PROJET CROAP

Conception et Réalisation d'Outils d'Aide à la Programmation

Localisation : *Sophia-Antipolis*

Mots-clés : environnement de programmation, génie logiciel, fiabilité du logiciel, programmation, sémantique, sémantique naturelle, Centaur, typage, langage de programmation, transformation de programme, parallélisme, preuve de programme, programmation à objets, édition structurée, interface homme-machine, mise au point de programme.

1 Composition de l'équipe

Responsable scientifique

Yves Bertot, Chargé de Recherche

Responsable permanent

Laurence Rideau, Chargé de Recherche, également responsable de la Valorisation de la Recherche en Industrie pour l'INRIA Sophia Antipolis

Assistante de projet

Sandrine Chevris, Technicien de la recherche

Personnel Inria

Isabelle Attali, Chargé de Recherche
Joëlle Despeyroux, Chargé de Recherche
Thierry Despeyroux, Chargé de Recherche
Gilles Kahn, Directeur de Recherche, à temps partiel dans le projet
Francis Montagnac, Ingénieur de Recherche
Valérie Pascual, Chargé de Recherche
Laurent Théry, Chargé de Recherche

Ingénieur expert

Janet Bertot

Chercheurs post-doctorants

Lena Magnusson, boursière du gouvernement suédois, jusqu'au 29/2/96
Savi Maharaj, boursière ERCIM, du 1/1/96 au 30/6/96

Chercheurs doctorants

Yann Coscoy, boursier DRET
Sidi Ould Ehmety, boursier CIES
Ranan Fraer, allocataire MENESR
Romain Guider, allocataire MENESR
Pierre Leleu, fonctionnaire ENPC
Olivier Pons, boursier INRIA
Christophe Roudet, allocataire MENESR

Stagiaires

Sylvain Lippi, DEA
Mouloud Bourbel, DEA
Lina Gegetieh, DEA, en commun avec le projet Safir

Collaborateurs extérieurs

Denis Caromel, Maître de Conférence, I3S-CNRS – UNSA
Anne-Marie Déry, Maître de Conférence, ESSI
André Hirschowitz, Directeur de Recherche, CNRS

2 Présentation du projet

L'objectif du projet est d'étudier les outils nécessaires pour produire du logiciel complexe et fiable, et la collaboration de ces outils dans des environnements de travail conviviaux et efficaces. La réalisation de cet objectif passe par trois thèmes principaux : la réalisation d'outils adaptés à des styles de programmation variés, l'étude formelle des langages de programmation et la maîtrise des fondements théoriques communs à de larges classes de ces langages.

Notre étude de différents styles de programmation se traduit par la réalisation d'environnements de programmation spécialement adaptés à plusieurs langages, chacun représentant une grande classe de styles et de langages de programmation. Ainsi, nous étudions des outils pour le langage ML (programmation fonctionnelle), les langages Eiffel et Java (programmation à objets et concurrence), le langage Sisal (programmation parallèle à flots de données) et le langage C (programmation impérative). Pour chaque langage, nous abordons des outils différents de l'environnement de programmation. Ainsi, nous étudions l'exécution symbolique d'Eiffel et le typage pour le langage ML. De façon plus prospective, nous étudions également des environnements spécialisés pour la conception conjointe de logiciel et de matériel ou pour la preuve de résultats mathématiques. Sans être directement des travaux sur des langages de programmation, ces expériences nous permettent d'aborder l'intégration d'outils dans des environnements hétérogènes, où les données manipulées sont également très diverses. Surtout, les travaux sur un environnement de preuve mathématique permettent également d'expérimenter avec le développement de programmes prouvés corrects.

Pour étudier les langages de programmation, nous utilisons notre propre méthode, dite *Sémantique Naturelle*, pour laquelle nous développons un laboratoire d'expérimentation très complet autour du formalisme TYPOL. Nous nous efforçons d'avoir des implantations réalistes pour ce formalisme. Ces implantations nous permettent de décrire formellement des outils de programmation tels que des compilateurs ou des interprètes pour des langages de programmation très variés.

Pour assurer la validité et la cohérence des différentes implantations du formalisme TYPOL, ainsi que pour en permettre l'extension à de nouvelles approches de spécification, il est également nécessaire d'inclure dans notre programme de recherche une étude théorique de ses fondements logiques. Cette étude théorique doit permettre d'intégrer au langage de spécification des paradigmes évolués comme la syntaxe abstraite d'ordre supérieur et de fournir un véritable atelier pour l'étude formelle des langages de programmation.

L'ensemble des expériences réalisées dans le projet utilise les compétences et les outils réunis dans un système de génération d'environnements de programmation interactifs, le système *Centaur*. Ce système est donc à la fois la cristallisation de notre savoir-faire et la panoplie d'outils nécessaire à nos nouvelles expériences.

Comme point marquant pour cette année, nous pouvons remarquer notre participation à la conclusion de la phase 1 du projet de collaboration Génie, qui a reçu une évaluation très favorable. Notons également le démarrage d'une action dans le cadre du G.I.E. Dyade et la déclaration d'intérêt de Novell, un des acteurs du marché des logiciels professionnels américains, pour nos travaux sur les langages à objets et parallèles.

3 Actions de recherche

3.1 Composants élémentaires d'environnements interactifs

3.1.1 Edition syntaxique guidée par menus

Participante : Valérie Pascual

Mots-clés : édition structurée, interface homme-machine, transformation de programme.

L'édition syntaxique guidée par des menus utilise des menus proposant des schémas d'arbres à insérer ou des transformations à effectuer sur les programmes. Ce mécanisme est un composant standard de Centaur qui permet de fournir des aides aux utilisateurs débutants et des outils évolués aux utilisateurs experts. Ces outils sont couramment utilisés pour l'édition de fichiers Lisp sous Centaur et dans l'environnement CtCoq. Le langage utilisé pour spécifier ces menus, TransForm, est un langage de règles de réécriture simple.

Diverses améliorations de ce composant ont été réalisées, le rendant plus efficace lors de l'affichage de menus hiérarchiques de taille importante. Jusqu'à présent, la structure des menus était calquée sur la syntaxe abstraite du langage. Des expériences ont été faites pour structurer les menus en définissant de nouveaux ensembles d'opérateurs et donc sans s'appuyer sur les phyla du langage. Cela permet en particulier de limiter les modifications du menu liés aux modifications de l'expression courante. Le problème qui se pose maintenant est celui de définir de « bons » menus pour un langage donné.

3.1.2 Spécifications de transformations de programmes

Participants : Isabelle Attali, Christophe Roudet

Mots-clés : transformation de programme, sémantique.

Nous souhaitons concevoir un formalisme de description sémantique spécialisé dans les transformations de programmes, qui travaille directement au niveau structures et qui soit adapté aussi bien pour une exécution purement automatique que pour une exécution interactive comme dans l'édition dirigée par menus, avec des performances acceptables. Enfin ce langage doit être assez déclaratif pour se prêter à la preuve de correction de transformations.

Nous avons conçu un langage appelé TrfL (prononcer « trèfle ») qui s'inspire à la fois des langages issus de Centaur et des langages existants dans la littérature sur les transformations de programmes. Pour l'instant, le langage TrfL est défini syntaxiquement ; la définition sémantique est en cours : la sémantique statique (notamment la vérification des schémas) est quasiment terminée et il reste à décrire la sémantique dynamique du langage. Ces travaux sont décrits dans [106]. Enfin, nous souhaitons intégrer ces nouveaux développements au sein du système Centaur (notamment avec les fonctionnalités de TransForm).

3.1.3 Accélérateurs clavier

Participante : Valérie Pascual

Mots-clés : interface homme-machine.

L'utilisation d'accélérateurs au clavier pour invoquer des commandes permet d'éviter de trop grands déplacements de la souris vers les menus, vers des boutons ou vers les barres de défilement des fenêtres. Ces clés sont facilement redéfinissables et l'utilisateur peut ainsi paramétrer son environnement d'édition selon ses habitudes.

3.1.4 Développement d'un outil de débogage pour PPML

Participants : Yves Bertot, Laurence Rideau

Mots-clés : environnement de programmation, interface graphique, mise au point de programme, interface homme-machine.

Dans Centaur, l'affichage des programmes est calculé par une machine d'affichage utilisant des règles d'affichages PPML (Pretty-Printer Meta-Language). Le choix de la règle à appliquer est basé sur du pattern matching : si plusieurs règles sont applicables, c'est la première qui est utilisée. En pratique, il devient difficile de savoir quelles règles sont utilisées pour une expression donnée.

Nous avons développé un outil permettant de montrer interactivement la ou les règle(s) utilisée(s) pour l'affichage d'un sous-arbre donné. Cet outil de débogage de spécifications PPML peut aussi être utilisé pour aider l'utilisateur final à décrire ses propres règles d'affichage : pour un sous arbre donné, le système lui montre la règle utilisée, il peut la copier, l'éditer pour la rendre spécifique à son application puis l'utiliser dans son contexte de travail.

3.1.5 Formateur incrémental graphique

Participant : Laurent Théry

Mots-clés : interface homme-machine, Java, Dyade.

Le travail de réimplantation de la partie graphique de l'afficheur Figue a été mené à bien. Le langage choisi pour cette réimplantation est C++. Pour la partie affichage il a été fait usage de la boîte à outils graphiques IlogViews. Le moteur d'affichage obtenu opère sur une structure de boîtes imbriquées. Une caractéristique importante du formatage est son *incrémentalité*. Une fois une structure affichée, toute modification sera répercutée au niveau de l'affichage par un recalcul minimal du formatage.

Le système Centaur manipule des arbres de syntaxe abstraite. Il utilise le langage PPML pour spécifier le lien entre syntaxe abstraite et structure de boîtes. Un prototype d'un nouveau compilateur PPML construisant des structures de boîtes du nouveau formateur a été entrepris. Il permet d'intégrer dans un environnement distribué hétérogène les outils existants de manipulation d'arbres Centaur avec le nouveau moteur d'affichage.

Enfin le phénomène WWW sur Internet semble constituer un terrain idéal pour la diffusion de nos résultats. Il a été donc entrepris une expérience de faisabilité du transfert de nos développements vers le langage Java. Ceci a conduit à un premier prototype. Un des objectifs de notre participation dans le GIE Dyade est de valoriser cette première expérience.

3.1.6 Documentation et aide aux utilisateurs

Participant : Thierry Despeyroux

Mots-clés : interface homme-machine, HTML.

Le système Centaur utilise déjà des facilités hypertextuelles permettant à l'utilisateur de faire le lien entre un message d'erreur ou ses composants et le texte source d'un programme.

Nous voulons élargir ces possibilités en permettant la liaison entre un environnement de programmation développé sous Centaur et la documentation qui lui est associée. Cela a été réalisé dans le cadre du système de messages d'erreur de Centaur qui permet dorénavant à l'utilisateur de se référer directement à la documentation HTML correspondant aux messages, et à partir de ces points d'entrée de naviguer à loisir dans la documentation. Nous utilisons pour cela la possibilité d'utilisation de Netscape en mode serveur.

Un exemple complet de documentation HTML, généré à partir d'un source LaTeX, a été développé pour le langage de spécification de syntaxe abstraite AS.

3.2 Environnements de programmation

3.2.1 Parallélisation de programmes à objets

Participants : Isabelle Attali, Denis Caromel(projet Sloop), Sidi Ould Ehmety, Sylvain Lippi

Mots-clés : sémantique, programmation à objets, héritage, parallélisme, TLA.

Nous étudions, à travers les langages Eiffel et Eiffel//, comment paralléliser des programmes à objets. Nous avons tout d'abord décrit la sémantique dynamique d'Eiffel en sémantique naturelle. La contribution majeure de cette sémantique repose sur la définition formelle de l'héritage et de la liaison dynamique en présence de renommage et redéfinition [93].

Nous avons ensuite modifié la sémantique d'Eiffel pour obtenir la sémantique d'Eiffel// (Eiffel parallèle). Les deux langages partagent la même syntaxe, seules leurs sémantiques diffèrent : les instances de classes héritant d'une classe prédéfinie « Process » deviennent des objets actifs (processus) et tout appel de routine sur un processus devient systématiquement un appel asynchrone.

La sémantique opérationnelle d'Eiffel// est décrite en termes de systèmes de transitions dont les états représentent les configurations globales du système. L'exécution d'un programme est modélisée par une séquence de configurations reliées par des transitions à partir d'une configuration initiale donnée. La sémantique d'un programme est donnée par le système de transitions représentant toutes ses exécutions possibles.

Cette sémantique a été définie dans le cadre de Centaur, et un environnement interactif pour le langage Eiffel// a ainsi pu être produit. Celui-ci permet actuellement la visualisation graphique des systèmes d'acteurs générés par Eiffel// [96], notamment on peut visualiser les références entre les différents objets (actifs et passifs), les valeurs des attributs, les requêtes en attente de service, l'activité de chaque objet, l'exécution concurrente des objets, et enfin l'attente par nécessité.

Il faut noter que cet environnement de visualisation est complètement adapté à la version séquentielle du langage, puisque la sémantique d'Eiffel// inclut celle d'Eiffel.

L'existence d'une spécification formelle pour Eiffel et Eiffel// permet d'étudier la parallélisation de programmes Eiffel et la preuve de leur correction. Les techniques classiques de vérification basées sur les systèmes de transition se heurtent à un problème d'explosion combinatoire ; il est donc nécessaire de considérer des méthodes de vérification d'ordre partiel, avec l'assurance que cet ordre partiel est compatible avec l'ensemble des exécutions possibles.

Enfin, nous avons exploré d'autres modélisations pour le langage Eiffel// afin de montrer l'équivalence de programmes séquentiels et parallèles ; en particulier, nous avons étudié comment modéliser des programmes Eiffel// en TLA+ [104]. La modélisation de l'arbre binaire de recherche est relativement compliquée car de bas niveau. Il a été possible de prouver l'équivalence entre la version séquentielle et la version parallèle de l'arbre binaire de recherche. La méthodologie employée repose sur la définition d'invariants (typage, contrôle des données, structures de données, requêtes) et sur une structuration fine des propriétés d'invariance (environ 150 cas différents à traiter). Il reste à généraliser la traduction à l'ensemble du langage Eiffel// au lieu d'un exemple particulier.

3.2.2 Transformations de programmes Sisal

Participants : Isabelle Attali, Denis Caromel(projet SLOOP), Romain Guider

Mots-clés : sémantique, programmation fonctionnelle, optimisation.

Le langage SISAL (Streams and Iteration in a Single Assignment Language), fortement typé, applicatif et à assignation unique, est particulièrement utilisé sur des architectures multiprocesseurs ou data-flow. Les applications privilégiées sont celles du calcul scientifique où l'on sait exhiber une régularité des données pour l'optimisation et la parallélisation. Malheureusement, il n'existe pas d'autre description précise de ces optimisations que dans le compilateur (appelé OSC – Optimizing Sisal Compiler), ce qui ne facilite ni l'extension du langage ni l'amélioration de la panoplie d'optimisations. Une nouvelle version du langage est en cours de définition pour ajouter des fonctions d'ordre supérieur, du polymorphisme, de la surcharge et une forme limitée d'inférence de type, et nous voulons participer à cet effort en fournissant une base formelle plus sûre.

Nous avons décrit la sémantique dynamique de Sisal [94], dont les points forts sont la description des expressions multivaluées, des fonctions d'ordre supérieur, des règles de visibilité, des boucles, des tableaux et des séquences. Nous avons ensuite décrit dans [95] le format intermédiaire de représentation de graphes IF1 utilisé par le compilateur et spécifié formellement la traduction de Sisal vers ce format et certaines des optimisations classiques sur ce format présentes dans le compilateur OSC. En utilisant des spécifications formelles et des transformations que l'on pourra prouver, nous espérons ainsi établir la base d'une définition et d'une implantation de compilateurs validés pour la nouvelle version de SISAL.

3.2.3 Environnement de programmation pour le langage ML

Participants : Laurence Rideau, Laurent Théry

Mots-clés : environnement de programmation, CRISTAL, explication d'inférence de types, ML, programmation fonctionnelle.

Les années précédentes, nous avons entamé le développement d'outils spécialement adaptés au langage ML en développant un éditeur structuré comprenant une interaction sophistiquée avec le vérificateur de type. Cette année, nous avons étendu ce travail aux explications de types.

La relative simplicité du vérificateur de type de CamlLight (langage développé dans le projet CRISTAL) a permis d'entreprendre le développement de variantes du vérificateur de type dédiées à l'utilisation interactive. D'une part, le diagnostic des erreurs de type permet de montrer interactivement les différents endroits du programme intervenant dans l'apparition d'une erreur. D'autre part, lors de l'inférence de types, des informations sont conservées afin de pouvoir ensuite, à la demande, fournir une explication du type d'une expression du programme. Cette explication est affichée dans un langage pseudo-naturel. Elle permet en particulier de montrer dans le programme source les différents endroits du programme qui ont permis d'inférer le type expliqué.

Parallèlement à ce développement, un gros travail a été fait pour augmenter la robustesse de notre environnement afin d'en permettre une utilisation fiable.

Dans l'avenir, ce travail va déboucher sur un environnement de travail pour la programmation en ML. Il doit également permettre de nouvelles expériences pour découvrir des utilisations originales de l'information de type pour l'aide à la construction de programmes.

3.2.4 Environnement pour Klone sous Centaur

Participante : Valérie Pascual

Mots-clés : environnement de programmation.

Le langage Klone est un langage de la famille Lisp développé par Bull. Ce langage est particulièrement adapté pour l'interfaçage avec des bibliothèques C ou C++, comme par exemple la librairie Motif. C'est donc un candidat idéal pour la fabrication d'interprètes embarqués fournissant les avantages conjoints

de bibliothèques compilées (efficacité) et d'un langage interprété (souplesse d'utilisation). Le développement de l'environnement pour ce langage s'est poursuivi dans le cadre du GIE Dyade.

3.2.5 Environnement pour Java sous Centaur

Participants : Thierry Despeyroux, Laurent Théry

Mots-clés : environnement de programmation, Java.

Le langage Java a servi de test pour les nouveaux formalismes de spécification. Sa syntaxe a été spécifiée à l'aide de AS, PPML et un parseur DCG (Prolog) généré à l'aide du nouveau langage de spécification de syntaxe concrète. Un parseur a aussi été spécifié en Metal pour pouvoir plus facilement le diffuser, en attendant une meilleure stabilité du nouveau formalisme.

3.2.6 Tranches de dépendance dans les programmes annotés

Participants : Ranan Fraer, Yves Bertot

Mots-clés : environnement de programmation, spécification formelle, vérification de programme, fiabilité du logiciel, analyse statique.

Le calcul de tranches de dépendance (*program slicing*) est une technique de plus en plus répandue dans les outils modernes de mise au point. Une tranche de dépendance par rapport à une variable x et un point donné dans un programme est formée par les instructions du programme qui affectent la valeur de x en ce point.

Il serait donc intéressant d'intégrer un calcul de tranches de dépendance dans un environnement de mise au point de programmes vérifiés. L'outil central, dans ces environnements, est un générateur de conditions de vérification à partir d'un programme annoté avec des assertions sur la valeur des variables du programme. Ces conditions logiques sont ensuite vérifiées par des outils de preuve automatique.

Dans le cas des programmes annotés, le critère d'intérêt n'est plus la valeur d'une variable mais la validité d'une assertion logique en un point du programme. Plus exactement, si cette assertion est fausse cela se traduit par l'insatisfiabilité d'une condition de vérification, et on aimerait isoler les instructions à partir desquelles cette condition a été générée.

La mise en pratique de cette idée est basée sur une extension du calcul des origines sur les conditions de vérification [99]. Afin de trouver les instructions qui contribuent effectivement à la génération d'une condition, on considère la réunion des origines de tous les sous-termes de cette condition. Cette approche produit seulement une approximation grossière, mais elle peut être raffinée afin de calculer avec précision un chemin d'exécution dans le programme (les instructions non significatives en moins) associé à la condition. Un aspect intéressant de ce travail est l'utilisation du générateur de conditions pour le calcul des tranches, alors que la méthode usuelle est basée sur des graphes de dépendance.

3.3 Environnements de preuve

Participants : Janet Bertot, Yves Bertot, Yann Coscoy, Gilles Kahn, Laurence Rideau, Francis Montagnac, Olivier Pons

Mots-clés : interface homme-machine, COQ, typage, déduction automatique, logique, Génie.

L'étude et la réalisation d'un environnement de travail, appelé CtCoq, pour le système Coq produit par le projet COQ de l'INRIA Rocquencourt et de l'Ecole Normale Supérieure de Lyon, se situe dans le prolongement de notre étude des environnements de programmation, avec ici l'objectif de la programmation certifiée. Le travail en cours dans ce domaine continue la consolidation entreprise l'année précédente et l'étude de nouveaux aspects relatifs au traitement de grandes preuves. Ces aspects ont été évalués par nos collaborateurs de Dassault-Aviation dans le cadre du projet Génie et ont montré que l'interface pouvait apporter le confort nécessaire à l'entreprise de grands développements formels.

Cette collaboration a également fourni de nouveaux sujets d'investigation comme la manipulation directe de formules algébriques décrite en 3.3.1.

3.3.1 Déplacements d'expressions algébriques

Participant : Yves Bertot

Lorsque le raisonnement formalisé en Coq comporte des nombres entiers ou réels, de nombreuses étapes élémentaires de raisonnement se font par réécriture. Lorsque ce type de raisonnement n'est pas effectué par une procédure automatique, la mise en œuvre de la réécriture est particulièrement fastidieuse, car il faut à la fois déterminer le théorème à appliquer et la position où la transformation doit avoir lieu. Nous avons développé un outil qui permet d'interpréter les mouvements de l'utilisateur avec la souris comme des déplacements de données dans les formules mathématiques, permettant ainsi une augmentation spectaculaire de la vitesse d'interaction. Cet outil est également extensible pour s'adapter rapidement au domaine choisi par l'utilisateur. Il complète l'outil de *preuve par sélection* déjà mis au point au cours des années précédentes, et qui ne s'adaptait pas bien à la réécriture.

3.3.2 Gestion de plusieurs tâches simultanées

Participante : Janet Bertot

L'organisation initiale de l'interface homme-machine impose pratiquement à l'utilisateur d'effectuer une preuve à la fois. Nous avons étendu cette organisation pour gérer plusieurs fenêtres d'interaction. Ceci permet d'ouvrir un espace de travail pour effectuer des développements annexes au développement principal sans abandonner ce développement principal. Une telle ouverture demande d'une part de reconsidérer l'ensemble des méthodes utilisées pour assurer la cohérence entre l'état du système tel qu'il est présenté à l'utilisateur et l'état effectif du moteur de preuve, d'autre part d'assurer le bon routage des données pour que chaque fenêtre d'interaction représente une « mini-session » avec CtCoq.

3.3.3 Approche structurée du retour-arrière

Participants : Olivier Pons, Laurence Rideau, Yves Bertot

La commande `Undo` fournie par Coq implémente une notion de retour en arrière historique. En particulier, il est impossible d'annuler l'effet d'une commande de preuve sans annuler l'ensemble des commandes effectuées ultérieurement. En fait, il est possible d'organiser les commandes effectuées lors d'une preuve en une structure arborescente de sorte que chaque commande ne dépende effectivement que des commandes apparaissant plus haut dans la structure arborescente. Nous avons développé des outils qui permettent de naviguer dans cette structure arborescente et de défaire une commande et uniquement les commandes qui en dépendent (retour arrière logique). Ces deux aspects sont importants lorsque l'on veut effectuer des preuves de grande taille. D'une part, la navigation dans la structure arborescente permet d'avoir plus facilement une vue d'ensemble sur une preuve, d'autre part le retour arrière logique permet de gagner du temps dans la réparation des erreurs.

3.3.4 Description textuelle des preuves

Participants : Yann Coscoy, Gilles Kahn

Ce travail sur la production de paraphrases en langue naturelle pour les preuves est la poursuite d'un effort déjà présent au cours des deux années précédentes. Le travail de cette année a permis de traiter de manière plus uniforme les définitions inductives et les définition non inductives (δ-définitions) : dans les termes de preuve du calcul des constructions, les expansions de définitions inductives sont toujours explicites tandis que les expansions de définitions non inductives sont toujours muettes. Le travail effectué permet d'avoir des expansions muettes pour des définitions inductives et réciproquement

des expansions explicites pour des δ-définitions. Ce travail permet donc d'estomper une distinction entre deux modes de définitions qui n'a parfois pas beaucoup de sens pour le lecteur. Un deuxième travail a permis de rendre plus uniformes les traitements répétitifs effectués sur des opérateurs logiques associatifs. Lorsqu'un opérateur logique (comme le «et» logique) est associatif, il est naturel pour l'implémentation du système de preuve de le manipuler comme un opérateur binaire. En revanche, il est naturel pour le mathématicien de s'abstraire de la structure binaire et de considérer cet opérateur comme prenant un nombre arbitraire d'arguments. L'outil de description textuelle des programmes suit maintenant cette démarche.

Sur un autre plan, l'outil d'affichage de preuve est maintenant intégré directement dans le système Coq et fait partie de sa distribution, ce qui permet d'élargir la communauté d'utilisateurs. Enfin, l'adaptation de l'outil de production textuelle à un autre langage a été effectuée et le Français est maintenant un langage possible.

3.3.5 Un environnement léger pour Alf

Participante : Lena Magnusson

Nous avons travaillé à la construction d'un environnement de travail léger pour Alf, qui consiste en quelques centaines de lignes de configuration pour l'éditeur de texte Xemacs. Cette interface a quelques traits en commun avec l'interface graphique de Alf et avec CtCoq : affichage structuré sensible à la souris, séparation des entrées et sorties.

3.4 Sémantique et preuves

Mots-clés : fiabilité du logiciel, sémantique, sémantique naturelle, COQ.

3.4.1 Aide au développement de preuves en sémantique naturelle

Participants : Salimeh Benhia, Thierry Despeyroux, Delphine Terrasse (du projet MEIJE)

Dans la lignée des travaux de thèse de Delphine Terrasse, nous nous sommes interessé à faciliter la mécanisation des preuves en Sémantique Naturelle en utilisant le système Coq. Les données fournies au système Coq étant obtenues par traduction des spécifications de syntaxe abstraite et des règles sémantiques en Coq, il est nécessaire de fournir à l'utilisateur un moyen de raisonner et de piloter la preuve au niveau des règles sources afin de comprendre les erreurs ou les défauts de cette sémantique révélés lors de l'essai de construction d'une preuve. D'autre part, il est nécessaire d'exhiber des tactiques de preuve spécifiques à la Sémantique Naturelle.

3.4.2 Environnement Centaur pour Z

Participante : Savi Maharaj

La notation Z est une notation mise au point pour spécifier formellement des systèmes informatiques. Nous avons utilisé le système Centaur pour donner une description formelle de de cette notation. Cette formalisation fournit la donnée de base pour décrire des analyses sur des documents écrits dans ces notations.

3.4.3 Développement formel en B d'un algorithme de graphes

Participant : Ranan Fraer

Mots-clés : vérification de programme.

La méthode B associe une notation proche de Z et un ensemble d'outils pour faciliter le développements de programmes répondant à une spécification formelle. Les études de cas disponibles sur cette méthode

sont souvent tirées des applications industrielles de taille réelle, et donc difficilement accessibles aux universitaires. Ainsi nous avons choisi d'illustrer par un exemple l'utilisation de la méthode B sur un algorithme bien connu dans la théorie des graphes [98]. Il s'agit de l'algorithme de Kruskal pour trouver un arbre recouvrant de coût minimum dans un graphe connexe.

Notre choix a été motivé également par les structures de données non-triviales sous-jacentes à cet algorithme : des queues de priorité implémentées commes des « min-heaps » et des représentations arborescentes des ensembles disjoints utilisés dans l'algorithme UNION-FIND de Tarjan. Ceci nous sort du cadre usuel des entiers et booléens manipulés dans les applications industrielles de B.

3.5 Algorithmes de calcul formel certifiés

Participants : Lina Gegetieh, Laurent Théry, Loïc Pottier (du projet SAFIR)

Mots-clés : vérification de programme, calcul formel, SAFIR.

Un travail a été commencé en collaboration avec Loïc Pottier du projet SAFIR sur le développement d'une bibliothèque certifiée de calcul formel. Il s'agit de prouver à l'aide du système Coq un certain nombre d'algorithmes classiques de calcul formel pour ensuite en extraire des programmes CAML dont la correction est certifiée. Dans ce cadre, une stagiaire de DEA, Lina Gegetieh, a travaillé sur la réalisation d'un algorithme de division de polynômes.

De plus il fournit un exemple d'utilisation de CtCoq comme plate-forme de développement de programmes certifiés.

3.5.1 Mécanisation des preuves dans les langages parallèles

Participants : Mouloud Bourbel, Joëlle Despeyroux, Eric Madelaine du projet MEIJE

Mots-clés : parallélisme, vérification de programme.

Les méthodes de vérification actuelles pour les langages parallèles et les systèmes distribués se divisent en deux grandes familles : les méthodes automatiques, souvent basées sur une représentation des programmes par des machines d'états finis, et les méthodes déductives traditionelles, utilisées dans des systèmes de preuve de théorèmes interactifs.

Ces deux familles de méthodes ont donné des résultats intéressants dans un certain nombre d'études de cas. Individuellement, aucune ne permet d'aborder des exemples de grande taille, ni ne couvre toutes les facettes des langages parallèles (structures infinies, passage de données, temps-réel, etc.). On veut donc pouvoir associer des techniques algorithmiques aux techniques déductives.

Une première expérience, menée dans le cadre d'un stage de DEA, a consisté à implanter en Coq le système d'inférence sous-jacent à Ecrins et à appliquer le système obtenu à la construction de preuves pour de petits exemples.

3.6 Vers un système de spécification d'ordre supérieur : théorie et pratique

Participants : Joëlle Despeyroux, Thierry Despeyroux, André Hirschowitz, Pierre Leleu

Mots-clés : sémantique, sémantique naturelle, théorie des types, COQ.

La *syntaxe abstraite d'ordre supérieur* est une technique de représentation des langages de programmation qui permet de formaliser les notions de variables liées et de substitution d'un langage. Elle permet également de formaliser les *side conditions* et les changements de contexte dans les règles d'inférence. Elle augmente donc nettement le niveau d'abstraction, et le confort, à la fois de la description des langages de programmation, et des preuves sur ces langages. Par ailleurs, la définition d'un type inductif fournit des schémas de récurrence et des principes d'induction qui permettent de faire des preuves générales sur des données de ce type. En particulier, si l'on représente la syntaxe d'un langage par un type inductif, on peut utiliser les principes associés pour faire des preuves sur la sémantique de ce langage.

Mais une syntaxe abstraite d'ordre supérieur, définie de manière naïve, ne forme pas un type inductif. Nous travaillons sur plusieurs solutions à ce problème, qui était ouvert depuis longtemps.

Ces travaux s'inscrivent dans une action à long terme, dont le but est d'élargir le champ d'application du système Centaur aux langages de programmation décrits à l'ordre supérieur.

3.6.1 Syntaxe abstraite d'ordre supérieur : spécification

Participants : Thierry Despeyroux, André Hirschowitz

Dans un travail pratiquement achevé, nous donnons une définition formelle (catégorique) de la syntaxe abstraite d'ordre supérieur, et des notions correspondantes d'induction et de récursion. Bien entendu, notre définition de syntaxe abstraite est compatible avec les systèmes existants, en particulier LF. Ce travail assure les fondements de l'implémentation décrite en 3.6.5.

3.6.2 Sémantique d'ordre supérieur

Participants : Thierry Despeyroux, André Hirschowitz

Dans un travail en cours, prolongeant celui de Joëlle Despeyroux et André Hirschowitz présenté en 1994, nous définissons la *sémantique d'ordre supérieur*. En effet notre thèse est que la sémantique telle qu'on peut l'écrire en LF ou en λ-prolog, n'est pas vraiment le formalisme adapté à la syntaxe d'ordre supérieur : comme nous l'expliquons dans le travail précédent, les sous-arbres d'un arbre de syntaxe abstraite d'ordre supérieur représentent des fonctions d'un nombre d'arguments qui peut croître à mesure qu'on s'enfonce dans l'arbre. Nous pensons que la sémantique adaptée doit manipuler des jugements dont le sujet est aussi une fonction d'un nombre variable d'arguments. Nous savons déjà définir cette sémantique d'ordre supérieur, pour laquelle nous étudions une formalisation en Coq et essayons de trouver une syntaxe concrète agréable. Le but final de cette recherche est de spécifier et implémenter un TYPOL d'ordre supérieur.

3.6.3 Récursion primitive

Participants : Joëlle Despeyroux, André Hirschowitz, Pierre Leleu

La syntaxe abstraite d'ordre supérieur, définie de manière standard, est incompatible avec les fonctions primitives récursives, ou avec les preuves par induction. Afin de réconcilier les deux paradigmes, il faut restreindre l'espace des fonctions utilisé pour décrire les arbres de syntaxe abstraite aux fonctions « paramétriques », c'est-à-dire uniquement construites avec les constructeurs du langage représenté.

Dans un premier pas vers la définition d'un système de type qui incorpore la méthodologie du système LF dans des systèmes comme Coq ou Alf, Joëlle Despeyroux, Frank Pfenning et Carsten Schürmann (Carnegie Mellon University) ont proposé [100] une extension d'un λ-calcul modal simplement typé par des opérateurs d'itération et de raisonnement par cas, étendus aux objets fonctionnels. L'opérateur $\Box$ de la logique modale est utilisé pour décomposer l'espace des fonctions primitives récursives $A \Rightarrow B$ en $\Box A \rightarrow B$, où $\rightarrow$ désigne l'espace des fonctions paramétriques. Ce système est une extension conservative du λ-calcul simplement typé, ce qui signifie qu'il préserve l'adéquation des représentations des langages utilisant la syntaxe abstraite d'ordre supérieur. La preuve de ce résultat est assez complexe.

Adoptant un point de vue théorique, André Hirschowitz et Pierre Leleu travaillent à affiner les définitions formelles données par Thierry Despeyroux et André Hirschowitz (3.6.1), de manière à pouvoir y confronter ce système.

3.6.4 Premiers résultats métathéoriques

Participants : Joëlle Despeyroux, André Hirschowitz, Pierre Leleu

Pour la sémantique (3.6.2) comme pour la récursion (3.6.3), il s'agit de définir de nouveaux langages qui poseront tous des problèmes de normalisation. Nous avons commencé par considérer la présentation du lambda-calcul modal S4 simplement typé proposée en 1995 par Frank Pfenning et Hao-Chi Wong. Nous avons prouvé les théorèmes de Church-Rosser et de forte normalisation de ce système en étendant au cas modal la technique de preuve que Healfdene Goguen a introduite dans sa thèse. Nous souhaitons étendre la preuve aux types dépendants, polymorphes et surtout inductifs.

3.6.5 Spécification de langages de programmation

Participant : Thierry Despeyroux

Mots-clés : environnement de programmation, langage de programmation.

Une première version d'un formalisme de spécification de syntaxe abstraite modulaires (AS) a été mise à disposition. Ce formalisme prend en compte des syntaxes d'ordre supérieur. Un manuel [101] a été écrit, et est disponible en version HTML.

Nous développons actuellement un formalisme de définition de syntaxe concrète modulaire dont l'implémentation se fait à l'aide de grammaires DCG. Une version préliminaire de ce formalisme a été testée pour C et Java et permet d'espérer un gain très important en vitesse de développement par rapport à la technique utilisant Metal et Yacc.

Le but de ces travaux est d'utiliser les connaissances acquises lors du développement de Centaur pour développer une chaîne complète de formalismes de spécification en mettant l'accent sur la rapidité, la sureté et la facilité de développement.

4 Actions industrielles

4.1 CNET

Participants : Olivier Pons, Laurence Rideau, Yves Bertot

Nous avons une collaboration suivie avec P. Crégut and J.-F. Monin du CNET à Lannion au sein d'une convention de trois ans, en commun avec le projet Coq. Cette collaboration porte sur l'aide au développement de grandes théories réutilisables pour le système Coq (thèse d'Olivier Pons – voir section 3.3.3).

4.2 Dassault-Aviation

Participants : Janet Bertot, Yves Bertot, Yann Coscoy, Gilles Kahn, Laurence Rideau, Joëlle Despeyroux

Dans le cadre de l'action Génie qui lie l'Inria à Dassault-Aviation, nous avons mené une collaboration autour de l'utilisation de CtCoq dans les développements formels effectués par Dassault Aviation pour la vérification de logiciels. Ce travail devrait se continuer dans la deuxième phase de Génie avec la preuve d'un compilateur.

4.3 SIMULOG

Participants : Isabelle Attali, Francis Montagnac, Christophe Roudet

La société SIMULOG est un partenaire privilégié de nos travaux de recherche. Le produit FORESYS qu'elle diffuse est construit à l'aide de notre générateur d'environnements de programmation, Centaur.

Cette année, nous avons fait bénéficier cette entreprise de nos résultats sur l'édition textuelle et nous avons partagé des corrections de problèmes dans le cadre d'une remise à niveau d'une version commune des modules de base. Nous collaborons aussi dans la conception du langage de transformation TrfL, qui les intéresse pour assurer la maintenance de codes industriels.

4.4 Bull – G.I.E. Dyade

Participants : Yves Bertot, Valérie Pascual, Laurent Théry

Nous prenons part au G.I.E. Dyade dans le cadre de l'action Koala (Kit d'outils pour Ateliers Logiciels Avancés) pour effectuer le transfert de nos compétences dans le développement de composants et d'outils d'aide à la programmation vers des outils industriels.

5 Actions nationales et internationales

5.1 Actions nationales

Joëlle Despeyroux, Laurence Rideau et Laurent Théry participent au groupe de travail Déduction et Calcul Formel du GDR/PRC Algèbre Mathématique et Informatique (AMI). Deux réunions de ce groupe ont eu lieu cette année.

Les travaux sur la sémantique de EIFFEL// s'inscrivent dans un projet commun avec le projet Inria SLOOP (avec Denis Caromel) et l'équipe de Patrick Sallé (IRIT, Toulouse) sur le thème « Outils pour la programmation en langages d'acteurs » soutenu par le PRC Programmation.

Isabelle Attali fait partie du comité de direction du GDR Programmation et anime le pôle « interfaces et environnements ».

Isabelle Attali participe aux groupes de travail SCOOP et FLASHI du GDR communication homme machine sur les thèmes liés aux interfaces homme machine : architectures logicielles, interopérabilité, spécifications formelles, etc.

5.2 Actions internationales

5.2.1 Europe de l'ouest

Laurent Théry s'est rendu à l'université de Rome *La Sapienza* dans le cadre de la préparation d'une proposition de constitution d'un TMR européen.

Joëlle Despeyroux est responsable local (*site leader*) du groupe de travail « Types », qui fait suite au projet européen BRA « Types pour les preuves et les programmmes ». Ce groupe de travail comprend 15 sites répartis en Europe et a commencé le 15 sept 96, pour une durée de 3 ans.

Dans le cadre de ce groupe de travail, Yves Bertot s'est rendu à l'université d'Edimbourg (Royaume-Uni) pour partager des résultats sur les interfaces homme-machine de systèmes de preuves, en particulier dans le domaine de l'interaction à la souris (*proof-by-pointing* et *drag-and-drop*).

5.2.2 Amérique

Les travaux autour de SISAL se poursuivent en collaboration avec Andrew Wendelborn (University of Adelaide, Australie) et Jean-Luc Gaudiot (University of Southern California, USA).

Joëlle Despeyroux et Pierre Leleu ont rendu visite trois jours à Frank Pfenning de Carnegie Mellon University (USA), alors en congé sabbatique à l'université de Darmstadt (Allemagne), en mai (collaboration sur la récursion primitive pour la syntaxe abstraite d'ordre supérieur [100]).

5.2.3 Asie et Océan Pacifique

Les travaux autour de Sisal donnent lieu à une collaboration avec l'université d'Adelaide en Australie (voir plus haut dans la section « Amérique »). Romain Guider a visité l'équipe de Andrew Wenldeborn de mai à août.

6 Diffusion des résultats

6.1 Conférences et congrès

Des membres du projet ont assisté aux conférences suivantes :
Algebraic Methodology and Software Technology à Munich, Allemagne, User Interfaces for Theorem Provers à York, Grande Bretagne, Computer Aided Deduction à New-Brunswick, Etats-Unis, Languages and Models for Future Computing Systems à Vienne, Autriche, International Symposium on Objects Technologies for Advanced Software, Kanazawa, Japon, conférence WWW à Paris, International Conference on Parallel Processing, Lyon, Rencontres sur le Parallélisme, à Bordeaux, Object-Oriented Programming : Systems, Languages, and Applications, San Diego, Etats-Unis, First B Conference à Nantes, et journées GDR Programmation à Orléans.

6.2 Conférences invitées, tutoriels, cours, etc.

Yves Bertot a donné un cours d'une journée sur la sémantique des langages de programmation à l'ecole jeunes chercheurs en programmation.

Gilles Kahn a donné un cours aux journées de formation au système de preuve Coq organisées à Lyon. Yves Bertot y est également intervenu comme assistant.

6.3 Actions d'enseignement

Isabelle Attali fait partie du conseil scientifique du DEA informatique de l'université de Nice Sophia Antipolis, et est responsable du module sémantique des langages de programmation. Elle est également responsable scientifique de l'ecole jeunes chercheurs en programmation.

Isabelle Attali, Laurence Rideau et Delphine Terrasse du projet MEIJE assurent un module de tronc commun de 40 heures au DEA Informatique de l'Université de Nice Sophia Antipolis.

Isabelle Attali, Yves Bertot et Joëlle Despeyroux enseignent une option « Sémantique des langages de programmation » au DEA de mathématiques discrètes et fondements de l'informatique (MDFI), à l'université de Marseille : 25 heures.

6.4 Thèses

- Habilitation à diriger des recherches :
  1. Isabelle Attali, « Sémantique Naturelle : expressivité et évaluation », université de Nice-Sophia Antipolis, septembre 96.
- Thèses en cours :
  1. Sidi Ould Ehmety, « Définition formelle d'un langage à objets parallèle et transformations de programmes », université de Nice-Sophia Antipolis, depuis octobre 93, dirigée par Isabelle Attali.
  2. Yann Coscoy, « Explication textuelle de preuves », Université de Nice-Sophia Antipolis, depuis octobre 95, dirigée par Gilles Kahn.

3. Ranan Fraer, « Environnements interactifs de vérification de programmes », Université de Nice-Sophia Antipolis, depuis novembre 94, dirigée par Yves Bertot.
4. Romain Guider, « Représentation data-flow de programmes à objets pour une parallélisation semi-automatique », université de Nice-Sophia Antipolis, depuis novembre 96, dirigée par Isabelle Attali.
5. Pierre Leleu, « Syntaxe abstraite d'ordre supérieur et induction dans les théories typées », ENPC, Paris, depuis novembre 95, dirigée par Joëlle Despeyroux.
6. Olivier Pons, « Aide au développement de grandes théories réutilisables dans les systèmes de preuve interactifs », CNAM, Paris, depuis novembre 95, dirigée par Laurence Rideau.
7. Christophe Roudet, « Conception et implantation d'un formalisme sémantique pour les transformations de programmes », université de Nice-Sophia Antipolis, depuis novembre 95, dirigée par Isabelle Attali.

6.4.1 Stages

Le projet a accueilli les stagiaires suivants :

– Mouloud Bourbel. (coencadré avec Eric Madelaine du projet MEIJE),
« Environnement interactif de développement de preuves pour des langages parallèles », DEA d'Informatique Lyonnais (DIL), 4 mois.

– Sylvain Lippi. (coencadré avec Denis Caromel du projet SLOOP) « Modélisation et preuve d'équivalence de programmes Eiffel// en TLA », DEA Université de Marseille Luminy, 3 mois.

6.5 Diffusion de logiciels

L'équipe continue de distribuer le générateur d'environnements de programmation Centaur (10 licences cette année). Nous avons également mis en accès la version v10 de l'environnement CtCoq au mois de Janvier (27 licences ont été distribuée cette année).

7 Publications

Documents d'habilitation à diriger des recherches

[92] I. ATTALI, *Sémantique Naturelle: expressivité et évaluation*, habilitation à diriger des recherches, Université de Nice Sophia Antipolis, septembre 1996.

Articles et chapitres de livre

[93] I. ATTALI, D. CAROMEL, S. OULD EHMETY, « A Natural Semantics for Eiffel Dynamic Binding », *ACM Transactions on Programming Languages and Systems (TOPLAS) 18*, 5, novembre 1996.

[94] I. ATTALI, D. CAROMEL, A. WENDELBORN, « A Formal Semantics and an Interactive Environment for Sisal », *in : Tools and Environments for Parallel and Distributed Systems*, Kluwer Academic Publishers, février 1996, ISBN 0-7923-9675-8.

Communications à des congrès, colloques, etc.

[95] I. ATTALI, D. CAROMEL, R. GUIDER, A. WENDELBORN, «Optimizing Sisal Programs: a formal approach», *in : Actes Euro-Par'96, International Conference on Parallel Processing, Lyon*, Springer-Verlag, août 1996. LNCS 1123-1124.

[96] I. ATTALI, D. CAROMEL, S. OULD EHMETY, S. LIPPI, «Semantic-based visualization for parallel object-oriented programming», *in : Actes OOPSLA'96 (Object-Oriented Programming: Systems, Languages, and Applications)*, *31, number 10*, ACM Press, Sigplan Notices, octobre 1996.

[97] J. BERTOT, Y. BERTOT, «CtCoq: A System Presentation», *in : Automatic Deduction, CADE-13*, M. McRobbie, J. Slaney (réd.), *LNAI*, 1104, Springer Verlag, p. 231–234, juillet 1996.

[98] R. FRAER, «Formal Development in B of a Minimum Spanning Tree Algorithm», *in : First B Conference*, novembre 1996.

[99] R. FRAER, «Tracing the Origins of Verification Conditions», *in : Fifth International Conference on Algebraic Methodology and Software Technology (AMAST '96)*, Springer-Verlag LNCS, juillet 1996.

Rapports de recherche et publications internes

[100] J. DESPEYROUX, F. PFENNING, C. SCHÜRMANN, «Primitive recursion for higher-order abstract syntax», *rapport de recherche n°CMU-CS-96-172*, Carnegie Mellon University, Pittsburgh, Pennsylvania, août 1996, Une version abrégée a été soumise pour publication.

[101] T. DESPEYROUX, «AS, for Abstract Syntax - Manual - V1.0», *Rapport Technique n°0197*, Inria, sep 1996, 36 pages, http://www.inria.fr/RRRT/RT-0197.html.

[102] A.-M. DÉRY, L. RIDEAU, «Distributed Architecture for Programming Environments», *rapport de recherche n°2918*, INRIA, juin 1996, http://www.inria.fr/RRRT/RR-2918.html.

[103] A. FELTY, L. THÉRY, «Interactive Theorem Proving with Temporal Logic», *Rapport de Recherche n°2804*, INRIA, 1996, http://www.inria.fr/RRRT/RR-2804.html.

[104] S. LIPPI, *Modélisation et preuve d'équivalence de programmes Eiffel// en TLA*, Rapport de stage DEA, Université de Marseille Luminy, juin 1996.

Divers

[105] I. ATTALI, D. CAROMEL, R. GUIDER, A. WENDELBORN, «Parallélisation de boucles sisal : génération de pipelines», novembre 1996, Journées du GDR Programmation, Orléans.

[106] I. ATTALI, C. ROUDET, «Trfl : un langage de description de transformations», novembre 1996, Journées du GDR Programmation, Orléans.

[107] J. BERTOT, Y. BERTOT, «The CtCoq Experience», août 1996, actes du colloque User Interfaces for Theorem Provers, York.

[108] O. PONS, L. RIDEAU, «Outils de navigation et de maintenance dans un script de preuve», novembre 1996, Journées du GDR Programmation, Orléans.

8 Abstract

The aim of the CROAP project is to study the tools that are needed in a programming environment to produce complex and safe software, and how to compose these tools in working environments that are user-friendly and efficient. To achieve this objective, our research is centered around three main themes: implementing tools adapted to various programming styles and languages, formally studying

the principles of programming languages, and mastering the common foundations to large classes of these languages.

Our study of programming styles leads us to implement programming environments that are specially suited to a variety of programming languages, each of which represents a family of programming methodology. Thus, we study tools for the ML language (functional programming), the Eiffel language (object-oriented programming), the Sisal programming language (parallel dataflow programming), and C (imperative programming). For each language, we address a different type of tool. For example, we studied symbolic execution for Eiffel and typing around ML. For a more prospective work, we also study specialised environments for software-hardware codesign or for symbolic computations like the mechanical proof of mathematical results. While not directly related to our work on programming languages, these experiments make it possible to address the issue of integrating tools in heterogeneous environments, with heterogeneous data. Above all, the work on proof environments makes it possible to experiment with the task of developping programs that are proved correct.

For programming language semantics, we use a method called *Natural Semantics*, implemented by the specification language TYPOL. This language can be executed fairly efficiently, which makes it possible to extract practical tools from programming language specifications. We also provide ways to reason formally about programming language properties and about the validity of the extracted tools.

To ensure the validity of our work around *Natural Semantics* and to make it possible to extend this method to new approaches, it is also necessary to study the logical foundations of this formal specification method. This theoretical study should make it possible to integrate new paradigms such as higher-order abstract syntax to the specification formalism and to construct a real workbench for the formal study of programming languages.

All these experiments use a system for generating interactif programming environments, the CENTAUR system. This system is both the sum of our competence and a toolkit for new experiments.

As landmarks for 96, we took part in the successful concusion of the first phase of the “Genie” project. Note also the starting of the Dyade activity and a declaration of interest by Novell, one of the major actors in the American market of professional software, for our work on object-oriented and parallel languages.