous forme de fichier BCG. BISIMULATOR permet de déterminer, soit si ces deux graphes sont équivalents au sens de la bisimulation forte, soit si l'un d'eux est inclus dans l'autre au sens du préordre associé à la bisimulation forte.

BISIMULATOR fonctionne *à la volée*, c'est-à-dire qu'il explore dynamiquement, au fur et à mesure des besoins, les parties utiles du graphe représenté implicitement, sans nécessiter de construire préalablement la totalité de ce graphe, ce qui pourrait s'avérer prohibitif.

De plus, l'utilisation des environnements génériques BCG et OPEN/CÆSAR rend BISIMULATOR totalement indépendant du langage source utilisé pour décrire les deux graphes à comparer. Ceci constitue un progrès significatif par rapport à des outils plus anciens — tels qu'ALDÉBARAN et FC2IMPLICIT — qui implémentent également certains algorithmes de

comparaison à la volée, mais de manière limitée, puisqu'ils imposent au graphe implicite d'être fourni sous la forme d'un produit d'automates communicants, alors que BISIMULATOR peut s'appliquer directement à un programme écrit dans un langage de haut niveau (par exemple, LOTOS).

L'approche utilisée dans BISIMULATOR consiste à formuler le problème de la comparaison de deux graphes en termes d'un système d'équations booléennes. Comparer les graphes à la volée revient alors à effectuer la résolution locale du système booléen, c'est-à-dire à calculer la valeur de vérité de la variable représentant l'équivalence des états initiaux des deux graphes.

BISIMULATOR comporte environ 2 500 lignes de code C et il réutilise les fonctionnalités de résolution de systèmes booléens avec production de diagnostics développées en 1999–2000 pour l'outil EVALUATOR 3.0 [10, 12]. Ces fonctionnalités ont été regroupées dans une bibliothèque générique appelée CAESAR_SOLVE (1 500 lignes de code C) qui est utilisée conjointement par EVALUATOR 3.0 et par BISIMULATOR.

Les expérimentations que nous avons effectuées sur des graphes allant jusqu'à 10 000 états montrent une vitesse d'exécution de BISIMULATOR environ 25 fois supérieure à celle des algorithmes de comparaison à la volée pour la bisimulation forte implémentés dans l'outil ALDÉBARAN. Nous n'avons pas pu poursuivre plus avant les expérimentations car, au-delà de 10 000 états, les temps d'exécution des algorithmes à la volée deviennent prohibitifs.

Une fonctionnalité importante de BISIMULATOR consiste en la production de diagnostics expliquant pourquoi deux graphes sont équivalents ou inclus l'un dans l'autre (*exemples*) ou pourquoi ils ne le sont pas (*contre-exemples*). En pratique, ces diagnostics ont diverses utilités :

- Les exemples produits par BISIMULATOR pour l'inclusion au sens du préordre de la bisimulation forte servent à améliorer les fonctionnalités du simulateur graphique OCIS, en lui permettant de relire interactivement les parties de graphes (traces d'exécution, scénarios, diagnostics, etc.) produites par d'autres outils de CADP tels que EXECUTOR, EXHIBITOR, EVALUATOR 3.0, etc.
- Les contre-exemples produits par BISIMULATOR permettent à l'utilisateur de comprendre pourquoi deux graphes ne sont pas équivalents ou inclus : ces contre-exemples sont des graphes acycliques rassemblant toutes les séquences de transitions qui, exécutées simultanément dans les deux graphes, mènent à des états non équivalents. De ce fait, les contre-exemples de BISIMULATOR sont (beaucoup) plus compacts que ceux d'ALDÉBARAN, qui se limitent à un ensemble de séquences d'exécution construites séparément.

Nous envisageons d'étendre BISIMULATOR pour traiter le cas des bisimulations faibles (équivalence observationnelle, équivalence de branchement, etc.), ce que permet l'approche basée sur les systèmes d'équations booléennes.

5.1.3 Développement de l'outil EVALUATOR 4.0

Participant : Radu Mateescu.

La boîte à outils CADP comporte actuellement deux évaluateurs de logique temporelle : XTL [9], qui permet de vérifier de façon énumérative des formules temporelles comportant des données et EVALUATOR 3.0 [10, 12], qui permet de vérifier à la volée des formules du μ-calcul régulier d'alternance 1 sans données.

Ces deux logiciels sont stables et robustes. Ils ont été utilisés avec succès pour valider 16 applications critiques, notamment en milieu industriel (par exemple, par BULL et ERICSSON). En 2001, ils ont été déposés à l'Agence pour la Protection des Programmes (APP).

Toutefois, dans un souci d'ergonomie, il serait souhaitable de proposer aux utilisateurs de CADP un seul outil d'évaluation de logique temporelle qui réunirait l'ensemble des fonctionnalités actuellement fournies par XTL et EVALUATOR 3.0 de manière séparée. C'est pourquoi, nous avons entrepris, en 2000, un travail d'unification d'XTL et d'EVALUATOR 3.0 qui devrait, à terme, aboutir à un outil de vérification unique, appelé EVALUATOR 4.0. A notre connaissance, il n'existe à l'heure actuelle aucun outil de vérification réunissant toutes ces fonctionnalités.

Cet outil permettra de vérifier à la volée des propriétés temporelles comportant des données. Le langage d'entrée d'EVALUATOR 4.0 sera basé sur des formules d'un μ-calcul régulier d'alternance 1 étendu avec des variables typées ; comme XTL, il offrira des primitives de manipulation des états et des transitions dans les formules logiques, permettant de définir des propriétés temporelles non-standard (comme par exemple le fait qu'un état possède une transition vers lui-même).

En 2001, nous avons poursuivi les travaux sur EVALUATOR 4.0 dans les directions suivantes :

- Suite au retour d'utilisation d'EVALUATOR 3.0 par BULL dans le cadre de FORMALFAME (voir § 5.3.1 et 6.2), nous avons étendu le langage d'entrée d'EVALUATOR 4.0 avec de nouveaux opérateurs réguliers permettant d'exprimer la répétition bornée (entre n_1 et n_2 fois, où n_1 et n_2 sont des constantes entières) d'une séquence régulière de transitions. Combinés avec les modalités de possibilité et de nécessité, ces nouveaux opérateurs autorisent une description concise des propriétés du type "*chaque action d'émission sera suivie, après 15 transitions maximum, d'une action de réception*". Nous avons défini la sémantique de ces opérateurs en les traduisant vers les opérateurs de point fixe paramétrés déjà présents dans le langage d'entrée d'EVALUATOR 4.0.
- Nous avons continué le développement de la "partie avant" d'EVALUATOR 4.0 qui effectue l'analyse syntaxique et sémantique du langage d'entrée (liaison des identificateurs, typage des expressions et des formules, résolution des surcharges), ainsi que diverses transformations sémantiques sur les formules temporelles. Développée à l'aide du générateur de compilateurs SYNTAX et du compilateur TRAIAN (voir § 4.2), la partie avant d'EVALUATOR 4.0 comporte désormais 15 000 lignes de code SYNTAX, LOTOS NT et C.

En 2001, nos efforts ont porté sur la traduction des formules temporelles en systèmes d'équations booléennes paramétrées par des variables typées. Nous traduisons d'abord ces formules vers des systèmes d'équations modales récursives ayant des variables propositionnelles en partie gauche et des formules PDL [FL79] en partie droite. Après avoir transformé ces systèmes afin que les parties droites des équations contiennent au plus un opérateur booléen ou modal, nous les traduisons finalement vers des systèmes d'équations modales récursives ayant des formules HML [HM85] en partie droite.

– Enfin, nous avons poursuivi nos recherches théoriques afin d'optimiser l'évaluation à la volée des formules de logique temporelle dans le cas particulier où le système de transitions à vérifier est un graphe sans circuit. Ce cas particulier présente un intérêt pratique, notamment pour valider des séquences d'exécution finies (produites en laissant s'exécuter un système réel) à l'aide de formules de logique temporelle ; cette approche est utilisée dans le cadre de FORMALFAME pour vérifier la correction des traces de grande taille (comportant des dizaines ou des centaines de milliers de transitions) obtenues par test aléatoire de l'architecture multi-processeur FAME développée par BULL.
Nous avons établi que, dans le cas des graphes sans circuit, le μ-calcul modal d'alternance quelconque possède la même expressivité que son fragment d'alternance 1. Pour démontrer ce résultat, nous avons proposé une traduction du μ-calcul modal vers le μ-calcul gardé (formules dans lesquelles tout appel récursif d'une variable propositionnelle est précédé par une modalité), puis un algorithme d'évaluation basé sur une réduction sémantique des formules du μ-calcul gardé (équivalence des plus petits et plus grands points fixes) lorsqu'elles sont évaluées sur des graphes sans circuit.
Ce résultat permet de réduire fortement la complexité de la vérification : au lieu d'une complexité exponentielle comme dans le cas des graphes avec circuits, les formules de μ-calcul modal d'alternance quelconque peuvent être vérifiées avec une complexité linéaire en taille du graphe (nombre d'états et de transitions) et quadratique en taille de la formule (nombre d'opérateurs). En particulier, notre algorithme ne stocke en mémoire que les états visités, mais non les transitions. Ce travail a donné lieu à une publication [20].

5.1.4 Développement de l'outil SVL 2.0

Participants : Hubert Garavel, Frédéric Lang.

Entre 1997 et 2000, nous avions conçu un langage appelé SVL (*Script Verification Language*), ainsi que le compilateur associé à ce langage, dans le but de simplifier et d'automatiser la mise en œuvre des techniques de vérification compositionnelle et pour rendre celles-ci réellement utilisables dans un contexte industriel (et notamment par BULL, qui souhaitait les utiliser). En 2000, de nombreuses améliorations avaient été apportées au langage SVL et le compilateur avait été complètement réécrit.

[FL79] M. J. FISCHER, R. E. LADNER, « Propositional Dynamic Logic of Regular Programs », *Journal of Computer and System Sciences*, 18, 1979, p. 194–211.

[HM85] M. HENNESSY, R. MILNER, « Algebraic Laws for Nondeterminism and Concurrency », *Journal of the ACM 32*, 1985, p. 137–161.

Dans son principe, SVL se présente comme un langage de haut niveau pour l'écriture de scénarios de vérification. Ce langage permet de décrire l'architecture du système à vérifier sous forme d'un système de processus communicants connectés par des opérateurs algébriques de composition parallèle. SVL offre aussi des opérateurs additionnels permettant de spécifier facilement les différentes étapes (réductions, comparaisons, abstractions, etc.) de la vérification compositionnelle.

Le compilateur SVL traduit chaque scénario de vérification en un *shell script* UNIX contenant la séquence de commandes (appels aux différents outils de vérification) correspondant à l'exécution de ce scénario. Les outils de vérification peuvent appartenir soit à CADP (ALDÉBARAN, CÆSAR, BCG, EXP.OPEN, OPEN/CÆSAR, PROJECTOR, etc.), soit à la boîte à outils FC2TOOLS développée dans l'ex-projet MEIJE (INRIA Sophia-Antipolis). Cette traduction est transparente pour l'utilisateur qui n'a plus à se soucier de la syntaxe d'appel propre à chaque outil. En outre, le compilateur SVL prend en charge la gestion des nombreux fichiers intermédiaires (abstractions, renommages, synchronisations, etc.) nécessaires à la vérification compositionnelle.

A l'origine, SVL a été conçu spécifiquement pour la vérification compositionnelle, en remplacement de l'outil DES2AUT [KM97]. En 2001, nous avons entrepris d'appliquer SVL dans un cadre beaucoup plus large que celui de la vérification compositionnelle, en l'utilisant pour simplifier les études de cas fournies dans la boîte à outils CADP comme exemples de démonstration. C'est ainsi que 19 études de cas (sur un total de 29) ont été réécrites en SVL. Ce faisant, nous avons pris conscience que SVL pouvait jouer le rôle d'un puissant langage de *scripts* dédié à la vérification, remplaçant avantageusement les *Makefiles* et *shell scripts* utilisés jusqu'alors, améliorant la lisibilité et supprimant un grand nombre de fichiers auxiliaires (désormais engendrés automatiquement par SVL).

Ces multiples expérimentations nous ont conduit à améliorer la définition du langage SVL :

- De nouvelles instructions permettant la recherche des blocages et des bouclages infinis, ainsi que l'évaluation de formules de μ-calcul modal ont été intégrées au langage.
- De multiples améliorations de la syntaxe ont été mises en œuvre : de nouveaux symboles en notation infixée ont été introduits pour l'opérateur d'abstraction, d'autres opérateurs ont été renommés, et les règles de précédence, d'associativité et de portée des opérateurs unaires et binaires ont été ré-étudiées. Le nombre des modifications et le soin apporté à chacune d'elles ont conduit à un langage plus homogène, donc d'apprentissage et d'utilisation plus faciles.
- Il est désormais possible, dans une description SVL, de créer et de manipuler explicitement des fichiers représentant des réseaux d'automates communicants au format EXP ou FC2. Auparavant, ces formats n'étaient que des formes intermédiaires produites et effacées automatiquement par SVL. Cette nouvelle fonctionnalité permet à présent d'ap-

[KM97] J.-P. KRIMM, L. MOUNIER, « Compositional State Space Generation from LOTOS Programs », *in : Proceedings of TACAS'97 Tools and Algorithms for the Construction and Analysis of Systems (University of Twente, Enschede, The Netherlands)*, E. Brinksma (éditeur), *Lecture Notes in Computer Science, 1217*, Springer Verlag, Berlin, avril 1997. Version étendue avec preuves parue comme rapport de recherche VERIMAG RR97-01.

pliquer plusieurs vérifications à la volée sur un même réseau d'automates communicants sans qu'il soit nécessaire de le regénérer à chaque vérification.

L'ergonomie du compilateur SVL 2.0 a également été améliorée sur de nombreux points :

- La syntaxe d'appel du compilateur SVL en ligne de commande a été simplifiée. Les paramètres passés à SVL ont été classés en deux ensembles distincts (commandes et options) et certains d'entre eux ont été renommés pour plus de clarté.
- Une nouvelle commande a été introduite, qui permet de visualiser le programme obtenu après la phase d'expansion syntaxique pendant laquelle certaines opérations complexes (dites *meta-opérations*) sont remplacées par des opérations plus simples.
- Les conventions de nommage des fichiers temporaires ont été améliorées : au lieu de générer, pour ces fichiers, des noms sans signification, SVL leur donne désormais des noms qui suggèrent à l'utilisateur l'origine et le contenu de ces fichiers, tout en garantissant l'unicité des noms.
- Durant l'exécution d'un script, SVL construit un fichier contenant la totalité des opérations effectuées (appels aux différents outils, résultats et messages produits par les outils, etc.) afin de fournir à l'utilisateur toutes informations utiles en cas d'échec de l'exécution, sans avoir à relancer SVL dans un mode de débogage.
- La nouvelle version de SVL détecte et corrige dynamiquement certaines erreurs commises par l'utilisateur, ce qui peut éviter d'interrompre des vérifications parfois très longues. Ainsi, SVL peut modifier (avec avertissement) certains paramètres lorsque la vérification ne peut pas être effectuée avec les paramètres choisis par l'utilisateur. Il signale également certaines erreurs fréquentes, telles que l'inadéquation entre les étiquettes et le mode de filtrage lors des masquages et renommages d'étiquettes, ou l'utilisation d'interfaces trop restrictives lors de l'abstraction de processus communicants.
- Des stratégies "intelligentes" ont été incorporées à SVL, pour lui permettre de résoudre divers problèmes comme le ferait un expert en vérification. Ainsi, si une minimisation par bisimulation avec ALDÉBARAN ou FC2 échoue, alors SVL essaie d'utiliser plutôt BCG_MIN si la relation d'équivalence choisie le permet. De même, si une minimisation pour une relation faible (par exemple, l'équivalence observationnelle) échoue, alors SVL tente d'effectuer d'abord une pré-minimisation avec une relation plus forte (équivalence de branchement ou équivalence forte) ayant plus de chances de réussir (car algorithmiquement moins coûteuse). En dernier recours, si toutes les tentatives ont échoué, SVL poursuit son exécution avec le système de transitions non réduit.

Ces multiples améliorations ont accru significativement la taille du compilateur SVL 2.0, qui est passée de 4 000 lignes de code (en 2000) à 6 300 lignes (soit 1 600 lignes de code SYNTAX et C, 2 900 lignes de code LOTOS NT et 1 800 lignes de *shell script*). Par ailleurs, la documentation du langage et du compilateur SVL a été mise à jour.

Distribué depuis juillet 2001, SVL 2.0 a été déjà utilisé par l'Université de Twente (Pays-Bas) pour la vérification compositionelle de systèmes stochastiques, ainsi que par les sociétés BULL et ERICSSON (Suède). Les premiers retours d'utilisation sont très positifs.

Enfin, les travaux consacrés à SVL ont donné lieu à une publication d'ensemble [17] et feront l'objet d'une démonstration publique [19] en 2002.

5.1.5 Développement de l'outil PROJECTOR 2.0

Participants : Hubert Garavel, Radu Mateescu, Gordon Pace.

Dans le prolongement direct du développement de SVL 2.0 (voir § 5.1.4), nous avons entrepris en 2001 des recherches pour améliorer les performances des techniques de vérification compositionnelle avec abstractions [GLS96,KM97] intégrées à la boîte à outils CADP.

Bien que ces techniques nous paraissent, dans leur principe, pertinentes pour lutter contre le problème de l'explosion d'états (voir § 2.2), leur implémentation dans l'outil PROJECTOR n'était pas optimale : des problèmes de lenteur et de consommation mémoire excessive pouvaient empêcher l'utilisation effective de la vérification compositionnelle.

Pour résoudre ces problèmes, nous avons entrepris de développer une nouvelle version de l'outil PROJECTOR (travail post-doctoral de G. Pace), en repartant uniquement des définitions théoriques [KM97] et sans aucunement considérer l'implémentation existante de PROJECTOR.

Bien que cette nouvelle version (550 lignes de code C) ne soit pas totalement achevée, les premières mesures de performances sont encourageantes puisque l'on obtient, sur certains exemples, des gains en rapidité pouvant atteindre un facteur 15.

5.1.6 Développement des outils DISTRIBUTOR 2.0 et BCG_MERGE 2.0

Participants : Adrian Curic, Hubert Garavel, Christophe Joubert, Radu Mateescu, Gilles Stragier.

Nous étudions depuis 1999 l'utilisation de machines parallèles pour améliorer les performances des algorithmes de vérification énumérative.

En effet, ces algorithmes — qui nécessitent l'exploration et le stockage de graphes de dimensions importantes (plusieurs millions d'états) — sont souvent limités par la puissance de calcul et l'espace mémoire des machines séquentielles actuelles. C'est pourquoi nous souhaitons repousser ces limites en exploitant au mieux les possibilités des machines parallèles de type MIMD à mémoire distribuée (grappes de PC, réseaux de stations de travail) disponibles dans les laboratoires de recherche.

Nos efforts se sont concentrés sur la parallélisation de l'algorithme de construction du graphe, qui constitue un goulot d'étranglement pour la vérification puisqu'il requiert un espace mémoire considérable pour stocker tous les états accessibles. La parallélisation de cet algorithme devrait permettre, en remplaçant la mémoire d'une seule machine par celle de di-

[GLS96] S. GRAF, G. LÜTTGEN, B. STEFFEN, « Compositional Minimisation of Finite State Systems using Interface Specifications », *Formal Aspects of Computation 8*, septembre 1996.

[KM97] J.-P. KRIMM, L. MOUNIER, « Compositional State Space Generation from LOTOS Programs », *in : Proceedings of TACAS'97 Tools and Algorithms for the Construction and Analysis of Systems (University of Twente, Enschede, The Netherlands)*, E. Brinksma (éditeur), *Lecture Notes in Computer Science, 1217*, Springer Verlag, Berlin, avril 1997. Version étendue avec preuves parue comme rapport de recherche VERIMAG RR97-01.

zaines ou de centaines de machines, de gagner plusieurs ordres de grandeur dans la complexité des graphes traités.

Nos travaux avaient abouti en 2000 à deux outils prototypes :

- DISTRIBUTOR 1.0, qui répartit la construction d'un graphe sur N machines communiquant deux à deux au moyen de *sockets*. Chaque machine est chargée de construire une partie du graphe sous forme d'un fichier au format BCG, les états étant répartis entre les mémoires locales des machines au moyen d'une fonction de hachage déterminée statiquement.
- BCG_MERGE 1.0, qui assemble les N parties de graphe construites sur chaque machine par DISTRIBUTOR afin d'obtenir un fichier BCG unique représentant le graphe complet.

Ces outils ont été expérimentés sur diverses architectures : stations de travail sous SOLARIS reliées par un réseau ETHERNET 100 Mbits et grappe de 12 PC sous LINUX interconnectés par un bus SCI. Les résultats obtenus sur des applications de grande taille (autour de 15 millions d'états) ont montré des accélérations importantes et un bon équilibrage de charge entre les machines. Ces résultats ont fait l'objet d'une publication [18].

En 2001, nous avons poursuivi nos travaux (dans le cadre des stages successifs d'A. Curic, de G. Stragier et de Ch. Joubert) afin d'augmenter les performances, la robustesse et l'ergonomie des outils et de permettre leur passage à l'échelle sur un nombre plus important de machines. Ces travaux ont abouti à un nouvel outil DISTRIBUTOR 2.0, entièrement réécrit (6 200 lignes de code C), qui apporte les améliorations suivantes :

- Les communications entre processus ont été optimisées : utilisation de *sockets* non-bloquantes (plus rapides que les *sockets* bloquantes utilisées dans DISTRIBUTOR 1.0), réduction du nombre de connexions nécessaires à la transmission des paramètres aux machines distantes, gestion optimisée des tampons logiques de communication, etc. Par ailleurs, le code de DISTRIBUTOR 2.0 est plus modulaire, car les mécanismes de communication entre processus ont été clairement séparés de l'algorithme d'exploration proprement dit.
- Dans son principe, l'outil DISTRIBUTOR est indépendant de tout langage de spécification particulier, puisqu'il s'appuie sur l'interface de programmation OPEN/CÆSAR [8] pour explorer le graphe à construire. Néanmoins, l'implémentation de DISTRIBUTOR 1.0 comportait certaines dépendances résiduelles par rapport au langage LOTOS, qui ont été supprimées de DISTRIBUTOR 2.0.
- Une innovation importante de DISTRIBUTOR 2.0 consiste en l'introduction d'un processus auxiliaire (processus *superviseur*) qui s'exécute pendant toute la durée de la construction du graphe. Ce processus est notamment responsable du lancement des processus de génération répartis sur chacune des N machines et du lancement de l'outil BCG_MERGE 2.0 après terminaison de la génération distribuée.
- DISTRIBUTOR 2.0 rassemble dans un fichier de configuration unique la totalité des paramètres nécessaires à la génération distribuée (nom des machines, répertoires de travail sur chaque machine, taille de la mémoire locale, taille des tampons de communication, etc.). Ce fichier de configuration permet de donner à ces paramètres des valeurs générales, valables pour la totalité des machines, valeurs pouvant ensuite être spécialisées pour certaines machines particulières. Cette approche est préférable à celle de DISTRIBUTOR 1.0

qui exigeait un fichier séparé pour chacune des machines, une solution fastidieuse et source d'erreurs lorsque le nombre de machines est élevé.
- Le cas de la terminaison "normale" de la génération distribuée était déjà traité par DISTRIBUTOR 1.0 au moyen d'un algorithme de détection fonctionnant sur un anneau virtuel à jeton. DISTRIBUTOR 2.0 prend également en compte le cas de la terminaison "anormale" qui survient, soit lorsqu'une machine doit s'arrêter par suite d'un problème grave (pénurie de mémoire, par exemple), soit lorsque l'utilisateur décide d'interrompre prématurément la génération distribuée. Ceci a nécessité l'ajout, dans DISTRIBUTOR 2.0, d'un protocole spécial entre le processus superviseur et les processus répartis sur les N machines afin de permettre l'arrêt d'urgence, tout en laissant le système dans un état cohérent (fermeture des *sockets* et des fichiers BCG ouverts, terminaison des processus répartis, etc.).
- DISTRIBUTOR 1.0 n'offrait pas à l'utilisateur la possibilité de suivre en temps-réel la progression de la génération distribuée. Pour remédier à cette lacune, DISTRIBUTOR 2.0 comporte une interface graphique (500 lignes de code TCL-TK) qui permet de visualiser en temps-réel diverses informations statistiques sur la construction du graphe : nombre d'états visités et explorés (sur chaque machine et globalement), nombre de transitions franchies (sur chaque machine et globalement), ensemble des étiquettes rencontrées, répartition des états entre les différentes machines, consommation mémoire, taux d'utilisation des processeurs, etc. Ces statistiques sont calculées localement sur chaque machine, puis transmises au processus superviseur qui les centralise et les affiche sur l'interface graphique.

Enfin, nous avons développé l'outil BCG_MERGE 2.0 (450 lignes de code C), qui constitue une refonte complète de BCG_MERGE 1.0 pour le rendre compatible avec la gestion des paramètres de configuration faite dans DISTRIBUTOR 2.0.

Les nouveaux outils DISTRIBUTOR 2.0 et BCG_MERGE 2.0 ont été expérimentés sur la grappe de 225 PC du projet APACHE. Les expériences ont porté sur diverses spécifications LOTOS (protocole du réseau HAVi, protocole d'arbitrage du bus SCSI-2, protocole d'élection sur des anneaux à jeton, etc.) traitées sur des configurations allant jusqu'à 70 machines. Les résultats observés confirment les résultats précédemment obtenus avec DISTRIBUTOR 1.0 et BCG_MERGE 1.0 sur des configurations plus réduites, à savoir une accélération importante et un bon équilibrage de charge entre les machines.

5.1.7 Autres développements d'outils

Participants : Hubert Garavel, Frédéric Lang, Stéphane Martin, Radu Mateescu, Bruno Ondet, Frédéric Perret.

En 2001, une part de nos travaux a été consacrée à l'intégration des résultats de recherche obtenus lors des dernières années au sein d'une version stable de CADP, appelée CADP 2001 "Ottawa" (voir § 8.1) :

- Nous avons effectué un test systématique et rigoureux des différents outils de CADP afin

de détecter les erreurs résiduelles.
- Nous avons finalisé le portage de CADP vers le système d'exploitation WINDOWS et fait les changements nécessaires pour prendre en compte les versions récentes de WINDOWS et du logiciel CYGWIN de REDHAT.
- Nous avons amélioré le simulateur OCIS sur divers points et mis à jour la documentation et l'aide en ligne pour cet outil.
- Nous avons étendu le format EXP utilisé en entrée par les outils ALDÉBARAN, EXP.OPEN et EXP2FC2, puis modifié ces trois outils afin qu'ils interprètent ce format de manière identique.
- Nous avons développé une extension des bibliothèques CAESAR_HIDE et CAESAR_RENAME de l'environnement OPEN/CÆSAR afin de prendre en compte les expressions régulières généralisées définies dans la norme POSIX.

Après la parution de CADP 2001, de nouvelles améliorations ont été effectuées :

- Un nouvel outil appelé SEQ.OPEN a été développé afin de répondre aux besoins de BULL dans le cadre de FORMALFAME (voir § 6.2). SEQ.OPEN permet d'appliquer à un ensemble de traces d'exécution encodées dans le format SEQ les différents outils disponibles dans l'environnement OPEN/CÆSAR. Un avantage important de SEQ.OPEN réside dans le fait qu'il fonctionne sans charger en mémoire la trace d'exécution et peut, par conséquent, s'appliquer à des traces de très grande taille.
- L'outil BCG_INFO a été étendu afin d'afficher des informations supplémentaires (facteur de branchement d'un graphe, liste des états de blocage, etc.)
- L'outil BCG_IO a été étendu pour produire le format DOT utilisé par les outils graphiques GRAPHVIZ développés par AT&T.
- Enfin, nous avons établi une connexion directe entre CADP et l'atelier MUCRL développé par le CWI (Pays-Bas), afin de résoudre certaines difficultés pratiques rencontrées par la société ERICSSON (Suède) qui utilise conjointement les outils CADP et MUCRL. Cette connexion permet désormais à l'outil INSTANTIATOR du CWI de produire directement des graphes au format BCG.

5.2 Technologie des langages – compilation

Mots clés : algèbre de processus, automate, compilation, concurrence, génération de code, génie logiciel, méthodes formelles, modélisation, parallélisme asynchrone, spécification formelle, synchronisation, système distribué, temps réel.

Résumé : *En 2001, nous avons entrepris des recherches sur la compilation des processus du langage LOTOS NT, en orientant nos travaux vers la modélisation d'applications embarquées sur cartes à puces.*

5.2.1 Compilation du langage E-LOTOS

Participants : Hubert Garavel, Frédéric Lang.

Les travaux consacrés depuis 1992 à la définition du langage E-LOTOS— travaux auxquels VASY a activement participé — ont trouvé leur consécration en 2001 avec l'adoption officielle de E-LOTOS comme norme internationale ISO/IEC [ISO01].

Forte de ce succès, l'équipe VASY a poursuivi ses efforts pour la compilation des méthodes formelles de nouvelle génération telles que E-LOTOS et LOTOS NT (voir § 2.3). Ces efforts portent dans plusieurs directions afin de traiter la partie données et la partie contrôle des langages E-LOTOS et LOTOS NT.

Concernant la partie données : nous avons continué à diffuser le compilateur TRAIAN (voir § 4.2) et à l'utiliser intensivement, au sein de l'équipe VASY, comme un méta-outil pour l'écriture de compilateurs. Une publication concernant cette approche a été rédigée [16] et fera l'objet d'une démonstration lors de la conférence *Compiler Construction* en 2002.

Concernant la partie contrôle : la compilation de E-LOTOS et LOTOS NT est un problème difficile, compte-tenu de la richesse de ces langages, qui combinent parallélisme asynchrone, temps quantifié et exceptions. Nous avons choisi d'aborder ces différents problèmes de manière progressive en nous concentrant, dans un premier temps, sur les aspects indispensables à la réalisation du contrat FORMALCARD (voir § 6.1) auquel participent la société SCHLUMBERGER et les projets VASY et VERTECS de l'INRIA.

Les applications embarquées sur cartes à puces pouvant être représentées sous forme d'automates étendus avec des variables, nos travaux se sont donc focalisés sur les processus séquentiels présents dans les langages E-LOTOS et LOTOS NT. Pour les modéliser, nous avons conçu un formalisme appelé NTIF (*New Technology Intermediate Form*) destiné à servir de langage intermédiaire dans la compilation des processus E-LOTOS et LOTOS NT.

Dans son principe, NTIF permet de spécifier des automates définis par un ensemble d'états et de transitions et paramétrés par des variables d'état typées. A chaque transition sont attachés une action (permettant la communication avec l'environnement selon la sémantique du rendez-vous propre aux algèbres de processus) ainsi qu'un fragment de code séquentiel permettant de consulter et/ou de modifier la valeur des variables d'états ; ce fragment de code est exprimé avec les structures de contrôle usuelles des langages algorithmiques et fonctionnels.

Nous avons défini la syntaxe et la sémantique du langage NTIF, puis montré que NTIF était suffisamment expressif pour généraliser un grand nombre de formalismes pré-existants pour la spécification des automates étendus.

Nous avons ensuite conçu deux logiciels (appelés NT2IF et NT2DOT) pour traiter les descriptions NTIF modélisant des applications embarquées sur cartes à puces. Développés à l'aide du générateur de compilateurs SYNTAX et du compilateur TRAIAN (voir § 4.2), ces deux logiciels (6 500 lignes de code) ont en commun les analyseurs lexicaux et syntaxiques, ainsi que

[ISO01] ISO/IEC, « Enhancements to LOTOS (E-LOTOS) », *International Standard n° 15437:2001*, International Organization for Standardization — Information Technology, Genève, septembre 2001.

les différentes phases de construction, de vérification et de transformation d'arbres abstraits. Leurs fonctionnalités sont les suivantes :

- NT2IF traduit les descriptions NTIF vers un formalisme de plus bas niveau, le modèle IOSTS (*Input Output Symbolic Transition Systems*) [RdJ00], qui est utilisé en entrée du générateur de tests symboliques STG développé par le projet VERTECS. Le modèle IOSTS — qui, concrètement, est représenté à l'aide du formalisme IF [BFG+99] développé au laboratoire VERIMAG — permet de décrire des automates étendus, dont chaque transition est étiquetée par une condition de franchissement portant sur la valeur des variables d'état, suivie d'une action de communication, d'une modification de l'état global par affectation des variables d'états, et enfin d'un branchement vers l'état suivant. La traduction s'effectue par "dépliage symbolique", en décomposant les transitions NTIF en transitions IF plus élémentaires.
  Grâce à l'existence du traducteur NT2IF, NTIF s'est rapidement imposé comme le langage d'entrée approprié pour l'utilisation de l'outil STG en milieu industriel. En effet, NTIF pallie les principales limitations du modèle IOSTS/IF : il possède des structures de contrôle évoluées (instructions "`case`", "`if-then-else`", "`while`", etc.) dont l'absence en IOSTS/IF oblige l'utilisateur à introduire de nombreux états et transitions intermédiaires, ainsi qu'à dupliquer de nombreuses conditions booléennes, ce qui constitue une source d'erreurs fréquentes. De plus, NTIF possède une sémantique plus intuitive, plus facile à apprendre, car très proche de la sémantique des langages algorithmiques.
- NT2DOT permet de visualiser sous forme graphique les descriptions NTIF en les traduisant vers le format DOT utilisé par les outils graphiques GRAPHVIZ développés par AT&T.
  Bien que l'outil STG dispose également de fonctionnalités de visualisation pour les modèles IOSTS/IF, l'expérience a démontré que les graphiques produits par NT2DOT étaient plus lisibles, car mieux structurés et de plus petite taille. Ce résultat s'explique par le fait que, d'une part, les modèles NTIF comportent moins d'états et de transitions que leurs équivalents IOSTS/IF, et que, d'autre part, STG attache les fragments de code séquentiels aux transitions du diagramme, alors que NT2DOT regroupe ces fragments sous une forme structurée à l'intérieur des états. Cette comparaison graphique a confirmé l'intérêt d'inclure des structures de contrôle évoluées en NTIF.

Dans le cadre de FORMALCARD, le langage NTIF et les outils associés ont été utilisés pour deux études de cas significatives portant sur les cartes à puces :

- Nous avons traduit en NTIF une spécification de porte-monnaie électronique multi-devises (EMV *Common Electronic Purse Standard* version 2.2 5-2000) à partir de la formalisation en IOSTS/IF (1 140 lignes de code) qu'en avait faite Duncan Clarke (pro-

[RdJ00] V. RUSU, L. DU BOUSQUET, T. JÉRON, « An Approach to Symbolic Test Generation », *in : Proceedings of the International Conference on Integrated Formal Methods IFM'00 (Dagstuhl, Germany)*, *Lecture Notes in Computer Science*, *1945*, Springer Verlag, p. 338–357, novembre 2000.

[BFG+99] M. BOZGA, J.-C. FERNANDEZ, L. GHIRVU, S. GRAF, J.-P. KRIMM, L. MOUNIER, « IF: An Intermediate Representation and Validation Environment for Timed Asynchronous Systems », *in : Proceedings of World Congress on Formal Methods in the Development of Computing Systems FM'99 (Toulouse, France)*, J. Wing, J. Woodcock (éditeurs), Springer Verlag, septembre 1999.

jet VERTECS). Cette expérience a permis de convaincre nos partenaires des qualités du langage NTIF : la description NTIF est beaucoup plus claire, plus concise (56% d'états et 50% de transitions en moins) et plus fiable (plusieurs erreurs ont été découvertes au cours de la traduction en NTIF). La description NTIF a ensuite été traduite en IOSTS/IF grâce à l'outil NT2IF, ce qui nous a permis d'améliorer notre traducteur et de détecter des erreurs résiduelles dans l'outil STG.

– François-Xavier Ponscarme (stagiaire SCHLUMBERGER) a décrit en NTIF les commandes administratives d'un système d'exploitation destiné aux cartes à puces pour la téléphonie mobile 3GPP (*3rd Generation Partnership Project*). Cette description NTIF (910 lignes de code), reprise et complétée par Elena Zinovieva (projet VERTECS), a été traduite en IOSTS/IF à l'aide de l'outil NT2IF (38% d'états et 56% de transitions en plus). Des tests symboliques ont été produits automatiquement par l'outil STG.

5.3 Etudes de cas et applications pratiques

Mots clés : activité de conception, algorithme distribué, application critique, application répartie, architecture multiprocesseur, architecture parallèle, atomicité, automate, cohérence de caches, concurrence, génération de code, génération de test, génie logiciel, ingénierie des protocoles, logique temporelle, mémoire répartie, modélisation, parallélisme asynchrone, protocole de communication, spécification formelle, synchronisation, système distribué, temps réel, test, travail coopératif, validation, vérification de programme.

Résumé : *Nous accordons une grande importance au traitement d'exemples réalistes, notamment issus de besoins industriels, qui nous permettent de vérifier l'adéquation de nos méthodes et outils, et d'identifier de nouvelles orientations de recherche pour résoudre les problèmes rencontrés.*

5.3.1 Protocole de cohérence de caches CC-NUMA "Fame"

Participants : Hubert Garavel, Radu Mateescu, Bruno Ondet, Frédéric Perret, Solofo Ramangalahy, Nicolas Zuanon.

Depuis octobre 1998, nous collaborons avec BULL dans le cadre de l'action FORMALFAME (voir § 6.2) qui cherche à promouvoir l'utilisation de méthodes formelles pour la vérification et le test d'architectures multiprocesseurs. Plusieurs méthodologies sont explorées. Toutes se basent sur l'établissement d'un modèle de référence de l'architecture visée grâce au langage de description formelle LOTOS.

Nos travaux portent sur FAME, une nouvelle architecture multi-processeurs CC-NUMA développée au centre BULL des Clayes-sous-Bois (France) et basée sur des processeurs INTEL ITANIUM 64 bits. Nous nous concentrons actuellement sur la validation et le test d'un circuit complexe (appelé B-SPS) ayant une double fonctionnalité : il effectue le routage entre les processeurs et les nœuds d'entrées/sorties et il implémente le protocole de cohérence de caches

de l'architecture FAME. La complexité de ce circuit provient à la fois du parallélisme interne inhérent au routage et des accès multiples aux données partagées dont il convient de préserver l'intégrité.

Le fonctionnement du circuit B-SPS est décrit par un document de référence (100 pages) combinant des spécifications informelles en langue naturelle avec des tables états/transitions. Le circuit lui-même est réalisé par un modèle en langage VERILOG.

Sur la base du document de référence, BULL a également développé un modèle du circuit B-SPS en langage C++. Celui-ci permet, soit d'émuler le comportement attendu du circuit, soit de valider le comportement produit par le modèle VERILOG.

Cette approche pragmatique est complétée par les travaux de FORMALFAME qui, en s'appuyant sur une description formelle en LOTOS du circuit B-SPS développée en 2000, introduisent des aspects formels dans la méthodologie de validation et de test utilisée par BULL.

En 2001, les travaux de FORMALFAME ont concerné les aspects suivants :

– *Vérification automatique de traces d'exécution* : nous avons poursuivi nos travaux sur l'analyse automatique des traces d'exécution produites soit par le modèle VERILOG, soit par l'émulateur C++ du circuit B-SPS. Ces traces sont issues de scénarios de test (déterministes ou aléatoires) et peuvent comporter plusieurs dizaines de milliers de messages. Leur analyse manuelle par relecture étant prohibitive, surtout dans le cas de transactions imbriquées, nous avons proposé et mis en œuvre deux méthodes de validation automatique basées sur les outils CADP :
  – La première méthode consiste à spécifier en logique temporelle des propriétés de correction sur les traces. Jusqu'alors, ces propriétés étaient codées manuellement à partir des tables états/transitions de la spécification informelle du protocole, puis vérifiées sur les traces (préalablement converties dans un format acceptable par les outils de CADP) au moyen de l'outil EVALUATOR 3.0.
    En 2001, cette procédure a été complètement automatisée grâce au développement de nouveaux outils (environ 5 000 lignes de PERL et de *shell script*) qui, à partir des tables états/transitions et d'un ensemble de traces, génèrent automatiquement les propriétés en logique temporelle et les vérifient sur chaque trace.
    Les résultats de ces vérifications fournissent une mesure précise de la couverture *fonctionnelle* du protocole (évaluée par rapport à la spécification) par l'ensemble de traces considéré. Cette couverture, différente de la couverture *structurelle* (évaluée par rapport au modèle VERILOG), n'était pas mesurable objectivement avec les méthodes de test utilisées antérieurement dans FORMALFAME.
  – La deuxième méthode consiste à vérifier que les traces d'exécution fournies par BULL sont acceptées par la description LOTOS du protocole de cohérence de caches. Pour cela, chaque trace est traduite vers une formule de logique temporelle comportant des expressions régulières, qui est vérifiée (à la volée) sur la description LOTOS grâce à l'outil EVALUATOR 3.0.

  Ces deux méthodes d'analyse ont été appliquées sur différents types de traces fournies par BULL et préalablement validées par l'implémentation en C++ du circuit B-SPS :
  – Pour les traces dites *de collision*, nous avons analysé plus de 130 méga-octets de traces

produites par le modèle VERILOG (représentant environ 24 000 transactions effectuées entre le circuit B-SPS et les autres modules) sans découvrir d'erreur dans le protocole de cohérence de caches. Naturellement, ceci ne constitue pas une garantie absolue, compte-tenu du nombre élevé de cas possibles.
- Pour les traces dites *d'interface*, l'objectif n'était pas de valider le circuit lui-même, mais de déterminer si les tests écrits manuellement par BULL assuraient une couverture exhaustive par rapport aux tables états/transitions. Sur les traces correspondant à ces tests, 761 propriétés temporelles ont été vérifiées, dont 216 n'étaient satisfaites par aucune trace. Après interprétation de ces résultats conjointement avec les concepteurs de FAME, il est apparu que ces propriétés non vérifiées se réduisaient à 24 situations "réelles" qui n'étaient pas testées. Ceci a permis de rajouter 2 nouveaux tests pour couvrir les 24 cas manquants.
- De même, pour les traces dites *directory*, 518 propriétés ont été vérifiées, dont 196 n'étaient pas satisfaites par l'ensemble de traces. Après analyse, ceci a permis de rajouter un test supplémentaire pour couvrir les cas manquants.

Grâce aux améliorations des outils CADP (en particulier, l'évaluation optimisée des propriétés sur les traces), l'ensemble de ces vérifications peut désormais être effectué en une nuit sur une machine standard (processeur Pentium III 700 MHz avec 1 giga-octet de mémoire vive).

- *Génération automatique de tests* : un autre objectif de FORMALFAME consiste à générer des cas de test en les dérivant automatiquement de la description LOTOS, ce qui garantit la correction des tests par rapport à la modélisation formelle. Ainsi, l'effort investi dans la modélisation trouve son utilité non seulement pour la vérification, mais aussi pour la génération automatique de tests qui viennent compléter les tests écrits manuellement.
  Initialement, il était prévu d'utiliser l'outil TGV afin de produire des tests déterministes. Toutefois, l'écriture manuelle des tests déterministes par l'équipe BULL aux Clayes-sous-Bois ayant atteint les objectifs fixés, nous avons orienté nos travaux vers la production de tests aléatoires.
  En 2001, nous avons conçu et étudié la faisabilité d'une approche basée sur le compilateur CÆSAR en mode "génération de code" (EXEC/CÆSAR). Dans cette approche, les tests sont obtenus par des exécutions aléatoires de la description LOTOS selon différentes distributions de probabilité. Nos premières expérimentations concluent à la possibilité de connecter le code C produit par EXEC/CÆSAR à la plate-forme de test de BULL afin que celle-ci puisse être directement pilotée par la description LOTOS.

Les résultats de FORMALFAME en 2001 ont été jugés positifs par BULL. Outre l'amélioration de la spécification informelle du protocole de cohérence de caches par la correction de deux imprécisions, les outils de vérification de CADP ont permis de mesurer précisément la couverture fonctionnelle des tests et de rajouter des cas de test manquants pour atteindre la totalité des objectifs du plan de test de l'architecture FAME.

Suite à ces résultats, l'équipe BULL des Clayes-sous-Bois a adopté la technologie basée sur LOTOS et CADP (en particulier, l'analyse automatique des traces d'exécution) afin de l'intégrer à sa propre méthodologie de validation utilisée pour l'architecture FAME.

5.3.2 Protocole de déploiement de composants Java

Participants : Hubert Garavel, Radu Mateescu, Frédéric Tronel.

Dans le cadre de PARFUMS (voir § 6.3), projet pré-compétitif du Réseau National des Technologiques Logicielles (RNTL, nous travaillons depuis octobre 2001 sur un protocole de déploiement conçu par le projet SIRAC. Ce protocole sert à installer et à configurer un ensemble d'agents mobiles (composants JAVA) sur des onduleurs fabriqués par la société MGE-UPS.

La contribution de VASY porte sur la validation de ce protocole, en étroite collaboration avec le projet SIRAC qui implémente ce protocole et la société MGE-UPS qui compte l'industrialiser.

En 2001, nous avons analysé la spécification en langue naturelle décrivant le protocole de déploiement afin d'en dériver une modélisation formelle en LOTOS qui servira de base à la vérification. Pour chacun des agents intervenant dans le protocole, un processus LOTOS correspondant devra être écrit manuellement.

Concernant la vérification, nous prévoyons l'emploi de techniques énumératives et compositionnelles sur des configurations comportant un nombre fini d'agents. Dans ce but, F. Tronel a écrit un traducteur (120 lignes de code CAML) qui, à partir de la description synthétique d'une configuration, construit automatiquement une description LOTOS formée de la mise en parallèle (avec les synchronisations et communications adéquates) des processus LOTOS correspondant aux agents.

5.3.3 Autres études de cas

Nous avons utilisé les outils CADP pour traiter plusieurs autres applications :

- Notre travail antérieur sur la spécification et la vérification d'un protocole de reconfiguration dynamique des applications à base d'agents mobiles JAVA a fait l'objet d'une publication [15].
- F. Tronel a modélisé en LOTOS un protocole de consensus [Rab83,MRT00] qu'il a étudié pendant sa thèse. La spécification LOTOS obtenue (400 lignes) servira d'exemple pour les travaux en cours sur la vérification compositionnelle.
- H. Garavel et R. Mateescu ont eu des échanges scientifiques suivis avec la société ERICSSON (Suède) afin de promouvoir l'utilisation des outils CADP pour la vérification de protocoles de télécommunication décrits dans le langage ERLANG. Ces échanges ont d'abord porté sur la vérification d'un protocole de verrouillage distribué conçu par ERICSSON à l'aide de l'outil EVALUATOR 3.0 [AE01], puis sur la modélisation en LOTOS de ce protocole

[Rab83] M. O. RABIN, « Randomized Byzantine Generals », *in : Proceedings of the IEEE Symposium on Foundations of Computer Science*, p. 403–409, 1983.

[MRT00] A. MOSTEFAOUI, M. RAYNAL, F. TRONEL, « The Best of Both Worlds: a Hybrid Approach to Solve Consensus », *in : Proceedings of the International Conference on Dependable Systems and Networks (FTCS/DCCA)*, IEEE, p. 513–522, juin 2000.

[AE01] T. ARTS, C. B. EARLE, « Development of a Verified Erlang Program for Resource Locking », *in :*

dans le but de concevoir un traducteur automatique d'ERLANG vers LOTOS.

D'autres équipes ont également utilisé la boîte à outils CADP pour diverses études de cas. Pour ne citer que les travaux publiés en 2001, on peut mentionner :

- la vérification du protocole TCAP (*Transaction Capabilities Application Part*) normalisé par l'ITU [AvL01] ;
- la spécification et la vérification de circuits asynchrones modulaires [Yoe01,YG01] ;
- la spécification et la vérification d'algorithmes distribués pour le recouvrement des points de contrôle (*checkpointing*) [GCM01] ;
- la vérification d'un système d'ascenseurs pour les véhicules lourds [GPW01] ;
- la spécification, la vérification et le test de circuits séquentiels [TH01] ;
- la spécification et la vérification du modèle de comptabilité de l'architecture TINA (*Telecommunications Information Networking Architecture*) [RSSW01] ;
- la vérification de l'architecture de coordination SPLICE [DvL01].

Par ailleurs, l'outil XTL a été utilisé pour développer un évaluateur de la logique modale FULL destinée aux programmes LOTOS comportant des données [BS01].

Proceedings of the 6th International Workshop on Formal Methods for Industrial Critical Systems FMICS'2001 (Paris, France), S. Gnesi, U. Ultes-Nitsche (éditeurs), Université Paris 7 – LIAFA and INRIA Rhône-Alpes, p. 131–145, juillet 2001.

[AvL01] T. ARTS, I. VAN LANGEVELDE, « Correct Performance of Transaction Capabilities », *in : Proceedings of the 2nd International Conference on Application of Concurrency to System Design ICACSD'2001 (Newcastle upon Tyne, UK)*, A. Valmari, A. Yakovlev (éditeurs), IEEE Computer Society, p. 35–42, juin 2001.

[Yoe01] M. YOELI, « Examples of LOTOS-Based Verification of Asynchronous Circuits », *Technical Report n° TR CS-2001-08*, Technion, Computer Science Department, Haifa, Israel, février 2001.

[YG01] M. YOELI, A. GINZBURG, « LOTOS/CADP-Based Verification of Asynchronous Circuits », *Technical Report n° TR CS-2001-09*, Technion, Computer Science Department, Haifa, Israel, mars 2001.

[GCM01] G. GODZA, V. CRISTEA, R. MATEESCU, « Formal Specification of Checkpointing Algorithms », *in : Proceedings of 13th International Conference on Control Systems and Computer Science CSCS 13 (Bucharest, Romania)*, Polytechnic University of Bucharest, p. 311–317, mai 2001.

[GPW01] J. F. GROOTE, J. PANG, A. WOUTERS, « A Balancing Act: Analyzing a Distributed Lift System », *in : Proceedings of the 6th International Workshop on Formal Methods for Industrial Critical Systems FMICS'2001 (Paris, France)*, S. Gnesi, U. Ultes-Nitsche (éditeurs), Université Paris 7 – LIAFA and INRIA Rhône-Alpes, p. 1–12, juillet 2001. Version étendue parue comme rapport technique CWI SEN-R0111, Amsterdam, mai 2001.

[TH01] K. J. TURNER, J. HE, « Formally-Based Design Evaluation », *in : Proceedings of the 11th IFIP WG 10.5 Advanced Research Working Conference on Correct Hardware Design and Verification Methods CHARME'2001 (Livingston, Scotland)*, T. Margaria, T. F. Melham (éditeurs), *Lecture Notes in Computer Science*, *2144*, IFIP, Springer Verlag, p. 104–109, septembre 2001.

[RSSW01] L. L. ROSSI, I. V. SOUZA, A. SEKKAKI, C. B. WESTPHALL, « Formal Specification and Verification of the Accounting Model of TINA Architecture using LOTOS and ALDEBARAN », *in : Proceedings of Congresso Brasileiro de Computação (Itajaí, Brazil)*, août 2001.

[DvL01] P. DECHERING, I. VAN LANGEVELDE, « Towards Automated Verification of SPLICE in μCRL », *SVC Deliverable n° 2D2*, Telematics Institute, Enschede, The Netherlands, octobre 2001.

[BS01] J. BRYANS, C. SHANKLAND, « Implementing a Modal Logic over Data and Processes using XTL », *in : Proceedings of the 21st IFIP WG 6.1 International Conference on Formal Techniques for Networked and Distributed Systems FORTE'2001 (Cheju Island, Korea)*, M. Kim, B. Chin, S. Kang, D. Lee (éditeurs), IFIP, Kluwer Academic Publishers, p. 201–216, août 2001.

6 Contrats industriels (nationaux, européens et internationaux)

6.1 Action FormalCard (Bull-Schlumberger)

Participants : Hubert Garavel, Frédéric Lang, Radu Mateescu.

Mots clés : activité de conception, compilation, génération de code, génération de test, modélisation, spécification formelle, vérification de programme.

La collaboration engagée en 1999 avec Bull SC&T (*Smart Cards and Terminals*), puis formalisée en juillet 2000 par le lancement de l'action FormalCard du Gie Bull-Inria Dyade, s'est poursuivie en 2001 en dépit des changements liés au rachat de Bull SC&T par la société Schlumberger, à la fin du Gie Dyade et à l'arrêt du projet Pampa (Rennes). Participent désormais à cette collaboration, à laquelle nous conservons le nom de FormalCard, les projets Vasy et Vertecs de l'Inria et la société Schlumberger.

FormalCard vise à réaliser des outils formels pour le développement des logiciels embarqués sur cartes à puces en vue de permettre leur certification. Cette certification répond à une demande croissante dans l'industrie, en raison de l'apparition de nouvelles normes (*critères communs*) qui préconisent, à partir d'un certain niveau de sécurité, l'utilisation de méthodes formelles. FormalCard comporte trois objectifs dont les deux premiers sont pris en charge par le projet Vasy, le troisième étant confié au projet Vertecs :

- *Conception d'un langage pour la spécification formelle d'applications pour cartes à puces* : en se basant sur l'expérience acquise par Vasy au cours de la normalisation du langage E-Lotos, il s'agit de définir un langage (sous-ensemble du langage Lotos NT) satisfaisant plusieurs critères :
  - Il doit être simple et intuitif pour pouvoir être utilisé par des ingénieurs qui ne sont pas nécessairement familiers avec les méthodes formelles.
  - Il doit être suffisamment expressif pour permettre une description concise du système d'exploitation et des applications embarquées sur cartes à puces.
  - Il doit être exécutable pour permettre le déboguage, la simulation et la vérification énumérative (*model checking*).
  - Il doit avoir une sémantique formelle pour permettre l'analyse symbolique et l'utilisation de techniques de preuve.
- *Vérification et génération de tests par des méthodes énumératives* : on cherche à spécialiser au domaine de la carte à puce certaines techniques bien établies pour lesquelles nous disposons d'outils efficaces. L'objectif est de réaliser, pour le langage de description formelle mentionné ci-dessus, un compilateur produisant du code C compatible avec l'interface Open/Cæsar [8]. Ceci devrait permettre de réutiliser les outils de simulation et de vérification disponibles dans Cadp pour valider les propriétés de sécurité des applications embarquées et d'utiliser l'outil Tgv pour générer des tests (non symboliques) de conformité.
- *Génération de tests symboliques* : les techniques énumératives de génération de tests ont

des limites inhérentes, dues à l'explosion combinatoire. De plus, les tests produits sont complètement instanciés, contrairement à la pratique industrielle [ISO96] qui veut que les cas de test soient de "vrais" programmes avec variables et paramètres. L'objectif est de proposer des méthodes de génération de test symboliques qui évitent ces inconvénients et de démontrer l'applicabilité de ces méthodes sur des études de cas industrielles.

En 2001, les travaux de VASY (voir § 5.2.1) ont porté sur la définition du formalisme NTIF, le développement des outils associés (NT2IF et NT2DOT) et leur utilisation dans deux études de cas industrielles (application porte-monnaie électronique et système d'exploitation pour carte à puce destinée à la téléphonie mobile de 3^{e} génération) en connexion avec l'outil STG conçu dans le projet VERTECS.

6.2 Action FormalFame (Bull)

Participants : Hubert Garavel, Radu Mateescu, Bruno Ondet, Frédéric Perret, Solofo Ramangalahy, Nicolas Zuanon.

Mots clés : activité de conception, algorithme distribué, application répartie, architecture multiprocesseur, architecture parallèle, automate, cohérence de caches, compilation, génération de code, génération de test, mémoire répartie, modélisation, parallélisme asynchrone, programmation parallèle, protocole de communication, spécification formelle, synchronisation, système distribué, vérification de programme.

Depuis 1995, nous entretenons une collaboration de longue durée avec BULL, collaboration à laquelle l'ex-projet PAMPA a participé jusqu'en décembre 2000. Il s'agit de démontrer que les méthodes formelles et les outils développés à l'INRIA pour la validation et le test des protocoles de télécommunication peuvent aussi être appliqués avec succès aux architectures multi-processeurs développées par BULL. L'objectif à long terme est d'offrir une chaîne complète et intégrée d'outils pour la spécification formelle, la simulation, le prototypage rapide, la vérification, la génération de tests et leur exécution.

Une première étape de cette collaboration a eu lieu entre 1995 et 1998 dans le cadre de l'action VASY du GIE BULL-INRIA DYADE. Elle a été consacrée à deux études de cas successives : le protocole d'arbitrage de bus de l'architecture POWERSCALE [CGM+96] et le protocole de cohérence de caches de l'architecture multiprocesseurs POLYKID [13]. Le résultat a été jugé positif : la faisabilité de l'approche proposée a été démontrée et BULL a manifesté

[ISO96] ISO/IEC, « The Tree and Tabular Combined Notation (TTCN) », *International Standard n° 9646-3*, International Organization for Standardization — Information Technology — Open Systems Interconnection — Conformance Testing Methodology and Framework, Genève, 1996.

[CGM+96] G. CHEHAIBAR, H. GARAVEL, L. MOUNIER, N. TAWBI, F. ZULIAN, « Specification and Verification of the PowerScale Bus Arbitration Protocol: An Industrial Experiment with LOTOS », *in : Proceedings of the Joint International Conference on Formal Description Techniques for Distributed Systems and Communication Protocols, and Protocol Specification, Testing, and Verification FORTE/PSTV'96 (Kaiserslautern, Germany)*, R. Gotzhein, J. Bredereke (éditeurs), IFIP, Chapman & Hall, p. 435–450, octobre 1996. Version intégrale parue comme rapport de recherche INRIA RR-2958.

son intérêt à poursuivre la collaboration avec de nouvelles architectures.

Depuis octobre 1998, nos travaux portent sur FAME, une architecture multi-processeurs CC-NUMA développée par BULL et basée sur des processeurs INTEL ITANIUM 64 bits. D'abord informelle, cette collaboration a été officialisée en 1999 sous la forme d'une action DYADE, intitulée FORMALFAME, qui a duré jusqu'à la fin du GIE DYADE en mars 2001. Depuis cette date, la collaboration se poursuit sous la forme d'un contrat BULL-INRIA, auquel nous avons conservé le nom de FORMALFAME. Prévu jusqu'à fin 2002, ce contrat réunit le projet VASY et l'équipe d'Anne Kaszynski au centre BULL des Clayes-sous-Bois (France), la coordination étant assurée par un ingénieur BULL (M. Zendri, puis N. Zuanon, puis S. Ramangalahy) installé dans les locaux de l'Unité de Recherche INRIA Rhône-Alpes.

FORMALFAME cible ses efforts sur les composants critiques de l'architecture FAME, successivement : le circuit CCS qui gère les communications pour un groupe de quatre processeurs, le circuit NCS qui gère les communications réseau et, depuis décembre 1999, le circuit B-SPS qui implémente le protocole de cohérence de caches. Pour chacun de ces composants, une description LOTOS a été élaborée, qui sert de modèle de référence pour la vérification et le test.

En 2001, les résultats obtenus par FORMALFAME concernent, d'une part, la validation et le test du circuit B-SPS (voir § 5.3.1) et, d'autre part, l'amélioration des différents outils de CADP.

6.3 Contrat RNTL Parfums (MGE-UPS, Silicomp)

Participants : Hubert Garavel, Radu Mateescu.

Mots clés : activité de conception, algorithme distribué, application répartie, concurrence, génie logiciel, ingénierie des protocoles, modélisation, langage à objets, langage d'interface, parallélisme asynchrone, protocole de communication, spécification formelle, synchronisation, système distribué, validation, vérification de programme.

PARFUMS (*Pervasive Agents for Reliable and Flexible UPS Management Systems*) est un projet pré-compétitif du Réseau National des Technologies Logicielles (RNTL). D'une durée de 2 ans, ce projet a débuté en mai 2001 et regroupe les sociétés MGE-UPS et SILICOMP, ainsi que les équipes SIRAC et VASY de l'INRIA.

L'objectif de PARFUMS est la mise en œuvre d'une architecture flexible et fiable à base de composants JAVA pour permettre l'administration d'onduleurs à distance depuis divers équipements (ordinateurs, téléphones portables, assistants personnels, etc.). Cette architecture s'appuie sur l'infrastructure logicielle développée dans le projet SIRAC : bus logiciel tolérant les pannes, modèle de programmation par agents (éventuellement mobiles) communiquant par messages, et outils de développement associés. Il s'agit de faciliter le déploiement des logiciels d'administration, leur surveillance, leur reconfiguration et leur mise à jour, tout en réduisant le coût de ces fonctions.

En 2001, la contribution de VASY au projet PARFUMS a porté essentiellement sur la

modélisation formelle du protocole de déploiement (voir § 5.3.2).

7 Actions régionales, nationales et internationales

7.1 Actions nationales

En 2001, nous avons collaboré avec plusieurs projets INRIA :

- APACHE (Rhône-Alpes) : utilisation de la plate-forme "grappe de PCs" développée par le projet APACHE pour l'expérimentation d'algorithmes de vérification massivement parallèles (voir § 5.1.6) ;
- SIRAC/SARDES (Rhône-Alpes) : collaboration dans le cadre du contrat RNTL PARFUMS (voir § 6.3) et organisation d'un séminaire de recherche commun SARDES-VASY à Autrans, France, du 5 au 7 décembre 2001.
- VERTECS (Rennes) : collaboration dans le cadre de FORMALCARD (voir § 6.1).

Nous entretenons également des relations scientifiques avec d'autres équipes françaises :

- Laboratoire ISIMA/LIMOS (Clermont-Ferrand) : méthodes de conception conjointe matériel-logiciel combinant LOTOS, LOTOS NT et VHDL (Pierre Wodey et Fabrice Baray) ;
- Laboratoire VERIMAG (Grenoble) : coopération pour l'amélioration des outils CADP (Laurent Mounier).

7.2 Actions internationales

7.2.1 Groupes de travail internationaux

- Nous sommes membre de FMICS (*Formal Methods for Industrial Critical Systems*), l'un des onze groupes de travail d'ERCIM. Depuis juillet 1999, H. Garavel coordonne ce groupe qui comprend actuellement 75 chercheurs appartenant à 29 organisations (voir http://www.inrialpes.fr/vasy/fmics).
- H. Garavel est membre du comité technique (*ETItorial Board*) de la plate-forme d'intégration logicielle ETI (*Electronic Tool Integration*) accessible en ligne par Internet (voir http://www.eti-service.org).

7.2.2 Relations bilatérales internationales

Nous entretenons des relations scientifiques avec plusieurs universités et centres de recherche internationaux. En 2001, nous avons notamment eu des contacts étroits avec :

- le centre de recherche d'ERICSSON en Suède (Thomas Arts),
- l'Université de Kent à Canterbury (Clara Benac Earle),
- l'Université Libre de Bruxelles (Thierry Massart),
- l'Université de Twente (Holger Hermanns).

7.3 Accueil de chercheurs français et étrangers

- Ilham Alloui, Christelle Chaudet et Flavio Oquendo, de l'Université de Savoie, nous ont rendu visite pour une journée de séminaires communs organisée le 12 mars 2001.
- Bertrand Jeannet, stagiaire post-doctoral à l'Université de Aalborg (Danemark) nous a rendu visite le 11 mai 2001. Il a donné un séminaire sur le partitionnement dynamique dans l'analyse de relations linéaires et la vérification de programmes synchrones.
- Jiri Barnat, de l'Université Masaryk de Brno (République Tchèque), nous a rendu visite du 1er au 5 octobre 2001.
- Thierry Massart, de l'Université Libre de Bruxelles (Belgique), nous a rendu visite du 18 au 20 décembre 2001.

8 Diffusion de résultats

8.1 Diffusion de logiciels

Le projet VASY diffuse principalement deux logiciels : la boîte à outils CADP (voir § 4.1) et le compilateur TRAIAN (voir § 4.2). En 2001, les faits marquants concernant la diffusion de CADP sont les suivants :

- Nous avons préparé et diffusé des beta-versions successives (99-v, ... 99-z, 2000-a, ..., 2000-d) des outils CADP. Ce travail a débouché sur une version stable (CADP 2001 "Ottawa") disponible depuis le 13 juillet 2001. Cette nouvelle version comprend cinq nouveaux outils (BCG_MIN, EVALUATOR 3.0, OCIS, SVL et TGV) et apporte de nombreuses améliorations aux outils déjà existants.
- A cette occasion, nous avons rédigé et publié un rapport technique [22] et le 5^{e} numéro de la lettre électronique "*The CADP Newsletter*" (voir `http://www.inrialpes.fr/vasy/cadp/news5.html`) présentant les nouvelles avancées de CADP.
- Nous avons effectué des démonstrations publiques de CADP pendant l'école d'été SFM-PA :01 (*School on Formal Methods — Process Algebra*, Bertinoro, Italie, 23–28 juillet 2001) et pendant la conférence FORTE'2001 (Cheju Island, Corée, 28–31 août 2001).
- Le nombre de contrats de licence signés pour CADP est passé de 225 à 246 et nous avons octroyé en 2001 des licences CADP pour 965 machines différentes dans le monde.

La distribution du compilateur TRAIAN s'est poursuivie : 55 sites différents ont téléchargé le compilateur en 2001.

8.2 Animation de la communauté scientifique

- Depuis juillet 1999, H. Garavel est responsable du groupe de travail FMICS d'ERCIM (voir § 7.2).
- H. Garavel a été membre du comité de programme de TACAS'2001 (*7th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, Gênes, Italie, 2–6 avril 2001).

- R. Mateescu a été membre du comité de programme de FMICS'2001 (*6th International ERCIM Workshop on Formal Methods for Industrial Critical Systems*, Paris, France, 16–17 juillet 2001).
- H. Garavel a été membre du comité de programme de SOFTMC'2001 (*Workshop on Software Model Checking*, Paris, France, 23 juillet 2001).
- H. Garavel est membre du comité de programme de TACAS'2002 (*8th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, Grenoble, France, 6–14 avril 2002) et responsable (*tool chair*) de la sélection des logiciels présentés.
- H. Garavel est, conjointement avec Rance Cleaveland, président du comité de programme de FMICS'2002 (*7th International Workshop on Formal Methods for Industrial Critical Systems*, Málaga, Espagne, 12-13 juillet 2002).
- H. Garavel est membre du comité de programme de PDMC'2002 (*Workshop on Parallel and Distributed Model Checking*, Brno, République Tchèque, 19 août 2002).
- H. Garavel est membre du comité de programme de FORTE'2002 (*IFIP TC6 WG 6.1 Joint International Conference on Formal Techniques for Networked and Distributed Systems*, Rice University, Houston, Texas, 11–14 novembre 2002).
- En collaboration avec Stefania Gnesi (IEI-CNR, Pise) et Ina Schieferdecker (GMD-FOKUS, Berlin), H. Garavel est responsable d'un numéro spécial de la revue SCP (*Science of Computer Programming*) à paraître en 2002, qui rassemble les meilleurs articles de FMICS'2000.

8.3 Enseignement universitaire

Le projet VASY est équipe d'accueil pour :

- le DEA "Informatique : Système et Communications" commun à l'Institut National Polytechnique de Grenoble et à l'Université Joseph-Fourier, et
- le DEA "Informatique : communication et coopération dans les systèmes à agents" de l'Université de Savoie.

En 2001 :

- R. Mateescu a dispensé le cours "Temps Réel" destiné aux étudiants en 3^{e} année de l'ENSIMAG (21 heures annuelles).
- R. Mateescu a assuré, conjointement avec Flavio Oquendo, le cours "Méthodes formelles pour l'ingénierie des logiciels : spécification et vérification de protocoles" destiné aux étudiants du DEA d'informatique de l'Université de Savoie (24 heures annuelles). Ce cours a été transmis simultanément en visioconférence aux étudiants du DEA de l'Ecole des Mines de Saint-Etienne.
- R. Mateescu a participé au jury de DEA d'Adrian Curic à Grenoble le 20 juin 2001.
- R. Mateescu a participé au jury de DEA de Pierre-Ange Albertini à Grenoble le 21 juin 2001.
- R. Mateescu a participé au jury de DEA de Bruno Ondet à Annecy le 28 juin 2001.
- H. Garavel a participé au jury de la thèse de doctorat de Fabrice Baray, intitulée "Contri-

bution à l'intégration de la vérification de modèle dans le processus de conception co-design", soutenue à l'Université Blaise Pascal (Clermont-Ferrand, France) le 2 juillet 2001.
- H. Garavel a participé au jury de la thèse de doctorat de Lars-Åke Fredlund, intitulée "*A Framework for Reasoning about ERLANG Code*", soutenue au KTH *Royal Institute of Technology* (Stockholm, Suède) le 14 septembre 2001.
- R. Mateescu a participé au jury de DEA de Moussa Amrani à Grenoble le 4 septembre 2001.
- H. Garavel est membre de la commission de spécialistes (CSE) de l'Institut National Polytechnique de Grenoble (sections 26 et 27).

8.4 Participation à des colloques, séminaires, invitations

Nous avons présenté des communications dans plusieurs conférences et colloques internationaux (voir à ce sujet la liste de nos publications). En outre :

- H. Garavel a donné un exposé intitulé "*An Overview of CADP and SVL*" au colloque ETI (*Electronic Tool Integration*), Gênes, Italie, 1er avril 2001.
- H. Garavel a participé à la conférence TACAS'2001 (*Tools and Algorithms for Construction and Analysis of Systems*), Gênes, Italie, 2–6 avril 2001.
- H. Garavel a participé au colloque PFM'2001 (*Proofs for Mobility*), Gênes, Italie, 7 avril 2001.
- H. Garavel et R. Mateescu ont participé au colloque FMICS'2001 (ERCIM *Working Group on Formal Methods for Industrial Critical Systems*), Paris, France, 16–17 juillet 2001.
- A. Curic, H. Garavel, F. Lang et R. Mateescu ont participé à l'école d'été SFM-PA :01 (*School on Formal Methods — Process Algebra*), Bertinoro, Italie, 23–28 juillet 2001. H. Garavel y a donné un exposé intitulé "*An Overview of CADP 2001*".
- H. Garavel a effectué une visite de l'Université de Twente (Pays-Bas) du 24 au 29 septembre 2001, où il a donné un exposé intitulé "*SVL : a Scripting Language for Compositional Verification*".
- R. Mateescu a donné un exposé intitulé "*Specification and Verification of Concurrent Critical Systems*" à l'occasion de l'inauguration officielle du Centre National pour la Technologie de l'Information roumain (CNTI COLABORATOR), futur noyau de collaboration pour l'enseignement et la recherche roumaine (Bucarest, Roumanie, 26–30 septembre 2001).
- G. Stragier a participé aux *Journées Systèmes et Logiciels Critiques*, Grenoble, France, 6–8 novembre 2001.
- H. Garavel et R. Mateescu ont participé au colloque ETI (*Electronic Tool Integration*), Dortmund, Allemagne, 23–24 novembre 2001.
- G. Pace a participé à un séminaire sur les langages synchrones à Dagstuhl, Allemagne, du 3 au 7 décembre 2001. A cette occasion, il a donné un exposé intitulé *An Embedded Language Approach to Hardware Compilation Verification.*
- Les autres membres du projet VASY ont participé à un séminaire commun avec le projet SARDES à Autrans, France, du 5 au 7 décembre 2001.

9 Bibliographie

Ouvrages et articles de référence de l'équipe

[1] J.-C. Fernandez, H. Garavel, A. Kerbrat, R. Mateescu, L. Mounier, M. Sighireanu, « CADP (CÆSAR/ALDEBARAN Development Package) : A Protocol Validation and Verification Toolbox », *in : Proceedings of the 8th Conference on Computer-Aided Verification (New Brunswick, New Jersey, USA)*, R. Alur, T. A. Henzinger (éditeurs), *Lecture Notes in Computer Science*, *1102*, Springer Verlag, p. 437–440, août 1996.

[2] H. Garavel, M. Jorgensen, R. Mateescu, C. Pecheur, M. Sighireanu, B. Vivien, « CADP'97 – Status, Applications and Perspectives », *in : Proceedings of the 2nd COST 247 International Workshop on Applied Formal Methods in System Design (Zagreb, Croatia)*, I. Lovrek (éditeur), juin 1997.

[3] H. Garavel, J. Sifakis, « Compilation and Verification of LOTOS Specifications », *in : Proceedings of the 10th International Symposium on Protocol Specification, Testing and Verification (Ottawa, Canada)*, L. Logrippo, R. L. Probert, H. Ural (éditeurs), IFIP, North-Holland, p. 379–394, juin 1990.

[4] H. Garavel, M. Sighireanu, « Towards a Second Generation of Formal Description Techniques – Rationale for the Design of E-LOTOS », *in : Proceedings of the 3rd International Workshop on Formal Methods for Industrial Critical Systems FMICS'98 (Amsterdam, The Netherlands)*, J.-F. Groote, B. Luttik, J. Wamel (éditeurs), CWI, p. 187–230, Amsterdam, mai 1998. Présentation invitée.

[5] H. Garavel, *Compilation et vérification de programmes LOTOS*, Thèse de doctorat, Université Joseph Fourier (Grenoble), novembre 1989.

[6] H. Garavel, « Compilation of LOTOS Abstract Data Types », *in : Proceedings of the 2nd International Conference on Formal Description Techniques FORTE'89 (Vancouver B.C., Canada)*, S. T. Vuong (éditeur), North-Holland, p. 147–162, décembre 1989.

[7] H. Garavel, « An Overview of the Eucalyptus Toolbox », *in : Proceedings of the COST 247 International Workshop on Applied Formal Methods in System Design (Maribor, Slovenia)*, Z. Brezočnik, T. Kapus (éditeurs), University of Maribor, Slovenia, p. 76–88, juin 1996.

[8] H. Garavel, « OPEN/CÆSAR : An Open Software Architecture for Verification, Simulation, and Testing », *in : Proceedings of the First International Conference on Tools and Algorithms for the Construction and Analysis of Systems TACAS'98 (Lisbon, Portugal)*, B. Steffen (éditeur), *Lecture Notes in Computer Science*, *1384*, Springer Verlag, p. 68–84, Berlin, mars 1998. Version étendue parue comme rapport de recherche INRIA RR-3352.

[9] R. Mateescu, H. Garavel, « XTL : A Meta-Language and Tool for Temporal Logic Model-Checking », *in : Proceedings of the International Workshop on Software Tools for Technology Transfer STTT'98 (Aalborg, Denmark)*, T. Margaria (éditeur), BRICS, p. 33–42, juillet 1998.

[10] R. Mateescu, M. Sighireanu, « Efficient On-the-Fly Model-Checking for Regular Alternation-Free Mu-Calculus », *in : Proceedings of the 5th International Workshop on Formal Methods for Industrial Critical Systems FMICS'2000 (Berlin, Germany)*, S. Gnesi, I. Schieferdecker, A. Rennoch (éditeurs), *GMD Report 91*, p. 65–86, Berlin, avril 2000. Paru également comme rapport de recherche INRIA RR-3899.

[11] R. Mateescu, *Vérification des propriétés temporelles des programmes parallèles*, Thèse de doctorat, Institut National Polytechnique de Grenoble, avril 1998.

[12] R. Mateescu, « Efficient Diagnostic Generation for Boolean Equation Systems », *in : Proceedings of 6th International Conference on Tools and Algorithms for the Construction and Analysis of Systems TACAS'2000 (Berlin, Germany)*, S. Graf, M. Schwartzbach (éditeurs), *Lecture Notes in Computer Science*, *1785*, Springer Verlag, p. 251–265, mars 2000. Version étendue parue comme rapport de recherche INRIA RR-3861.

Articles et chapitres de livre

[13] H. Garavel, C. Viho, M. Zendri, « System Design of a CC-NUMA Multiprocessor Architecture using Formal Specification, Model-Checking, Co-Simulation, and Test Generation », *Springer International Journal on Software Tools for Technology Transfer (STTT) 3*, 3, juillet 2001, p. 314–331, paru également comme rapport de recherche INRIA RR-4041.

[14] R. Mateescu, M. Sighireanu, « Efficient On-the-Fly Model-Checking for Regular Alternation-Free Mu-Calculus », *Science of Computer Programming*, 2002, à paraître.

Communications à des congrès, colloques, etc.

[15] M. A. Cornejo, H. Garavel, R. Mateescu, N. de Palma, « Specification and Verification of a Dynamic Reconfiguration Protocol for Agent-Based Applications », *in : Proceedings of the 3rd IFIP WG 6.1 International Working Conference on Distributed Applications and Interoperable Systems DAIS'2001 (Krakow, Poland)*, A. Laurentowski, J. Kosinski, Z. Mossurska, R. Ruchala (éditeurs), IFIP, Kluwer Academic Publishers, p. 229–242, septembre 2001. Version étendue parue comme rapport de recherche INRIA RR-4222.

[16] H. Garavel, F. Lang, R. Mateescu, « Compiler Construction using LOTOS NT », *in : Proceedings of the International Conference on Compiler Construction CC 2002 (Grenoble, France)*, N. Horspool (éditeur), *Lecture Notes in Computer Science*, Springer Verlag, avril 2002. A paraître.

[17] H. Garavel, F. Lang, « SVL : a Scripting Language for Compositional Verification », *in : Proceedings of the 21st IFIP WG 6.1 International Conference on Formal Techniques for Networked and Distributed Systems FORTE'2001 (Cheju Island, Korea)*, M. Kim, B. Chin, S. Kang, D. Lee (éditeurs), IFIP, Kluwer Academic Publishers, p. 377–392, août 2001. Version étendue parue comme rapport de recherche INRIA RR-4223.

[18] H. Garavel, R. Mateescu, I. Smarandache, « Parallel State Space Construction for Model-Checking », *in : Proceedings of the 8th International SPIN Workshop on Model Checking of Software SPIN'2001 (Toronto, Canada)*, M. B. Dwyer (éditeur), *Lecture Notes in Computer Science*, *2057*, Springer Verlag, p. 217–234, Berlin, mai 2001. Version révisée parue comme rapport de recherche INRIA RR-4341 (décembre 2001).

[19] F. Lang, « Compositional Verification using SVL Scripts », *in : Proceedings of the International Conference on Tools and Algorithms for Construction and Analysis of Systems TACAS'2002 (Grenoble, France)*, J.-P. Katoen, P. Stevens (éditeurs), *Lecture Notes in Computer Science*, Springer Verlag, avril 2002. A paraître.

[20] R. Mateescu, « Local Model-Checking of Modal Mu-Calculus on Acyclic Labeled Transition Systems », *in : Proceedings of the 8th International Conference on Tools and Algorithms for the Construction and Analysis of Systems TACAS'2002 (Grenoble, France)*, P. Stevens, J.-P. Katoen (éditeurs), *Lecture Notes in Computer Science*, Springer Verlag, avril 2002. A paraître.

Rapports de recherche et publications internes

[21] A. Curic, « Techniques et outils pour la vérification de programmes parallèles comportant des données », *Mémoire de DEA, Université Joseph-Fourier (Grenoble)*, juin 2001.

[22] H. Garavel, F. Lang, R. Mateescu, « An Overview of CADP 2001 », *Rapport technique RT-254*, INRIA, décembre 2001.

[23] B. Ondet, « Vérification d'un langage formel de description d'architectures logicielles dynamiques », *Mémoire de DEA, Université de Savoie*, juillet 2001.

Divers

[24] H. Garavel, « Sixth ERCIM FMICS International Workshop », Ercim News 47, octobre 2001.