Rapport INRIA 1994 — Programme 2

Spécification et programmation des systèmes communicants et temps réel

PROJET SPECTRE

3 mai 1995

PROJET SPECTRE

Spécification et programmation des systèmes communicants et temps réel

Localisation : *Grenoble*

Mots-clés : concurrence (1), génération de code (1), génie logiciel (1), interprétation abstraite (1), langage synchrone (1), logique temporelle (1), parallélisme (1), programmation parallèle (1), protocole de communication (1), sémantique (1), spécification formelle (1), système hybride (1), temps réel (1), vérification de programme (1).

1 Composition de l'équipe

Responsable scientifique et permanent

Joseph Sifakis, DR, CNRS

Secrétariat

Chantal Costes, AAR, CNRS
Christine Servonnet, SAR contractuelle

Personnel INRIA

Hubert Garavel, CR

Personnel IMAG

Saddek Bensalem, MC, UJF (Université Joseph Fourier)

Ahmed Bouajjani, Mc, Ujf, détaché au Cnrs du 1-10-1994 au 30-9-95
Paul Caspi, Cr, Cnrs
Jean-Claude Fernandez, Mc, Ujf, détaché à l'Inria du 1-10-1994 au 30-9-95
Susanne Graf, Cr, Cnrs
Nicolas Halbwachs, Dr, Cnrs
Fabienne Lagnier, Mc, Ujf
Oded Maler, Cr, Cnrs
Florence Maraninchi, Mc, Ujf
Laurent Mounier, Mc, Ujf
Xavier Nicollin, Mc, Inpg (Institut National Polytechnique de Grenoble)
Pascal Raymond, Cr, Cnrs
Ghislaine Thuau, Mc, Inpg

Chercheurs invités

Amir Pnueli, Weizmann Institute, Revohot, Israel
Yassine Lakhnech, Université de Kiel, Allemagne

Chercheurs post-doctorants

Sergio Yovine, boursier
Eric Conquet, Matra Marconi Space
Marc Pouzet, Ater, Ujf

Chercheurs doctorants

Conrado Daws, boursier Mesr
Alain Girault, boursier Cifre
Peter Habermehl, boursier Mesr
Muriel Jourdan, boursier région Rhône-Alpes
Alain Kerbrat, boursier Mesr
Claire Loiseaux, Ater Inpg
Alfredo Olivero, boursier Cies
Yann-Eric Proy, boursier Cifre
Riadh Robbana, boursier du gouvernement français

Stagiaires

Sébastien Bornot, stagiaire Dea
David Lesens, stagiaire Dea
Radu Mateescu, stagiaire Dea
Renaud Ruffiot, élève-ingénieur Cnam

Mihaela Sighireanu, élève-ingénieur Institut Polytechnique de Bucarest

2 Présentation du projet

SPECTRE est un projet commun entre l'INRIA et le CNRS, localisé à Grenoble et intégré dans l'Unité Mixte VERIMAG.

L'évolution de l'informatique a conduit au développement de systèmes complexes, constitués de composants qui coopèrent. Bien que des réalisations de tels systèmes existent à l'heure actuelle, leurs méthodes de conception sont loin d'être maîtrisées et restent encore peu explorées. Ceci est dû au fait que la mise en œuvre des systèmes parallèles et distribués soulève de nombreux problèmes concernant le choix et l'utilisation des langages de description et des méthodes de programmation et de validation associées.

Répondre aux besoins des concepteurs des systèmes reviendrait à fournir des environnements qui supportent des langages de spécification et de programmation adéquats et qui intègrent des outils d'analyse et de génération de code. A l'heure actuelle, cet objectif paraît trop ambitieux si l'on veut l'atteindre dans sa généralité.

L'objectif du projet est d'aider le concepteur de certains types d'applications parallèles et temps réel, pour lesquelles le développement d'une méthode de conception rigoureuse semble à la fois nécessaire et réaliste à l'heure actuelle. Il s'agit de systèmes informatiques critiques dont le rôle est d'interagir de manière permanente avec un environnement. Des exemples typiques de tels systèmes sont les protocoles de communication et les systèmes de commande temps réel.

Dans ce domaine le projet s'attache à fournir des méthodes et des outils d'aide pour les trois tâches principales de la conception : la programmation, la spécification et la validation.

Programmation : par programmation on entend l'écriture d'algorithmes qui décrivent le fonctionnement d'un système. Nous nous intéressons à la définition, l'expérimentation et l'implantation de langages de haut niveau qui permettent une expression directe du parallélisme et des contraintes temps réel. Dans ce cadre, nous étudions la sémantique des langages et les techniques de compilation associées.

Concernant les langages de programmation, les recherches entreprises dans le cadre du projet sont menées selon deux axes :

- L'étude formelle et la compilation de langages existants, comme ESTELLE ou LOTOS, dans le but d'admettre ces langages en entrée de nos outils de validation.
- La conception et la compilation de nouveaux langages, mieux adaptés, à notre avis, à certains types d'applications. C'est le cas des langages synchrones LUSTRE et ARGOS.

Ces travaux sont menés d'une part sur le plan formel, par une étude approfondie de la sémantique des langages, considérée comme un prérequis de la compilation et de la validation, et d'autre part en vue de la réalisation d'outils (compilateurs), dont les performances sont un critère prépondérant.

Spécification : la spécification est la description du comportement d'un système vis-à-vis de l'ensemble de ses utilisateurs. De nombreux langages ont été proposés pour la spécification des systèmes parallèles. Selon leur type, on peut distinguer deux approches :

- La première approche consiste à spécifier un système en donnant une description dont la sémantique est fondée sur les systèmes de transitions (sémantique opérationnelle). Des spécifications de ce type permettent de décrire le comportement d'un système comme la composition de comportements élémentaires. Les réseaux de Petri, les graphes d'états, les algèbres de processus et des langages tels que ESTELLE ou LOTOS, sont quelques exemples de formalismes utilisés.
- La deuxième approche consiste à exprimer les spécifications par un ensemble de propriétés, une propriété représentant une classe de systèmes. Le langage de spécification utilisé dans ce cas est un langage de type déclaratif, le plus souvent le langage des formules d'une logique. Les logiques de programmes, notamment les logiques temporelles, sont des exemples de formalismes utilisés pour l'expression des propriétés.

Nous pensons que les formalismes existants sont essentiellement complémentaires et que leur combinaison peut s'avérer intéressante. En effet, les langages déclaratifs tels que les logiques sont mieux adaptés à l'expression de propriétés globales — par exemple, l'exclusion mutuelle,

l'absence de blocage ou l'absence de famine — tandis que les formalismes à base de systèmes de transitions se prêtent mieux à l'expression des relations de causalité directe et du séquencement. Des résultats concernant l'intégration d'une logique temporelle et d'une algèbre de processus dans un même langage ont déjà été obtenus dans le cadre de ce projet.

Validation : les méthodes de validation sont nécessaires depuis les spécifications initiales jusqu'à l'implémentation. Elles permettent d'assurer que les choix effectués satisfont les besoins et éventuellement de réajuster ces choix. Elles concernent des activités diverses comme la vérification formelle, la simulation et le test. Elles doivent permettre d'intégrer, d'interpréter et de corriger les erreurs.

L'étude des méthodes formelles de validation et leur application effective aux systèmes parallèles ont connu un regain d'intérêt, pour deux raisons essentielles :

- La validation par des méthodes empiriques devient problématique : les systèmes réactifs, notamment temps réel, sont souvent des systèmes critiques, dont certaines propriétés doivent être absolument satisfaites. Par ailleurs, le non-déterminisme inhérent au parallélisme asynchrone rend l'expérimentation non reproductible et affaiblit considérablement la confiance que l'on peut accorder à une validation par test.
- D'autre part, on a constaté que les propriétés cruciales d'un système parallèle peuvent être étudiées sur des abstractions qui admettent souvent des modélisations finies. Alors les problèmes relatifs à la validation se simplifient, voire deviennent décidables.

Les méthodes de validation étudiées et mises en œuvre dans le cadre du projet SPECTRE sont fondées sur l'analyse d'un modèle sémantique des programmes.

Ce sont des méthodes automatiques, mais partielles, au sens où,

- soit elles ne s'appliquent qu'à certaines classes particulières de problèmes : c'est le cas des méthodes fondées sur les modèles finis, qui s'appliquent aux programmes n'ayant qu'un nombre fini (et raisonnable) d'états ou à la vérification de propriétés analysables sur une abstraction finie du programme.
- soit elles ne fournissent que des résultats partiels : sous cette rubrique, on peut ranger la génération de séquences de test (le test

vise à prouver que le programme est faux, mais ne peut en général servir à prouver qu'il est correct), mais aussi les méthodes approchées fondées sur l'interprétation abstraite des programmes, qui peuvent fournir le résultat inverse : si la vérification d'une propriété réussit, alors la propriété est vraie, mais l'échec de la vérification est sans signification.

Trois facteurs interviennent dans la conception et la réalisation des outils actuels :

- l'utilisation de langages de programmation de haut niveau, tels que ESTELLE, LOTOS, LUSTRE ou ARGOS ;
- une recherche de performances pour aborder la vérification de systèmes réels ;
- un souci de réutilisabilité des composants logiciels développés afin de réduire l'effort de programmation.

Le projet consiste en un ensemble d'actions orientées vers des domaines d'applications particuliers — systèmes réactifs (voir section 3.1), systèmes distribués (voir section 3.2), systèmes temporisés (voir section 3.3) — et une action transversale fournissant la technologie de base pour la validation (voir section 3.4).

3 Actions de recherche

3.1 Langages synchrones et systèmes réactifs

Participants : Paul Caspi, Alain Girault, Nicolas Halbwachs, Fabienne Lagnier, Muriel Jourdan, David Lesens, Florence Maraninchi, Marc Pouzet, Pascal Raymond, Ghislaine Thuau

Les langages synchrones[1] constituent une famille de langages de haut niveau, dédiés à la programmation des systèmes réactifs. Principalement développés en France, ces langages ont donné lieu à une activité intense et à une collaboration étroite entre les équipes concernées (CMA/ENSM, INRIA-Sophia, IRISA et SPECTRE).

Deux langages synchrones sont étudiés dans le cadre du projet :

[1] cf. "Synchronous programming of reactive systems", par N. Halbwachs, Kluwer Academic Pub., 1993.

- le langage déclaratif LUSTRE adopte une approche "flot de données". Développé depuis 1984, ce langage fait l'objet d'une industrialisation par la société VERILOG.
- le langage ARGOS [24, 25], inspiré des STATECHARTS, est fondé sur une description en termes d'automates parallèles hiérarchisés.

Dans le passé, les recherches autour de ces langages ont porté sur leur sémantique, leur compilation — vers du code séquentiel, réparti, ou vers des circuits —, et la spécification et la vérification de programmes synchrones.

Certains de ces thèmes sont encore étudiés, et de nouvelles voies de recherche ont été abordées, concernant la combinaison des deux langages, la synthèse de programmes de contrôle, l'extension temporisée d'ARGOS (voir §3.3.2) et l'extension récursive de LUSTRE. Enfin, la collaboration visant à définir et développer les formats communs aux langages synchrones se poursuit, notamment, depuis cette année, dans le cadre du contrat EUREKA-SYNCHRON.

3.1.1 Compilation

Une nouvelle version, solidifiée, du compilateur LUSTRE est en cours d'écriture par Fabienne Lagnier et Pascal Raymond.

Une nouvelle technique de compilation d'ARGOS a été décrite dans la thèse de Muriel Jourdan [2] : elle consiste à ramener la phase critique du processus de compilation à la résolution d'un système d'équations booléennes. Ce système est ensuite codé par un diagramme de décision binaire (BDD).

Nous avons aussi proposé une technique symbolique — à base de BDD— pour analyser complètement les problèmes de causalité, en ARGOS comme en LUSTRE.

Enfin, les compilateurs ont été modifiés pour produire des sorties aux formats communs GC et OC5.

3.1.2 Répartition de code

La répartition de programmes synchrones est un problème difficile, mais crucial dans la mesure où les applications réparties se rencontrent fréquemment en contrôle-commande, et où leur programmation est source

d'erreur lorsqu'on les programme séparément. Un répartiteur de programmes synchrones, a été défini et réalisé dans le cadre de la thèse d'Alain Girault [1][17]. Appelé OC2REP, cet outil prend en entrée un programme OC[2] obtenu à partir d'un programme LUSTRE, ESTEREL ou ARGOS, et des directives de répartition écrites par le programmeur ; il fournit plusieurs programmes OC bien coordonnés, qui garantissent un fonctionnement équivalent au programme initial. En revanche, OC2REP n'aborde pas la question des performances du programme réparti, en ce sens qu'il se conforme aux directives de l'usager et qu'il ne cherche pas à les optimiser.

D'autre part, il importe que le programmeur comprenne le sens de ces programmes répartis, qui correspond à un relâchement de l'hypothèse synchrone : à cet effet, une sémantique de LUSTRE en ordres partiels a été étudiée, en collaboration avec l'équipe SIGNAL de l'IRISA.

3.1.3 Vérification de programmes

L'expérience montre que, dans le domaine des systèmes réactifs, les propriétés critiques d'un programme sont presque toujours des propriétés de "sûreté", exprimant que quelque chose de fâcheux ne se produit jamais au cours de l'exécution du programme. L'approche synchrone fournit une méthode originale pour spécifier ce type de propriétés, par l'écriture d'un programme "observateur" qui lit les entrées et les sorties du programme à vérifier et décide à chaque instant si la propriété est satisfaite. On doit alors vérifier que la composition parallèle du programme initial et de son observateur ne signale jamais de violation de la propriété. Nous avons développé des outils de vérification fondés sur l'examen de l'automate global, soit en extension, soit représenté symboliquement à l'aide de BDDS. Un noyau de vérification symbolique, appelé BAC (pour "Boolean Automaton Checker") a été développé, qui peut être utilisé aussi bien à partir de LUSTRE que d'ARGOS. Dans le cas de LUSTRE, qui permet l'usage de variables numériques, cette vérification n'est qu'approchée, puisque l'automate n'est qu'une abstraction finie du programme. Nous étudions donc aussi des extensions numériques [22] de notre outil de vérification, fondées sur l'utilisation du module POLKA de manipulation de systèmes de contraintes linéaires (voir §3.4.2). Notre approche a été validée, cette année, par le traitement complet d'un logiciel de contrôle de

[2]OC est le code objet commun aux langages synchrones.

centrale nucléaire. Cette étude de cas a été menée par Schneider-Electric, en collaboration avec nous, dans le cadre d'un contrat DRET [35].

3.1.4 Combinaison impératif/déclaratif

Il est reconnu que, pour programmer les systèmes réactifs, les approches impératives et déclaratives sont plus complémentaires que concurrentes. Certaines applications se programment mieux en ARGOS ou ESTEREL qu'en LUSTRE, et inversement. Une même application peut présenter des parties "impératives" et des parties "déclaratives". Il est donc important d'arriver à combiner les deux styles de programmation dans un même langage [23]. Une partie de la thèse de Muriel Jourdan [2] est ainsi consacrée à la notion de programmation multilangages pour les systèmes réactifs. L'idée est de construire la syntaxe du langage mixte en mêlant les structures d'ARGOS et de LUSTRE. La sémantique est obtenue en fusionnant les sémantiques des deux langages, exprimées dans les mêmes termes.

3.1.5 Synthèse de programmes de contrôle

La synthèse de contrôleurs basée sur la restriction d'automates a été intensivement étudiée par P. J. Ramadge et W. M. Wonham. Nous étudions l'application de ces travaux dans le cas de LUSTRE, et les possibilités d'extension fournies par les spécificités du langage et des méthodes de vérification que nous développons. Une première approche de ces problèmes a fait l'objet du DEA de David Lesens [36], qui commence cette année une thèse sur le sujet.

3.1.6 Extensions récursives de Lustre

Des réflexions récentes nous ont montré la possibilité d'introduire de la récursivité en flots de données synchrones, sans perdre les propriétés de réaction à mémoire et temps bornés ; elles doivent permettre de combler le fossé séparant au niveau conceptuel les styles impératifs et déclaratifs de programmation synchrone. Une première description, dans le style graphique de l'atelier SAGA peut être trouvée dans [18].

Plus profondément, ces réflexions nous ont mis sur la voie d'un langage parallèle d'usage général: LUCID synchrone, dont LUSTRE récursif est un sous-ensemble réactif ; LUCID synchrone est obtenu à partir d'un langage fonctionnel de suites (LUCID), en lui appliquant des contraintes

d'horloges à la LUSTRE-SIGNAL: le résultat est un sous-ensemble de LUCID se compilant efficacement, aussi bien en séquentiel qu'en réparti.

3.1.7 L'environnement SYNCHRONE

Nous participons activement au projet SYNCHRONE, issu de l'opération Langages Synchrones de C[3] et de l'Action Coopérative de Recherche C2A. Le but de ce projet est de définir et de normaliser un ensemble de formats d'échange, communs aux langages synchrones, afin de constituer un socle commun à partir duquel des outils compatibles sont développés. Trois formats ont été définis[3]: deux formats orientés vers les langages source, l'un de style impératif (IC, pour *imperative code*), l'autre de style déclaratif (GC, pour *graph code*), et un format orienté vers le code cible séquentiel, OC (pour *object code*). Ce projet, soutenu jusqu'à présent par le Ministère de l'Industrie et la DRET, est devenu cette année le projet EUREKA SYNCHRON.

3.2 Protocoles et systèmes distribués

La spécification des protocoles et des systèmes répartis nécessite l'utilisation de langages de haut niveau permettant une description aisée de la dispersion du contrôle et des problèmes liés à la synchronisation et à la communication entre entités séparées. Comparés aux langages synchrones, ces langages doivent permettre la description de comportements asynchrones et ils doivent fournir les moyens pour une description aisée de données complexes et variées. Toutefois, avec l'apparition de protocoles à contraintes temporelles fortes — protocoles utilisés dans les réseaux hauts-débits — le besoin apparaît pour des langages qui combinent les caractéristiques des langages synchrones et asynchrones.

Depuis 1986, nous nous intéressons au langage LOTOS, une norme proposée par l'ISO et l'ITU pour la spécification des protocoles et des systèmes répartis. Il s'agit d'un langage de spécification algébrique de systèmes de processus communicants qui intègre les calculs de processus CCS et CSP pour la description du contrôle et les types abstraits algébriques pour la description des données.

[3]Voir "*Projet SYNCHRONE, Les formats communs des langages synchrones*", Rapport Technique INRIA n° 157, juin 1993.

Nous avons développé deux compilateurs efficaces, baptisés CÆSAR et CÆSAR.ADT, pour le langage LOTOS qui, avec l'outil de vérification ALDÉBARAN, constituent le socle de notre boîte à outils pour l'ingénierie des protocoles.

3.2.1 Compilation des algèbres de processus

Participant : Hubert Garavel

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.

En 1994, outre des corrections d'anomalies, les travaux sur CÆSAR ont été marqués par la confrontation avec une application de grande taille : la description formelle du service de transport étendu OSI 95 (développé par l'équipe d'A. Danthine à l'Université de Liège). Cette description fait un usage intensif de la programmation par contraintes et comporte plusieurs centaines de processus en parallèle.

Cette caractéristique nous a conduit à une approche compositionnelle : les réseaux de Petri des processus parallèles sont engendrés séparément, puis réduits, avant d'être recomposés pour construire le réseau global du service. Cette approche est en cours d'implémentation dans le compilateur.

3.2.2 Compilation des types abstraits algébriques

Participants : Hubert Garavel, Mihaela Sighireanu

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...) qui sont identifiées automatiquement et implémentées de manière optimale.

En 1994, nous avons modifié la partie avant (*front end*) commune aux deux compilateurs CÆSAR et CÆSAR.ADT pour la rendre conforme à la

norme ISO-8807. En effet, cette partie avait été développée en 1987 sur la base d'un avant-projet de norme et ne permettait pas de combiner renommage et instanciations de types paramétrés [42].

Nous avons ensuite étudié le problème de la génération de code C pour les types paramétrés. Ceci n'étant pas toujours possible, nous avons énoncé un ensemble de conditions nécessaires — plus strictes que les contraintes de sémantique statique figurant dans la norme LOTOS — qui caractérisent les cas où la génération de code a un sens.

Enfin, nous nous sommes intéressés à l'amélioration de la qualité du code C engendré, notamment pour les types. Nous avons étudié des méthodes pour détecter de nouvelles classes de types LOTOS susceptibles de recevoir une implémentation optimale, ainsi que diverses techniques pour optimiser les structures de données C produites par CÆSAR.ADT.

3.2.3 Réalisation d'une interface utilisateur

Participants : Hubert Garavel, Alain Kerbrat

La boîte à outils CÆSAR/ALDÉBARAN n'est pas forcément d'un abord facile, en raison du nombre et de la diversité des outils qui la composent. Il nous est apparu indispensable d'en faciliter l'accès en réunissant les outils au sein d'une interface utilisateur homogène.

Sur la base d'une interface déjà existante pour ALDÉBARAN, et en collaboration étroite avec nos partenaires belges et canadiens du projet EUCALYPTUS, nous avons développé une interface utilisateur qui intègre nos outils et ceux de Liège, Montréal et Ottawa.

Cette interface fonctionne sous le système de fenêtrage X11 et a été réalisée à l'aide du générateur d'interfaces XTPANEL de l'Université Stanford qui, à l'usage, s'est révélé être assez mal adapté au développement d'une interface de cette taille (environ 5 000 lignes de code).

Plus que les détails de réalisation, ce sont les concepts mis en œuvre dans cette interface qui sont à retenir : l'approche "à objets" que nous avons proposée semble simple et naturelle aux utilisateurs ; elle contribue à donner une vision unifiée de la boîte à outils.

3.2.4 Contribution à l'action de normalisation E-LOTOS

Participant : Hubert Garavel

Une révision de la norme LOTOS est actuellement en cours à l'ISO : elle devrait conduire à un nouveau langage, baptisé E-LOTOS (*Extended-*LOTOS) intégrant de nouveaux concepts, notamment les aspects de temps quantifié.

Nous avons participé en tant que délégué AFNOR aux réunions d'édition ISO qui se sont tenues à Madrid (janvier 1994) et à Southampton (juillet 1994). Comme coordinateur du groupe de travail 1 du projet *Cost-248*, nous avons organisé deux séminaires, l'un à Brighton (juillet 1994) consacré au problème du temps et l'autre à Evry (septembre 1994) consacré aux types de données.

Nous avons fourni deux contributions à la normalisation E-LOTOS : la première [31] propose un système de typage des interactions lors des rendez-vous ; la seconde [32] suggère des améliorations à la partie processus de LOTOS. Ces deux contributions ont été retenues et incluses dans le document de travail de l'ISO.

3.2.5 Etudes de cas

Participants : Laurent Mounier, Alain Kerbrat

Dans le cadre du projet VTT, nous avons étudié l'exemple d'un routeur de messages dans un réseau de type téléphonique. Cet exemple a donc été décrit formellement en LOTOS puis vérifié à l'aide de la boîte à outils CÆSAR/ALDÉBARAN [39].

Dans le cadre du projet EUCALYPTUS, nous avons étudié un central téléphonique POTS (*Plain Ordinary Telephony Service*) décrit en LOTOS par Patrik Ernberg du SICS (Suède). A l'aide de la boîte à outils CÆSAR/ALDÉBARAN nous avons vérifié un ensemble de propriétés décrivant le fonctionnement attendu du central.

D'autres équipes ont également utilisé la boîte à outils pour diverses études de cas : interactions de services téléphoniques (H. Korver, CWI), commutateur ATM (A. Février et E. Najm, ENST), systèmes de téléphonie mobile GSM (L. Logrippo, Université d'Ottawa), etc.

3.3 Systèmes temporisés et hybrides

Les travaux sur les systèmes ayant une base de temps continu ont connu un essor spectaculaire pendant les trois dernières années, et ceci essentiellement pour deux raisons : d'une part, la découverte de modèles sémantiques fondés sur les systèmes de transitions a permis d'étendre aux systèmes temporisés les méthodes classiques d'analyse des programmes ; d'autre part, de récents résultats ont montré que l'analyse de certaines classes de ces systèmes n'est pas plus difficile que l'analyse des classes correspondantes ayant une base de temps discret.

Du point de vue pratique, les résultats sur les systèmes temporisés permettent d'élargir le champ d'application de nos méthodes aux systèmes hybrides, systèmes contenant des composantes discrètes et continues. Ils permettent également une analyse plus fine des systèmes temps réel qui tient compte de leur environnement souvent modélisé avec des variables continues.

3.3.1 Modèles des systèmes temporisés et hybrides

Participants : Xavier Nicollin, Alfredo Olivero, Joseph Sifakis, Sergio Yovine

Nos travaux sur les systèmes ayant une base de temps continu nous ont conduit à la définition d'un modèle général pour systèmes hybrides [6]. Ce modèle peut être considéré comme un automate étendu avec un ensemble de variables. L'état d'un système représenté dans ce modèle est déterminé par un état de l'automate (état de contrôle) et une valuation des variables. L'état peut changer soit de manière discrète en exécutant des transitions de durée zéro soit de façon continue en laissant progresser le temps. Les changements d'état sont spécifiés dans le modèle en étiquetant le graphe de l'automate :

- par des commandes gardées simples, des paires (garde, affectation), associées aux transitions de l'automate. Une transition est franchissable si la garde correspondante est satisfaite. Son exécution entraîne la modification des variables selon l'affectation suivie du changement de l'état de contrôle correspondant.
- par un système de contraintes portant sur les dérivées des variables et un invariant associés aux états de l'automate. Dans un état de l'automate, les valeurs des variables peuvent changer selon les lois

spécifiées par le système des contraintes si l'invariant reste vrai au cours de cette évolution.

Nous avons défini une sémantique de ce modèle en termes de systèmes de transitions. Différentes sous-classes du modèle général sont étudiées.

- Les *automates hybrides linéaires* sont des systèmes hybrides où toutes les conditions portant sur les variables ou leurs dérivées sont des contraintes linéaires et les termes des affectations sont des expressions linéaires.
- Les *automates temporisés* sont des automates hybrides linéaires dont les variables sont des horloges (leurs dérivées sont égales à 1 dans tout état de contrôle), les invariants et les gardes sont des combinaisons booléennes de conditions de comparaison d'une horloge à une constante et les affectations peuvent être seulement des remises à zéro.
- Des classes se situant entre les deux classes mentionnées comme les *automates à intégrateurs* qui diffèrent des automates temporisés seulement par le fait que les dérivées des horloges peuvent prendre aussi la valeur 0. Le blocage de la progression d'une horloge permet de mesurer et de comparer des durées.

3.3.2 Argos temporisé et hybride

Participants : Nicolas Halbwachs, Muriel Jourdan, Florence Maraninchi, Yann-Eric Proy

Nous étudions des extensions du langage ARGOS pour permettre la description structurée et hiérarchique de graphes temporisés et d'automates hybrides.

L'ajout d'une construction temporelle dans ARGOS permet d'utiliser le langage comme un jeu d'opérateurs pour décrire de manière structurée des graphes temporisés. La sémantique du langage étendu, la production du format d'entrée de l'outil KRONOS (voir § 3.3.3), ainsi que l'application à la vérification de propriétés temps réel quantitatives, sont décrites dans la thèse de Muriel Jourdan [2].

Une extension plus générale permet la description d'*automates hybrides linéaires*. Elle consiste simplement à annoter un programme ARGOS avec des indications concernant les variables continues :

- A un état d'un automate, on peut associer un système de contraintes linéaires portant sur les dérivées des variables, et un invariant linéaire conditionnant la station dans l'état considéré.
- A une transition d'un automate peuvent être associés une garde linéaire et un ensemble d'actions d'affectations linéaires aux variables.

Le compilateur ARGOS est utilisé pour composer les automates élémentaires, en reportant sur l'automate produit les annotations hybrides. L'automate hybride résultant peut alors être traduit vers les formats d'entrée de KRONOS (voir §3.3.3) ou POLKA (voir §3.4.2).

3.3.3 Spécification logique des systèmes hybrides

Participants : Ahmed Bouajjani, Conrado Daws, Oded Maler, Riadh Robbana

Nous exprimons les propriétés des systèmes hybrides dans des logiques temporelles dites "temps réel" qui permettent d'exprimer des contraintes où intervient le temps quantitatif défini sur les réels. Un exemple de telles logiques est TCTL une extension de la logique CTL dont les modalités "possible" et "inévitable" sont paramétrées par des contraintes sur le temps où l'argument de ces opérateurs devient vrai. Ainsi, on peut exprimer dans cette logique des propriétés de la forme, *il est inévitable que* ou *il est possible que* dans un certain temps un événement se produise.

Nous avons étendu la logique TCTL en utilisant le concept de *durée* : étant donné une séquence d'exécution, la durée d'une propriété est le temps accumulé durant lequel cette proposition a été vraie (éventuellement de manière intermittente). En particulier, le temps écoulé sur une séquence d'exécution est simplement la durée de la proposition *vrai*. Ainsi, nous avons proposé une logique appelée DTL (Duration Temporal Logic) qui est une extension de TCTL où l'on peut exprimer des contraintes très générales sur les durées.

Plusieurs résultats concernant le problème de la vérification des systèmes temporisés et des systèmes hybrides pour des propriétés exprimables en TCTL et DTL ont été obtenus.

Nous avons déjà étudié une méthode de vérification symbolique de propriétés temps réel exprimées en TCTL pour des automates temporisés, implémentée dans le cadre de l'outil KRONOS. Par ailleurs, nous

avons montré que le problème de l'atteignabilité pour toute la classe des automates à intégrateurs est indécidable.

Les résultats de cette année se résument ainsi :

- Nous avons montré la décidabilité du problème de la vérification pour des fragments non triviaux de DTL et une classe d'automates à intégrateurs. Nous considérons notamment des propriétés qui comprennent des imbrications de propriétés d'invariance et d'inévitabilité. Dans [15], nous montrons en particulier que le problème de la vérification pour toute la logique DTL est indécidable.
- Dans [14] et [16], nous exhibons d'autres classes d'automates à intégrateurs pour laquelle la vérification des propriétés d'invariance est décidable.
- Dans [12], le problème de la vérification des propriétés d'invariance est considéré pour des systèmes hybrides définis par la donnée d'un ensemble de variables réelles qui évoluent de manière continue. La dynamique de ces variables est définie par une partition de l'espace des états (vecteurs des valeurs des variables du système) en un nombre fini de régions dans chacune desquelles les dérivées des variables sont constantes. Nous avons prouvé que le problème d'atteignabilité pour de tels systèmes est décidable en dimension 2 (avec au plus deux variables), et dans [12], il est montré que ce problème devient indécidable à partir de la dimension 3.

3.4 Méthodes de vérification

La vérification d'applications réparties ou de systèmes réactifs consiste à comparer une forme intermédiaire (modèle) d'un programme avec la propriété que l'on veut vérifier. En fonction de la nature de cette forme intermédiaire, on distingue plusieurs classes de méthodes, notamment les méthodes "énumératives", qui traitent des modèles finis, et les méthodes "symboliques", qui traitent des modèles finis ou infinis.

3.4.1 Vérification énumérative

Les méthodes énumératives sont mises en œuvre dans la boîte à outils CÆSAR/ALDÉBARAN. Ces méthodes rencontrent un problème crucial,

celui de l'"explosion d'états" qui survient lorsque le nombre d'états du système à vérifier dépasse les capacités de la machine. En fonction de ce problème on peut vérifier :

- soit à partir d'une représentation explicite du graphe en gardant en mémoire l'ensemble de ses états et transitions,
- soit à partir d'une représentation implicite consistant à garder en mémoire seulement un fragment "utile" du graphe.

Nous avons déjà développé l'outil de vérification ALDÉBARAN pour la comparaison et la réduction de graphes modulo une relation de d'équivalence appropriée. Nous avons également développé l'environnement OPEN/CÆSAR permettant l'application de différentes méthodes d'analyse aux modèles engendrés, dont la vérification à la volée et la simulation pas à pas.

En 1994, le développement des méthodes à la volée et compositionnelles s'est poursuivi ; notre travail a également porté sur la représentation explicite des graphes à l'aide du format BCG, l'évaluation des propriétés logiques et le traitement d'études de cas significatives.

Vérification à la volée *Participants* : Hubert Garavel, Jean-Claude Fernandez, Mats Kindahl, Laurent Mounier, Renaud Ruffiot

Les interfaces d'OPEN/CÆSAR étaient trop liées au langage LOTOS et au compilateur CÆSAR. Elles ont été modifiées pour permettre à d'autres langages/outils de se connecter à l'environnement OPEN/CÆSAR.

Nous avons développé un outil permettant l'évaluation à la volée (dans l'environnement OPEN/CÆSAR) d'un sous-ensemble des formules de CTL (formules avec un seul niveau d'imbrication des modalités temporelles).

Durant son séjour, Mats Kindahl (de l'Université d'Uppsala) a ajouté à OPEN/CÆSAR une nouvelle bibliothèque qui permet de cacher ou de renommer "dynamiquement" les actions d'un programme au moment de sa vérification (sans avoir à modifier le programme ni à le recompiler).

Représentation explicite des graphes *Participants* : Hubert Garavel, Radu Mateescu, Renaud Ruffiot

Nous avons défini un format baptisé BCG (*Binary Coded Graphs*) qui met en œuvre des techniques de compression de graphes d'états étiquetés. Ce format est indépendant du langage source et des outils de

vérification. En outre, il contient suffisamment d'information pour que les outils qui l'exploitent puissent fournir à l'utilisateur des diagnostics précis en termes du programme source.

En 1994, nous avons poursuivi les travaux engagés autour du format BCG. Une définition rigoureuse du format BCG a été élaborée [30]. Un environnement logiciel a été réalisé [40], qui se compose d'outils et de bibliothèques (environ 30 000 lignes de code). Cet environnement a été intégré dans la boîte à outils CÆSAR/ALDÉBARAN. Les gains en espace mémoire obtenu grâce à l'utilisation de BCG sont importants. Par exemple, pour représenter un graphe ayant 1 million d'états et 5 millions de transitions, il faut 70 méga-octets en utilisant le format FC2 et 180 méga-octets en utilisant le format de l'outil ALDÉBARAN, tandis que 13 méga-octets suffisent pour une représentation en BCG.

Parmi les outils actuellement disponibles, on peut en mentionner trois :

- L'outil `bcg_io` effectue des conversions entre le format BCG et une douzaine d'autres formats utilisés dans des outils de vérification (dont le format FC2 du projet MEIJE et le format TRANS du projet PAMPA).
- L'outil `bcg_open` permet d'appliquer à des graphes BCG les outils de l'environnement OPEN/CÆSAR pour la vérification à la volée.
- L'outil `bcg_draw` permet d'afficher en PostScript un graphe BCG. Cet outil a été développé en collaboration avec Claude Jard et Thierry Jéron de l'IRISA : il utilise une représentation 2D, complétée par des heuristiques de placement des états et des étiquettes qui donnent de bons résultats sur des graphes de quelques dizaines d'états.

Vérification de propriétés logiques *Participants* : Hubert Garavel, Radu Mateescu

Certaines propriétés à vérifier s'expriment plus aisément à l'aide de formules de logique temporelle qu'à l'aide de relations de bisimulation. Or, à l'heure actuelle, la boîte à outils CÆSAR/ALDÉBARAN ne comporte aucun évaluateur de formules de logique temporelle. La connexion avec des évaluateurs tels que XESAR, MEC, CONCURRENCY WORKBENCH est possible, mais pas entièrement satisfaisante, car ces outils ne peuvent exploiter toutes les informations contenues dans le format BCG.

C'est pourquoi nous avons défini un méta-langage, baptisé XTL (*eXtended Temporal Logic*) pour l'expression d'algorithmes de vérification par évaluation de formules de logiques temporelles. D'inspiration fonctionnelle, ce méta-langage permet la définition de fonctions récursives qui accèdent aux informations contenues dans les états et les étiquettes des transitions, ainsi qu'aux fonctions "successeurs" et "prédécesseurs " de la relation de transition. Grâce à ces fonctions, il est possible d'exprimer les prédicats de base, ainsi que des calculs des points fixes sur des ensembles d'états et de transitions.

XTL permet l'expression des algorithmes d'évaluation pour les logiques temporelles usuelles (CTL, HML, LTAC). De plus, il semble possible d'y intégrer les calculs relatifs à la génération de diagnostics.

A terme, l'objectif de ce travail est de développer un compilateur capable d'évaluer, sur un système de transitions représenté avec le format BCG, des formules temporelles exprimées en XTL.

La syntaxe, la sémantique statique et la génération de code — et plus généralement l'architecture complète du compilateur XTL — ont été formellement spécifiées [38]. Le travail d'implémentation a commencé : le pré-processeur du langage XTL a été réalisé, ainsi que les phases d'analyse lexicale et syntaxique.

3.4.2 Vérification symbolique et interprétation abstraite

Participants : Saddek Bensalem, Conrado Daws, Jean-Claude Fernandez, Susanne Graf, Nicolas Halbwachs, Alain Kerbrat, Claire Loiseaux, Xavier Nicollin, Alfredo Olivero, Yann-Eric Proy, Joseph Sifakis, Sergio Yovine

Une direction importante de recherche en vérification symbolique concerne l'application de techniques d'interprétation abstraite. L'application de telles techniques d'analyse approchée se justifie dans le cas de systèmes d'états infinis, mais aussi lorsque la taille des problèmes rend les méthodes exactes inopérantes. Nous participons sur ce thème à un projet soutenu par le MESR, en collaboration avec Bull et le LIENS (équipe de P. Cousot).

Génération de modèle minimal : L'explosion combinatoire du nombre des états d'un modèle est souvent due à la prolifération d'états équivalents. S'il existe des méthodes efficaces pour éliminer ces états

équivalents — en particulier les méthodes appliquées par ALDÉBARAN — celles-ci ne sont applicables qu'après la génération explicite du graphe d'états et n'évitent donc pas l'explosion au cours de celle-ci. Nous avons proposé un algorithme de minimisation à la volée, au cours de la génération du graphe. Cet algorithme nécessite la représentation et la manipulation symbolique d'ensembles d'états, puisqu'il manipule, non pas explicitement des états, mais des classes d'états équivalents. Il a été implémenté dans l'outil MAGEL (thèse d'Alain Kerbrat [3]) qui permet d'analyser des réseaux de Petri interprétés (avec valeurs booléennes ou entières) produits par CÆSAR ou bien des automates communicants au format ALDÉBARAN.

Vérification de systèmes temporisés : KRONOS est un outil de vérification permettant de comparer des automates temporisés avec des propriétés temps réel exprimées dans la logique TCTL (thèses de Xavier Nicollin et de Sergio Yovine).

La méthode de vérification consiste à calculer, à partir d'un automate temporisé et d'une formule à vérifier, l'ensemble caractéristique de la formule sous forme d'union de contraintes linéaires sur les horloges de l'automate. Ces contraintes ont la particularité qu'elles s'expriment comme conjonctions de conditions de comparaison d'une horloge à une constante ou de la différence de deux horloges à une constante. Les principaux résultats obtenus au cours de cette année sont :

- l'amélioration des performances de l'outil et son utilisation pour la vérification de contraintes temporelles fortes de systèmes temporisés non triviaux (thèse d'Alfredo Olivero [5]). Certains systèmes ont été obtenus à partir de spécifications structurées en Argos temporisé.
- L'étude de méthodes de traduction de deux sous-classes d'automates hybrides linéaires, vers les automates temporisés. La première s'applique à des automates hybrides linéaires à dérivées constantes non nulles pour obtenir des automates temporisés bisimilaires et la seconde donne des automates temporisés qui sont des abstractions des automates initiaux. Ces résultats étendent le domaine d'application de KRONOS à deux sous-classes importantes des automates hybrides linéaires [27].

Analyse d'invariants linéaires : Pour traiter certains programmes numériques simples, nous implémentons une technique particulière de l'interprétation abstraite concernant la synthèse d'invariants linéaires. Cette technique consiste à calculer automatiquement, en chaque état du programme, une approximation supérieure de l'ensemble des valeurs possibles des variables numériques, sous forme d'un système de contraintes linéaires. Ces résultats sont utilisés pour démontrer des propriétés de sûreté (inaccessibilité de certains états), exclusion mutuelle entre événements ou même pour choisir les valeurs de certains paramètres numériques. Cette technique a été implantée dans deux outils :

- POLKA qui prend en entrée un automate interprété et a été appliqué à l'analyse des délais dans les programmes synchrones, et à la vérification de systèmes temporisés et hybrides [21]
- MAGEL qui prend en entrée un réseau de Petri interprété généré par le compilateur CÆSAR pour l'analyse de programmes LOTOS [3] (thèse d'Alain Kerbrat) [19].

Vérification par calcul d'abstractions : Nous avons étudié des relations d'abstraction particulières qui préservent des classes de propriétés décrites par des logiques temporelles. Ces relations correspondent à la relation de simulation de Milner, paramétrée par une relation entre domaines de données. Ces relations d'abstraction préservent la validité des propriétés significatives exprimables par de formules d'un fragment du μ-calcul arborescent [9].

Nous avons implémenté un outil qui vérifie de manière symbolique des propriétés sur des systèmes de processus, où chaque processus est décrit par un programme à commandes gardées sur des variables booléennes. Pour un système donné, une relation entre le domaine de ses variables et un domaine abstrait, l'outil calcule automatiquement un programme abstrait optimal sur lequel ensuite les propriétés sont vérifiées. L'outil permet de définir les abstractions de manière compositionnelle et de simplifier ainsi le calcul du programme abstrait (thèse de Claire Loiseaux [4]).

Cette approche s'est également avérée très utile pour la vérification de systèmes infinis. Dans ce cas, le programme abstrait ne peut pas en général être calculé automatiquement. Une méthode de calcul consiste à remplacer toute opération sur le domaine concret par une opération

appropriée sur le domaine abstrait choisi ; la preuve que cette opération représente effectivement une abstraction est généralement simple, et peut être faite à l'aide d'un démonstrateur de logique de prédicats de premier ordre. Nous avons ainsi vérifié de manière élégante à l'aide de CÆSAR/ALDÉBARAN un système de mémoire cache — utilisant des types de données complexes, comme des buffers, des mémoires, des ensembles de tuples... — étudié dans le cadre du projet React [20]. Le langage LOTOS s'est avéré bien adapté à notre méthode puisque le passage du programme concret vers le programme abstrait consiste simplement à changer les domaines des variables du programme et les définitions de types associés.

Pour pouvoir appliquer facilement cette approche, nous envisageons la création d'une bibliothèque de telles opérations abstraites "standard" qui pourront être utilisées pour le calcul systématique de programmes abstraits.

3.4.3 Classes décidables de systèmes infinis

Participants : Ahmed Bouajjani, Peter Habermehl, Riadh Robbana

La motivation de ce travail est la vérification des systèmes infinis en suivant une approche "incrémentale". En effet, des modèles des systèmes infinis peuvent être obtenus en étendant les modèles réguliers :

- soit par adjonction de types de données infinis. On peut étendre ainsi les automates pour obtenir les systèmes hybrides (automates à variables réelles), les automates temporisés (automates à horloges), les automates à compteurs, à files, etc.
- soit par adjonction d'opérateurs ou de constructions engendrant des comportements non réguliers. On peut étendre ainsi les algèbres de processus réguliers en autorisant la récursion à travers la composition séquentielle (pour définir des processus hors contexte) ou parallèle.

L'objectif du travail est l'identification de classes de systèmes et de propriétés pour lesquelles des méthodes de décision existent.

Spécification et analyse de systèmes infinis : Nous nous intéressons à la vérification de propriétés de systèmes infinis décrites dans des formalismes de spécification basés sur des logiques temporelles.

Les résultats positifs qui existent sur ce sujet concernent l'extension aux processus hors-contexte et aux automates à pile des techniques de vérification pour des propriétés exprimables dans un fragment du μ-calcul propositionnel arborescent. Ces propriétés sont des propriétés *régulières*, c'est-à-dire que l'on peut caractériser par des automates *finis*. Cependant, les propriétés qui portent sur les valeurs de variables non bornées ne sont pas régulières de manière générale. En particulier, sont non-régulières des propriétés portant sur les nombres d'occurrences de certains événements dans une séquence d'exécution. Par exemple, des propriétés d'un protocole telles que *entre le début et la fin d'une session, il y a autant de demandes que de permissions* ou *durant une session, le nombre des demandes est toujours supérieur à celui des permissions.*

Pour pouvoir exprimer de telles propriétés, nous avons défini dans [15] une nouvelle logique appelée PCTL (Presburger CTL), qui est une extension de la logique temporelle CTL où des contraintes sur des nombres d'occurrences d'événements sont décrites à l'aide de formules de l'arithmétique de Presburger.

Ainsi, le problème que nous considérons est celui de la vérification de propriétés *non régulières* pour des systèmes *non réguliers*.

Nous montrons dans [15] que le problème de la vérification de PCTL est indécidable même pour des systèmes réguliers, mais cependant, pour un large fragment de cette logique, que nous appelons PCTL$^+$, ce problème est décidable pour les processus hors-contexte. Dans [13], nous étendons ce résultat de décidabilité à la classe des processus avec récursion à travers la composition séquentielle et parallèle.

Finalement, nous nous sommes intéressé à des systèmes infinis avec des variables continues et des structures de données discrètes. Dans [14] et [29], nous montrons la décidabilité du problème de la vérification des propriétés d'invariance pour des classes de systèmes hybrides linéaires avec une pile et certains types de compteurs.

4 Actions industrielles

Coopération avec VERILOG **:** le projet SPECTRE fait partie de l'Unité Mixte VERIMAG, laboratoire associé au CNRS, l'Institut National Polytechnique de Grenoble, l'Université Joseph Fourier et la société VERILOG. La coopération avec VERILOG dans le cadre de cette unité

vise à transférer les résultats du projet sur les méthodes et outils d'aide à la conception.

Cette année le travail commun s'est focalisé plus particulièrement sur les environnements de programmation développés autour du langage LUSTRE.

Nous participons conjointement au projet EUREKA SYNCHRON dont VERILOG est le coordinateur.

Coopération avec Schneider-Electric : Cette année a vu se terminer le contrat Cifre sur la répartition de code synchrone (thèse d'A. Girault). Un nouveau contrat Cifre a été initialisé, concernant la modélisation et l'analyse de réseaux électriques au moyen des modèles de systèmes hybrides. Enfin, dans le cadre d'un contrat DRET, une expérimentation en vraie grandeur des outils de vérification de programmes LUSTRE a été menée au département SES de Schneider-Electric.

Coopération avec Bull : nos travaux sur l'utilisation des techniques symboliques pour l'analyse statique et la vérification des systèmes bénéficient d'un contrat du MESR dans le cadre de l'action Informatique 92. Nos partenaires dans ce contrat sont Bull (équipe de F. Anceau) et le LIENS (équipe de P. Cousot). L'objectif de cette action et de mettre en commun et de permettre la coopération entre un certain nombre de modules et techniques de base pour l'analyse symbolique.

Coopération avec Matra Marconi Space : dans le cadre d'un contrat de collaboration de recherche Post-Doc Entreprise passé entre le CNRS et cette société, nous accueillons Eric Conquet. Ce dernier a effectué sa thèse en coopération avec Matra Marconi Space sur l'intégration des méthodes d'évaluation dans le développement d'un système informatique. L'objet de la coopération est "l'étude de méthodes et outils pour l'analyse des propriétés d'un système informatique en cours de développement".

Coopération avec le CNET : nous démarrons, dans le cadre d'une convention avec le CNET, une coopération sur la spécification et l'analyse des systèmes temporisés. L'objectif est l'application de nos résultats pour l'expression et l'analyse d'exigences de "qualité de service" des systèmes multimedia.

5 Actions nationales et internationales

Projets et groupes de travail :

GDR-PRS : Nous participons à son comité scientifique et nous animons le thème "Description formelle et vérification".

Groupe de travail VTT : Nous avons participé à une étude de cas réalisée dans le cadre de l'opération VTT (Vérification, Types et Temps). Cette opération, financée par le MESR, regroupe quatre laboratoires français : le LABRI, le LRI, le LIENS, et VERIMAG, Grenoble. Ses objectifs sont d'évaluer et de comparer différentes méthodes de vérification autour d'une même étude de cas.

Groupe de travail C2A : nous participons à l'action coopérative de recherche C2A (CAO-Automatique).

Projet franco-israélien : nous participons à un projet commun avec l'Institut Weizmann sur les systèmes hybrides dans le cadre de la collaboration Franco-Israélienne (MESR).

Projets ESPRIT-BRA **React et Concur-2 :** nous participons actuellement aux projets ESPRIT-BRA React et et Concur-2 qui regroupent des équipes européennes travaillant sur la spécification et la vérification des systèmes en utilisant respectivement des logiques et des algèbres de processus.

Projet ESPRIT **Euro-Canadien Eucalyptus :** depuis octobre 1992, nous participons à Eucalyptus, projet de coopération entre la CEE et le Canada, ayant pour objectif le développement concerté d'outils pour la vérification des spécifications LOTOS.

Projet sc Eureka Synchron : nous participons à ce projet pour le développement d'un environnement multilangages pour la programmation synchrone. Partenaires : TNI, ILOG, VERILOG, LOGIKKONSULT, INRIA, Schneider Electric, SAAB.

Action européenne COST 247 : nous participons à ce projet sur la vérification et la validation de descriptions formelles ; nous coordonnons le groupe de travail consacré à LOTOS et E-LOTOS.

Conférences et journaux

Organisation de Colloque : nous avons organisé le colloque franco-canadien "Communicating Informatics and Distributed Systems,

new Technologies, new Requirements" dans le cadre des 7èmes Entretiens du Centre Jacques Cartier à Grenoble du 30 novembre au 2 décembre.

Comités de programme : nous participons au *steering committee* de la conférence "Computer Aided Verification" (CAV) et aux comités de programme des conférences CAV, AMAST, FTRTFTS, CONCUR, PSTV ET RCS.

J. Sifakis participe au comité de lecture du journal "*Formal Methods in System Design*" et est éditeur invité d'un numéro spécial de TCS sur les systèmes hybrides.

Autres contacts universitaires : INRIA-Sophia Antipolis, CMA-Ecole des Mines, IRISA, LAAS, LORIA, GMD, les Universités d'Edimbourg, d'Eindhoven, de Kiel, de Passau, de Liège, d'Oxford, de Crète, Carnegie Mellon, Cornell, Stanford, de Montréal et d'Ottawa.

Sergio Yovine a séjourné pendant un mois à l'Université Cornell en tant que chercheur invité.

Mats Kindahl de l'Université d'Uppsala a séjourné pendant un mois dans notre projet.

6 Diffusion des résultats

6.1 Diffusion de produits

La diffusion de la boîte à outils CADP (regroupant les outils de vérification ALDÉBARAN, BCG, CÆSAR, CÆSAR.ADT et OPEN/CÆSAR) s'est poursuivie. En 1994, 20 nouveaux sites ont demandé une mise à disposition, ce qui porte le nombre total de sites à 94. Nous avons effectué de nombreuses démonstrations publiques de la boîte à outils, notamment pendant une journée industrielle spécialement organisée à Ottawa (juin 1994), ainsi qu'à l'occasion des conférences HPN (Grenoble, juin 1994), CAV (Stanford, juin 1994), CONCUR (Uppsala, août 1994), COST-247 (Evry, septembre 1994), FORTE (Berne, octobre 1994) et des Entretiens Jacques Cartier (Grenoble, décembre 1994).

Les logiciels suivants sont distribués par ftp public :

- ARGONAUTE : Environnement de programmation et vérification de systèmes réactifs, multi-langages et multi-outils.

- KRONOS : Outil de vérification de systèmes temporisés par évaluation symbolique : comparaison d'automates temporisés à des propriétés exprimées en TCTL.
- LÉSAR : Outil de vérification de propriétés de programmes écrits en LUSTRE.
- POLKA : Outil d'analyse des propriétés numériques des programmes, fondé sur une interprétation abstraite à base de polyèdres convexes.
- BAC : Outil de vérification d'automates booléens à base de BDD.
- MAGEL : Outil de réduction à l'aide de BDD et d'analyse à l'aide de polyèdres convexes s'appliquant aux réseaux de Petri produits à partir de programmes LOTOS.

6.2 Actions d'enseignement

6.2.1 Enseignement universitaire

Tous les membres permanents du projet, et en particulier les enseignants-chercheurs des établissements d'enseignement supérieur de Grenoble, participent activement aux formations de ceux-ci : INPG et UJF.

Nous donnons des cours spécifiquement dédiés aux thèmes développés dans le projet, dans le cadre de DEA de Grenoble.

Par ailleurs, nous avons assuré les cours suivants :

Compilation : cours 2e année, Ecole Normale Supérieure de Lyon (J.-C. Fernandez).

Méthodes de vérification de systèmes finis : cours de DEA IMA commun à l'Ecole Normale Supérieure, l'Ecole Polytechnique, et les universités Paris VI, VII et XI (N. Halbwachs).

6.2.2 Séminaires et formation permanente

Les membres du projet ont participé à de nombreux séminaires de recherche, conférences invitées, ainsi qu'aux actions de formations permanentes et écoles d'été suivantes :

Ecole d'été MOVEP'94 : X. Nicollin a assuré un cours sur les "Automates temporisés" à l'école d'été "Modélisation et Vérification des Processus Parallèles" à Nantes du 21 au 24 juin 1994.

Tutoriel CONCUR'94 : J. -Cl. Fernandez a assuré (conjointement avec R. de Simone du projet Meije) un tutoriel intitulé "Model based verification methods and tools" lors de la conférence CONCUR du 22 au 25 août à Uppsala.

6.2.3 Jurys de thèse

Nous avons été rapporteurs et participé aux jurys de thèses suivantes :
P. Caspi, Benont Caillaud, Rennes I
P. Caspi, Nai-Cheng Nhsiao, Orsay
N. Halbwachs, thèse de F. Mignard, Ecole des Mines.
J. Sifakis, thèse de Ulf Carlsen, Université Paris XI à Orsay.
J. Sifakis, thèse de Patrice Godefroid, Université de Liège.
J. Sifakis, habilitation de Claude Jard, Rennes.

Nous avons participé aux jurys de thèses suivants :
P. Caspi, Nacera Bennacer, CNAM
H. Garavel, thèse de Marcos Veloso Peixoto, Paris VII.
N. Halbwachs, thèse de F. Masdupuy, Ecole Polytechnique.
N. Halbwachs, habilitation de F. Ouabdesselam, UJF.
F. Maraninchi, thèse de F. Mignard, Ecole des Mines.
J. Sifakis, thèse de F. Laroussinie, INPG.

7 Publications

Thèses

[1] A. GIRAULT, *Sur la répartition de programmes synchrones*, thèse de doctorat, Institut National Polytechnique de Grenoble, janvier 1994.

[2] M. JOURDAN, *Etude d'un environnement de programmation et de vérification des systèmes réactifs, multi-langages et multi-outils*, thèse de doctorat, Université Joseph Fourier, Grenoble I, septembre 1994.

[3] A. KERBRAT, *Méthodes symboliques pour la vérification de Processus Communicants : étude et mise en oeuvre*, thèse de doctorat, Université Joseph Fourier, Grenoble I, novembre 1994.

[4] C. LOISEAUX, *Vérification symbolique de programmes réactifs à l'aide d'abstractions*, thèse de doctorat, Université Joseph Fourier, Grenoble, février 1994.

[5] A. Olivero, *Modélisation et analyse de systèmes temporisés et hybrides*, thèse de doctorat, Institut National Polytechnique de Grenoble, septembre 1994.

Articles et chapitres de livre

[6] R. Alur, C. Courcoubetis, N. Halbwachs, T. Henzinger, P. Ho, X. Nicollin, A. Olivero, J. Sifakis, S. Yovine, «The Algorithmic Analysis of Hybrid Systems», *Theoretical Computer Science B, vol. 137, janvier 1995*, 1994.

[7] H. Garavel, R.-P. Hautbois, «Experimenting LOTOS in Aerospace Industry», *in : Theories and Experiences for Real-Time System Development*, T. Rus et C. Rattray (réd.), *Amast Series in Computing*, *2*, World Scientific, 1994, ch. 11.

[8] T. Henzinger, X. Nicollin, J. Sifakis, S. Yovine, «Symbolic model checking for real-time systems», *Information and Computation 111*, 2, 1994, p. 193–244.

[9] C. Loiseaux, S. Graf, J. Sifakis, A. Bouajjani, S. Bensalem, «Property Preserving Abstractions for the Verification of Concurrent Systems», *Formal Methods in System Design, à paraître* , 1994.

[10] X. Nicollin, J. Sifakis, «The algebra of timed processes ATP: theory and application», *Information and Computation*, 1994, à paraître .

Communications à des congrès, colloques, etc.

[11] R. Alur, C. Courcoubetis, T. Henzinger, P. Ho, X. Nicollin, A. Olivero, J. Sifakis, S. Yovine, «The Algorithmic Analysis of Hybrid Systems», *in : 11th. International Conference on Analysis and Optimization of Systems: Descrete Event Systems*, G. Cohen, J.-P. Quadrat (réd.), Lecture Notes in Control and Information Sciences 199, Springer-Verlag, Sophia-Antipolis, France, juin 1994.

[12] E. Asarin, O. Maler, «On some Relations between Dynamical Systems and Transition Systems», *in : ICALP'94*, LNCS 820, Springer Verlag, 1994.

[13] A. Bouajjani, R. Echahed, P. Habermehl, «Verifying Infinite State Processes with Sequential and Parallel Composition», *in : accepté à 22nd Symp. on Principles of Programming Languages (POPL'95)*, ACM, 1994.

[14] A. Bouajjani, R. Echahed, R. Robbana, «Verification of Context-Free Timed Systems using Linear Hybrid Observers», *in : CAV'94*, LNCS 818, 1994.

[15] A. Bouajjani, R. Echahed, R. Robbana, « Verification of Nonregular Temporal Properties for Context-Free Processes », *in : CONCUR'94*, LNCS 836, Springer Verlag, 1994.

[16] A. Bouajjani, R. Echahed, R. Robbana, « Verifying Invariance Properties of Timed Systems with Duration Variables», *in : FTRTFT'94*, LNCS 863, Springer Verlag, 1994.

[17] P. Caspi, A. Girault, D. Pilaud, «Distributing Reactive Systems », *in : Seventh International Conference on Parallel and Distributed Computing Systems, PDCS'94*, ISCA, octobre 1994.

[18] P. Caspi, « Towards recursive block diagrams », *in : 19th IFAC Workshop on Real Time Programming WRTP'94*, IFAC, p. 81–86, juin 1994.

[19] C. Daws, A. Olivero, S. Yovine, « Verifying ET-LOTOS programs with KRONOS », *in : Proc. FORTE'94*, Bern, Switzerland, octobre 1994.

[20] S. Graf, « Verification of a distributed Cache memory by using abstractions », *in : Conference on Computer Aided Verification CAV'94, Stanford*, LNCS 818, Springer Verlag, juin 1994.

[21] N. Halbwachs, Y.-E. Proy, P. Raymond, « Verification of linear hybrid systems by means of convex approximations», *in : International Symposium on Static Analysis, SAS'94*, B. LeCharlier (réd.), LNCS 864, Springer Verlag, Namur (belgium), septembre 1994.

[22] N. Halbwachs, « About synchronous programming and abstract interpretation », *in : International Symposium on Static Analysis, SAS'94*, B. LeCharlier (réd.), LNCS 864, Springer Verlag, Namur (belgium), septembre 1994.

[23] M. Jourdan, F. Lagnier, P. Raymond, F. Maraninchi, « A Multiparadigm Language for Reactive Systems », *in : 5th IEEE International Conference on Computer Languages*, IEEE Computer Society Press, Toulouse, mai 1994.

[24] M. Jourdan, F. Maraninchi, « A Modular State/Transition Approach for Programming Real-Time Systems », *in : ACM Sigplan Workshop on Language, compiler and tool support for real-time systems*, Available through FTP anonymous on site `cs.umd.edu`, in file : `/pub/faculty/pugh/sigplan_realtime_workshop_94/jourdan.ps.Z` and through WWW page : `http://www.cs.umd.edu/~pugh/sigplan_realtime_workshop_94/`), Orlando, FL, juin 1994.

[25] M. Jourdan, F. Maraninchi, «Studying Synchronous Communication Mechanisms by Abstractions », *in : IFIP Working Conference on Programming Concepts, Methods and Calculi*, Elsevier Science Publishers, San Miniato, Italy, juin 1994.

[26] A. KERBRAT, «Reachable state space analysis of LOTOS programs», *in : 7th international Conference on Formal Description Techniques for Distributed Systems and Communication Protocols, Bern, Switzerland*, octobre 1994.

[27] OLIVERO, J. SIFAKIS, S. YOVINE, «Using abstractions for the verification of linear hybrid systems», *in : 6th. Computer-Aided Verification*, D. Dill (réd.), Lecture Notes in Computer Science 818, Sprinver-Verlag, p. 81–94, California, juin 1994.

Rapports de recherche et publications internes

[28] S. BORNOT, *Comparaison observationnelle des systèmes temporisés*, Mémoire, Ecole Polytechnique, Paris, juillet 1994.

[29] A. BOUAJJANI, R. ECHAHED, R. ROBBANA, «Decidability Results for the Verification Problem of Pushdown Linear Hybrid Systems», *Rapport Technique n°Spectre-94-6*, VERIMAG, Grenoble, Grenoble, 1994.

[30] H. GARAVEL, «Binary Coded Graphs — Definition of the BCG Format (version 1.0)», *Rapport Technique n°à paraître* , INRIA Rhône-Alpes, Grenoble, 1994.

[31] H. GARAVEL, «On the Introduction of Gate Typing in E-LOTOS», *Rapport Technique n°94-3*, VERIMAG, Grenoble, Grenoble, février 1994, Annex C of ISO/IEC JTC 1/SC 21/WG 1 N 1314 Revised Draft on Enhancements to LOTOS.

[32] H. GARAVEL, «Six improvements to the process part of LOTOS», *Rapport Technique n°94-7*, VERIMAG, Grenoble, Grenoble, juin 1994, ISO/IEC JTC 1/SC 21/WG 1 N 1337. Annex L of the Revised Draft on Enhancements to LOTOS.

[33] A. KERBRAT, «Simultaneous composition in BDD based model checking», *Rapport Technique n°Spectre-94-2*, VERIMAG, Grenoble, 1994.

[34] Y. KESTEN, O. MALER, A. PNUELI, «Alternately, Quantification for Free», soumis pour publication, 1994.

[35] F. LAGNIER, P. RAYMOND, C. DUBOIS, «Vérification de propriétés de programmes SAGA/LUSTRE», *Rapport final du contrat n° 92 34 463 00 470 7501*, DRET, juillet 1994.

[36] D. LESENS, *Synthèse de programmes réactifs*, Mémoire, Ecole Polytechnique, Paris, juillet 1994.

[37] A. MALER, O. PNUELI, J. SIFAKIS, «On the Synthesis of Discrete Controllers for Timed Systems», soumis à STACS'95, 1994.

[38] R. MATEESCU, *Définition et compilation d'un méta-langage pour l'implémentation des logiques temporelles*, Mémoire, Institut National Polytechnique de Grenoble, juin 1994.

[39] L. MOUNIER, «A LOTOS Specification of a Transit-Node», *Rapport Technique n°Spectre-94-8*, VERIMAG, Grenoble, Grenoble, mars 1994.

[40] R. RUFFIOT, «Définition et réalisation d'un atelier logiciel pour l'étude des systèmes de transitions», *Mémoire d'ingénieur CNAM*, Institut IMAG, Grenoble, décembre 1994.

[41] P. SCHAAR, «Un environnement de programmation pour le langage graphique Argos», *Mémoire d'ingénieur CNAM*, VERIMAG, Grenoble, mars 1994.

[42] M. SIGHIREANU, «Implémentation optimisée des types abstraits algébriques du langage LOTOS», *Mémoire d'ingénieur de l'Institut Polytechnique de Bucarest*, VERIMAG, Grenoble, Grenoble, septembre 1994.

8 Abstract

The SPECTRE project aims at aiding designers of concurrent and real-time systems, for which the use of formal methods appears both necessary and reasonably feasible. Methods and tools are provided for the main design tasks: specification, programming and validation. The SPECTRE project is involved in research in the three following directions:

- Design and use of specification languages for concurrent and real-time systems. Declarative languages (temporal logics) are particularly considered, together with their connections with imperative formalisms (process algebras and transition system based languages).
- Design and/or implementation of programming languages for concurrent and real-time systems. The semantics of such languages and the associated compilation techniques are studied.
- Definition and implementation of validation methods, well-suited for these languages. In the considered application domains (communication protocols, real-time control systems and hardware), automated validation seems achievable. The problem of error identification and correction is also addressed.

One goal of the work is to confront theoretical results to practical problems raised by actual system design. It is intended to produce, on one

hand, prototype tools open to industrialization, and on the other hand, practical design methods.

The project consists of several coordinated research actions oriented to specific applications — reactive systems, communication protocols, timed systems — and one transversal action providing the basic technology for validation.

SPECTRE is a project of VERIMAG, a joint laboratory of CNRS, Institut Polytechnique de Grenoble, Université Joseph Fourier and VERILOG SA. The objectives of VERIMAG include the transfer of research results of the project.

SPECTRE participates in several international projects: the ESPRIT-BRA React project, the ESPRIT-BRA Concur-2 project, the CEE-Canada Eucalyptus project for the development of protocol validation tools, and a France-Israel cooperation project on the specification and analysis of hybrid systems.

Table des matières