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

Joanna Payart [jusqu'au 31 octobre 2000]

Béatrice Claudio [jusqu'au 30 novembre 2000]

Véronique Roux [à partir du 1er novembre 2000]

Personnel Inria

Radu Mateescu [CR2 INRIA]

Personnel Bull

Massimo Zendri [responsable d'action DYADE, jusqu'au 31 mars 2000]

Nicolas Zuanon [responsable d'action DYADE, à partir du 15 mai 2000]

Ingénieurs experts

Moëz Cherif [ingénieur Reutel (ALCATEL), jusqu'au 31 octobre 2000]

Marc Herbert [ingénieur DYADE, jusqu'au 30 septembre 2000]

Frédéric Lang [ingénieur DYADE, depuis le 25 juillet 2000]

Stéphane Martin [ingénieur Reutel (ALCATEL), depuis le 20 novembre 2000]

Frédéric Perret [ingénieur DYADE, depuis le 20 novembre 2000]

Chercheur post-doctorant

Irina Smarandache [boursière INRIA Rhône-Alpes, jusqu'au 31 octobre 2000]

Stagiaires

Adrian Curic [élève-ingénieur UPB puis stagiaire DEA, depuis le 25 mars 2000]

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

Parmi les différentes approches existantes 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 ses *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 comportementales 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 trois 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

[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.

de sémantique formelle assez inquiétante 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 devrait conduire à un nouveau langage nommé E-LOTOS (*Extended*-LOTOS) [Que00] qui vise à conjuguer une expressivité 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 du langage E-LOTOS, appelée LOTOS NT (LOTOS *Nouvelle Technologie*) [4], dans laquelle nous avons introduit les 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 une *partie modules*, qui autorise 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).

– Depuis 1999, dans le cadre d'une collaboration avec ALCATEL et le projet PAMPA (Rennes), nous nous intéressons à la notation UML pour la modélisation des systèmes informatiques. Issue des travaux de Booch, Jacobson et Rumbaugh, cette notation unifie trois méthodes de développement à objets très utilisées, dans le souci de faciliter le développement d'outils inter-opérables.

[Que00] J. QUEMADA, EDITOR, « Information Technology – Enhancements to LOTOS (E-LOTOS) », ISO/IEC FDIS 15437 ballot, novembre 2000.

Dans ce contexte, nous cherchons à réutiliser au mieux les outils de CADP, en les connectant et en les adaptant à UML.

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 :** arbitrage de bus, cohérence de caches, conception conjointe matériel-logiciel ;
- **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 :** contrôle de trafic aérien ;
- **systèmes répartis :** mémoire virtuelle, systèmes de fichiers répartis, ingénierie concourante, algorithmes d'élection ;
- **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], Moëz Cherif, Marc Herbert, 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 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 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 développer des outils de simulation, de vérification et de génération de tests sur des graphes représentés sous forme implicite. Ces outils peuvent être réalisés 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, qui évalue à la volée des formules de μ-calcul régulier sans alternance,
  - 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 permet d'afficher en POSTSCRIPT une représentation 2D d'un graphe,
  - BCG_EDIT, qui permet de modifier interactivement la représentation graphique produite par BCG_DRAW,
  - BCG_IO, qui effectue des conversions entre BCG et d'autres formats d'automates,
  - BCG_LABELS, qui permet de masquer et/ou de renommer par des expressions régulières les étiquettes d'un graphe BCG,
  - BCG_MIN, qui permet de minimiser des graphes BCG 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*) 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 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 servant à calculer des prédicats de base et des modalités temporelles portant sur les 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.

[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 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.

A ces outils s'ajoutent ceux développés par le laboratoire VERIMAG et/ou le 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 le 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], Marc Herbert, 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 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.

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 2000, 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 du simulateur OCIS

Participants : Moëz Cherif, Hubert Garavel.

Nous avons poursuivi nos travaux relatifs au simulateur OCIS (OPEN/CÆSAR *Interactive Simulator*) dont le développement avait débuté en 1998, à la demande de nos partenaires de BULL qui souhaitaient disposer d'un outil pour faciliter la mise au point des descriptions formelles de protocoles en LOTOS. En 1999, le développement d'OCIS a pris un nouvel essor dans le cadre de la participation de VASY au contrat REUTEL-2000 (voir § 6.4) dont les partenaires ont souhaité utiliser OCIS sur des descriptions UML en l'interfaçant avec l'outil UMLAUT en cours de développement dans le projet PAMPA.

A terme, l'objectif de ce travail est de disposer, pour le développement de protocoles et de systèmes distribués, d'un simulateur offrant des fonctionnalités comparables à celles présentes dans les débogueurs et environnements de développement existant pour les langages séquentiels. A la différence des simulateurs ordinaires, OCIS n'est pas lié à un langage source précis : par construction, il est possible de l'utiliser avec tout langage dont le compilateur implémente l'interface de programmation OPEN/CÆSAR [8].

Les principales améliorations apportées à OCIS en 2000 sont les suivantes :

- Plusieurs erreurs ont été corrigées et certaines parties du simulateur ont été réécrites afin de parvenir à un code plus simple et plus robuste : parmi elles, on peut mentionner la récupération des exceptions levées par le traitement des données et la recompilation "en ligne", qui permet de modifier et recompiler la description source (LOTOS, UML, etc.) sans interrompre la session de simulation en cours.

- Les performances du simulateur ont été améliorées pour permettre de traiter efficacement les scénarios de simulation complexes. Ceci a été obtenu en optimisant le moteur de simulation (écrit en langage C) ainsi que le protocole de communication entre ce moteur et l'interface graphique (développée avec les outils TCL/TK et TIX).

- L'ergonomie de l'interface utilisateur a été accrue, notamment en affichant, pour chaque état, le nombre de transitions sortantes explorées et non explorées, et en coloriant chaque état de manière différente selon qu'il s'agit d'un état puits ou selon le niveau d'exploration des transitions sortant de cet état (toutes, certaines, ou aucune).

- La possibilité de manipuler le scénario de simulation courant — c'est-à-dire les parties du graphe explorées — et de le sauvegarder dans un fichier BCG a été affinée. Ainsi, OCIS permet désormais d'éditer le scénario courant pour en supprimer certaines branches.

De plus, il est désormais possible de réexécuter un scénario contenu dans un fichier BCG avec une description source différente de celle qui a servi à produire ce scénario, OCIS éliminant automatiquement les parties du scénario qui ne sont pas acceptées par la nouvelle spécification.

En 2000, OCIS a fait l'objet d'une utilisation quotidienne par Nicolas Zuanon dans le cadre de l'action FORMALFAME du GIE DYADE (voir § 6.2). Ce test intensif a permis à OCIS d'atteindre un degré de maturité suffisant pour qu'il soit intégré à la boîte à outils CADP.

5.1.2 Développement de l'outil BCG_MIN

Participants : Moëz Cherif, 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" (connus aussi sous le nom de "processus de décision markoviens à temps discret"), qui comportent à la fois des transitions ordinaires et des transitions étiquetées par une probabilité $p \in [0, 1]$,
- des systèmes de transitions "stochastiques" (connus aussi sous le nom de "processus de décision markoviens à temps continu"), qui comportent à la fois des transitions ordinaires et des transitions étiquetées par un paramètre réel λ qui détermine une loi de la forme $prob(x > t) = e^{-\lambda t}$.

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].

En 2000, les améliorations apportées ont été les suivantes :

- L'outil BCG_MIN a été intégré dans la boîte à outils CADP, ce qui a permis de l'utiliser intensivement sur plusieurs études de cas, notamment dans le cadre de l'action FORMALFAME (voir 6.2). Quelques erreurs résiduelles ont ainsi été détectées et corrigées.

[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.

- Les performances de BCG_MIN sont désormais reconnues : ainsi, BCG_MIN est présenté 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. A titre d'exemple, le plus gros graphe traité à ce jour par BCG_MIN comporte 6 millions d'états et 11 millions de transitions. Par ailleurs, l'utilisation du format très compact BCG entraîne des gains à la fois en place disque et en rapidité.

- Les modèles probabilistes (*resp.* stochastiques) traités par BCG_MIN ont été généralisés. BCG_MIN est désormais capable de minimiser des graphes comportant — outre des transitions ordinaires et des transitions étiquetées par une probabilité p (*resp.* un paramètre λ) — des transitions "mixtes" étiquetées simultanément par une action ordinaire et par une probabilité p (*resp.* un paramètre λ). Cette extension permet d'utiliser BCG_MIN sur de nombreux modèles probabilistiques[1] et stochastiques[2].

Enfin, à partir de novembre 2000, nous avons entrepris d'optimiser les structures de données internes de BCG_MIN afin d'être en mesure de traiter des graphes de taille encore plus grande (travail de F. Perret).

5.1.3 Développement de l'outil SVL 2.0

Participants : Hubert Garavel, Marc Herbert, Frédéric Lang.

Lorsque l'on essaie de vérifier des systèmes assez complexes, le problème de l'explosion d'états (voir § 2.2) survient fréquemment. Les techniques de vérification compositionnelle avec abstractions [GLS96,KM97] intégrées à la boîte à outils CADP visent à éviter l'explosion d'états

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.*

[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. Also available as Technical Report SEN-R0008, CWI, Amsterdam, March 2000.

[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. Extended version with proofs available as Research Report VERIMAG RR97-01.

et fournissent souvent une réponse appropriée à ce problème, y compris sur des études de cas industrielles significatives [CGM+96].

En 1997–1998, 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 de la vérification compositionnelle et pour rendre celle-ci réellement utilisable dans un contexte industriel (et notamment par BULL, qui souhaitait utiliser ces techniques).

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.

En 2000, nous avons effectué une réécriture complète du compilateur SVL. Plusieurs éléments nous ont conduits à cette décision. L'évolution de la boîte à outils CADP (notamment l'ajout de l'outil BCG_MIN) exigeait d'étendre le langage et le compilateur SVL. Mais cette modification était rendue difficile, du fait que le générateur de compilateurs FNC-2 utilisé pour construire SVL n'était plus maintenu et qu'il contenait encore de nombreuses bogues résiduelles (voir le rapport d'activité 1999 de VASY pour une évaluation de l'outil FNC-2).

Pour développer la version 2.0 du compilateur SVL, nous avons donc choisi de remettre en cause nos choix antérieurs et de nous appuyer sur une technologie plus simple et plus fiable : l'utilisation combinée de l'outil SYNTAX développé à l'INRIA Rocquencourt (pour les aspects lexicaux et syntaxiques) et du langage LOTOS NT (pour les aspects sémantiques). Par rapport à la version antérieure, SVL 2.0 apporte les améliorations suivantes :

- SVL 2.0 prend en compte l'outil BCG_MIN, dont les bonnes performances apportent une efficacité accrue au processus de vérification compositionnelle.

- La syntaxe du langage SVL 2.0 a été étendue. Il est désormais possible d'insérer directement, au sein d'un programme SVL, des lignes de *shell-script* qui seront reproduites telles

[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. Full version available as INRIA Research Report RR-2958, `http://www.inria.fr/rrrt/rr-2958.html`.

quelles dans le code *shell* produit par SVL. On peut ainsi utiliser, dans un programme SVL, toute la puissance des instructions *shell* : conditionnelles, boucles, variables, fonctions, etc. Ceci a aussi permis de simplifier la syntaxe SVL en supprimant certaines instructions qui sont désormais déléguées au niveau du *shell*.

- Les paramètres de la vérification (outils à utiliser, options à transmettre aux outils, noms de fichiers choisis, etc.) sont désormais représentés par des variables *shell* ordinaires, ce qui permet à l'utilisateur de les manipuler aisément s'il ne souhaite pas se reposer sur les valeurs par défaut de ces paramètres. Ceci rend le langage SVL très évolutif, puisque l'ajout de nouveaux paramètres, notamment ceux liés à l'intégration de nouveaux outils, ne nécessite pas de modifier la grammaire du langage SVL.

- SVL comporte des macro-opérateurs algébriques qui permettent de spécifier globalement des opérations (abstraction, renommage, minimisation) devant être appliquées à chaque processus séquentiel ou même, par propagation récursive, à chaque fois que deux processus sont composés en parallèle. Ces macro-opérateurs sont expansés par des transformations algébriques (règles de distributivité, de permutation d'opérateurs, etc.). En SVL 2.0, ces règles ont été redéfinies pour être plus "intelligentes" et plus efficaces.

- Enfin, l'ergonomie de l'outil SVL 2.0 a été améliorée en de nombreux points : localisation des fichiers produits, options permettant d'effacer les fichiers produits, uniformisation des règles de priorité et d'associativité des opérateurs SVL, possibilité d'obtenir le programme SVL expansé après élimination des macro-opérateurs algébriques, refonte totale de la documentation, mise à jour des exemples de démonstration figurant dans la distribution CADP afin d'utiliser SVL 2.0.

Le compilateur SVL 2.0 a été intégré à la boîte à outils CADP. Il comporte 1 300 lignes de code SYNTAX et C, 1 900 lignes de code LOTOS NT et 800 lignes de code *Bourne shell*. L'utilisation combinée des technologies SYNTAX, LOTOS NT et TRAIAN a donné entière satisfaction, tant pour la qualité du compilateur SVL produit que pour la facilité et la rapidité du processus de développement lui-même. En outre, l'utilisation du compilateur TRAIAN sur ce projet de taille significative a permis de détecter et de corriger quelques bogues de TRAIAN.

5.1.4 Développement de l'outil EVALUATOR 3.0

Participant : Radu Mateescu.

L'outil EVALUATOR 3.0 permet de vérifier une propriété exprimée en logique temporelle (μ-calcul régulier d'alternance 1) sur un système de transitions étiquetées. Il utilise un algorithme de résolution locale de systèmes d'équations booléennes qui fonctionne à la volée (voir § 2.2), de manière à contourner le problème de l'explosion d'états : cette approche permet, dans de nombreux cas, de détecter des erreurs dans une spécification sans construire entièrement le système de transitions avant de commencer la vérification d'une propriété, mais en le générant dynamiquement au fur et à mesure des besoins.

EVALUATOR 3.0 est développé au moyen de l'environnement OPEN/CÆSAR [8] : de ce fait, il ne dépend pas d'un langage de spécification particulier, mais peut être utilisé pour tout langage

dont le compilateur implémente l'interface de programmation OPEN/CÆSAR pour l'exploration des systèmes de transitions étiquetées. Enfin, EVALUATOR 3.0 contient un algorithme original permettant de fournir à l'utilisateur des diagnostics (par exemple, une séquence de transitions étiquetées justifiant la valeur de vérité d'une formule logique).

Les travaux relatifs à EVALUATOR 3.0 ont fait l'objet d'une publication dans un colloque [13] ; une version étendue de cet article devrait paraître dans une revue [12]. Notre algorithme de génération de diagnostics a aussi donné lieu à publication [14].

En 2000, nos travaux ont essentiellement porté sur l'optimisation des performances d'EVALUATOR 3.0. Des améliorations importantes ont été obtenues en *spécialisant* EVALUATOR 3.0 pour qu'il traite de manière optimale deux cas particuliers fréquemment rencontrés en pratique :

1. Nous avons d'abord traité le cas particulier où le système de transitions à vérifier est un graphe au format BCG (auquel on applique EVALUATOR 3.0 par l'intermédiaire de l'outil BCG_OPEN qui fournit une connexion entre les environnements BCG et OPEN/CÆSAR).

   Ce cas particulier sort du cadre de la vérification à la volée, puisqu'il suppose la construction exhaustive du système de transitions préalablement à la vérification et se trouve, de fait, soumis au risque d'explosion d'états. Néanmoins, si le graphe BCG peut être construit entièrement, cette approche est intéressante en pratique, surtout lorsque l'on doit évaluer plusieurs formules logiques, puisque la construction du graphe (opération coûteuse) n'est effectuée qu'une seule fois. De plus, il existe certaines formules, notamment les *invariants* (prédicats qui doivent être satisfaits par chaque état) dont la validité ne peut être décidée qu'après exploration de tous les états accessibles.

   Deux améliorations ont été apportées à EVALUATOR 3.0 lorsqu'il travaille sur un graphe au format BCG :

   - En utilisant de manière judicieuse la représentation des états mise en œuvre dans l'environnement BCG, nous avons optimisé la représentation interne des données utilisées par l'algorithme à la volée d'EVALUATOR 3.0. Ceci a permis d'obtenir une réduction moyenne de 5% de la mémoire et de 20% du temps d'exécution (chiffres mesurés sur les divers exemples de démonstration fournis avec CADP).
   - Nous avons étendu l'outil EVALUATOR 3.0 avec un algorithme de vérification *énumératif* permettant d'évaluer une formule logique sur un graphe BCG. Cet algorithme effectue une résolution itérative globale de systèmes d'équations booléennes, c'est-à-dire qu'il calcule la valeur de vérité de toutes les variables du système (ce qui équivaut au calcul de l'ensemble des états du graphe qui satisfont la formule). Cet algorithme énumératif utilise la relation "prédécesseur" entre les états du graphe, contrairement à l'algorithme à la volée qui exploite la relation "successeur" entre états.

     Comme l'algorithme à la volée, l'algorithme énumératif a une complexité linéaire en taille du graphe (nombre d'états et de transitions) et de la formule à vérifier (nombre d'opérateurs). Toutefois, l'algorithme énumératif gère plus efficacement les états explorés lorsque le graphe est connu entièrement, ce qui permet des gains

conséquents en mémoire : jusqu'à 50% sur les exemples de démonstration fournis avec CADP.

2. Nous avons ensuite traité le cas particulier où le système de transitions à vérifier est un graphe sans circuit.

   Nous nous sommes intéressés à ce problème dans le cadre de l'action FORMALFAME du GIE DYADE (voir § 5.3.1 et 6.2) : le test aléatoire de l'architecture multi-processeur FAME développée par BULL produisait des traces (séquences d'exécution finies) comportant des dizaines ou des centaines de milliers de transitions dont BULL souhaitait s'assurer de la cohérence. Nous avons proposé d'exprimer les propriétés de cohérence (ordonnancement des événements conforme au comportement attendu de l'architecture) par des formules de logique temporelle et d'utiliser EVALUATOR 3.0 pour vérifier que les traces satisfaisaient ces formules.

   Bien que les performances fournies par EVALUATOR 3.0 soient acceptables, il est apparu que l'algorithme à la volée pouvait être optimisé dans le cas des graphes sans circuit (en particulier, dans le cas des traces). Nous avons conçu et mis en œuvre un algorithme spécialisé qui permet d'atteindre une complexité moyenne optimale en n'explorant que le fragment du graphe strictement nécessaire à l'évaluation de la formule.

   Sur les exemples de traces fournies par BULL, nous avons pu constater des gains de 30% en mémoire et de 10% en temps d'exécution par rapport à l'algorithme à la volée d'EVALUATOR 3.0.

5.1.5 Développement de l'outil EVALUATOR 4.0

Participant : Radu Mateescu.

La boîte à outils CADP comprend 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 [13, 12], qui permet de vérifier à la volée des formules du μ-calcul régulier d'alternance 1 sans données.

Ces deux logiciels sont robustes et ont été utilisés avec succès pour valider plusieurs applications industrielles critiques. Toutefois, dans un souci d'ergonomie, il serait souhaitable de proposer aux utilisateurs de CADP un seul outil d'évaluation de logique temporelle qui offrirait l'ensemble des fonctionnalités fournies actuellement 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.

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). A notre connaissance, il n'existe à l'heure actuelle aucun outil de vérification réunissant toutes ces fonctionnalités.

Nous avons défini la syntaxe et la sémantique du langage d'entrée d'EVALUATOR 4.0, ainsi que la transformation du problème de la vérification en la résolution locale d'un système d'équations booléennes paramétré par des variables typées.

Nous avons également commencé l'implémentation de la "partie avant" d'EVALUATOR 4.0, en choisissant l'outil SYNTAX comme générateur de compilateurs et LOTOS NT comme langage de développement du noyau de compilation ; le compilateur TRAIAN (voir § 4.2 et 5.2.2) est utilisé pour traduire les modules LOTOS NT en C.

La version courante d'EVALUATOR 4.0 comprend environ 6 500 lignes de code LOTOS NT et 7 200 lignes de code SYNTAX et C. Elle effectue l'analyse syntaxique et sémantique du langage d'entrée (liaison des identificateurs, typage des expressions et des formules, résolution des surcharges), ainsi que les transformations préliminaires sur les formules temporelles (élimination des opérateurs dérivés et traduction en forme normale positive).

5.1.6 Parallélisation d'algorithmes de vérification

Participants : Hubert Garavel, Radu Mateescu, Irina Smarandache.

Nous avons entrepris en 1999 des travaux sur l'utilisation de machines parallèles pour améliorer les performances des algorithmes de vérification énumérative (travail post-doctoral d'I. Smarandache).

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.

En 2000, nous avons travaillé pour paralléliser 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 dizaines ou de centaines de machines, de gagner plusieurs ordres de grandeur dans la complexité des graphes traités.

A cette fin, nous avons développé deux outils prototypes :

- DISTRIBUTOR est un outil qui répartit la construction d'un graphe sur N machines (numérotées de 0 à $N-1$) communiquant deux à deux au moyen de *sockets*. Le graphe à construire est défini au moyen de l'interface de programmation OPEN/CÆSAR [8], ce qui rend l'outil DISTRIBUTOR indépendant de tout langage de spécification particulier.

  Chaque machine est chargée de construire une partie du graphe sous forme d'un fichier au format BCG. Les états sont répartis entre les mémoires locales des machines au moyen d'une fonction de hachage (déterminée statiquement) qui calcule pour chaque état S le numéro $h(S)$ de la machine responsable du traitement et du stockage de l'état S. Cette fonction doit être choisie judicieusement afin d'obtenir un bon équilibrage de charge entre les machines.

La construction du graphe prend fin lorsque chaque machine a traité tous les états qui lui sont attribués par la fonction de hachage et que tous les canaux de communication entre machines sont vides : cette terminaison est détectée par un algorithme réparti à base d'anneau virtuel à jeton implémenté dans DISTRIBUTOR.

– BCG_MERGE est un outil qui permet, une fois terminée l'exécution de DISTRIBUTOR, d'assembler les différentes parties de graphe construites sur chaque machine afin d'obtenir, au moyen d'un algorithme de renumérotation d'états, un fichier BCG unique contenant le graphe complet.

Nous avons expérimenté les outils prototypes DISTRIBUTOR et BCG_MERGE dans deux types d'environnements de calcul : d'une part, le réseau local de l'équipe VASY (stations de travail fonctionnant sous SOLARIS ou LINUX et connectées par ETHERNET 100 Mbits) et d'autre part, la grappe de PCs développée par les projets INRIA SIRAC et APACHE ; dans la configuration utilisée, cette grappe comportait une douzaine de PCs sous LINUX interconnectés par un bus SCI.

Nos expériences sur la grappe de PCs ont concerné diverses applications de grande taille (protocole HAVi de PHILIPS, protocoles d'élection sur un anneau à jeton, protocole d'arbitrage du bus SCSI-2). Les premiers résultats montrent la possibilité de générer des graphes de grande taille (autour de 15 millions d'états) avec des gains de vitesse (*speedup*) importants et indiquent que la fonction de hachage choisie assure un bon équilibrage de charge. Ce travail a donné lieu à un article soumis à publication.

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 : *Nous avons maintenu et amélioré nos outils de compilation pour LOTOS, qui sont utilisés pour le traitement d'études de cas de complexité significative (notamment industrielles). Nous avons aussi poursuivi le développement du compilateur TRAIAN pour le langage LOTOS NT, utilisé actuellement pour la génération de compilateurs et prochainement pour le développement d'applications embarquées sur cartes à puces.*

5.2.1 Compilation des types du langage LOTOS

Participants : Adrian Curic, Hubert Garavel, Radu Mateescu.

Le compilateur CÆSAR.ADT permet de traduire en C les types abstraits algébriques LOTOS. Nos travaux récents portent sur la compilation des types LOTOS qui sont directement ou mutuellement récursifs (listes, arbres, files d'attente, etc.). Pour traduire en C ces types LOTOS, on utilise des pointeurs vers des structures ou des unions avec discriminants.

Bien que ce schéma de traduction soit celui que choisirait "naturellement" un programmeur humain, et bien que le compilateur CÆSAR.ADT possède diverses heuristiques visant à minimiser le nombre de pointeurs en "cassant" les dépendances cycliques entre types, l'utilisation de types pointeurs reste onéreuse en mémoire, notamment dans le cadre de la vérification énumérative (*model-checking*), où il est souvent nécessaire de mémoriser simultanément un nombre important (plusieurs millions) de valeurs contenues dans tous les états accessibles du système.

En effet, outre le coût individuel en mémoire de chaque pointeur, on est confronté au problème des copies multiples (certaines structures de données identiques sont allouées plusieurs fois) et au problème de la désallocation (certaines structures de données doivent être libérées lorsqu'elles ne sont plus référencées). Jusqu'à présent, l'approche suivie pour CÆSAR.ADT consistait à réduire autant que possible le nombre de copies multiples grâce au partage de pointeurs entre plusieurs structures de données (*structural sharing*) et à confier la désallocation à un ramasse-miettes conservatif [Boe93].

En 2000, nous avons expérimenté une approche alternative consistant à représenter les structures de données sous forme canonique, ce qui revient à détecter les sous-termes algébriques identiques afin de les mettre en facteur. Pour ce faire, nous avons développé une bibliothèque logicielle générique (3600 lignes de code C) pour la factorisation des termes algébriques typés. Dans son principe, cette bibliothèque repose sur une table de hachage permettant de conserver chaque structure de données allouée de manière unique et, par conséquent, d'éviter qu'une même structure de données ne soit allouée plusieurs fois.

Outre le gain résultant de la factorisation, cette approche permet également de réduire la taille des structures de données en remplaçant chaque pointeur vers une structure de données par un entier (plus court qu'un pointeur), qui correspond à l'indice de cette structure de données dans la table de hachage. Ce remplacement permet également de réduire la taille des états.

Nous avons développé une variante prototype du compilateur CÆSAR.ADT qui engendre un code C modifié pour tirer parti de la nouvelle bibliothèque.

Cette approche a été expérimentée sur deux *benchmarks* EVALEXP(n) et EVALSYM(n) transmis par Jan-Friso Groote (CWI, Amsterdam). Ces *benchmarks* servent à tester les performances de diverses implémentations de langages fonctionnels en produisant un nombre de sous-termes identiques exponentiel en fonction de la valeur du paramètre n. Alors que les performances maximales publiées pour les outils concurrents s'arrêtaient à $n = 27$, la version modifiée de CÆSAR.ADT a pu atteindre $n = 35$ sur une machine DELL POWEREDGE 2300 fonctionnant sous LINUX. De plus, les gains en mémoire s'accompagnent d'une légère augmentation (5–10%) de la rapidité d'exécution par rapport à la version antérieure de CÆSAR.ADT utilisant le ramasse-miettes conservatif.

Nous avons ensuite utilisé la version modifiée de CÆSAR.ADT pour vérifier énumérativement un protocole de reconfiguration dynamique d'agents mobiles développé au sein du projet SIRAC. En comparant les résultats avec ceux précédemment obtenus sur ce même protocole [Agu99],

[Boe93] H. J. BOEHM, «Space Efficient Conservative Garbage Collection», *in: Proceedings of the ACM SIGPLAN '93 Conference on Programming Language Design and Implementation*, ACM SIGPLAN Notices 28, 6, p. 197–206, juin 1993.

[Agu99] M. AGUILAR CORNEJO, «Etude comparative et application des techniques de description formelle LOTOS et E-LOTOS à des protocoles pour les systèmes répartis», *Mémoire de DEA*, Université

on constate que la taille mémoire allouée pour les termes algébriques chute de 295 Moctets à 1.2 Moctets, que la taille de chaque état est réduite de 20% et que la quantité totale de mémoire nécessaire à la vérification est divisée par deux.

Enfin, la nouvelle bibliothèque étant suffisamment générique pour contenir d'autres informations que des termes algébriques, nous l'avons employée pour stocker des fragments d'états, selon la méthode proposée en [GRRV89] : dans cette approche, le vecteur d'état produit par CÆSAR est découpé en fragments correspondant aux sous-processus parallèles du système, chacun des fragments étant conservé dans une table de hachage séparée. Nos expériences ont montré que l'on pouvait ainsi diviser par quatre la quantité totale de mémoire nécessaire à la vérification.

5.2.2 Développement du compilateur TRAIAN pour LOTOS NT

Participants : Hubert Garavel, Marc Herbert, Frédéric Lang.

En 2000, nous avons poursuivi nos travaux relatifs au langage LOTOS NT et au compilateur TRAIAN (voir § 4.2).

Les améliorations importantes apportées en 1999 au compilateur TRAIAN (voir le rapport d'activité 1999 de VASY) ont été finalisées. Ce travail a débouché en février 2000 sur une version 2.0 de TRAIAN remplaçant la précédente version 1.0 issue de la thèse de Mihaela Sighireanu (septembre 1998). Cette nouvelle version fonctionne sous quatre systèmes d'exploitation (SUNOS, SOLARIS, LINUX et WINDOWS). Elle comporte des bibliothèques prédéfinies, de la documentation en ligne et quatre exemples de démonstration du compilateur.

Cette nouvelle version de TRAIAN a fait l'objet d'une utilisation intensive par l'équipe VASY, notamment comme outil de développement de compilateurs. En particulier, TRAIAN a été utilisé avec succès pour deux outils de vérification avancée : SVL 2.0 (voir § 5.1.3) et EVALUATOR 4.0 (voir § 5.1.5). L'approche suivie utilise conjointement l'outil SYNTAX développé à l'INRIA Rocquencourt (pour produire les analyseurs lexicaux et syntaxiques) et l'outil TRAIAN (pour traduire en C les programmes LOTOS NT décrivant les arbres de syntaxe abstraite, les opérations de manipulation de ces arbres, les traitements sémantiques et la génération de code).

Cette utilisation intensive a permis de tester de nouveaux pans du langage LOTOS NT et de corriger quelques erreurs résiduelles dans le compilateur TRAIAN. Par ailleurs, le manuel d'utilisation du langage LOTOS NT [16] a été entièrement révisé. Ces améliorations ont conduit à une nouvelle distribution TRAIAN 2.1 (novembre 2000).

5.2.3 Développement et portage de logiciels

Participants : Moëz Cherif, Hubert Garavel, Marc Herbert, Radu Mateescu, Stéphane

Joseph Fourier et Institut National Polytechnique de Grenoble, juin 1999.

[GRRV89] S. GRAF, J.-L. RICHIER, C. RODRÍGUEZ, J. VOIRON, « What are the Limits of Model Checking Methods for the Verification of Real Life Protocols? », *in : Proceedings of the 1st Workshop on Automatic Verification Methods for Finite State Systems (Grenoble, France)*, J. Sifakis (éditeur), *Lecture Notes in Computer Science, 407*, Springer Verlag, p. 275–285, juin 1989.

Martin.

En 2000, nous avons effectué diverses activités logicielles relatives à la boîte à outils CADP :

- Pour permettre aux outils CADP de fonctionner avec les versions récentes du système d'exploitation LINUX, nous avons dû faire migrer tous les programmes CADP de la version 5 de la bibliothèque C (LIBC5) vers la version 6 (LIBC6). Nous avons également dû résoudre divers problèmes de compatibilité existant entre les distributions REDHAT et DEBIAN. Enfin, nous avons pris en compte le cas des ordinateurs portables fonctionnant avec LINUX.

- Le portage des outils CADP vers le système d'exploitation WINDOWS de MICROSOFT entrepris en 1999 s'est poursuivi, notamment dans le cadre du contrat REUTEL-2000 (voir § 6.4). En 2000, nos efforts se sont focalisés sur les programmes C de CADP (le portage des interfaces graphiques, des *shells-scripts* et de la documentation de CADP ayant été achevé en 1999). Nous avons d'abord mis à jour notre environnement de développement multi-plateformes en installant les dernières versions du compilateur croisé GCC et de la bibliothèque MINGW32 qui permettent la production de programmes exécutables WINDOWS depuis un environnement UNIX. Ensuite, nous avons progressivement porté vers WINDOWS tous les outils de CADP écrits en C, ce qui a nécessité d'importants efforts pour tester le bon fonctionnement de ces programmes et les modifier si nécessaire.

  Ce travail de portage nous a permis de mettre en évidence deux anomalies dans les outils CYGWIN de la société REDHAT et une anomalie dans la version WINDOWS de TCL/TK de la société AJUBA (anciennement SCRIPTICS). Ces anomalies ont été signalées et corrigées dans les versions récentes CYGWIN et TCL/TK.

- Nous avons collaboré avec Solofo Ramangalahy et Séverine Simon du projet PAMPA (Rennes) pour intégrer le générateur de tests TGV au sein de l'interface graphique EUCALYPTUS. Nous avons fourni au projet PAMPA les outils de développement et le support technique pour porter l'outil TGV vers WINDOWS.

- Nous avons collaboré avec Laurent Mounier du laboratoire VERIMAG pour améliorer les outils ALDEBARAN, DES2AUT, EXP.OPEN et PROJECTOR. Nous avons étendu ces outils pour leur permettre d'utiliser le format BCG et les bibliothèques CAESAR_HIDE et CAESAR_RENAME. Nous avons effectué des tests minutieux pour répertorier les anomalies de fonctionnement connues à ce jour, dont neuf ont été corrigées par nos soins. Nous avons porté ces outils vers LINUX LIBC6 et WINDOWS. Enfin, nous avons mis à jour la documentation correspondante.

- L'interface graphique EUCALYPTUS a été étendue pour intégrer le nouvel outil BCG_MIN (voir § 5.1.2) et modifiée en profondeur pour permettre à l'utilisateur de lancer simultanément plusieurs outils de vérification (*multithreading*). L'outil de visualisation de graphes BCG_EDIT a été amélioré, notamment en ce qui concerne l'affichage de graphes de grande taille.

– Nous avons simplifié la procédure de configuration des outils CADP, afin de minimiser le nombre de variables d'environnement que l'utilisateur doit positionner lui-même en fonction du système d'exploitation et de la configuration locale. Désormais, toutes ces variables (sauf une) reçoivent une valeur par défaut appropriée, ceci quel que soit le système d'exploitation considéré. Par ailleurs, le programme TST, qui vérifie si les outils CADP ont été correctement installés, a été revu et étendu afin de détecter automatiquement les problèmes de configuration les plus couramment rencontrés.

5.3 Etudes de cas et applications pratiques

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

Résumé : *Nous accordons une grande importance au traitement d'exemples réalistes, qui nous permet de vérifier l'adéquation de nos méthodes et outils, et d'identifier de nouvelles orientations de recherche pour résoudre les problèmes rencontrés. En 2000, nous avons traité plusieurs études de cas, notamment dans le cadre de notre coopération avec BULL et avec le projet BIP.*

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

Participants : Hubert Garavel, Radu Mateescu, Massimo Zendri, Nicolas Zuanon.

En collaboration avec le projet PAMPA (Rennes), nous travaillons depuis octobre 1998 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. Cette collaboration a été officialisée en 1999 par la création de l'action FORMALFAME du GIE BULL-INRIA DYADE (voir § 6.2).

L'action FORMALFAME 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. Les outils de simulation et de vérification CADP sont utilisés pour s'assurer de la correction du modèle de référence. L'outil TGV est employé pour produire des jeux de tests "abstraits" déduits du modèle de référence ; ces tests abstraits sont ensuite traduits automatiquement en tests exécutables qui, appliqués dans l'environnement de test utilisé par BULL, serviront à valider l'implémentation du produit final.

Depuis décembre 1999, les travaux de FORMALFAME portent sur une nouvelle version de l'architecture FAME à base de processeurs INTEL ITANIUM et, plus précisément, sur la validation d'un circuit appelé B-SPS qui implémente le protocole de cohérence de caches de l'architecture FAME. Le fonctionnement de ce protocole est décrit par un document de référence (100 pages) combinant des spécifications informelles en langue naturelle avec des tables états/transitions.

L'implémentation du circuit B-SPS est décrite par un modèle en langage VERILOG. Un simulateur logiciel en langage C++ de ce circuit est également disponible pour valider les résultats générés par le modèle VERILOG.

En 2000, les travaux de FORMALFAME ont concerné les points suivants :

- *Description formelle du protocole de cohérence de caches* : nous avons réalisé une modélisation du circuit B-SPS et de son environnement (processeurs). Cette description, qui prend en compte les dernières évolutions du protocole et le traitement des collisions d'accès à une même adresse dues à la concurrence des transactions, comporte 12 processus concurrents (5 000 lignes de code LOTOS).

  Dans un second temps, une variante plus abstraite de cette description LOTOS a été développée. Comportant 7 processus LOTOS (4 500 lignes de code), cette variante est destinée à la validation des traces (voir ci-après) ; elle se concentre sur le fonctionnement observable du circuit B-SPS en faisant abstraction de l'environnement (processeurs) et de certains aspects d'implémentation.

- *Simulation et vérification des descriptions formelles* : la mise au point des deux descriptions LOTOS a été facilitée par la disponibilité, dès mars 2000, du simulateur graphique OCIS (voir § 5.1.1). L'action FORMALFAME a joué un rôle significatif pour le test en vraie grandeur et la mise au point de cet outil. Une vérification énumérative des descriptions LOTOS a été brièvement tentée, mais s'est heurtée au problème de l'explosion d'états et n'a pu être poursuivie faute de temps.

- *Génération automatique de tests* : un des objectifs initiaux de l'action FORMALFAME consistait à générer des cas de test grâce à l'outil TGV en les dérivant automatiquement de la description LOTOS. Cette tâche a été retardée du fait des délais liés au remplacement de M. Zendri. En effet, à la date d'embauche de N. Zuanon, l'écriture manuelle des tests par l'équipe BULL aux Clayes-sous-Bois avait atteint les objectifs fixés (plus de 30 000 cas de test exécutés). En conséquence, l'action FORMALFAME s'est réorientée vers des objectifs plus prioritaires.

- *Vérification automatique de traces d'exécution* : l'action FORMALFAME a travaillé à l'analyse automatisée des traces d'exécution produites soit par le modèle VERILOG, soit par le simulateur 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 des propriétés de correction sur les traces en utilisant la logique temporelle acceptée par l'outil de vérification EVALUATOR 3.0. 700 propriétés de séquencement ont ainsi été spécifiées et vérifiées par Solofo Ramangalahy (projetPAMPA). Ce résultat a été rendu possible grâce à plusieurs améliorations apportées aux outils CADP pour réduire le temps de vérification :
    - Les primitives de lecture du format binaire BCG dans lequel sont encodées les traces ont été rendues plus rapides ;

  - L'outil EVALUATOR 3.0 a été optimisé pour prendre en compte les graphes sans circuit (dont les traces sont un cas particulier) ;
  - Un outil prototype appelé SEQ.OPEN a été développé pour permettre de manipuler, via l'interface de programmation OPEN/CÆSAR [8], des traces encodées simplement en format ASCII.
- La deuxième méthode consiste à utiliser la description LOTOS (variante abstraite) comme un accepteur de traces. Par rapport à l'approche précédente, cette seconde méthode offre l'avantage d'éviter l'écriture de formules en logique temporelle (celles-ci étant remplacées par la description LOTOS) et d'effectuer une vérification plus fine qui prend en compte la totalité du comportement du protocole. Dans ce but, nous avons utilisé deux approches différentes afin de vérifier si une trace est compatible avec le comportement du circuit B-SPS décrit en LOTOS :
  - soit la trace à vérifier est encodée comme une expression régulière dont on détermine l'acceptation grâce à l'outil EXHIBITOR,
  - soit la trace à vérifier est encodée comme une formule de logique temporelle (comportant des expressions régulières) et évaluée grâce à l'outil EVALUATOR 3.0.

Les résultats obtenus par FORMALFAME en 2000 ont été jugés positifs. Le travail de modélisation formelle a soulevé 10 questions qui, adressées aux développeurs de l'architecture FAME, ont contribué à éliminer divers problèmes (imprécisions, ambiguïtés, incomplétudes ou incohérences) dans la description informelle du protocole de cohérence de caches. De plus, la vérification des traces a mis en évidence un problème majeur (ambiguïté dans la spécification informelle du protocole, également détectée par le simulateur C++ du circuit B-SPS) qui a confirmé l'intérêt des travaux menés dans l'action FORMALFAME.

5.3.2 Vérification de tâches robotiques

Participants : Hubert Garavel, Radu Mateescu.

En collaboration avec Alain Girault, Eric Rutten et Daniel Simon du projet BIP et Soraya Arias du service des moyens robotiques de l'INRIA Rhône-Alpes, nous avons poursuivi le travail entrepris en 1999 dans le cadre de l'action de recherche coopérative TOLERE. Ce travail vise à interconnecter la boîte à outils CADP, l'atelier logiciel pour le langage synchrone ESTEREL et l'environnement ORCCAD destiné à spécifier, programmer et vérifier les applications de contrôle-commande complexes que l'on trouve sur les systèmes embarqués tels que les robots mobiles.

Une première connexion a été effectuée au niveau des systèmes de transitions (ou graphes) obtenus à partir de descriptions graphiques ORCCAD. Ces descriptions sont traduites automatiquement par ORCCAD en programmes ESTEREL, à partir desquels la chaîne de compilation d'ESTEREL produit des graphes au format FC2. Cette connexion a permis d'apporter à ORCCAD les fonctionnalités offertes par l'environnement BCG :

- L'outil BCG_IO est d'abord utilisé pour traduire les graphes FC2 en format BCG.

- L'outil BCG_LABELS permet ensuite d'abstraire les étiquettes des transitions en cachant et/ou renommant certaines informations pour ne conserver que celles nécessaires à la vérification : ainsi, on élimine les informations concernant les signaux ESTEREL absents puisque les propriétés à vérifier ne portent que sur les signaux présents.

- L'outil BCG_MIN sert à minimiser par bisimulation les graphes obtenus après abstraction.

- Finalement, les outils BCG_DRAW et BCG_EDIT sont employés pour inspecter visuellement les graphes obtenus après minimisation.

Nous avons réalisé une deuxième connexion entre CADP et ORCCAD permettant d'utiliser EVALUATOR 3.0 (voir § 5.1.4) pour vérifier les propriétés temporelles de tâches robotiques décrites avec ORCCAD. Ce travail a permis d'étendre les fonctionnalités de vérification disponibles pour ORCCAD, qui étaient précédemment limitées à la comparaison de graphes par bisimulation.

Afin de faciliter l'utilisation d'EVALUATOR 3.0 conjointement avec ORCCAD, nous avons poursuivi nos efforts pour établir un catalogue de propriétés temporelles à vérifier. En 1999, nous avions développé une bibliothèque de propriétés réutilisables, exprimées en logique temporelle ACTL [NV90,NFGR91]. En 2000, nous avons réécrit cette bibliothèque en μ-calcul régulier d'alternance 1 et généralisé la formulation de ces propriétés. La nouvelle bibliothèque comprend à la fois des propriétés génériques devant être vérifiées par toutes les tâches robotiques que l'on peut décrire avec ORCCAD et des propriétés spécifiques exprimant des séquencements particuliers d'événements qui dépendent de l'application à vérifier.

L'interconnexion de CADP, ESTEREL et ORCCAD a été testée sur une étude de cas réelle : la vérification des missions du robot sous-marin expérimental VORTEX conçu par l'IFREMER pour tester et mettre en œuvre des lois de commande, des architectures de contrôle et des programmes de missions.

5.3.3 Autres études de cas

D'autres équipes ont également utilisé la boîte à outils CADP pour diverses études de cas. Pour ne citer que les travaux publiés en 2000, on peut mentionner :

- la conception conjointe matériel-logiciel [BW00] ;

[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.

[NFGR91] R. D. NICOLA, A. FANTECHI, S. GNESI, G. RISTORI, « An Action-based Framework for Verifying Logical and Behavioural Properties of Concurrent Systems », *in : Proceedings of 3rd Workshop on Computer Aided Verification CAV '91 (Aalborg, Denmark)*, K. G. Larsen, A. Skou (éditeurs), *Lecture Notes in Computer Science*, *575*, Springer Verlag, p. 37–47, Berlin, juillet 1991.

[BW00] F. BARAY, J.-P. WODEY, « Verification in the Codesign Process by means of LOTOS Based Model-Checking », *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. 87–108, Berlin, avril 2000.

- la vérification du système de contrôle d'une chaudière à vapeur (*steam-boiler system*) [CC00] ;
- la vérification de systèmes embarqués de contrôle-commande utilisés par la société Hollandse Signaalapparaten [DvL00b,DvL00a] ;
- le test d'un protocole de conférence [dRS+00] ;
- la détection d'interactions indésirables entre services téléphoniques [FHLS00] ;
- la spécification, la vérification et le test de circuits séquentiels [He00,HT00] ;
- la vérification de protocoles d'authentification [LG00] ;
- la spécification et la vérification d'architectures logicielles [RL00] ;
- la spécification et la vérification du protocole OM/RR pour le contrôle du trafic routier [WTK00].

[CC00] P. J. F. Carreira, M. E. F. Costa, «Automatically Verifying an Object-Oriented Specification of the Steam-Boiler System», *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. 345–360, Berlin, avril 2000.

[DvL00b] P. Dechering, I. van Langevelde, « Towards Automated Verification of Splice in muCRL», *Technical Report n° SEN-R0015*, CWI, Amsterdam, The Netherlands, mai 2000.

[DvL00a] P. Dechering, I. van Langevelde, «On the Verification of Coordination», *in : Proceedings of the 4th International Conference on Coordination Models and Languages (Limassol, Cyprus)*, A. Porto, G.-C. Roman (éditeurs), *Lecture Notes in Computer Science, 1906*, Springer Verlag, p. 335–340, septembre 2000.

[dRS+00] L. du Bousquet, S. Ramangalahy, S. Simon, C. Viho, A. Belinfante, R. G. de Vries, «Formal Test Automation: the Conference Protocol with TGV/TorX», *in : Proceedings of the 13th IFIP International Conference on Testing of Communicating Systems TestCom'2000 (Ottawa, Canada)*, H. Ural, R. L. Probert, G. v. Bochmann (éditeurs), University of Ottawa, Kluwer Academic Publishers, août 2000.

[FHLS00] Q. Fu, P. Harnois, L. Logrippo, J. Sincennes, « Feature Interaction Detection: a LOTOS-based Approach», *Computer Networks 32*, 4, 2000, p. 433–448.

[He00] J. He, *Formal Specification and Analysis of Digital Hardware Circuits in LOTOS*, PhD thesis, University of Stirling, août 2000, Available as Technical Report CSM-158, University of Stirling, Scotland.

[HT00] J. He, K. J. Turner, « Verifying and Testing Asynchronous Circuits using 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'2000 (Pisa, Italy)*, T. Bolognesi, D. Latella (éditeurs), IFIP, Kluwer Academic Publishers, p. 267–283, octobre 2000.

[LG00] G. Leduc, F. Germeau, « Verification of Security Protocols using LOTOS - Method and Application», *Computer Communications 23*, 12, juillet 2000, p. 1089–1103.

[RL00] S. Rongviriyapanish, N. Lévy, « Variations sur le style architectural *Pipe & Filter*», *in : Actes du 3ème Colloque sur les Approches Formelles dans l'Assistance au Développement de Logiciels AFADL'2000 (Grenoble, France)*, P. Berlioux, Y. Ledru (éditeurs), ADER and LSR-IMAG, p. 81–95, Grenoble, janvier 2000.

[WTK00] T. Willemse, J. Tretmans, A. Klomp, « A Case Study in Formal Methods: Specification and Validation of the OM/RR Protocol», *in : Proceedings of the 5th International Workshop on Formal*

Par ailleurs, d'autres équipes de recherche ont adopté les composants logiciels offerts par CADP (et notamment par les environnements BCG et OPEN/CÆSAR) pour développer leurs propres outils. Nous pouvons citer les réalisations suivantes :

- la connexion à BCG de l'atelier B afin de permettre la vérification des propriétés temporelles des spécifications B [BC00,Cav00] ;
- la connexion à CADP du langage à objets OBLOG [CC00] ;
- la connexion à OPEN/CÆSAR de l'environnement UMLAUT pour UML [Gue00] ;
- l'utilisation de CADP pour développer un outil de génération compositionnelle de systèmes de transitions étiquetées [Liu00] ;
- la connexion à CADP de l'environnement CTTE (*Concur Task Trees Environment*) pour la modélisation et la vérification d'interfaces graphiques [PS00] ;
- la connexion à CADP des contrôleurs logiques programmables (*Programmable Logic Controllers*) [Wil99].

6 Contrats industriels (nationaux, européens et internationaux)

6.1 Action FormalCard (Dyade)

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 le centre de recherche et développement de BULL

Methods for Industrial Critical Systems FMICS'2000 (Berlin, Germany), S. Gnesi, I. Schieferdecker, A. Rennoch (éditeurs), *GMD Report 91*, p. 331–344, Berlin, avril 2000.

[BC00] D. BERT, F. CAVE, «Construction of Finite Labelled Transition Systems From B Abstract Systems», *in : Proceedings of the 2nd International Conference on Integrated Formal Methods IFM'2000 (Schloss Dagstuhl, Germany)*, W. Grieskamp, T. Santen, B. Stoddart (éditeurs), *Lecture Notes in Computer Science, 1945*, Springer Verlag, p. 235–254, novembre 2000.

[Cav00] F. CAVE, «Passage de systèmes abstraits B à des systèmes de transitions étiquetées finis», *Mémoire de DEA*, Université Joseph Fourier et Institut National Polytechnique de Grenoble, juin 2000.

[Gue00] A. L. GUENNEC, «Méthodes formelles avec UML. Modélisation, validation et génération de tests», *in : Actes du Colloque Francophone pour l'Ingénierie des Protocoles CFIP'2000 (Toulouse, France)*, M. Diaz, J.-P. Courtiat, P. Senac (éditeurs), Hermès, Paris, octobre 2000.

[Liu00] W. LIU, «Interaction Abstraction for Compositional Finite State Systems», *in : Proceedings of the 7th SPIN Workshop (Stanford University, California)*, K. Havelund, J. Penix, W. Visser (éditeurs), *Lecture Notes in Computer Science, 1885*, Springer Verlag, p. 148–162, Berlin, septembre 2000.

[PS00] F. PATERNÒ, C. SANTORO, «Integrating Model-Checking and HCI Tools to Help Designers Verifying User Interface Properties», *in : Proceedings of the 7th International Workshop on Design, Specification and Verification of Interactive Systems DSV-IS'2000 (Limerick, Ireland)*, P. Palanque, F. Paternò (éditeurs), Eurographics, juin 2000.

[Wil99] H. WILLEMS, «Compact Timed Automata for PLC Programs», *Technical Report n° CSI-R9925*, University of Nijmegen, Nijmegen, The Netherlands, novembre 1999.

SC&T (*Smart Cards and Terminals*) situé à Louveciennes s'est concrétisée en juillet 2000 par le lancement d'une nouvelle action DYADE, appelée FORMALCARD, à laquelle participent le projet VASY et le projet PAMPA (Rennes).

L'objectif principal de l'action FORMALCARD est de réaliser un environnement de validation formelle de logiciels embarqués sur cartes à puces en vue de permettre leur certification par des méthodes formelles. 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.

L'action FORMALCARD comporte trois objectifs dont les deux premiers sont pris en charge par le projet VASY, le troisième étant confié au projet PAMPA :

- *Conception d'un langage pour la spécification formelle d'applications pour cartes à puces* : il s'agit de définir formellement (syntaxe, sémantique statique et dynamique) un langage qui sera utilisé pour décrire les applications à valider. Ce langage doit satisfaire 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 la carte à puce.
  - Il doit être exécutable pour permettre le déboguage, la simulation, et la validation par 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.

[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.

La conception du langage de description formelle reprend les travaux de normalisation du langage E-LOTOS effectués au sein de l'ISO de 1992 à 2000 : un sous-ensemble du langage LOTOS NT apte à décrire les applications embarquées (automates à commandes gardées et actions d'entrée/sortie) a été sélectionné. En 2000, nos travaux ont porté sur l'utilisation intensive du langage LOTOS NT et l'amélioration du compilateur TRAIAN et de sa documentation (voir § 5.2.2). Ce travail nous a permis d'expérimenter en vraie grandeur la technologie de construction de compilateurs basée sur la partie données de LOTOS NT qui sera utilisée dans la suite de l'action FORMALCARD afin de développer un compilateur pour la partie contrôle de LOTOS NT.

6.2 Action FormalFame (Dyade)

Participants : Hubert Garavel, Marc Herbert, Radu Mateescu, Frédéric Perret, Massimo Zendri, 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 dans le cadre du GIE BULL-INRIA DYADE. Cette collaboration, à laquelle participe également le projet PAMPA (Solofo Ramangalahy et César Viho), est coordonnée par un ingénieur BULL (M. Zendri, puis N. Zuanon) installé dans les locaux de l'Unité de Recherche INRIA Rhône-Alpes. Elle vise à démontrer que les méthodes formelles et les outils développés à l'INRIA pour la validation et le test des protocoles de télécommunications 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 pris fin en 1998 avec l'achèvement de l'action VASY de DYADE, 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 [15, 11]. Le résultat de ces expérimentations a été jugé positif : la faisabilité de l'approche proposée a été démontrée et BULL a manifesté son intérêt à poursuivre l'application de cette approche sur 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

[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. Full version available as INRIA Research Report RR-2958, `http://www.inria.fr/rrrt/rr-2958.html`.

informelle, la collaboration a été officialisée en 1999 sous la forme d'une nouvelle action DYADE, intitulée FORMALFAME, que nous menons en coopération avec l'équipe d'Anne Kaszynski au centre BULL des Clayes-sous-Bois (France).

L'action 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 est élaborée, qui constitue un modèle de référence à partir duquel on peut effectuer la simulation et la vérification (en utilisant la boîte à outils CADP) et la génération de tests (en utilisant l'outil TGV).

En 2000, les résultats obtenus par FORMALFAME portent, d'une part, sur la validation et le test du circuit B-SPS (voir § 5.3.1) et, d'autre part, sur l'amélioration des différents outils CADP.

6.3 Action SmartTools (Dyade)

Participant : Hubert Garavel.

Mots clés : compilateur, compilation, environnement de programmation, génération de code, génie logiciel.

Depuis 1999, nous participons à l'action SMARTTOOLS du GIE BULL-INRIA DYADE. Dirigée par Isabelle Attali et Didier Parigot, cette action réunit les projets COQ, LANDE, OASIS et VASY de l'INRIA, avec comme partenaires industriels BULL SC&T et MICROSOFT RESEARCH. Elle vise à construire un environnement de développement convivial et interactif pour JAVACARD (architecture de programmation standard pour cartes à puces). Nous nous intéressons aux outils génériques SMARTTOOLS, qui permettent de réaliser des environnements de programmation complets (éditeurs structurés, vérificateurs syntaxiques et sémantiques, interprètes, traducteurs, outils d'aide à la mise au point, etc.) pour différents langages informatiques.

6.4 Contrat Reutel-2000 (Alcatel)

Participants : Moëz Cherif, Hubert Garavel.

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

Depuis avril 1999, nous participons à la deuxième phase du contrat REUTEL-2000, un contrat de recherche entre ALCATEL et l'INRIA s'inscrivant dans le contexte de l'accord-cadre de collaboration ALCATEL/INRIA. Les équipes ADP, COMPOSE, EP-ATR et PAMPA de l'INRIA Rennes sont également engagées dans le contrat REUTEL-2000 dont Claude Jard est le coordonnateur à l'INRIA.

L'objectif de cette collaboration est la maîtrise du développement logiciel d'applications de télécommunications réutilisables, par la conception d'outils de manipulation formelle à l'intérieur d'une chaîne de développement définie par ALCATEL, en combinant les approches à objets et les modèles de parallélisme synchrone et asynchrone.

L'industrie des télécommunications est soumise à de fortes contraintes visant à réduire les coûts et les délais de développement tout en améliorant la qualité du logiciel. Compte-tenu de la taille et de la complexité des applications mises en œuvre, les spécifications et les programmes doivent être conçus de manière suffisamment générique pour pouvoir fonctionner dans des configurations hétérogènes, ainsi que pour assurer une flexibilité et une réactivité élevées en regard des évolutions du marché et des technologies naissantes.

Le cadre de développement considéré par ALCATEL prend en compte la norme CORBA (*Common Object Request Broker Architecture*). Les spécifications sont élaborées selon une méthodologie de conception à objets compatible avec la notation UML (*Unified Meta Language*). Le développement repose sur l'écriture de schémas de programmes dans des langages d'interface inspirés d'IDL (*Interface Definition Language*) mais étendus pour faire apparaître des informations comportementales. L'utilisation de tels langages d'interface offre une certaine indépendance vis-à-vis des différents langages utilisés pour la programmation des objets logiciels et des différentes plates-formes d'exécution. Dans cette approche, la mise en œuvre d'une application sur une plate-forme donnée se fait, lorsque cela est possible, par génération automatique de code (C, C++, JAVA, etc.).

Dans ce contexte, l'INRIA soutient l'utilisation de méthodes formelles et d'outils associés permettant la manipulation des schémas de programmes écrits dans les langages d'interface (génération de code, analyse, transformation et optimisation de code, vérification, génération de tests) afin d'assister les concepteurs d'applications et améliorer la maîtrise du développement.

La seconde phase de REUTEL comporte quatre axes de recherche : nous intervenons principalement dans le thème intitulé "Outils de validation et de génération de code pour UML". L'objectif est la réalisation d'une plate-forme de démonstration pour UML offrant des fonctionnalités de simulation, de vérification et de génération de tests. En 2000, notre contribution a porté sur les points suivants :

- Nous avons poursuivi le développement du simulateur OCIS dans le but d'en faire un outil robuste utilisable pour UML (voir § 5.1.1).

- Nous avons terminé le portage des outils CADP sous le système WINDOWS afin de permettre leur utilisation dans l'environnement de développement d'ALCATEL (voir § 5.2.3).

- En collaboration avec Solofo Ramangalahy et Séverine Simon (équipe PAMPA), nous avons finalisé l'intégration du générateur de tests TGV au sein de la boîte à outils CADP.

- Nous avons assisté Alain Le Guennec (équipe PAMPA) pour la connexion du compilateur UMLAUT pour UML à l'environnement OPEN/CÆSAR.

6.5 Contrat RNTL Parfums

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). Ce projet 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'appuiera, pour une grande part, sur les techniques développées dans l'action A3 du GIE DYADE : 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.

La contribution de l'équipe VASY au projet PARFUMS concernera la vérification de l'infrastructure logicielle et des protocoles correspondants.

7 Actions régionales, nationales et internationales

7.1 Actions nationales

En 2000, nous avons collaboré avec plusieurs projets INRIA :

APACHE et SIRAC (Rhône-Alpes) : utilisation de la plate-forme "grappe de PCs" développée par les projets APACHE et SIRAC pour l'expérimentation d'algorithmes de vérification massivement parallèles (voir § 5.1.6) ;

BIP (Rhône-Alpes) : connexion des outils de vérification BCG et EVALUATOR 3.0 à l'environnement de développement pour le langage synchrone ESTEREL et à l'atelier ORCCAD pour la conception de contrôleurs de tâches robotiques voir § 5.3.2) ;

PAMPA (Rennes) : collaboration dans le cadre des actions FORMALCARD et FORMALFAME du GIE DYADE (voir § 6.1 et 6.2) et du contrat REUTEL-2000 (voir § 6.4).

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 LIAFA (Paris) : développement du compilateur TRAIAN (Mihaela Sighireanu) ;

Laboratoire LSR-IMAG (Grenoble) : collaboration afin d'interconnecter l'atelier B et la boîte à outils CADP (Didier Bert, Francis Cave et Marie-Laure Potet) ; R Mateescu a co-encadré le stage de DEA de F. Cave ;

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 neuf groupes de travail d'ERCIM. Depuis juillet 1999, H. Garavel coordonne ce groupe qui comprend actuellement 72 chercheurs appartenant à 28 organisations (voir `http://www.inrialpes.fr/vasy/fmics`).

- H. Garavel est membre du comité technique (*ETItorial Board*) pour la plate-forme d'intégration logicielle ETI (*Electronic Tool Integration*) développée à l'Université de Dortmund et accessible en ligne par Internet (voir `http://www.eti-service.org`). Marc Herbert a contribué au test de la nouvelle version du service ETI et du logiciel client correspondant.

7.2.2 Relations bilatérales internationales

Nous entretenons des relations scientifiques avec plusieurs universités et centres de recherche internationaux. En 2000, nous avons notamment eu des contacts étroits avec :

- l'IEI-CNR Pise (équipe du professeur Stefania Gnesi),
- l'Université de New York à Stony Brook (équipe des professeurs Rance Cleaveland et Scott Smolka),
- l'Université d'Ottawa (équipes des professeurs Gregor v. Bochmann et Luigi Logrippo),
- l'Université Polytechnique de Bucarest (Gavril Godza),
- l'Université de Twente (Holger Hermanns).

7.3 Accueil de chercheurs français et étrangers

- Solofo Ramangalahy, ingénieur-expert dans le projet PAMPA, nous a rendu visite le 27 juillet 2000 dans le cadre de l'action FORMALFAME du GIE DYADE.
- Guillaume Brat, Charles Pecheur et Wilhelm Vissers, du centre de recherche AMES de la NASA, nous ont rendu visite (ainsi qu'au projet BIP et à la société de technologie POLYSPACE) le 11 septembre 2000.

- Fabrice Baray et Pierre Wodey, du laboratoire ISIMA/LIMOS nous ont rendu visite le 19 octobre 2000. Pierre Wodey a donné un séminaire sur l'utilisation de la boîte à outils CADP pour la vérification de systèmes matériels.

- Duncan Clarke, Thierry Jéron, Vlad Rusu et Elena Zinovieva, du projet PAMPA, nous ont rendu visite le 19 octobre 2000 dans le cadre de l'action FORMALCARD du GIE DYADE.

- Gavril Godza, maître-assistant à l'Université Polytechnique de Bucarest a séjourné dans le projet VASY du 10 octobre au 15 décembre 2000. Il a donné deux exposés, l'un sur les fondements de la tolérance aux pannes, l'autre sur la modélisation en LOTOS et la vérification des différents algorithmes de *checkpointing*.

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 2000, les faits marquants concernant la diffusion de CADP sont les suivants :

- Nous avons préparé et diffusé des beta-versions successives (99-e, 99-f, ... 99-u) des outils CADP.

- Le nombre de contrats de licence signés pour CADP est passé de 198 à 225 et nous avons octroyé des licences CADP pour 770 machines différentes dans le monde.

- Nous avons enrichi les pages Web consacrées à CADP (`http://www.inrialpes.fr/vasy/cadp`), notamment celles décrivant les 53 études de cas réalisées avec CADP et celles présentant les 10 outils développés au moyen des composants logiciels réutilisables fournis par CADP.

Les principaux résultats obtenus en 2000 pour la diffusion de TRAIAN sont les suivants :

- Nous avons préparé et diffusé deux nouvelles versions (2.0 et 2.1) du compilateur TRAIAN, qui a été déposé à l'Agence de Protection des Programmes.

- 74 sites différents ont téléchargé ces nouvelles versions.

- Les pages Web concernant E-LOTOS (`http://www.inrialpes.fr/vasy/elotos`) et TRAIAN (`http://www.inrialpes.fr/vasy/traian`) ont été mises à jour.

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 FMICS'2000 (*5th International ERCIM Workshop on Formal Methods for Industrial Critical Systems*, Berlin, Allemagne, 3–4 avril 2000).
- I. Smarandache a été responsable locale du symposium doctoral de la conférence ASE'2000 (*Automated Software Engineering*, Grenoble, France, 11 septembre 2000).
- H. Garavel a été membre du comité de programme de CFIP'2000 (8^{e} Colloque Francophone sur l'Ingénierie des Protocoles, Toulouse, France, 17–20 novembre 2000).
- H. Garavel est membre du comité de programme de TACAS'2001 (*6th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, Gênes, Italie, 2–6 avril 2001).
- 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 2001, 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 2000 :

- H. Garavel, R. Mateescu et N. Zuanon ont 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).
- I. Smarandache a assuré des travaux pratiques "Langages et programmation" à l'Université Joseph Fourier (filière "Réseaux Informatiques et Communication Multimedia", 18 heures annuelles).
- I. Smarandache a assuré des cours et travaux dirigés de logiciel de base et compilation à l'ENSIMAG (35 heures annuelles).
- R. Mateescu a participé au jury de DEA de Francis Cave à Grenoble le 20 juin 2000.
- 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 :

- R. Mateescu et M. Zendri ont participé à la Journée Technologique DYADE (Bull, Les Clayes sous Bois) le 21 janvier 2000. A cette occasion, ils ont effectué des démonstrations de la boîte à outils CADP.
- H. Garavel a participé à la réunion de l'action SMARTTOOLS de DYADE à l'INRIA Sophia-Antipolis, le 21 mars 2000.
- H. Garavel et R. Mateescu ont participé aux colloques et conférences FMICS'2000 et TACAS'2000 (voir § 8.2).
- H. Garavel a visité, du 22 au 29 septembre 2000, les équipes des Professeurs Gregor v. Bochmann et Luigi Logrippo à l'Université d'Ottawa où il a donné, le 25 septembre 2000, un exposé consacré à la boîte à outils CADP. Cette visite a été prise en charge par l'ambassade de France à Ottawa.
- H. Garavel a visité, du 2 au 6 octobre 2000, l'équipe des Professeurs Rance Cleaveland et Scott Smolka à l'Université de New York à Stony Brook (SUNY) où il a donné, le 2 octobre 2000, un exposé intitulé "*CADP 2000 and beyond*".
- R. Mateescu a été invité à visiter l'équipe de Stefania Gnesi (IEI-CNR, Pise) les 9 et 10 octobre 2000. Il a donné le 9 octobre un exposé intitulé "*On the fly model checking with diagnostic based on boolean equation systems*".
- R. Mateescu a assisté à la conférence FORTE/PSTV'2000 (Pise, 11–13 octobre 2000).

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. Invited lecture.

[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.

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

Articles et chapitres de livre

[11] 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)*, 2001, à paraître.

[12] R. Mateescu, M. Sighireanu, «Efficient On-the-Fly Model-Checking for Regular Alternation-Free Mu-Calculus», *Science of Computer Programming*, 2001, à paraître.

Communications à des congrès, colloques, etc.

[13] 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. Also available as INRIA Research Report RR-3899, `http://www.inria.fr/rrrt/rr-3899.html`.

[14] 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. Full version available as INRIA Research Report RR-3861, `http://www.inria.fr/rrrt/rr-3861.html`.

Rapports de recherche et publications internes

[15] H. Garavel, C. Viho, M. Zendri, «System Design of a CC-NUMA Multiprocessor Architecture using Formal Specification, Model-Checking, Co-Simulation, and Test Generation», *Research Report n° RR-4041*, INRIA, novembre 2000, `http://www.inria.fr/rrrt/rr-4041.html`.

Divers

[16] M. Sighireanu, «LOTOS NT User's Manual (Version 2.1)», INRIA projet VASY.