Rapport INRIA 1994 — Programme 2

Dérivation de Spécifications et de Programmes

Projet PROGRAIS

3 mai 1995

Projet PROGRAIS

Dérivation de Spécifications et de Programmes

Mots-clés : communication (8), démonstration automatique (18), environnement de programmation (8), lambda-calcul (20), logique (20), logique temporelle (12, 15, 17), parallélisme (17), programmation (1), programmation parallèle (12), qualité du logiciel (4, 15, 17), raisonnement (1), réseau informatique (9), réutilisation de logiciel (4), spécification (6), spécification formelle (12, 18), temps réel (9), transformation de programme (12, 18), typage (20), vérification de programme (8, 15).

Localisation : *Nancy*[1]

1 Composition de l'équipe

Responsable scientifique

Jean-Pierre Finance, professeur, Université Henri Poincaré (UHP-Nancy 1)

Responsable permanent

Jeanine Souquières, professeur, Nancy 2

Secrétaire

Nathalie Pierre

[1] Projet commun à l'Inria Lorraine et au Centre de Recherche en Informatique de Nancy (CRIN, laboratoire des universités de Nancy 1, Nancy 2 et de l'INPL, associé au CNRS URA 262

Personnel Inria

Khaled Bsaies, chargé de recherche
Philippe de Groote, chargé de recherche
Alain Quéré, chargé de recherche, tiers temps[2]
Christian Retoré, chargé de recherche, à partir du 1er octobre 1994

Personnel CNRS

Didier Galmiche, chargé de recherche, jusqu'au 30 septembre 1994

Personnel Université

Dominique Méry, professeur, UHP-Nancy 1
Francis Alexandre, maître de conférences, UHP-Nancy 1
Didier Galmiche, maître de conférences, UHP-Nancy 1, retour de détachement le 1er octobre 1994
Jacques Guyard, maître de conférences, UHP-Nancy 1
Jean-Pierre Jacquot, maître de conférences, UHP-Nancy 1
Jacques Jaray, maître de conférences, Inpl
Nicole Lévy, maître de conférences, UHP-Nancy 1
André Schaff, maître de conférences, UHP-Nancy 1
Serge Vorobyov, maître de conférences associé, UHP-Nancy 1, Jusqu'au 30 septembre 1994
Naima Brown, ATER, Metz
Olivier Festor, ATER, UHP-Nancy 1
Jean-Michel Hufflen, ATER, UHP-Nancy 1, jusqu'au 31 aout 1994
Jean-Yves Marion, ATER, UHP-Nancy 1, à partir du 1er septembre 1994
Abdellilah Mokkedem, ATER, UHP-Nancy 1

Chercheurs invités

Peter Ladkin, research fellow, University of Stirling (Grande Bretagne), à partir du 1er avril 1994
Jim Lipton, assistant professor, Wesleyan University (Etats Unis), du 15 mai 1994 au 15 juin 94
John Mullins, professeur assistant, Université d'Ottawa (Canada), du 1er mai 1994 au 30 juin 1994

[2] A. Quéré est responsable des moyens informatiques de l'Inria Lorraine et du CRIN

Ingénieur expert

Najib El Cadi, sous contrat CNET

Chercheurs post-doctorants

Robert Darimont, boursier CEE, à partir du 1^{er} juillet 1994
Paul Gibson, boursier CEE, à partir du 1^{er} juillet 1994
Graeme Smith, boursier DRED, jusqu'au 31 janvier 1994

Chercheurs doctorants

Denis Béchet, allocataire moniteur normalien
Eric Boudinet, boursier MESR
Abdelkader Dekdouk, chercheur sous contrat CNET
Eric Dillon, boursier MESR, à partir du 1^{er} octobre 1994
Odile Hermann, professeur certifié à l'IUFM
Paul Kaboré, boursier du Burkina Faso
Thomas Lambolais, boursier MESR
Guy Perrier, professeur agrégé
Catherine Pilière, boursier MESR
Dominique Quesnot, boursier CIFRE (SEMA-METRA)
Angèle Rivituso, professeur certifié
Denis Roegel, boursier MESR

2 Présentation du projet

L'objectif de l'équipe Prograis est de comprendre et d'exprimer le processus de construction du logiciel aux différents stades de son cycle de vie. Nos recherches visent donc à proposer des techniques de conception de logiciels fondées sur une manipulation explicite du processus de développement et à étudier des techniques permettant de faire évoluer et de réutiliser les logiciels existants.

Notre thème de recherche étant très étendu, l'activité de notre équipe s'est cristallisée autour de cinq axes dont chacun est devenu quasi-autonome au cours des années :

- l'axe PLANO qui tend à modéliser les mécanismes de développement et de réutilisation de spécifications et de programmes,
- l'axe RESEAUX qui étudie les problèmes de validation de logiciels dans le cas particulier des protocoles de communication,

- l'axe COMETE qui étudie et développe des méthodes formelles de développement de systèmes parallèles et distribués,

- l'axe SPES qui se focalise sur le développement de programmes par transformation de spécifications dans le cadre particulier de la programmation logique,

- l'axe TYPES qui étudie et exploite les méthodes de programmation basées sur la théorie de la démonstration, en général, et celle des types, en particulier.

Une réorientation du projet est prévue. Le projet PROGRAIS, en tant que tel, est donc amené à disparaitre fin 1994.

3 Actions de recherche

3.1 PLANO : Construction Raisonnée de Spécifications et de Programmes

Participants : Najib El Cadi, Robert Darimont, Jean-Michel Hufflen, Jean-Pierre Finance, Jean-Pierre Jacquot, Thomas Lambolais, Nicole Lévy, Dominique Quesnot, Angèle Rivituso, Graeme Smith, Jeanine Souquières

Les travaux réalisés dans cet axe concernent la définition d'un *modèle* et d'*outils* permettant de décrire et de maîtriser l'activité de *développement* de spécifications et de programmes. Ce modèle supporte le développement incrémental et facilite la construction et la réutilisation de logiciels. Il décrit un développement comme une suite de couples état-transition, la transition, appelée étape, consiste à sélectionner une tâche du plan de travail et à lui appliquer un opérateur de développement. Au cours d'une étape, des décisions sont prises pour atteindre un objectif, ces décisions portant à la fois sur le produit à concevoir (programme ou spécification) et sur la suite des tâches à réaliser pour atteindre l'objectif.

3.1.1 Validation et extension du modèle de développement

Participants : Robert Darimont, Jean-Pierre Jacquot, Jean-Pierre Finance, Dominique Quesnot, Jeanine Souquières

Les tactiques. L'expérimentation du modèle sur des études de cas réels montre qu'à chaque étape du développement un grand nombre de tâches et d'opérateurs peuvent être choisis. En fait, un opérateur de développement permet de décrire comment résoudre une tâche mais aucune aide n'est fournie au développeur pour déterminer le moment le plus propice à l'application des opérateurs ainsi que la motivation sous-jacente. Le modèle de développement a été enrichi [65], notamment avec les concepts de stratégies et tactiques [44]. Quatre composants caractérisent une tactique : une **situation** qui décrit ses conditions d'application; une **heuristique** précise ses critères d'application; une **motivation** qui justifie son application en termes de buts de développement et un **guide de développement** qui indique quelles tâches et quels opérateurs choisir et dans quel ordre. Une stratégie est structurée de la même façon; les trois premiers composants ont la même sémantique que pour une tactique. Le guide de développement indique les tactiques qui peuvent être appliquées. Des exemples de tactiques ont été approfondis.

Description d'interfaces. Une approche intégrée de la conception de logiciels nécessite de prendre en compte les aspects d'interface homme-machine. Dans ce domaine, deux axes ont été approfondis. D. Quesnot a poursuivi son travail de définition d'un modèle et d'un formalisme de spécification d'interface. L'originalité de ce travail réside dans l'utilisation de notre modèle de développement pour décrire une interaction entre un utilisateur et un logiciel. Notre objectif est de couvrir l'espace situé entre "l'analyse des tâches" (au sens ergonomique) et les outils de réalisation (boîtes à outils ou systèmes de génération d'interface utilisateur). Actuellement, un outil d'édition du formalisme (sous Concerto) ainsi qu'un prototypeur permettant d'exécuter les spécifications (compilateur vers Tcl/Tk) sont disponibles.

La syntaxe concrète d'un formalisme peut aussi être étudiée dans une perspective d'interface homme-machine. En particulier, se pose le problème du compromis entre les facilités d'expression qu'il est souhaitable d'offrir au rédacteur et les possibilité d'implanter le langage. A. Valdenaire, dans le cadre d'un stage CNAM, a étudié cette question dans le

cas de Glider en lui adjoignant un analyseur syntaxique autorisant l'utilisation d'opérateurs misfix [70]. Une des originalités du travail a été d'aborder le problème sous un angle pragmatique et quantitatif. Les résultats obtenus seront présentés à la prochaine conférence *Requirement Engineering.*

Maintenance et architecture logicielle. Le séjour de J.-P. Jacquot à *Carnegie Mellon University* avait pour objectif d'étudier l'application du modèle de développement à de nouveaux domaines. Un premier domaine concerne la maintenance des programmes, qui reste une activité assez mal comprise actuellement. Une étude de cas [27], présentée à l'*International Conference on Software Maintenance* a montré l'intérêt que notre modèle présente dans le domaine. La perspective ouverte devrait mener à des approfondissements substantiels dans un proche futur. Un second domaine concerne l'architecture logicielle, c'est à dire la phase suivant immédiatement la spécification. Force est de constater que ce domaine reste encore mal défini, les concepts de base commençant juste à émerger. Un travail de formalisation d'un cadre de programmation distribuée est en cours afin de mieux identifier les opérations de développement caractéristiques de cette phase.

Réutilisation dans le cadre d'un outil de gestion de publications. A partir d'un outil développé localement, Bibgest, et d'un ensemble d'améliorations souhaitées par les utilisateurs de cet outil, une étude a été réalisée sur ses différentes possibilités d'évolution. Les mécanismes de réutilisation utilisés ont porté à la fois sur l'outil (la base de la nouvelle version de l'outil est une version figée de Bibcard, outil disponible sur Sun) et sur les concepts disponibles dans l'outil local [62].

3.1.2 Construction de spécifications

Participants : Thomas Lambolais, Nicole Lévy, Graeme Smith, Angèle Rivituso, Jeanine Souquières

Modélisation d'une approche. Une bibliothèque d'opérateurs permet de modéliser une approche en proposant des schémas de raisonnement caractéristiques. Ainsi l'approche orientée objet a été décrite par une bibliothèque d'opérateurs de développement pour [38]. Ceci peut être

généralisé pour modéliser l'approche orientée objet de la programmation. Une bibliothèque d'opérateurs de développement de programmes Eiffel est en cours de définition. Dans le modèle de développement, il est possible de manipuler l'historique de l'application des opérateurs lors d'un développement. Ceci autorise à retrouver des informations qui n'apparaissent plus dans le texte de la spécification. C'est en particulier le cas pour les *classes* qui peuvent être identifiées alors que dans la spécification Z elles n'apparaissent pas clairement. Il est alors possible de faire de la vérification et du raffinement de manière modulaire [39].

Opérateurs indépendants du langage. Suite à la définition pour Z et Glider de bibliothèques d'opérateurs de développement, il nous a paru important de les comparer. Pour cela nous avons étudié la possibilité de définir des opérateurs de manière indépendante d'un langage de spécification ou plutôt en parallèle pour plusieurs. Ceci a amené à séparer précisément dans la définition des opérateurs, la partie méthode de la partie dépendante du langage. Les opérateurs décrivant l'approche orientée objet ont été redéfinis pour pouvoir s'appliquer aussi bien à une spécification basée modèle en Z qu'à une spécification algébrique en Glider [56, 35] ou en PLUSS [40].

Spécifications LOTOS

Notre objectif est d'isoler des étapes de construction d'architectures de protocoles de communication. Nous avons étudié des exemples classiques en LOTOS, dans le but, soit de les reconstruire entièrement, soit de les obtenir par transformation à partir d'autres spécifications. Ces dernières peuvent par exemple être des spécifications partielles du même problème ou d'un problème simplifié, ou encore des spécifications d'un problème voisin, toujours dans le même formalisme. Nous n'envisageons pas, pour le moment, le problème de la transformation d'un formalisme (par exemple une description en logique) vers un autre. En revanche, le cas du passage d'une spécification semi-formelle vers une description formelle fait partie de nos objectifs. Deux démarches de construction opposées ont été observées, débouchant sur des descriptions entièrement différentes :

- L'une réserve la description des données pour la fin, après avoir explicité l'ensemble du comportement. Dans une telle démarche, le pouvoir d'expression des types n'est pas utilisé.

- La deuxième donne une description abstraite de l'ensemble des fonctionnalités attendues du système. Le contrôle est réservé pour clarifier l'ordonnancement temporel des opérations.

Nous avons montré que la seconde démarche n'est pas réaliste pour la construction de protocoles. Différents types d'opérateurs interviennent implicitement à plusieurs niveaux et sont en cours d'explicitation.

3.1.3 Environnement

Participants : Najib El Cadi, Jean-Michel Hufflen

Evolution de la maquette. La maquette Icarus est disponible [63, 51, 50]. Le traducteur GLIDER←FISC a été finalisé dans le cadre de l'opération SALSA et intégré dans la maquette [42]. Un module générique de gestion de bibliothèques a été rajouté [67]. Le vérificateur de types développé par nos collègues du projet Icarus a été intégré dans la maquette.

Intégration de Lotos dans la maquette. En vue d'intégrer le langage LOTOS dans l'environnement développé dans le cadre d'Icarus, nous réalisons un environnement LOTOS sous CENTAUR. Un effort particulier est porté sur la partie analyse sémantique et sur la réutilisation et l'adaptation d'un simulateur développé dans le cadre du projet GIPE-II qui n'admet actuellement que des spécifications de taille réduite.

3.2 Modélisation et validation des protocoles de communication

3.2.1 Validation de protocoles de communication

Participants : Laurent Andrey, André Schaff

Pour représenter les protocoles de communication et les algorithmes distribués, nous avons expérimenté, via le formalisme **GAPP**[3] [6, 69] l'utilisation des grammaires attribuées pour décrire les entités de protocole et en dériver de manière semi-automatique une implantation. Un prototype a été développé dans ce cadre. Or lors de la conception

[3]Grammaires Attribuées Pour les Protocoles

de ces algorithmes distribués des erreurs peuvent être commises. Nous avons étudié comment nous pouvions ajouter des aspects validation au prototype GAPP [49].

3.2.2 Prise en compte du temps dans la construction de spécifications

Participants : Abdelkader Dekdouk, André Schaff

Le respect des contraintes temporelles est très important pour la sûreté des équipements et pour la sécurité des utilisateurs. D'où le recours à des spécifications formelles qui nous permettent de prévoir dès la conception le comportement de l'application. Les langages normalisés ne permettent pas une bonne spécification des contraintes temporelles utilisées dans la description informelle d'un protocole à part la temporisation. Notre objectif est d'améliorer ces formalismes pour leur permettre de décrire ces applications temps-réel distribuées. Pour cela, le travail entrepris vise à

1. étudier l'impact de l'introduction des opérateurs temporels au formalisme LOTOS,
2. proposer une méthodologie de développement de spécification LOTOS en tenant compte des contraintes temporelles spécifiées dans la description informelle du protocole,
3. proposer une approche de validation des spécifications LOTOS avec contraintes temporelles.

Comme les aspects tests ci-dessus, cette recherche s'intègre dans le contrat CNET mené avec l'action 3.1.

3.2.3 Description et test du comportement des objets gérés

Participants : Olivier Festor, Paul Kaboré, André Schaff

Dans le cadre de l'administration des réseaux d'ordinateurs et/ou de télécommunications, il existe deux approches : le standard SNMP[4] de l'IAB et les normes CMIS/CMIP[5] et GDMO[6] de l'ITU-T, ex CCITT,

[4]Simple Network Management Protocol

[5]Common Management Information Service/Protocol

[6]Guidelines for the Definition of Managed Objects

et de l'ISO. Chaque approche propose son modèle de description des ressources de gestion, modèle spécifique basé sur une approche formelle et comprenant un langage de spécification des ressources gérées.

Dans l'approche SNMP le langage de spécification est basé sur une description des ressources par des variables simples sans prendre en compte la spécification des opérations possibles sur les interfaces des ressources gérées. La simplicité est l'avantage d'une telle approche mais son inconvénient est de ne permettre ni la définition d'opérations complexes sur les ressources, ni la description formelle du comportement associé. Dans sa version actuelle, l'approche normalisée comporte le langage GDMO, basé sur les concepts orientés objet, pour la spécification des ressources de gestion. C'est le cadre de ce travail de recherche. Un ensemble de neuf formulaires ou gabarits (à savoir classe, module, attribut, groupe d'attributs, action, notification, paramètre, comportement et corrélation de noms) permet une description modulaire et normalisée des objets gérés et de leur organisation au sein d'une base d'information de gestion ou MIB[7]. Dans ce contexte normalisé il est nécessaire que les techniques de description formelle utilisées supportent les concepts de l'approche orientée-objet et s'intègrent parfaitement aux formalismes normalisés pour la spécification des ressources. Nous proposons d'étendre le langage normalisé GDMO pour supporter la description formelle du comportement associé aux objets de gestion ainsi que la spécification formelle des interfaces associées à tout objet géré [3, 66, 20].

Elément essentiel dans la production des systèmes distribués et normalisés, le test des protocoles dans les couches hautes et dans les systèmes d'administration de réseaux est un problème crucial pour toutes les sociétés qui développent actuellement des réalisations dans ce domaine. Contrairement aux couches basses où les aspects contrôle sont prédominants, la couche application est vue comme une combinaison de protocoles et ne peut être définie par un système à transitions car la structure complexe des données et l'orientation objet y sont très largement prépondérants. Les recherches sur les tests de conformité n'ont pour l'instant pas tellement abordé ces aspects. Aucune approche, ni procédure, n'a été proposée pour tester si les opérations des utilisateurs conduisent effectivement aux modifications attendues sur les ressources réelles du réseau. Or la conformité des implantations aux spécifications des besoins des utilisateurs est un élément essentiel. Une méthodolo-

[7] Management Information Base

gie de test complète se doit donc de prendre en compte non seulement les protocoles OSI d'administration mais également les composantes non protocolaires des applications d'administration. C'est l'objet de nos propositions faites dans [53, 30, 29].

Le prototype MODE[8] comporte dès à présent une approche simulative des spécifications de comportement. La génération semi-automatique de code pour les agents de gestion et génération de séquences de tests sont en cours de développement.

3.2.4 Description de données et administration réseaux

Participants : Eric Dillon, Jacques Guyard, André Schaff

L'étude des notations de description des données et l'élaboration d'outils d'aide associés aux diverses phases du développement des logiciels pour applications distribuées font partie intégrante de nos objectifs. Notre objectif plus général est de proposer une approche formelle intégrée pour la conception, la spécification, l'implantation et le test des applications d'administration de systèmes distribués sur la base des notations normalisées par l'OSI. Dans le cadre de cette proposition de travail, nous entreprendrons des recherches s'articulant autour d'axes liés à la répartition des applications et à l'administration des systèmes hétérogènes. Cette proposition de travail a été récemment retenue dans le cadre du Centre Charles Hermite, centre lorrain de compétence en modélisation et calcul à hautes performances.

3.3 COMETE

Participants : Naïma Brown, Paul Gibson, Jacques Jaray, Peter Ladkin, Dominique Méry, Abdelillah Mokkedem, Denis Roegel

Le travail se focalise sur l'étude des systèmes parallèles et distribués et les techniques utilisées sont principalement les méthodes formelles concernant divers aspects du développement du logiciel (spécification, programmation, implantation, vérification). Tous les aspects du génie logiciel ne sont pas couverts par notre activité et nous nous spécialisons sur quelques approches développées par des chercheurs extérieurs à ce groupe mais aussi sur le développement de techniques propres. Nos résultats sont, du reste, modestes: il est vrai que le parallélisme recouvre

[8] Managed Object Development Environment

de nombreuses activités et techniques. Ce rapport tend à mettre en évidence les résultats importants de notre activité:

1. la proposition d'une logique temporelle modulaire, MTL, comparable à TLA, et que nous voulons développer pour en juger de l'expressivité par rapport à des problèmes de télécommunications.
2. l'analyse des distributions d'actions UNITY suivant plusieurs politiques de répartition et sur différents types de langages de programmation (OCCAM,C LINDA).

Nos activités concernent les méthodes formelles permettant de prendre en compte durant le développement les aspects liés aux systèmes parallèles ou/et distribués: TLA$^+$, UNITY, MTL, δ-NU, B sans négliger les méthodes comme RAISE, VDM, Z. Nous décrivons plus précisèment notre activité dans les sous-sections qui suivent.

3.3.1 Conception de programmes parallèles à partir de spécifications par affinement successif

Participants : Naïma Brown, Jacques Jaray, Dominique Méry, Abdelillah Mokkedem, Denis Roegel

L'affinement successif de programmes parallèles à partir de spécifications formelles permet de s'assurer de la correction du programme obtenu par rapport à la spécification initiale, puisque la transformation est fondée sur la préservation de propriétés telles que celles d'*invariance* et celles de *fatalité*. Cependant, les programmes obtenus sont écrits dans une notation abstraite éloignée des langages de programmation parallèles classiques et il convient de développer des techniques de transformation de programmes (ensembles d'actions) en des programmes écrits dans des langages de programmation parallèle (C LINDA, OCCAM). Bien entendu, cela conduit à analyser les techniques possibles de parallélisation de systèmes d'actions ou de distribution de systèmes d'actions. Le travail s'appuie sur le formalisme UNITY de Chandy et Misra pour lequel nous avons construit un outil d'aide à la preuve dans l'outil B et nous avons examiné plusieurs types de transformations de programmes UNITY en d'autres langages ainsi que des stratégies de répartition.

Transformation des programmes *Unity* en programmes *UCL* Nous avons développé des **techniques de parallélisation** des programmes *Unity*. L'idée de base est une **analyse des dépendances**

entre les variables du programme, cette dépendance nous a permis de mettre en évidence quatre classes de variables :

- les variables uniquement modifiables, ce sont les variables non communicables pour le calcul d'une autre variable du programme,
- les variables de lecture, qui ne sont pas modifiables, les constantes du programme sont des variables de lecture,
- les variables de communication, ce sont les variables modifiables et communicables pour le calcul d'autres variables du programme,
- les variables locales, qui forment une sous-classe de la classe des variables de communication. Une variable x est locale à une autre variable y si la valeur de x est nécessaire uniquement pour le calcul de y.

Nous avons défini ensuite des **stratégies de répartition** de ces classes de variables pour construire un ensemble de processus. Trois stratégies de répartition ont été proposées :

- la première, est une stratégie de répartition qui minimise la communication.
- La seconde, est une stratégie avec un maximum de parallélisme.
- La troisième stratégie se veut un compromis entre le parallélisme et la communication. Il est clair que ce n'est pas l'unique stratégie qui puisse être développée dans ce but. Cette stratégie consiste à associer un processus à chaque variable uniquement modifiable et de communication. Nous introduisons ainsi de la communication pour la classe des variables de communication mais pas pour la classe des variables uniquement modifiables. En effet, par définition, ces variables ne sont pas communicables pour le calcul d'autres variables du programme. Nous ajoutons ensuite les variables locales et de lecture là où leur utilisation est nécessaire. Aucune communication n'est introduite au cours de cette deuxième étape.

La correction des schémas de traduction est fondée sur l'équivalence des comportements observables du programme UNITY et du programme UCL. Le langage UCL est ensuite utilisé comme nouveau point de départ de la transformation et permet de produire soit un programme C LINDA, soit un programme OCCAM. Donc, le processus de transformation comprend:

- une phase de transformation du programme UNITY en un programme UCL.
- une phase de transformation du programme UCL en C LINDA ou en OCCAM.

Cette approche favorise la réutilisation du programme UNITY, afin d'en donner d'autres versions par transformation avant de choisir une architecture donnée. Nous décrivons les deux approches:

Transformation des programmes *UCL* en programmes *C Linda* Nous avons premièrement défini des règles de transformations des actions de base, c'est-à-dire, les actions d'affectation et les actions de communication. Ensuite, nous avons donné des règles de transformation des opérateurs de composition, composition indéterministe équitable (ACT) et composition parallèle des processus ($\|$).

Nous avons démontré que ce schéma de transformation est correct, dans le sens qu'il préserve la sémantique des programmes *UCL*. De plus, il n'introduit pas d'interblocage et conserve les propriétés d'invariance et de fatalité des programmes *UCL*.

La démarche de construction de programmes que nous avons adoptée à consister à dériver à partir d'une spécification abstraite *Unity* et par transformations successifs un programme parallèle *C Linda*. Les preuves de correction des programmes *CLinda* obtenus par ce processus de transformation sont faites à un niveau logique, celui de la logique d'*Unity* à l'aide de B, et preservées tout au long du processus de transformation.

Transformation des programmes *UCL* en programmes *Occam* Nous avons ainsi une implantation des programmes abstraits *Unity* dans le langage de programmation *Occam*. Pour garantir la correction du schéma de traduction d'*UCL* dans *Occam*, nous avons proposé un algorithme d'ordonnancement dont le but est de d'éviter l'introduction d'interblocage au cours du processus de traduction.

En conclusion, nous pouvons dériver à partir d'une spécification abstraite *Unity* un programme parallèle *CLinda* ou *Occam*. La transformation est effectuée par des transformations successives qui nous assurent de la correction des programmes *C LINDA* ainsi obtenus. Ce travail a fait l'objet de publications dans les conférences [12, 13, 14].

Cette recherche s'intègre dans l'activité du Centre Charles Hermite.

3.3.2 Spécifier et raisonner dans MTL

Participants : Paul Gibson, Dominique Méry, Abdelillah Mokkedem

Le problème d'abstraction de la logique temporelle TL Nous montrons que le défaut majeur de la logique TL qui empêche une application acceptable à la vérification compositionnelle de programmes parallèles est étroitement lié au problème d'abstraction posé par l'opérateur *next* et la quantification locale [37, 57]. Nous explorons ce point de vue et proposons une sémantique plus raffinée pour ces deux opérateurs de la logique temporelle. Nous obtenons une nouvelle formulation de la logique TL, appelée MTL, dont la sémantique possède la propriété de *stuttering-closure* [37, 57]. Nous donnons une axiomatisation de la logique MTL et prouvons sa correction [37, 57]. La logique temporelle MTL a deux avantages majeurs. Le premier réside dans son pouvoir d'abstraction varié; le niveau d'abstraction d'une spécification (ou formule) est exprimé dans la formule elle même sous forme d'un index qui représente l'ensemble des variables observables. Le deuxième avantage est que d'importants concepts, notamment la *communication* et le *parallélisme*, s'avèrent axiomatisés rigoureusement dans la nouvelle logique temporelle [57]. D'autre part, outre la vérification compositionnelle et le développement systématique des programmes parallèles, le raffinement d'un programme par un autre programme à faible granularité est formalisé dans la logique MTL d'une manière assez élégante.

Vérification compositionnelle dans MTL Pour justifier les avantages de la logique MTL, nous l'appliquons à une notation de programmation parallèle appelée IPL [57]. Le langage IPL est une simple modification du langage introduit par Manna et Pnueli et basé sur la notion de *module*. Les modifications que nous avons introduites ont pour but principal d'établir une sémantique compositionnelle pour le langage IPL. Nous établissons une correspondance entre la sémantique des formules MTL et celle des programmes IPL. Cela nous fournira un langage de spécification pour la description du comportement des programmes IPL. Nous complétons ensuite le système de preuve de MTL par des axiomes et des règles d'inférence pour la démonstration de propriétés de programmes IPL. Le système de preuve obtenu est compositionnel, c'est-à-dire que les propriétés d'un programme sont établies à partir

des propriétés de ses composants sans connaître *a priori* leur structure interne.

Raffinement de programmes dans MTL Un autre aspect que nous avons étudié est le raffinement de programmes parallèles dans la logique MTL. D'une part, la quantification existentielle sur les variables locales modulo *stuttering* permet, d'une manière rigoureuse, l'introduction de variables locales, et d'autre part, l'invariance des opérateurs temporels par rapport aux w[9]-*stutterings* permet de raffiner une action en une séquence d'actions plus atomiques en préservant les propriétés du système. Nous avons montré comment utiliser MTL pour prouver qu'un programme parallèle raffine un autre programme parallèle (plus abstrait). Le raffinement est exprimé par l'*implication* entre deux formules logiques dans MTL et les règles de raffinement sont fondées sur les règles de vérification de programmes parallèles. En utilisant ces règles, une spécification abstraite d'un programme parallèle peut être progressivement raffiné en une implantation concrète.

Comparaison avec d'autres travaux Lamport est le premier à avoir défini une logique (TLA) dans laquelle les programmes sont décrits par des formules faisant abstraction du *stuttering*. La logique MTL est une reformulation de la logique TL, ayant les mêmes avantages que TLA.

Récemment, Manna et Pnueli ont obtenu les mêmes résultats dans le cadre de la logique temporelle TLR, une autre reformulation de la logique TL, introduisant une nouvelle définition des modèles qui considérant les états de position impaire comme états de changement et les états de position paire comme états de silence. Ils décrivent l'opérateur next-time comme un opérateur qui déplace d'une position impaire à la position paire voisine. Avec une telle inteprétation, l'axiome d'induction $(p \Rightarrow \bigcirc p) \supset (p \Rightarrow \Box p)$ n'est plus toujours valide. Cependant, cet axiome est important dans l'utilisation de la logique temporelle pour la preuve de propriétés d'invariance des programmes. Pour maintenir cet axiome, ils considèrent les programmes dont les variables sont *continues à gauche*, i.e., toute variable du programme ne peut changer de valeur que lors d'une transition d'un état impair vers un état pair. Contrairement à TLR, dans MTL l'axiome d'induction est toujours valide et nous

[9] w désigne l'ensemble des variables observables d'un module.

ne faisons aucune restriction sur les variables de programmes quand nous appliquons la logique MTL aux programmes [57].

L'idée de ramener la preuve qu'un programme parallèle *raffine* un autre programme parallèle (plus abstrait) à la preuve d'une *implication* entre deux formules logiques dépend de la capacité d'abstraire le comportement des variables locales tel les transitions vides (*stuttering steps*) dans la logique utilisée [57]. Ceci permet de rendre les spécifications indépendantes de l'implantation des variables locales et de la mesure précise du temps; une affectation peut être effectuée en une, deux, trois, ou un nombre quelconque de transitions.

En effet, la preuve qu'un programme implante une spécification (qui décrit essentiellement la sémantique temporelle d'un autre progamme plus abstrait) est un moyen permettant l'utilisation de la logique temporelle pour prouver le raffinement de programmes parallèles. Cependant, lorsque l'on veut utiliser la logique temporelle pour décrire la sémantique des programmes, l'opérateur *next-time* devient nécessaire pour exprimer l'effet des transitions sur les états. Cet opérateur est l'une des causes de la sensibilité de la logique TL au le *stuttering* qui complique la preuve de raffinement. Dans TLA, Lamport surmonte ce problème en exigeant que l'opérateur $'$ (pour les *primed variables*) ne peut être utilisé que dans des sous-formules d'une forme spécifique, où il ne cause aucun mal. Dans MTL [37, 57] la sémantique de l'opérateur next-time et de la quantification flexible a été raffinée, de manière à utiliser ces opérateurs dans les formules sans aucune restriction tout en garantissant l'invariance au *stuttering*.

3.3.3 Exprimer le calcul parallèle dans TLA$^+$

Participants : Paul Gibson, Peter Ladkin, Dominique Méry, Denis Roegel

Le langage de spécification TLA$^+$ défini par Lamport et le langage de programmation PCN sont les points de départ de cette recherche. L'objectif reste d'obtenir un programme parallèle correct PCN à partir d'une spécification TLA$^+$ obtenue par raffinements successifs.

La conception de programmes parallèles un tant soit peu importants nécessite une méthodologie et des outils adaptés, choses qui sont actuellement inexistantes. Afin de cerner le problème, on s'est intéressé

au langage de spécification TLA$^+$ introduit par Lamport en 1991 et au langage de programmation PCN défini par Chandy et Taylor à la même époque.

Une importante étude de cas a permis d'élaborer une première méthodologie pour la transformation de spécifications TLA$^+$ en programmes PCN et de dégager des résultats théoriques. En particulier, certaines règles de raffinement ont été identifiées.

Une autre étude de cas sur des raffinements de spécifications exprimant des problèmes de cohérence de caches est en cours actuellement en collaboration avec Peter Ladkin, Leslie Lamport, Bryan Olivier sur *Lazy Caching: An Assertional View.*

Des expérimentations ont été faites avec le prouveur TLP – une version spéciale du prouveur Larch pour TLA$^+$ – afin d'estimer son adéquation à nos problèmes.

Ce travail se fonde sur le développement d'études de cas du domaine du calcul à haute performances et s'intègre dans le Centre Charles Hermite.

3.4 SPES : Transformation de spécifications

Participants : Francis Alexandre, Khaled Bsaïes, Jean-Pierre Finance, Alain Quéré

3.4.1 Problématique

L'objectif général du thème SPES est d'étudier des mécanismes de construction systématique de programmes à partir d'énoncés de problèmes. On distingue deux étapes, la première est la conception d'une spécification formelle à partir d'un énoncé. La seconde étape consiste à dériver un programme efficace à partir de cette spécification.

Dans ce cadre les spécifications sont exprimées sous forme de clauses, les programmes résultants sont exprimés en clauses de Horn. Nous nous intéressons à l'approche transformationnelle et à l'approche déductive qui proposent un cadre formel de développement de programmes.

A côté des transformations bien connues de pliage et de dépliage, nous utilisons d'autres transformations de base telles l'introduction de nouveaux symboles de prédicats ou de fonctions et l'utilisation de propriétés. L'étude de ces transformations consiste à montrer leur correction par

rapport à certaines sémantiques, mais aussi à trouver des stratégies d'application.

L'approche déductive consiste à synthétiser un programme logique en prouvant une formule du premier ordre. Pour cela on dispose d'un ensemble de règles de déduction qui sont des extensions des transformations de dépliage et pliage.

3.4.2 Résultats obtenus

L'écriture d'une spécification initiale à partir d'un énoncé n'a fait l'objet que de peu de recherche, dans [9], une méthodologie de construction basée sur des paradigmes est proposée, elle décrit une approche descendante de construction de spécifications. Les spécifications ainsi construites possèdent des propriétés de modes entrées-sortie et de types et sont donc candidates à la dérivation.

Des stratégies de transformations basées sur des schémas ont été proposées. Un schéma est la donnée de la tête de la clause cible et des appels récursifs de son corps. Deux problèmes apparaissent, le premier, appelé "problème du dépliage", est d'obtenir par dépliage une clause dont la tête est une variante de la tête de la clause cible. Ce problème n'est pas décidable [9]. Une classe de programmes pour laquelle le problème du dépliage est décidable est mise en évidence [9]. Un deuxième problème est lié à la réussite du pliage afin d'obtenir les appels récursifs du corps de la clause cible. Des stratégies de synthèse de propriétés pour forcer le pliage ont été proposées [16, 15]. Des formules logiques du premier ordre sont générées par ces stratégies. La correction de ces stratégies est liée à la validité de ces formules.

L'approche suivie, qui fait intervenir les aspects fondamentaux de l'informatique (analyse des programmes, sémantique, démonstration automatique), est aussi expérimentale puisque les études théoriques décrites précédemment ont fait l'objet d'implantation des transformations, des stratégies et d'un démonstrateur dans le système SPES.

3.4.3 Perspectives

Etude de l'efficacité des programmes transformés Mesurer l'efficacité (ou la complexité) des programmes logiques avant et après transformation est un sujet sur lequel des tentatives ont été réalisées.

Dans la pratique on se limite souvent à des mesures de temps d'exécution ce qui ne suffit pas notamment pour guider les transformations.

Notre objectif est de définir pour les programmes des critères d'efficacité, d'identifier des stratégies de transformation guidées par les critères précédemment définis et d'implanter les fonctions de mesures de performances et les stratégies précédemment proposées.

Stratégies pour la preuve automatique de formules L'automatisation des stratégies de construction de propriétés nécessite que les preuves de formules générées soient aussi automatisée. Il est donc indispensable de découvrir des stratégies de preuves. L'objectif de ces stratégies est de réduire l'indéterminisme à la fois dans le choix des règles de déduction et dans le choix des atomes sur lequels s'appliquent ces règles.

3.5 Théorie des types: preuves et programmes

Participants : Denis Béchet, Eric Boudinet, Didier Galmiche, Philippe de Groote, Odile Hermann, Jean-Yves Marion, Guy Perrier, Catherine Pilière, Christian Retoré

3.5.1 Problématique

L'objectif de cet axe est l'étude et la conception de systèmes de synthèse automatisée dans des logiques constructives où les preuves ont un contenu algorithmique. Il consiste donc a étudier la mise en œuvre de l'isomorphisme de Curry-Howard et du paradigme *proofs as programs* ainsi que les liens entre les concepts de la logique et ceux de la programmation dans différents cadres logiques comme la théorie constructive des types, la logique intuitionniste du second ordre, la logique linéaire, la logique classique et plus généralement dans des λ-calculs typés [4].

Cette étude de la programmation avec des preuves est abordée suivant différent aspects complémentaires: les cadres logiques (λ-calcul typés) pour spécifier des logiques objets, les preuves et programmes dans des λ-calcul typés, l'étude du contenu algorithmique des preuves, la recherche de preuves (méthodes, automatisation) et la construction de programmes, les concepts de la programmation à travers la logique classique (contrôle, non-déterminisme, exceptions), l'étude de logiques et de λ-calculs pour le parallélisme.

3.5.2 Logique classique et λ-calcul

Dans le cadre des liens entre λ-calcul et logique classique, une étude du $\lambda\mu$-calcul de M. Parigot a été développée. Un premier travail a permis de traduire le $\lambda\mu$-calcul dans le λ-calcul, par CPS transformation [17]. Celle-ci induit, au niveau du typage, une interprétation de la logique classique en logique intuitionniste. De plus, on y prouve la normalisation du $\lambda\mu$-calcul. Un second travail a conduit à proposer une traduction du $\lambda\mu$-calcul dans un sous-système du λ_c-calcul de Felleisen et à démontrer que ces deux calculs sont isomorphes [18]. Concernant le non-déterminisme en λ-calcul, l'étude d'un λ-calcul typé non-déterministe dont le système de type est le calcul LK de Gentzen a été poursuivie. Pour permettre une interprétation calculatoire, on considère une version plus faible de ce calcul où la réduction modélise la réduction de preuve [19].

3.5.3 Recherches de preuves et programmes

L'étude du développement de preuves et de programmes dans ces cadres logiques de la théorie des types a été poursuivi en mettant l'accent sur la notion de recherche de preuves ainsi que sur la mécanisation du processus de développement, mettant ainsi en évidence la spécificité des liens entre preuves, programmes et types de données dans ces cadres logiques. Notons l'organisation par D. Galmiche et L. Wallen, d'un workshop, sur le thème de la recherche de preuves en théorie des types [1].

Preuves et programmes en AF_2. L'étude de la mécanisation du développement de preuves (et de programmes) en logique du second ordre (AF_2) a été poursuivie par l'étude de schémas d'induction adéquats pour la synthèse de programmes et la définition de procédures d'aide aux choix des variables d'induction et des types d'induction et à la synthèse de plans de preuve [22]. Un travail complémentaire a consisté à étudier la recherche de preuves dans ce cadre en utilisant un codage dans un logique d'ordre supérieur et a conduit à une implantation de démonstrateurs (et donc de synthétiseurs) automatique et interactif en MALI-λProlog.

Recherche de preuves en Logique Linéaire. Le problème de la déduction automatique en théorie des types étant au centre de ces travaux et la logique linéaire présentant des potentialités pour exprimer la concurrence et le parallélisme nous avons abordé l' étude de la construction de preuves dans différents fragments de la logique linéaire et de la notion de réseaux de preuves.

Un travail important a consisté à étudier la recherche de preuves et la normalisation de preuves en LL, à partir des notions de permutabilité d'inférence, de mouvement d'inférence dans une preuve [7]. Cela a conduit à définir la notion de preuve normale ou canonique et une méthode générale de conception de telle preuve [25] pour faire de la déduction automatique dans des fragments de LL autant en chaînage arrière qu'en chaînage avant. A partir de cette normalisation, il est possible de raffiner ces résultats et de proposer des techniques de recherche de preuves [23]. Ces méthodes ont été implantées sous forme de démonstrateurs automatique et interactif en Quintus-Prolog.

Ces résultats et leur spécialisation à la Logique Linéaire Intuitionniste (ILL) permettent d'aborder la recherche de preuves de propriétés et de spécifications de systèmes distribués [21]. De plus, du point de vue déduction automatique, on voit l'importance des preuves canoniques pour faire de la programmation logique en LL [24] et aborder également la programmation logique concurrente dans des fragments de ILL.

Etude des réseaux. Concernant la normalisation de réseaux, un travail a consisté à réduire le nombre de constructions dans les preuves en logique linéaire et à obtenir un mécanisme simple de normalisation de réseaux de preuves par une implémentation directe avec des réseaux d'interactions. L'étude de ces derniers a été poursuivie et a conduit à une nouvelle approche de l'evaluation partielle basée sur les notions de *valeurs alternatives* et d'*événements*, travaux qui devraient être diffusés très rapidement.

L'arrivée dans le groupe de C. Rétore va nous conduire à étudier des techniques de preuves pour le calcul ordonné [59] et leurs applications, entre autres, à la linguistique [33] et à la programmation logique concurrente. Celle de Jean-Yves Marion permet d'aborder ces travaux autour du λ-calcul et la logique linéaire du point de vue de la complexité [11].

4 Actions industrielles

4.1 Collaboration avec IBM

Nos collaborations contractuelles avec le laboratoire ENC d'IBM à Heidelberg concernent l'administration de réseaux (Voyez l'action de recherche 3.2).

4.2 Collaboration avec le CNET

Notre projet avec le CNET-Lannion a pour but d'appliquer nos techniques de construction et de réutilisation de spécifications formelles au domaine particulier des réseaux et des protocoles de communication [45, 52]. Ce projet est réalisé conjointement par les chercheurs des axes 3.1 et 3.2.

Une étude de cas est en cours de réalisation; elle concerne la spécification d'interfaces-usagers réseaux pour le RNIS. Les documents de départ sont des documents internationaux informels (rédigés en anglais) constitués à partir d'une norme et de l'histoire de son évolution. A ces textes informels sont joints un document complémentaire reformulant le texte informel sous forme de schémas LDS. Une spécification différentielle, texte réduit par rapport aux recommandations internationales précise les options nationales. Les normes françaises sont elles-mêmes constituées des textes informels et de la spécification différentielle. Cette étude nous a amené à certaines extensions du modèle de développement, cf. paragraphe 3.1.1 et à l'étude de démarches particulières 3.1.2.

4.3 Icarus

Le projet Esprit2 Icarus s'est terminé fin juin 1994 [46]. Dans ce cadre, nous avons développé une grosse étude de cas, "The Radio Advertising System", avec l'un de nos partenaires industriel, Decision Support System, Dublin. La maquette support est diffusée [43].

5 Actions nationales et internationales

5.1 Actions nationales

Nous participons aux diverses actions nationales suivantes :

- SAGACE, projet du PRC Programmation (ce projet regroupe des équipes de Strasbourg, Grenoble, Paris (LIENS), Evry et Orsay),
- ADER, projet du PRC Programmation (ce projet regroupe, en plus de notre équipe, les équipes SCOP (IMAG–LGI, Grenoble), Génie Logiciel de l'ONERA–CERT à Toulouse et l'IRIT de Toulouse),
- Réseaux Informatiques (mise en place par le MESR),

- Thème 2.1 du GDR Parallélisme, Réseaux et Systèmes distribués (description formelle et vérification, supervisé par Jean-Claude Fernandez),
- Réseau Grand Est (regroupement des équipes de recherche de de J.-J. Pansiot (Strasbourg), J.-P. Thomesse (Crin-Ensem), M. Tréhel (Besançon) et d'A. Schaff (Crin-Inria), sur une proposition initiale de Michel Tréhel)
- Modèles Logiques de la Programmation (projet inter PRC du CNRS dirigé par M. Parigot, dont les principaux sites sont Paris 7, Orsay, Lille, Marseille, Sophia-Antipolis, Chambéry et Nancy).

5.2 Actions internationales

Nous sommes impliquées dans plusieurs actions internationales:

- Réseau Médicis (ce réseau Capital Humain et Mobilité de la CEE, concernant les méthodes de développement de spécifications formelles, regroupe des équipes universitaires d'Allemagne, de Belgique, d'Espagne, de Grande-Bretagne, du Portugal et de Pologne—ces derniers, dans le cadre du programme européen PECO),
- Projet Icarus (projet du programme Esprit 2 de la CEE),
- Projet Gentzen (*Common foundations of logical and functional languages*, Esprit Basic Research Action, CEE),
- Collaboration avec l'Université Technique de Berlin (équipe du Prof. Jähnichen, dans le cadre d'une action Procope),
- Collaboration avec le Centre de Logique de l'Université Catholique de Louvain (dans le cadre d'une action Tournesol).

5.3 Invitations

Le projet a accueilli, dans le cadre de ses séminaires, plusieurs chercheurs, dont :

- Jenny Harvey, Adelaide University (Australie),
- Dominique Geniet, LRI-ORSAY,
- Patrice Godefroid, Université de Liège (Belgique),
- Hubert Garavel, Inria Rhônes Alpes,

- Thierry Cattel, NRC-IIT, Ottawa (Canada),
- Leslie Lamport, DEC, Palo Alto (Etats Unis),
- Gaétan Hains, Université de Montréal (Canada),
- Rachida Dssouli, Université de Montréal (Canada),
- Qiwen Xu, Abo Akademi, Turku (Finlande).

5.4 Participation à des comités de programmes et à des comités éditoriaux

Jean-Pierre Finance a fait partie des comités de programmes des conférences ICSE16 (Sorrento, 1994) et l'Intelligence Artificielle (Rabat, avril 1994), de la conférence IFIP TC2 (juin 1994), du workshop "Logic of Theory for Program Construction", à Dagstuhl (mars 1994).

Jeanine Souquières fait partie du comité de programmes de la troisième Conférence Internationale *on Achieving Quality in Software, AQuis 96* (Venise, janvier 1996). Elle fait partie du comité de rédaction de la revue TSI et est responsable d'un numéro spécial sur la maîtrise du développement de logiciels à paraître courant 1995.

6 Diffusion des résultats

6.1 Enseignement

Outre les cours de bases dispensés en deug, licence, maîtrise, écoles d'ingénieurs ..., les membres de l'équipe assurent les enseignements suivants:

- Concepts Avancés du Parallélisme, cours de DEA, UHP-Nancy 1 (Dominique Méry et Jacques Jaray)
- Programmation Fonctionnelle et Programmation Logique : Concepts Fondamentaux, cours de DEA, UHP-Nancy 1 (D. Galmiche).
- Théorie des types et Logiques d'ordre supérieur, cours de DEA, UHP-Nancy 1 (D. Galmiche).
- Concepts et Sémantiques des Systèmes Distribués et Communicants, cours de DEA, UHP-Nancy 1, (A. Schaff),

- Ingénierie des Protocoles, cours de DEA, UHP-Nancy 1, (A. Schaff, Ye-Qiong Song).

Laurent Andrey, Olivier Festor et André Schaff ont participé, en tant qu'enseignants, à l'Ecole d'été « Réseaux de communication et techniques formelles » de l'école nationale supérieure de télécommunication (Paris, septembre 1994) [66, 69].

6.2 Participation à des conférences, colloques et séminaires

Outre les colloques et conferences qui ont fait l'objet de communications (Voyez la section 7), les membres de notre équipe ont participé à divers événements.

Jeanine Souquières a présenté ses travaux aux séminaires du CNET à Issy-les Moulineaux et Lannion (mars et avril 1994), aux journées de rencontre du groupe « Logiciel-Zéro-Défault » de l'AFCET (septembre 1994).

Jeanine Souquières et Didier Galmiche ont présenté leurs travaux respectifs au *International Workshop on Logical Theory for Program Construction* (Dagstuhl, Allemagne, mars 1994).

Jeanine Souquières et Jean-Michel Hufflen ont présenté la maquette Icarus lors de la conférence *Formal Methods Europe, FME'94* (Barcelone, Espagne, octobre 1994).

Philippe de Groote a présenté ses travaux au séminaire de logique de Paris VII (décembre 1993), au *Isabelle's Lunch* de l'université de Cambridge (Grande Bretagne, avril 1994), aux séminaires de logique et de logique et catégories du centre de logique et du département de mathématiques de l'université catholique de Louvain (Louvain-la-Neuve, Belgique, décembre 1994), au séminaire d'informatique de l'université de Savoie (Chambéry, décembre 1994).

Christian Retoré, a présenté ses travaux à la journée « Grammaire et Théorie de la Preuve » de l'action ATALA du CNRS (Paris, décembre 1994), à la réunion du groupe de recherche en algorithmique et logique de l'université de Caen (décembre 1994).

Christian Retoré, Didier Galmiche, Jean-Yves Marion et Denis Béchet ont présenté leurs travaux au *Workshop on Proofs, Nets and Types* (Marseille, novembre 94).

Jean-Pierre Jacquot et Olivier Festor ont participé à la réunion de travail du thème 2.1 du GDR Parallélisme, Réseaux et Systèmes distribués (Grenoble, décembre 1994).

Nicole Lévy a présenté ses travaux aux Journées du GDR Programmation (Lille, septembre 1994) et au séminaire du groupe *programming system* de la *Carnegie Mellon University* (Etats-Unis, décembre 1994).

André Schaff a participé aux réunions de travail de l'action nationale « Réseaux Informatiques » (Toulouse, mars et juin 1994)

Jean-Michel Hufflen a présenté ses travaux au *workshop on Abstract Data Type* (mai 1994, Santa Margherita, Italie) et aux journées Cost 247 d'Evry (septembre 1994).

Jean-Yves Marion a présenté ses travaux au séminaire d'informatique du LABRI (Bordeaux, décembre 1994).

Abdelkader Dekdouk a participé aux aux journées Cost 247 de Brighton (Grande Bretagne, juillet 1994).

6.3 Organisation de colloques et de cours

Didier Galmiche et L. Wallen (Oxford University) ont organisé un *Workshop CADE* sur le thème *Proof search in type-theoretic languages* en juin 1994 [1].

Didier Galmiche a organisé la réunion du projet Inter-PRC « Modèles Logiques de la Programmation », les 28, 29 et 30 novembre 1994 à Nancy.

7 Publications

Livres et monographies

[1] D. Galmiche, L. Wallen (réd.), *CADE-12 Workshop on Proof search in type-theoretic languages*, Nancy, June 1994.

Thèses

[2] N. Brown, *Vérification et Mise en œuvre Distribuée des Programmes Unity*, thèse de doctorat, Université Henri Poincaré–Nancy 1, Octobre 1994.

[3] O. Festor, *Formalisation du comportement des objets gérés dans le cadre du modèle OSI*, thèse de doctorat, Université Henri Poincaré–Nancy 1, octobre 1994.

[4] D. Galmiche, *Des preuves aux programmes dans des logiques constructives*, Habilitation à diriger des recherches, spécialité informatique, Université Henri Poincaré, Nancy 1, Juin 1994.

[5] A. Mokkedem, *Virification et Raffinement de programmes parallhles dans une logique temporelle compositionnelle – Application au langage SDL*, thèse de doctorat, Institut National Polytechnique de Lorraine, Octobre 1994.

Articles et chapitres de livre

[6] L. Andrey, A. Schaff, «Description et prototypage d'entités de protocole OSI par des grammaires attribuées», *Réseaux et Informatique Répartie*, 1994.

[7] D. Galmiche, G. Perrier, «On Proof Normalization in Linear Logic», *Theoretical Computer Science 135*, 1, December 1994, p. 67–110.

[8] P. B. Ladkin, B. B. Simons, «Static Deadlock Analysis for CSP-Type Communications», *in : Responsive Computer Systems: Toward Integration of Fault-Tolerance and Real Time*, D. Fussell (réd.), Kluwer, 1994, p. 87–100.

Communications à des congrès, colloques, etc.

[9] F. Alexandre, K. Bsaïes, «A Methodology for Constructing Logic Programs», *in : Proceeding of the ICLP'94 Post-conference workshop on Application of Logic Programming to Software Engineering*, L.Sterling (réd.), p. 120–138, 1994.

[10] B. Amar, Y. Benoit, J. Guyard, J.-P. Jacquot, «Implication of Practitioners in a Post-Graduate Curriculum, A Successful Collaboration», *in : Proceedings 7th SEI CSEE Conference*, J. L. Diaz-Herrera (réd.), Lecture Notes in Computer Science, 750, Springer Verlag, p. 251–261, 1994.

[11] P. Bradford, J.-Y. Marion, L. Moss, «The additive fragment of Linear Logic is NC^1-complete», *in : Workshop on Logic complexity and computation*, D. Leivant (réd.), Indianapolis, October 1994.

[12] N. Brown, «Correctness-Preserving Transformations for the Design of Parallel Programs», *in : ECOOP '94 Workshop: Models and Languages for Coordination and Parallelism and Distribution*, 4-8 July 1994. Bologna, Italie.

[13] N. Brown, «A Sound Mapping from Abstract Algorithms to Occam Programs», *in : Transputer Research and Applications Conference*, H. Arabnia (réd.), IOS Press, p. 218–231, 1994.

[14] N. Brown, «Vérification et Mise en œuvre Distribuée des Programmes Unity», *in : Rencontres francophones du Parallélisme (RenPar '6)*, L. Bougé, M. Cosnard, P. Fraigniaud (réd.), Ecole Normale Supérieure de Lyon, p. 259–262, 1994.

[15] K. Bsaïes, «Implementing the Synthesis of Properties in Unfold/Fold Transformations», *in : International Conference on Programming Language Implementation and Logic Programming (PLILP'94)*, M. Hermenegildo (réd.), Lecture Notes in Computer Sciences, 844, Springer Verlag, 1994.

[16] K. Bsaïes, «A Schema Guided "Eureka Step" Discovery for Transforming Logic Programs», *in : 10^{th} Logic Programming Workshop, LPW'94*, N.E.Fichs, G.Gottlob (réd.), 1994.

[17] P. de Groote, «A CPS-Translation of the $\lambda\mu$-Calculus», *in : Proceedings of the 19th International Colloquium on Trees in Algebra and Programming (CAAP'94)*, S. Tison (réd.), Lecture Notes in Computer Science, 787, Springer Verlag, p. 85–99, 1994.

[18] P. de Groote, «On the Relation between the $\lambda\mu$-Calculus and the Syntactic Theory of Sequential Control», *in : Proceedings of the 5th International Conference on Logic Programming and Automated Reasoning–LPAR'94*, Lecture Notes in Computer Science, 822, Springer Verlag, p. 31–43, 1994.

[19] P. de Groote, «Strong Normalization in a Non-Deterministic Typed Lambda-Calculus», *in : Proceedings of the the 3rd international symposium on Logical Foundations of Computer Science–St Petersburg'94*, Lecture Notes in Computer Science, 813, Springer Verlag, p. 142–152, 1994.

[20] O. Festor, «OSI Managed-Objects Development with LOBSTERS», *in : Proceedings of the 5^{th} annual workshop on Distributed Systems and Operations Management*, Institut de Recherche en Informatique de Toulouse, Université Paul Sabatier, 1994.

[21] D. Galmiche, E. Boudinet, «Proof search for programming in Intuitionistic Linear Logic», *in : CADE-12 Workshop on Proof search in type-theoretic languages*, D. Galmiche, L. Wallen (réd.), p. 24–32, Nancy, France, 1994.

[22] D. Galmiche, O. Hermann, «Automated Inductive Proofs in a second order logical framework», *in : CADE-12 Workshop on Automation of Proof by Mathematical Induction*, A. Bundy, T. W. M. Rusinowitch (réd.), Nancy, France, 1994.

[23] D. Galmiche, G. Perrier, «Foundations of Proof Search Strategies Design in Linear Logic», *in : Logic at St Petersburg '94, Symposium on Logical Foundations of Computer Science*, Lectures Notes in Computer Science, 813, Springer Verlag, p. 101–113, 1994.

[24] D. Galmiche, «Canonical Proofs for Linear Logic Programming Frameworks», *in : ICLP'94 Workshop on Proof-theoretical extensions of logic programming*, A. Momigliano, M. Ornaghi (réd.), p. 2–9, Santa Margherita Ligure, Italy, 1994.

[25] D. Galmiche, «Computation with Proofs in Linear Logic», *in : International Workshop on Logical Theory of Program Construction*, Dagsthul Schloss, Germany, 1994.

[26] J.-P. Jacquot, A. Valdenaire, «Improving Legibility in Specification Languages: the Parsing Problem», *in : Proceedings RE'95 (to appear)*, P. Zave (réd.), IEEE, York, England, March 1995.

[27] J. P. Jacquot, «Programming through Disciplined Modification», *in : Proceedings International Conference on Software Maintenance*, H. A. Muller, M. Georges (réd.), IEEE Computer Society Press, p. 362–371, 1994.

[28] J.-P. Jacquot, «The Process of Teaching Process», *in : Proceedings of CSEE'95*, R. Ibrahim (réd.), SEI, Springer Verlag, New Orleans, Lo, March 1995.

[29] P. Kaboré, A. Schaff, «Du test de conformité des systèmes d'administration de réseaux», *in : Actes du deuxième Colloque Africain sur la Recherche en Informatique, CARI'94, Ouagadougou*, J. Tankoano (réd.), Inria-Orstom, p. 797–810, 1994.

[30] P. Kaboré, A. Schaff, «Objets de gestion de réseaux : une approche de test de conformité», *in : JISI'94, Informatique répartie, état de l'art et perspectives, Recueil des communications*, Tunis, p. 413–427, 1994.

[31] P. Ladkin, B. Simons, «Static Analysis of Multiway Synchronization», *in : CASCON '94: Integrated Solutions*, J. Botsford, A. Gawman, M. Gentlemen, E. Kidd, K. Lyones, J. Slonim (réd.), IBM Software Solutions Toronto Laboratory and National Research Council of Canada, p. 142–156, 1994.

[32] P. B. Ladkin, S. Leue, «Four Issues concerning the Semantics of Message Flow Graphs», *in : Participants' Proceedings of the Seventh International Conference on Formal Description Techniques: FORTE 94*, D. Hogrefe, S. Leue (réd.), University of Berne, Switzerland, p. 343–360, 1994.

[33] A. Lecomte, C. Retoré, «Pomset Logic as an alternative Categorial Grammar», *in : Proof Theory, Linear Logic and Categorial Grammar*,

M. Abrusci, C. Casadio, M. Moortgat (réd.), Dipartimento di Studi Filosofici ed Epistemologici, Università di Roma La Sapienza, DYANA-2: European Esprit Basic Research Project 6852, 1994.

[34] D. Leivant, J.-Y. Marion, «Predicative Recurrence over free algebras and Computability in Polynomial Space», *in : CSL '94*, Kazimierz, Poland, 1994.

[35] N. Lévy, G. Smith, «A Language-Independent Approach to Specification Construction», *in : Symposium on the Foundations of Software Engineering, New Orleans, USA*, S. issue of the ACM Software Engineering notes (réd.), 1994.

[36] D. Méry, D. Roegel, «Refining Formal Specifications to Get Efficient, Structured and Correct Concurrent Programs», *in : Proceedings Workshop High Performance Computing*, J. A. Z. et al. (réd.), IEEE/USP-LSI, p. 69–82, 1994.

[37] A. Mokkedem, D. Méry, «A Stuttering Closed Temporal Logic for Modular Reasoning about Concurrent Programs», *in : First International Conference on Temporal Logic*, D. M. Gabbay, H. J. Ohlbach (réd.), Lecture Notes in Artificial Intelligence, 827, Springer Verlag, p. 382–397, 1994.

[38] G. Smith, «A Development Framework for Object-Oriented Specification and Refinement», *in : Proceedings of TOOLS EUROPE '94*, B. Magnusson, B. Meyer, J.-M. Nerson, J.-F. Perrot (réd.), Prentice Hall, p. 173–183, 1994.

[39] G. Smith, «An Object-Oriented Development Framework for Z», *in : Proc. of the 8th Z User Meeting (ZUM'94), Cambridge*, J. Bowen, J. Hall (réd.), Workshops in Computing, Springer Verlag, p. 89–107, 1994.

[40] J. Souquières, M. Heisel, «How to Manage Formal Specifications? », *in : International Workshop on Logical Theory of Program Construction*, Dagsthul Schloss, Germany, 1994.

[41] S. Vorobyov, «Extensions of $F_{\leq}$ with Decidable Typing», *in : Proceedings of the 22nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages POPL'95*, 1995. To appear.

Rapports de recherche et publications internes

[42] D. Bert, M. Bidoit, C. Choppy, R. Echahed, J.-M. Hufflen, J.-P. Jacquot, M. Lemoine, N. Lévy, J.-C. Reynaud, C. Roques, F. Voisin, «Exemples, difficultés et points obscurs des fisc. Complément au rapport final de l'opération SALSA», *rapport de recherche n° 94–R–091*, CRIN, Nancy, may 1994.

[43] G. Bosch, J.-M. Hufflen, «ICARUS Tool—Installation Guide», *rapport de recherche n°Assist–011–R*, ICARUS, Nancy, July 1994.

[44] R. Darimont, J. Souquières, «An Extended Development Model To Guide Requirements Specifications», *rapport de recherche n°94-R-213*, CRIN, 1994.

[45] A. Dekdouk, N. E. Cadi, J. Hufflen, P. Kaboré, T. Lambolais, N. Lévy, A. Schaff, J. Souquières, «Aide à la construction et à la réutilisation de spécifications formelles», *rapport de recherche n°94-R-111*, CRIN, 1994.

[46] E. Dubois, J. Hagelstein, A. van Lamsweerde, F. Orejas, J. Souquières, P. Wodon, «Overview of ESPRIT Project 2537», *rapport de recherche*, ICARUS, November 1994.

[47] L. Franz, *Expérimentation de $\pi o \beta \lambda$, méthode de programmation orientée objet pour programme parallèle*, Mémoire, Université Henri Poincaré Nancy 1, Septembre 1994.

[48] O. Galibert, *Une extension objet C++ au paradigme Linda*, Mémoire, Institut National Polytechnique de Lorraine, Septembre 1994.

[49] F. Gauché, *Validation réduite de protocoles de communication*, Mémoire, Université Henri Poincaré–Nancy 1, 1994.

[50] J.-M. Hufflen, «GLIDER Version 1.2—Reference Manual», *rapport de recherche n°Assist–012–R*, ICARUS, Nancy, July 1994.

[51] J.-M. Hufflen, «Implementation Guide for the Tools of the Product Part. Application to the Tools around GLIDER», *rapport de recherche n°Assist–009–R*, ICARUS, Nancy, July 1994.

[52] P. Kaboré, A. Schaff, J. Souquières, «Aide à la Construction et à la Réutilisation de Spécifications Formelles», *rapport de recherche n°2*, CNET-CNRS, 1994.

[53] P. Kaboré, A. Schaff, «OSI management conformance testing : an approach for Managed Objects», *rapport de recherche n°94-R-236*, CRIN, 1994.

[54] P. B. Ladkin, B. B. Simons, «Static Analysis of Multiway Synchronization», *rapport de recherche n°94-R-079*, CRIN, 1994.

[55] P. B. Ladkin, «Analysis of a Technical Description of the Airbus A320 Braking System», *rapport de recherche n°94-R-101*, CRIN, 1994.

[56] N. Lévy, G. Smith, «A Language-Independent Approach to Object-Oriented Specification Construction», *rapport de recherche n°94-R-071*, CRIN, 1994.

[57] A. Mokkedem, D. Méry, « On using temporal logic for refinement and compositional verification of concurrent systems », *rapport de recherche n° 93-R-324*, CRIN, 1994.

[58] A. M. D. Moreira, P. B. Ladkin, R. G. Clark, « Formalizing OO Analysis with LOTOS », *rapport de recherche n° 94-R-078*, CRIN, 1994.

[59] C. Retoré, « Pomset Logic », *rapport de recherche*, Inria, 1994.

[60] S. Vorobyov, « Hierarchies of decidable extensions of bounded quantification », *rapport de recherche n° 94-R-120*, CRIN, 1994.

[61] S. Vorobyov, « Theory of finite trees revisited: Application of model theoretic algebra », *rapport de recherche n° 94-R-135*, CRIN, 1994.

Divers

[62] J.-M. Antoine, « Evolution de la Gestion des Bibliographies et Etude de l'Interfaçage entre Logiciels », Mémoire CNAM, Centre de Nancy, juillet 1994.

[63] G. Bosch, J. Souquières, « Prototype Assistant: its function and its design », Esprit Project 2537 ICARUS, Deliverable #48, January 1994, Development Editor User Manual.

[64] M. Bouzid, P. Ladkin, « Simple Reasoning with Time-Dependent Propositions », 1994, Internal Note.

[65] R. Darimont, N. Levy, J. Souquières, « The Process Model: Extensions and Enrichments, Modelling the Rationale », Task Process, Process-036-R, January 1994.

[66] O. Festor, « Gestion de réseaux OSI », Actes de l'Ecole d'été *Réseaux de communication et techniques formelles*, Ecole Nationale Supérieure de Télécommunication, Paris, 1994.

[67] J.-M. Hufflen, N. Lévy, « Prototypes: Their Functions and their Design », Icarus Task Tools, January 1994.

[68] P. Ladkin, L. Lamport, B. Olivier, D. Roegel, « Lazy Caching: An Assertional View », 1994, Internal Note.

[69] A. Schaff, L. Andrey, « Grammaires attribuées et protocoles », Actes de l'Ecole d'été *Réseaux de communication et techniques formelles*, Ecole Nationale Supérieure de Télécommunication, Paris, 1994.

[70] A. Valdenaire, « Lisibilité dans les langages de spécification », Mémoire CNAM, Centre de Nancy, juillet 1994.

8 Abstract

The Prograis project aims at understanding and expressing the process of software development. More precisely, we are focusing our attention on the five following topics:

- the modelling of specification and program developments,
- specification and validation of protocols,
- formal methods in the development of parallel and distributed systems.
- interactive synthesis of logic programs from formal specifications,
- the application of proof-theory and type theory to programming.

Table des matières