INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Projet mimosa

Migration et Mobilité : Sémantique et Applications

Sophia Antipolis

THÈME 1C

Table des matières

1. Composition de l'équipe

MIMOSA *est un projet commun entre l'INRIA, le Centre de Mathématiques Appliquées (CMA) de l'Ecole des Mines de Paris et le Centre de Mathématiques et d'Informatique (CMI) de l'Université de Provence.*

Responsable Scientifique
Gérard Boudol [Directeur de Recherche, Inria]

Responsable Permanent
Ilaria Castellani [Chargée de Recherche, Inria]

Assistantes de Projet
Catherine Juncker [Inria, jusqu'à septembre]
Evelyne Largeteau [Inria, à partir de septembre]
Dominique Micollier [Armines]

Personnel Inria
Davide Sangiorgi [Directeur de Recherche, Inria, jusqu'à octobre]
Manuel Serrano [Chargé de Recherche, Inria]

Personnel CMA et CMI
Roberto Amadio [Professeur, université de Provence]
Frédéric Boussinot [Maître de Recherches, École des Mines de Paris]
Silvano Dal-Zilio [Chargé de Recherche, CNRS]
Jean-Ferdy Susini [Ingénieur Armines]

Chercheurs Doctorants
Raul Acosta-Bermejo [Allocataire CONACYT, 2ème année]
Christian Brunette [Allocataire MENRT, 1ère année]
Damien Ciabrini [Allocataire MENRT, à partir de septembre]
Yuxin Deng [Allocataire Egide, projet Profundis, à partir de septembre]
Cédric Lhoussaine [Allocataire MENRT, jusqu'à avril]
Ana Matos [Allocataire Gouv. Portugais, à partir de mars]
Charles Meyssonnier [Elève ENS Lyon, 1ère année]
Dimitri Petoukhov [Allocataire École des Mines de Paris, 2ème année]
Vincent Vanackère [Allocataire ENS Lyon, 2ème année]
Pascal Zimmer [Allocataire ENS Lyon, 2ème année]

Chercheur Post-Doctorant
Xudong Guan [Allocataire Egide, projet Mikado, à partir de septembre]

Professeur Invité
António Ravara [de mars à août]

Collaborateur Extérieur
Yannis Bres [à partir d'octobre]

Stagiaires
Damien Ciabrini [DEA Informatique, Nice]
Thomas Gazagnaire [Magistère ENS Lyon]
Pascal Fauconnier [DEA Informatique, Nice]
Damien Pous [Magistère ENS Lyon]
Alexander Samarin [DEA RSD, Nice]

2. Présentation et objectifs généraux

Le projet MIMOSA est commun avec le Centre de Mathématiques Appliquées de l'ENSMP et le Centre de Mathématiques et d'Informatique de l'université de Provence. L'objectif général du projet est de concevoir

et d'étudier des modèles de la programmation répartie et mobile, d'en tirer des primitives pour cette programmation, et d'élaborer des techniques de raisonnement et de vérification concernant spécifiquement les problèmes liés au code migrant. Nous développons deux approches : l'approche interactive, fondée sur l'idée de processus indépendants et communicants, et l'approche réactive, où les processus réagissent de concert à des événements diffusés. Nous visons particulièrement à intégrer les primitives de migration dans la programmation réactive, de façon à pouvoir programmer des « agents » participant à des « assemblées réactives » (jeux en réseau par exemple). Nos axes de recherche principaux sont les suivants :

- Modèles et langages pour la mobilité, approche interactive. Dans cet axe de recherche l'objectif est d'étudier les primitives pour la mobilité fondées en particulier sur le modèle du π-calcul et de ses extensions réparties, et sur le calcul des Ambients.
- Modèles et langages pour la mobilité, approche réactive. Les objectifs sont de poursuivre le développement des outils de la programmation réactive et d'y intégrer des primitives pour la migration.
- Méthodes de raisonnement. Nous développons en particulier des méthodes de vérification pour des problèmes de sécurité (vérification de protocoles cryptographiques, analyse de flux d'information dans les programmes).
- Types. Les systèmes de types sont utilisés, dans les modèles de la mobilité, à la fois pour éviter des erreurs, imposer des disciplines de sécurité, spécifier des comportements, mais aussi pour valider des transformations et des équivalences.

Relations internationales et industrielles :

- CTI France Télécom R&D, sur la modélisation et la vérification des objets migrants.
- projet IST Profundis (Proofs of functionality for mobile distributed systems), avec le KTH de Stockholm, la Faculté des Sciences et Technologies de Lisbonne, l'Université de Pise.
- projet IST Mikado (Mobile calculi based on domains), avec l'INRIA Rhone-Alpes (projet SARDES), France Télécom R&D, les universités de Sussex, Lisbonne, Florence et Turin, l'Ecole Nationale Supérieure des Télécommunications.

3. Fondements scientifiques

3.1. Sémantique de la mobilité et sécurité

Mots clés : *concurrence*, *sémantique*, *parallélisme asynchrone*, *typage*, *langages fonctionnels*, *objets concurrents*, *mobilité*, *λ-calcul*, *π-calcul*, *protocoles cryptographiques*, *non-interférence*.

Participants : Roberto Amadio, Gérard Boudol, Ilaria Castellani, Yuxin Deng, Xudong Guan, Cédric Lhoussaine, Ana Matos, Charles Meyssonnier, Davide Sangiorgi, Vincent Vanackère, Pascal Zimmer.

La *mobilité* est devenue l'un des aspects importants des systèmes informatiques, et particulièrement des systèmes distribués. Notre projet s'intéresse à la notion de « code mobile » - et donc à une notion de mobilité logique, plutôt que physique. Il s'agit de dégager les concepts utilisés dans les langages de programmation récemment proposés pour permettre la mobilité des calculs et d'en formaliser la sémantique. D'autre part, les préoccupations concernant la *sécurité* sont évidemment très importantes s'agissant de code mobile, et plus généralement de systèmes partagés. Nous étudions des méthodes de vérification de propriétés de sécurité.

Les modèles sur lesquels nous développons principalement notre approche formelle de la mobilité sont le *π-calcul* de Robin Milner & al. et le calcul des « Ambients » de Luca Cardelli & al. Le premier est un modèle similaire à celui que le λ-calcul offre pour la programmation séquentielle et fonctionnelle, mais dans lequel le parallélisme est pris en compte, ainsi qu'une certaine forme de mobilité puisque dans le π-calcul les processus se communiquent des noms de canaux de communication. En fait, le π-calcul contient plus ou moins directement le λ-calcul, sa puissance d'expression est donc déjà bien établie de ce simple fait - ceci en regard de la programmation séquentielle.

Le second explore plutôt le concept de migration : il est fondé sur une notion très générale de « domaine », appelé « Ambient », dans lequel des calculs peuvent s'effectuer, et qui possède une structure hiérarchique. Dans le calcul des « Ambients », la structure d'imbrication des domaines les uns dans les autres évolue dynamiquement, et tout le calcul réside dans ce mouvement (et aussi dans la possible disparition des frontières). Ce mécanisme est en fait extrêmement puissant, puisqu'on peut l'utiliser pour simuler les machines de Turing.

Nos recherches sur ces formalismes, considérés en tant que modèles pour la programmation des systèmes distribués et mobiles, s'articulent autour de plusieurs axes. Nous étudions en particulier comment peut s'étendre la notion de type, et comment elle peut être utile dans le raisonnement à propos des programmes répartis ou migrants. Cette notion de type, familière dans la programmation séquentielle, paraît encore plus importante dans un cadre distribué, parce que les sources d'erreurs y sont bien plus nombreuses et subtiles. Plus généralement, nous étudions ce que peuvent être des méthodes de raisonnement relatives aux programmes répartis et migrants. Nous visons à appliquer ces méthodes tout spécialement aux problèmes de sécurité, par exemple dans la vérification de protocoles cryptographiques ou l'analyse de flux d'information dans les programmes.

3.2. Programmation réactive et fonctionnelle

Participants : Raul Acosta-Bermejo, Gérard Boudol, Frédéric Boussinot, Christian Brunette, Damien Ciabrini, Dimitri Petoukhov, Manuel Serrano, Jean-Ferdy Susini, Pascal Zimmer.

Mots clés : *Concurrence*, *programmation*, *parallélisme synchrone*, *langages fonctionnels*, *langages réactifs*, *sémantique dénotationnelle et opérationnelle*, *threads*, *WWW*.

L'approche réactive a pour but d'étudier divers modèles de programmation concurrente et parallèle dans lesquels existe une horloge logique définissant des instants globaux. La programmation fonctionnelle privilégie la notion de fonction. Elle repose fortement sur le λ-calcul. Ces deux approches sont complémentaires et reposent sur des bases théoriques solides. Un de nos objectifs est de les marier afin d'obtenir un modèle de programmation généraliste permettant la concurrence tout en conservant une sémantique formelle. Nos travaux sur la programmation réactive, initiés avec le langage Reactive-C en 1988, visent à ajouter aux langages existants des primitives de programmation de haut niveau, fondées sur les paradigmes de la programmation synchrone : notion d'instant global, d'événement et de diffusion instantanée.

Plusieurs langages et formalismes réactifs ont été définis et implémentés : Reactive-C, puis, sur cette « couche de base », réseaux de processus réactifs, langage synchrone SL, objets réactifs. Ils sont décrits dans un livre qui présente également Reactive-C en détail [5]. L'approche réactive au dessus de JAVA (plus particulièrement les SugarCubes) est aussi décrite dans un livre [4]. Plus récemment, nous nous sommes intéressés à la programmation réactive dans un cadre fonctionnel.

Le traitement de la concurrence dans les langages de programmation usuels reste très empirique et fondé sur des concepts datés. Par exemple, JAVA propose les « threads » préemptifs pour la gestion de la concurrence. Ces « threads » posent de nombreux problèmes en particulier à cause de leurs faiblesses sémantiques [6]. Nous proposons une alternative appelée FairThreads reposant sur l'approche réactive qui allie une sémantique déterministe et une grande simplicité de programmation.

Un de nos objectifs est d'étudier les interactions entre les aspects concurrents et algorithmiques des programmes, en particulier, en construisant une sémantique formelle (dénotationnelle ou opérationnelle) recouvrant ces deux aspects. Pour cela, il est bien sûr préférable de partir de langages séquentiels ayant une sémantique formelle, ce qui est le cas des langages fonctionnels. De plus, l'absence de construction permettant la concurrence facilite l'expérimentation linguistique à partir de ceux-ci. Nous avons ainsi introduit les FairThreads en Scheme, dans le système Bigloo. Il ne s'agit pas d'une simple expérimentation car nous gardons le souci de conserver l'efficacité du système.

Nous proposons également une technique de programmation entièrement graphique, à base de combinaisons de comportements. Les comportements sont en fait du code réactif, combiné de différentes façons par des

manipulations d'icônes à l'écran. Les domaines d'applications principaux sont ceux des interfaces hommes machines et des jeux.

4. Domaines d'application

4.1. Télécommunications

Mots clés : *télécommunications*, *programmation réactive*, *mobilité*.

Le domaine d'application « naturel » de nos travaux est celui des télécommunications. Nos contributions y portent à la fois sur la spécification et sur la programmation fiable à l'aide de formalismes adaptés de haut niveau.

4.1.1. Modélisation de systèmes mobiles

La spécification *formelle* des systèmes répartis, par exemple exprimés comme systèmes d'objets concurrents, reste encore très largement à faire, et les besoins dans ce domaine sont encore mal couverts. Nos travaux dans ce domaine ont fait l'objet d'une collaboration suivie avec France Télécom R&D, dans le cadre de deux CTI successives, puis au sein du projet RNRT MARVEL, et maintenant dans le projet IST Mikado.

4.1.2. Serveurs Web

Les FairThreads conçus par l'équipe MIMOSA sont, a priori, bien adaptés aux architectures logicielles client/serveur et plus particulièrement au développement de serveurs Web. En effet, des recherches récentes ont montré qu'une architecture à base de boucles d'événements et d'entrées-sorties asynchrones permet d'obtenir des serveurs plus rapides que ceux actuellement déployés (comme par exemple le serveur Apache) utilisant des threads. Les FairThreads permettent de programmer naturellement et simplement des boucles d'événements. Les FairThreads devraient donc ouvrir le champ à de nouvelles implantations pour serveurs Web plus simples et efficaces que les implantations classiques. Une expérimentation approfondie est prévue pour l'année à venir afin de valider cette hypothèse.

4.1.3. IHM graphiques

L'utilisation des Icobjs apparaît naturelle pour les interfaces homme-machine télécom, en particulier lorsqu'il s'agit de représenter des animations d'icônes, comme par exemple dans la supervision de réseau. Une autre utilisation possible des Icobjs est la conception de systèmes permettant à l'utilisateur final de programmer ses propres services, par des combinaisons graphiques d'Icobjs. Enfin, le domaine des jeux, en particulier des jeux en réseaux, semble être un domaine dans lequel les Icobjs peuvent être utilisés, par exemple pour la programmation des avatars de joueurs.

5. Logiciels

5.1. Les logiciels du projet

La plupart des logiciels développés dans le projet, y compris les plus « anciens », qui ne font pas l'objet d'un paragraphe ci-dessous lorsqu'il n'y a pas eu de travail les concernant effectué cette année, sont librement accessibles, en particulier depuis les pages web de l'INRIA (http://www.inria.fr/valorisation/logiciels/langages.fr.html) et la page du projet (http://www-sop.inria.fr/mimosa/).

5.2. Programmation fonctionnelle : Bigloo

Participants : Yannis Bres, Damien Ciabrini, Bernard Serpette [projet Oasis], Manuel Serrano [correspondant].

Mots clés : *Programmation fonctionnelle*, *Scheme*, *Fair Threads*.

L'environnement de programmation Bigloo est disponible librement sur le site Web de l'INRIA Sophia depuis le 1er octobre 2001 à l'adresse http://www.inria.fr/mimosa/fp/Bigloo. Cette distribution comprend un compilateur optimisant pour Scheme produisant indifféremment du code natif ou du code JVM, un debogueur, un profiler et un ensemble de bibliothèques permettant le développement d'applications réalistes. Les travaux récents ont porté sur la réalisation d'un nouveau debogueur, la production de code pour machine JVM, ainsi que la programmation concurrente et réactive en Scheme. En particulier, le langage Senior (Scheme réactif inspiré de Junior) a vu une nouvelle implantation, poursuivant ainsi les travaux de J. Demaria qui a quitté l'équipe en début d'année. Néanmoins, la principale innovation a été l'intégration des FairThreads dans Bigloo. Ces derniers sont en effet maintenant disponibles dans la distribution officielle.

Bigloo était initialement destiné à la création d'applications sous Unix. Un premier portage direct sous Windows a été effectué. Nous sommes toutefois conscients de ses limites. S'il permet d'exécuter du code Scheme sous Windows, il ne permet pas toutefois d'utiliser toutes les fonctionnalités, par exemple graphiques, de ce système. Un nouvel effort a donc été entrepris pour mieux intégrer Bigloo sous Windows. Ce travail fait l'objet d'une aide de l'INRIA par le biais d'une ODL, et d'un soutien financier de Microsoft.

5.3. Programmation fonctionnelle : Scribe

Participants : Erick Gallesio [Université de Nice], Manuel Serrano [correspondant].

Mots clés : *Programmation fonctionnelle*, *Scheme*, *HTML*, *XML*, *Web*.

La nécessité de disposer de « vrais » langages de programmation pour produire des documents techniques (articles, documentations de logiciels, pages web) est de plus en plus avérée. La mode actuelle s'articule autour de XML et sa batterie d'outils ainsi que des librairies pour manipuler des documents par le biais de programmes informatiques. Nous pensons que cette tendance qui repose sur de nombreux formalismes différents introduit une complexité d'utilisation peu justifiée. Nous avons conçu un langage qui est à la fois un langage de description de documents et un langage de programmation. L'avantage d'une telle approche est que toutes les opérations à effectuer sur le document (transformations, mise en forme, ...) sont spécifiées dans un formalisme unique.

Ce langage, Scribe, est fortement inspiré du langage Scheme. Cette filiation n'est pas très étonnante puisque Scheme est, par construction, bien adapté à la représentation de structures d'arbre qui sont généralement utilisées pour la représentation des textes. Scribe étend toutefois Scheme syntaxiquement et sémantiquement afin de permettre une représentation compacte des textes. Scribe est disponible sur le serveur web de l'Inria Sophia Antipolis à l'adresse http://www.inria.fr/mimosa/fp/Scribe. Une première présentation de Scribe a été publiée dans l'article [21].

5.4. Programmation réactive : Rejo/Ros

Participant : Raul Acosta.

Rejo est une extension de Java qui permet de définir des objets réactifs, c'est-à-dire des objets qui encapsulent des données et des mélanges d'instructions Java et d'instructions réactives. Le modèle d'exécution est celui de l'approche réactive synchrone, dans lequel les instructions réactives sont exécutées d'une façon atomique. Ces objets peuvent être considérés comme des agents mobiles car ils peuvent migrer en utilisant une plate-forme, appelée ROS, qui offre les fonctionnalités nécessaires. Les travaux récents ont porté sur : 1) l'ajout de la propriété de persistence dans les agents, 2) l'implémentation des Icobjs en Rejo (Ricobjs), et 3) l'étude du contrôle d'accès des agents migrants sur Ros. Ceci est décrit plus en détail dans la section « résultats nouveaux ».

La version actuelle du compilateur de Rejo est la 2.0. Cette version utilise la version 1.3 de Java et génère du code Junior version 2.1 (dans ses différentes implémentations, Rewrite, Replace, Storm, Simple) qui peut être exécuté sur les plate-formes Linux, Macintosh et Windows (Cygwin). Elle est accessible à l'url http://www-sop.inria.fr/mimosa/rp/.

5.5. Programmation réactive : SugarCubes

Participants : Alexander Samarin, Jean-Ferdy Susini [correspondant].

Les SugarCubes forment une bibliothèque Java permettant l'implémentation de systèmes réactifs dynamiques et modulaires. Cette bibliothèque apporte une alternative au mécanisme standard des threads Java dans la programmation de systèmes concurrents. La modularité induite par la programmation réactive est renforcée par un modèle de composant réactifs particuliers appelés Cubes.

Ces travaux poursuivis maintenant depuis quelques années ont été orientés cette année vers la réalisation par Alexander Samarin d'un framework graphique dédié à la simulation physique, reposant sur le mécanisme des occurrences d'événements valués introduit par la version 3 des SugarCubes. La distribution de la version 3 des SugarCubes a été publiée sur le Web (http://www-sop.inria.fr/mimosa/rp/) et quelques corrections ont été apportées. Un certain nombre d'exemples illustrant les possibilités offertes par la programmation en SugarCubes ont été publiés.

5.6. Programmation réactive : FairThreads en C

Participant : Frédéric Boussinot.

Dans l'implémentation en C des FairThreads, disponible à l'url http://www-sop.inria.fr/mimosa/rp/, les schedulers sont des serveurs de synchronisation auxquels les threads peuvent se lier dynamiquement. Les threads liés à un même scheduler sont exécutés de manière coopérative et au même rythme. La communication se fait par des événements diffusés à tous les threads liés au scheduler. La sémantique opérationnelle de la partie coopérative des FairThreads est décrite sous forme de règles de réécriture.

Les FairThreads qui ne sont liés à aucun scheduler sont exécutés par l'OS à leur propre rythme, dans un mode préemptif. Il est ainsi possible, avec les FairThreads en C, de mélanger des threads coopératifs et des threads préemptifs dans un même cadre. De plus, on peut définir des threads spéciaux, appelés automates, qui ne nécessitent pas la puissance d'un thread natif dédié pour s'exécuter. Les automates consomment moins de ressources et sont utiles dans les systèmes formés d'un grand nombre de tâches concurrentes.

Dans les FairThreads en C, les schedulers et les threads déliés peuvent être exécutés en réel parallélisme sur des processeurs distincts, permettant ainsi de profiter des architectures multi-processeurs (en particulier de type SMP).

5.7. Typage et exécution des mixins

Participants : Pascal Zimmer.

Mots clés : *inférence de type*, *récursion*, *objets*, *mixins*.

Nous avons implémenté, en OCAML, une extension de l'algorithme de typage pour la récursion généralisée dans un langage à la ML proposé par Boudol [28]. L'extension concerne aussi bien le langage de base, qui contient des constructions standards sur les booléens, les entiers, les listes, etc., que le système de type, qui autorise des types récursifs. Cette implémentation permet l'expérimentation d'un style de programmation orientée-objet basé sur les *mixins*. De plus, nous avons étendu la machine abstraite proposée dans [15], et nous avons intégré l'implémentation de cette machine dans notre logiciel.

5.8. Ambients en Icobjs

Participants : Frédéric Boussinot, Damien Pous, Pascal Zimmer [correspondant].

Le calcul des Ambients a été implémenté en utilisant les Icobjs par Damien Pous, sous la direction de Frédéric Boussinot et Pascal Zimmer. Il s'agit d'une implémentation graphique, donc parfaitement adaptée au modèle des Ambients qui se présente naturellement sous cette forme. Le logiciel, disponible sur le Web à l'url http://www-sop.inria.fr/mimosa/ambicobjs, permet la programmation et l'exécution d'expressions du calcul des Ambients, en exploitant la notion d'atomicité offerte par la programmation réactive pour résoudre les problèmes de synchronisation complexes posés par les mouvements des Ambients.

5.9. Vérification de protocoles cryptographiques

Participant : Vincent Vanackère.

Le vérificateur de protocoles TRUST [23] est une implémentation optimisée, réalisée en OCAML, d'un algorithme symbolique (développé avec R. Amadio et D. Lugiez [27]) de vérification de protocoles cryptographiques pour le cas sans itération. TRUST est disponible à l'url http://www.cmi.univ-mrs.fr/~vvanacke/trust.html.

6. Résultats nouveaux

6.1. Sémantique de la mobilité

Participants : Roberto Amadio, Silvano Dal-Zilio, Xudong Guan, Cédric Lhoussaine, Charles Meyssonnier, Davide Sangiorgi, Pascal Zimmer.

Mentionnons ici que nous avons produit, dans le cadre du projet MIKADO, un rapport [29] qui présente un état de l'art en ce qui concerne les modèles de la mobilité. Ce travail couvre quinze modèles, étudiés sous les points de vue de la répartition, de la mobilité et de la sécurité. Ce rapport est disponible à http://mikado.di.fc.ul.pt/output/deliverables.html.

6.1.1. Travaux sur le π-calcul

Davide Sangiorgi a étudié le problème de la terminaison dans le π-calcul. Dans le domaine du code mobile, la propriété de terminaison revêt une importance particulière, car elle apparaît dans les préoccupations de sécurité. Par exemple, il est souhaitable qu'un programme migrant ne s'exécute pas indéfiniment, mobilisant les ressources du système hôte, provoquant ainsi une forme de « deni de service ». De même, il est souhaité que toute requête à un service, distant ou non, donne lieu à une transaction qui termine. Dans la programmation fonctionnelle, on peut utiliser les systèmes de types pour aborder la question de la terminaison. Il est bien connu par exemple que le λ-calcul simplement typé (sans récursion) a la propriété de normalisation forte, c'est-à-dire que toute stratégie de calcul termine. Sangiorgi adapte au π-calcul la technique de preuve de ce résultat, ce qui permet d'ailleurs de retrouver des résultats de terminaison pour diverses stratégies d'évaluation pour des λ-calculs typés, via la traduction de ces calculs en π-calcul. Ce travail a fait l'objet d'une présentation invitée à la conférence CONCUR [19].

Cédric Lhoussaine a soutenu sa thèse [10]. Celle-ci portait sur l'étude du π-calcul distribué réceptif (voir les rapports précédents), et en particulier sur l'inférence de types [25]. Les résultats obtenus par Amadio, Boudol et Lhoussaine sur l'expressivité du π-calcul réceptif ont fait l'objet d'un article accepté pour publication dans le journal Fundamenta Informaticae [26]. De même, les travaux de Roberto Amadio et Charles Meyssonnier sur la décidabilité de l'accessibilité dans plusieurs fragments du π-calcul asynchrone ont fait l'objet d'un article publié dans un journal [12].

6.1.2. Travaux sur le calcul des « Ambients »

Xudong Guan a repris le travail de Pascal Zimmer sur l'expressivité du calcul des « Ambients », qui montrait (voir les rapports précédents) que le π-calcul pouvait être codé dans le calcul des Ambients « pur », sans communication, donc en utilisant uniquement les mouvements (et bien entendu le nommage) des Ambients. Ce travail éludait en effet la preuve d'une propriété de confluence nécessaire pour établir une correspondance opérationnelle forte entre les termes du π-calcul et leur traduction. Xudong Guan obtient ce résultat par une méthode élégante et générale : il réécrit la traduction du π-calcul vers une variante des « Safe Ambients » de Levi et Sangiorgi, et utilise un système de types nouveau, qui garantit certaines lois algébriques. Ainsi, il suffit de montrer que les termes obtenus par traduction sont typables pour en déduire, par un raisonnement purement algébrique, le résultat visé [32].

Pascal Zimmer a étudié, avec la collaboration de David Teller et Daniel Hirschkoff de l'ENS Lyon, une formalisation d'un problème de contrôle d'usage de ressources dans le calcul des Ambients. Plus précisément, il s'agit de faire en sorte que dans un système donné, il n'y ait pas plus de requêtes que de ressources

disponibles. Ceci est modélisé dans le calcul des Ambients, sous la forme suivante : chaque Ambient peut être considéré comme une ressource, et le fait de migrer vers un Ambient est une façon d'utiliser la ressource qu'il représente. Dans leur article [22], Zimmer, Teller et Hirschkoff utilisent en fait un raffinement des « Safe Ambients » de Levi et Sangiorgi, où les « co-capacités » s'exercent non seulement dans l'Ambient visé par une migration, mais aussi dans l'Ambient « ambiant » - si l'on ose dire -, ce qui introduit plus de possibilités de contrôle des mouvements. Les auteurs introduisent ensuite un système de type, et montrent que le fait d'être typable garantit que le programme concerné respecte la discipline fixée (par le contexte de typage) s'agissant de l'usage des ressources.

Dans [18], Davide Sangiorgi, en collaboration avec Daniel Hirschkoff et Etienne Lozes de l'ENS Lyon, a poursuivi l'étude de la logique des Ambients proposée par Cardelli. Ils obtiennent toute une série de résultats liés à la caractérisation en terme d'équivalence comportementale de cette logique, à son axiomatisation, à la construction de « formules caractéristiques » pour les processus, à la décidabilité de l'équivalence logique, ceci pour divers fragments du calcul des Ambients.

Dans cette section nous rendons compte également du travail de Silvano Dal-Zilio, en collaboration avec Denis Lugiez de l'Université de Provence, sur l'interrogation de données semi-structurées [24]. En effet, les travaux de Cardelli sur la logique des Ambients ont conduit à constater que cette logique était, dans son principe, très bien adaptée au raisonnement sur des données structurées sous forme d'arbres, comme par exemple les documents XML. D'un autre côté, il est naturel de penser à utiliser des automates d'arbres pour raisonner sur de telles données, et également d'essayer d'appliquer la connexion classique qui relie automates, logiques et langages de requêtes. Cette approche a déjà été suivie par plusieurs équipes de recherche, aussi bien sur le plan théorique que sur le plan pratique, et elle a permis d'obtenir des résultats intéressants, tout spécialement dans l'étude des DTD (Document Type Definition), le premier et le plus simple des standards pour la validation de documents XML. Néanmoins, des travaux sont nécessaires pour pouvoir prendre en compte les schémas XML, un nouveau standard mis en avant par le W3C pour « typer » les documents XML. Dans leur travail, Dal-Zilio et Lugiez ont défini une nouvelle classe d'automates d'arbres - les « sheaves automata » -, et une logique modale associée, dérivée de la logique des Ambients, dédiés à la manipulation des documents et des schémas XML. Les problèmes liés aux schémas XML sont plus compliqués que dans le cas des DTD. En particulier, alors qu'il est possible de se limiter aux automates d'arbres réguliers pour traiter les DTD, ce modèle n'est pas assez puissant dans le cas des schémas XML. Une des raisons majeures de cette insuffisance est la présence d'un opérateur associatif/commutatif dans le langage des schémas, qui est hérité de l'opérateur d'agrégation de SGML, couplé aux limitations des automates réguliers lorsqu'il s'agit de manipuler des algèbres associatives/commutatives. Le résultat principal de l'article [24] est que la logique associée à la nouvelle classe d'automates d'arbres qui y est considérée est décidable.

6.2. Sécurité

Participants : Roberto Amadio, Gérard Boudol, Ilaria Castellani, Ana Matos, Vincent Vanackère.

6.2.1. Protocoles cryptographiques

Les prototocoles cryptographiques jouent un rôle important dans la sécurisation d'applications réparties. Il existe de nombreuses techniques de spécification des protocoles cryptographiques (et des propriétés qu'ils sont censés vérifier) dans des logiques variées : logique du premier ordre, logique linéaire, algèbres de processus, logique équationnelle... Un certain nombre d'expériences montrent que les spécifications peuvent s'exprimer naturellement dans des fragments de la logique du premier ordre. Notre objectif est de contribuer à la vérification de ce type de protocoles en utilisant des méthodes symboliques définies sur ces fragments logiques. Les résultats obtenus dans ce domaine font l'objet d'une implantation - le système TRUST - décrite dans la section « logiciels ».

Roberto Amadio a poursuivi, en collaboration avec Witold Charatonik, le travail sur les modèles de protocoles cryptographiques. Dans l'article [14], nous étudions le problème de l'accessibité du contrôle dans le modèle de Dolev-Yao dans le cas où les principaux sont représentés par des processus avec génération de noms et récursion terminale. Cet article propose une approximation du problème par réduction à une

sémantique opérationnelle quotient et nous introduisons des conditions syntaxiques vérifiables qui entrainent l'équivalence de la sémantique standard et de la sémantique quotient. Ensuite, nous introduisons une analyse, basée sur les contraintes ensemblistes, de la sémantique quotient. Cette analyse est toujours correcte et nous caractérisons une situation dans laquelle elle est exacte. Enfin, nous décrivons comment notre cadre peut être utilisé pour spécifier des propriétés de secret et de fraîcheur de protocoles cryptographiques.

6.2.2. Non-interférence

Nous avons poursuivi nos recherches, en ce qui concerne la sécurité, sur le problème de la *non-interférence*. Le problème est ici le suivant : étant donnée une « politique de sécurité », qui prend la forme d'un treillis de niveaux de sécurité, il s'agit de trouver des moyens de garantir que l'exécution d'un programme ne peut donner lieu à des « fuites d'information » d'un niveau de sécurité vers un autre qui ne lui est pas supérieur. Par exemple, on veut garantir que deux « applets » partageant des données mais ayant également chacune des données « privées » - donc de niveaux de sécurité incomparables - peuvent coopérer sans qu'il y ait une fuite de données privées de l'une vers l'autre. Ce problème est posé depuis longtemps, mais a trouvé assez récemment une solution très élégante, avec les travaux de Volpano et Smith qui ont formalisé une technique d'analyse due à Denning sous la forme d'un système de types, obtenant ainsi une preuve de correction de cette analyse.

Le travail de Boudol et Castellani à ce sujet, décrit dans le rapport d'activité de l'an passé, est désormais publié, dans un volume spécial du journal Theoretical Computer Science, édité en l'honneur de Maurice Nivat [13]. Ce travail est poursuivi maintenant par Ana Matos, qui commence une thèse. Dans [13], nous avions envisagé le cas de programmes parallèles (en nombre fixés), et soumis à un ordonnancement arbitraire. Nous nous intéressons maintenant au modèle de la programmation réactive (voir la section « programmation réactive et fonctionnelle »), où le parallélisme n'est pas non-déterministe, mais s'apparente à une exécution en « coroutines », et où le mécanisme de synchronisation (attente d'événements diffusés) est assez différent des mécanismes considérés d'habitude dans l'approche « interactive » (typiquement : rendez-vous point à point). L'objectif, à moyen terme, est de pouvoir tester les résultats théoriques sur un modèle de programmation dont nous avons la maîtrise, et qui est bien expérimenté dans le projet MIMOSA. Les résultats préliminaires obtenus par Ana Matos semblent indiquer que l'approche de Boudol et Castellani peut s'adapter au cas réactif, même s'il est encore trop tôt bien entendu pour valider la méthode sur des applications.

6.3. Programmation fonctionnelle

Participants : Gérard Boudol, Frédéric Boussinot, Yannis Bres, Damien Ciabrini, Erick Gallesio [Université de Nice], Bernard Serpette [projet Oasis], Manuel Serrano, Pascal Zimmer.

6.3.1. Mise au point des programmes

La mise au point des programmes informatiques est un domaine qui a très peu évolué ces dernières décennies. De façon assez surprenante, aujourd'hui encore, on utilise des techniques et des outils conçus il y a presque vingt ans. Malgré leur âge, ces outils sont conceptuellement limités et peu pratiques à l'usage. Le problème est encore plus criant pour les langages évolués.

Nos travaux se focalisent sur le domaine du déboguage dynamique. Pour tenter de répondre au besoin de nouveaux outils, nous avons développé BUGLOO, un débogueur dynamique pour la JVM (Java Virtual Machine). Il permet d'instrumenter l'exécution d'un programme écrit en Scheme pour le compilateur Bigloo ou en Java, et d'obtenir des informations sur la valeur de ses différentes variables durant son exécution.

BUGLOO utilise JVMDI, l'API standard de déboguage fournie par Sun, et offre un contrôle d'exécution équivalent à celui de GDB. Dans le but de le rendre plus agréable et plus pratique à utiliser que les débogueurs dynamiques traditionnels, nous avons rajouté des fonctionnalités supplémentaires comme l'enregistrement de traces durant l'exécution, le déboguage d'allocation mémoire ou l'interprète Scheme embarqué dans le programme débogué.

6.3.2. Compilation vers du code octet

Dans un premier temps, nous avons ajouté un nouveau générateur de code octet JVM pour le compilateur Scheme Bigloo qui, jusqu'à présent, ne produisait que du code natif. Nous avons utilisé ce nouveau compilateur pour mesurer les performances des JVM. En particulier, nous avons mesuré les performances de nombreux programmes Scheme compilés vers du code natif et vers du code JVM. De cette expérience, nous avons conclu que les plateformes JVM ne sont pas plus de deux ou trois fois plus lentes que les plateformes natives. Nous pensons qu'à l'avenir cet écart devrait encore diminuer. Il nous semble donc parfaitement judicieux de choisir ce type de plateforme d'exécution pour compiler des langages de haut niveau (fonctionnels, objet, ...). Ce travail a été décrit dans l'article [20].

Dans une seconde étape, nous considérons la plateforme d'exécution de Microsoft .NET. Cette nouvelle machine, qui partage de nombreuses caractéristiques avec la JVM semble toutefois mieux adaptée à l'exécution de langages fonctionnels (par exemple elle devrait permettre une meilleure compilation des récursions qui sont très fréquentes dans le style fonctionnel). Ce travail tout récemment entrepris fait l'objet d'un soutien de l'INRIA au moyen d'une ODL ainsi qu'un financement de Microsoft dans le cadre de son action ROTOR.

6.3.3. Programmation des interfaces graphiques

Nous avons terminé nos travaux sur Biglook, une librairie pour la construction d'interfaces graphiques en Scheme. Cette librairie repose fortement sur l'exploitation de la couche objet de Bigloo qui elle-même repose sur le modèle de Clos. Dans Biglook, les classes sont utilisées pour représenter les widgets (objets graphiques) et les fermetures sont utilisées pour implanter les comportements. Allier les styles de programmation fonctionnel et objet nous a permis de concevoir une interface de programmation originale qui a le mérite principal d'établir une séparation nette entre le code destiné à encoder les parties graphiques des interfaces et le code destiné à encoder les comportements de ces mêmes interfaces. Cela permet un codage particulièrement compact des interfaces graphiques. Ces travaux sont décrits dans les articles [31][17].

6.3.4. FairThreads en Bigloo

Nous avons ajouté les FairThreads à Bigloo [33]. Les FairThreads sont des threads qui s'exécutent selon un schéma coopératif et progressent à la même cadence en se synchronisant au moyen de signaux. Ils ont une sémantique précise et déterministe, et sont donc faciles à programmer et à mettre au point. Les FairThreads supportent des opérations concurrentes pour traiter efficacement les services systèmes asynchrones (par exemple les entrées/sorties non bloquantes).

Les FairThreads n'imposent pas de contraintes sur l'implantation de la bibliothèque d'exécution du langage qui les accueille. En particulier, ils ne nécessitent pas de réentrance. En conséquence, ajouter des FairThreads dans un système ne dégrade pas les performances de ce dernier.

6.3.5. Récursion et sémantique des objets

Un effort a été entrepris par le passé, notamment dans le cadre des contrats « Sémantique des Objets Concurrents » avec l'université de Lisbonne et « Objets Migrants » avec France Télécom R&D, pour obtenir une formalisation de la programmation orientée objet, particulièrement importante dans le cadre des plates-formes réparties. L'objectif principal était d'obtenir un modèle autorisant l'inférence de type à la ML - ce qui n'est pas le cas des « calculs d'objets » à la Abadi-Cardelli ou Fisher-Mitchell par exemple. Un candidat naturel est le modèle proposé initialement par Cardelli, et repris par Wand en ce qui concerne le typage, où les concepts de la programmation objet sont dérivés de notions simples et traditionnelles, et où en particulier un objet est vu comme un *enregistrement récursif*. Toutefois, il est nécessaire d'adapter ce modèle à un langage en appel par valeur, pour pouvoir modéliser des objets possédant un état interne. Une difficulté spécifique se pose alors, qui est d'étendre le mécanisme de la récursion de ML, beaucoup trop restrictif - il ne permet, pour l'essentiel, de construire que des fonctions.

Le travail de Boudol sur ce sujet, décrit dans le rapport d'activité de l'an passé, et qui résolvait le problème posé par la récursion généralisée, sera prochainement publié dans le Journal of Functional Programming [28]. Ce travail laissait toutefois de côté la question de l'implémentation du calcul de la récursion, qui est non-standard. Nous avions certes montré que ce calcul ne pouvait conduire à une situation de blocage où la valeur

d'une variable récursive aurait été nécessaire pour le calcul de cette même valeur, mais ceci ne donnait pas d'indication sur la façon de compiler la récursion. Une analyse plus fine de ce problème a montré que les variables récursives sont seulement passées en argument à des fonctions (ce fait est établi dans la version à paraître du travail de Boudol), et a ouvert la voie à la conception d'une machine abstraite pour la compilation de la récursion. Cette machine est une extension de la machine de Krivine usuelle pour le λ-calcul en appel par valeur. La description de cette machine, ainsi que la preuve de sa correction, a fait l'objet de l'article [15]. Un prototype de cette machine a été implémenté et expérimenté par Pascal Zimmer.

Ce travail se poursuit actuellement dans la direction suivante : le calcul d'objets présenté dans l'article de Boudol peut être étendu en un calcul de *prototypes*, où un objet peut être réutilisé pour créer (par clonage et modification) de nouveaux objets. Malheureusement, si le codage semble bien marcher sur le plan opérationnel, le typage à la ML s'avère insuffisant : il n'offre pas assez de polymorphisme pour pouvoir typer certaines constructions de la programmations par « prototypes ». C'est pourquoi nous étudions maintenant le système de types avec intersection, qui est beaucoup plus puissant en termes de polymorphisme que le typage à la ML. Nous étudions actuellement le problème de l'inférence de types pour ce système.

6.4. Programmation réactive

Participants : Raul Acosta, Frédéric Boussinot, Christian Brunette, Dimitri Petoukhov, Jean-Ferdy Susini.

6.4.1. FairThreads

Les threads sont généralement considérés comme étant bien adaptés pour les systèmes constitués d'un petit nombre de tâches effectuant des calculs lourds, communiquant peu et exécutées dans un contexte préemptif. L'intérêt des threads préemptifs est moins clair dans le cas où l'on a un grand nombre de tâches nécessitant des synchronisations fortes et de nombreuses communications. Les threads purements coopératifs semblent mieux adaptés à ces cas. A l'inverse des threads préemptifs, les threads coopératifs ont une sémantique claire et sont donc plus faciles à utiliser. Cependant, les threads coopératifs ne peuvent pas profiter des machines multi-processeurs et nécessitent des mécanismes ad-hoc pour traiter les entrées-sorties. La programmation par threads est difficile car les threads ont une sémantique « faible ». C'est particulièrement vrai dans le cas des threads préemptifs car leur sémantique repose fortement sur les caractéristiques particulières du scheduler et de la politique de scheduling qu'il utilise.

C'est dans ce contexte que nous proposons une nouvelle sorte de threads, appelés « FairThreads », inspirés de l'approche réactive et possédant comme elle une sémantique formelle. Les FairThreads sont des threads coopératifs exécutés par un scheduler qui leur distribue le processeur équitablement. Les FairThreads communiquent par des événements diffusés. Ils peuvent être contrôlés finement, ce qui permet à l'utilisateur de coder ses propres stratégies d'exécution. La première version des FairThreads a été définie dans le contexte du langage JAVA. On a vu ci-dessus que les FairThreads ont été intégrés au langage Scheme. Nous avons également défini une version des fair threads en C [30]. Leur implémentation utilise la librairie pthread. Dans cette version, il est possible de mélanger les FairThreads coopératifs avec des threads natifs, soumis au contrôle préemptif de l'OS. Dans cette optique, les schedulers sont des serveurs de synchronisation auxquels les threads peuvent se lier dynamiquement au cours de leur exécution. Cette version des FairThreads permet de profiter des architectures multiprocesseurs : les divers schedulers ainsi que les threads qui ne sont liés à aucun scheduler peuvent être exécutés en réel parallélisme sur des processeurs distincts.

6.4.2. Rejo/Ros

Les travaux récents ont porté sur l'ajout de la propriété de persistence. Cette propriété consiste à sauver les données d'un agent mobile ainsi que son état d'exécution (partie réactive). Grâce à cette propriété le programmeur peut sauver des simulations et les reexécuter ultérieurement, et reprendre l'exécution des agents en cas de panne du système. Nous avons également travaillé sur l'implémentation des Icobjs en Rejo (Ricobjs). Les Icobjs permettent de programmer des comportements réactifs à l'aide d'une interface graphique. La programmation des Icobjs en Rejo a les avantages des deux formalismes, en permettant de programmer des agents qui ont une représentation graphique. Enfin nous avons étudié le contrôle d'accès des agents migrants

sur Ros, aboutissant à la proposition d'une nouvelle instruction pour garantir qu'un système coopératif coopère vraiment. Pour ces deux études nous avons travaillé sur le calcul du nombre d'instructions exécutées dans l'instant et l'exécution d'actions atomiques bornées en temps.

6.4.3. Icobjs et Physique

Une nouvelle version des Icobjs a été développée pour permettre une programmation plus souple des mécanismes de combinaison de comportements d'icobjs [16]. Cette version, mise en chantier dans le cadre du projet PING, a été utilisée par Damien Pous dans le cadre d'un stage pour implémenter diverses familles d'ambients.

Lors de son stage de DEA, Alexander Samarin a étudié comment utiliser la programmation réactive pour modéliser les phénomènes physiques. Plus précisément, il a implémenté certains algorithmes standards (Runge-Kutta,...) en utilisant les instants de l'approche réactive. Un ensemble de démos est disponible sur le Web en http://www.inria.fr/mimosa/rp/SimulationInPhysics.

6.4.4. Machines Virtuelles Réactives

Une notion de machine virtuelle réactive a été définie et implémentée en Java et en C. Les cibles sont les systèmes embarqués avec de faibles ressources, (PDA, GBA, etc). Une telle machine permet une programmation concurrente en absence de threads ou même d'OS. Elle devrait également être extrêmement efficace car très proche des ressources matérielles.

7. Contrats industriels

7.1. CTI FT-R&D : Objets Migrants

Participants : Raul Acosta, Roberto Amadio, Gérard Boudol, Frédéric Boussinot, Cédric Lhoussaine, Pascal Zimmer.

Les objectifs de ce contrat, qui a débuté en janvier 2000 pour une durée de 3 ans, sont, d'une part, de développer des techniques de modélisation, de spécification et de vérification en ce qui concerne les objets migrants, et d'autre part d'expérimenter la programmation et la validation d'agents mobiles, en se fondant sur le travail déjà accompli en ce qui concerne les objets et Scripts Réactifs. Nous portons un effort particulier sur les aspects de typage, aussi bien en ce qui concerne le modèle objet lui-même que pour ce qui touche aux aspects de migration.

7.2. Projet Rotor

Participants : Yannis Bres, Bernard Serpette [projet Oasis], Manuel Serrano.

L'objet de ce financement de Microsoft est le portage de Bigloo sur la plateforme d'exécution .NET. Pour ce travail, notre objectif est double. Il s'agit, d'une part, d'ajouter un nouveau générateur et une nouvelle bibliothèque d'exécution pour Bigloo. En ce sens, ce travail est comparable à celui que nous avons réalisé pour la JVM. Mais il s'agit également de parvenir à « intégrer » Bigloo dans l'environnement de développement de Microsoft, c'est-à-dire en adoptant la « philosophie » de développement de Microsoft. Ce contrat porte sur une durée d'un an.

8. Actions régionales, nationales et internationales

8.1. Actions nationales

8.1.1. Actions VERNAM et AZURCRYPT

Participants : Roberto Amadio, Vincent Vanackère.

Nous coordonnons l'Action Concertée Incitative Cryptologie VERNAM sur la Vérification automatique de protocoles cryptographiques. L'action comprend le projet PROTHÉO de l'INRIA-Lorraine et l'ENS-Cachan. Nous participons à l'ACI AZURCRYPT, avec l'IML de Marseille et l'Université de Toulon.

8.1.2. Action Méthodes Formelles pour la Mobilité

Participants : Roberto Amadio, Silvano Dal-Zilio.

Il s'agit d'une Action Spécifique Math-STIC, à laquelle participent des équipes du LIENS (ENS Ulm), du LIP (ENS Lyon) du LIPN et de PPS (Paris). Le projet MIMOSA est concerné par les directions « sécurité » (analyse de la complexité des programmes mobiles) et « données semi-structurées » de cette action.

8.2. Actions européennes

8.2.1. Projet IST MIKADO

Participants : Gérard Boudol, Ilaria Castellani, Xudong Guan, Ana Matos, Pascal Zimmer.

Le projet MIKADO regroupe l'INRIA (Rhone-Alpes, projet SARDES, et Sophia Antipolis, projet MIMOSA), contractant principal, France Télécom R&D, les universités de Sussex, Lisbonne, Florence et Turin, et l'Ecole Nationale Supérieure des Télécommunications (sous-traitant de l'INRIA). Ce projet du programme FET Global Computing a pour objectifs l'étude de modèles de programmation pour les applications réparties et mobiles, la conception de nouvelles constructions pour ce type de programmation, l'étude des formalismes de spécification et d'analyse relatifs aux modèles considérés, la conception de machines virtuelles prototypes.

8.2.2. Projet IST PROFUNDIS

Participants : Roberto Amadio, Silvano Dal-Zilio, Yuxin Deng, Davide Sangiorgi.

Dans le projet PROFUNDIS nous avons pour partenaires le KTH de Stockholm (contractant principal), la Faculté des Sciences et Technologies de Lisbonne et l'Université de Pise. Ce projet du programme FET Global Computing a pour objectifs l'étude des modéles formels et techniques de preuves (logiques, équivalences comportementales, systèmes de types) pour les systèmes mobiles.

8.3. Visites, et invitations de chercheurs

Nous avons eu la visite d'António Ravara, de l'Instituto Superior Técnico de Lisbonne pour six mois, et de Catuscia Palamidessi, pour quelques jours. Rémi Douence et Mario Sudholt, de l'Ecole des Mines de Nantes, nous ont présenté leurs travaux sur la programmation par aspects. Christian Queinnec nous a donné un exposé sur la programmation de sites web.

9. Diffusion des résultats

9.1. Animation de la communauté scientifique

• Roberto Amadio est responsable de l'équipe MOVE (Modélisation et Vérification) du Laboratoire d'Informatique Fondamentale de Marseille (CNRS, FRE2246). Il était co-président du comité d'organisation et membre du comité scientifique des semaines *Logique et Interaction* (CIRM, Marseille), membre du comité de programme de ICALP'02 (track B) et CONCUR'02. Il était membre du jury de thèse de Cédric Lhoussaine. Il a participé aux conférences FLOC'02 (Copenhague) et CONCUR'02 (Brno).
• Gérard Boudol a participé aux conférences ETAPS et FLOC, ainsi qu'à l'Ecole de Printemps « Sémantique des Langages de Programmation ». Il était rapporteur de l'Habilitation à Diriger des Recherches de Giuseppe Castagna, et membre du jury de l'Habilitation à Diriger des Recherches de Davide Sangiorgi, et de la thèse de Cédric Lhoussaine. Il a co-organisé, avec Ilaria Castellani, deux réunions du projet MIKADO à Sophia, et participé aux réunions de ce projet tenues à Lisbonne, où il a donné un exposé sur [15], et Florence. Il a donné un exposé invité à l'Université de Turin, sur le modèle de la programmation par mixins.

• Frédéric Boussinot a été rapporteur de l'Habilitation à Diriger des Recherches de Michel Augeraud (La Rochelle) et des thèses de Philippe Boinot (Rennes) et Grégoire Hamon (Paris 6). Raul Acosta, Frédéric Boussinot, Christian Brunette, Dimitri Petoukhov et Jean-Ferdy Susini ont participé au workshop Synchron'02.
• Christian Brunette a participé à la conférence OOPSLA'02, où il a présenté son travail [16], dans le cadre du workshop DSVL.
• Ilaria Castellani a été membre du jury des thèses de Cédric Lhoussaine et de Sylvain Conchon. Elle a participé aux semaines « Logique et Interaction », et à l'Ecole de Printemps « Sémantique des Langages de Programmation ». Elle a également co-organisé les réunions du projet Mikado, à Sophia en février et septembre, et participé à celles de Lisbonne et Florence. Enfin elle a participé à l'atelier du CNRS « Sexes et Genres dans le Travail Scientifique ». Elle a organisé les séminaires croisés des doctorants de l'INRIA Sophia.
• Silvano Dal-Zilio est responsable d'une ATIP « jeunes chercheurs » du CNRS, sur les fondements de l'interrogation de données semi-structurées. Il est co-responsable du séminaire d'informatique du Laboratoire d'Informatique de Marseille, et responsable de l'organisation des événements satellites à la conférence CONCUR'03. Il a participé à l'Ecole de Printemps « Sémantique des Langages de Programmation ».
• Davide Sangiorgi a été membre du comité de programme des conférences et workshops JFLA'02, FWAN, FTRTFT'02, FST-TCS'02, POPL'03. Il a donné un cours à l'Ecole d'Eté sur les Types, et un exposé invité à la conférence CONCUR'02. Il a organisé la première réunion du projet PROFUNDIS à Sophia. Davide Sangiorgi et Pascal Zimmer ont créé et maintiennent la liste « moca » (mobile calculi) qui est un forum de discussions sur les modèles de la mobilité.
• Manuel Serrano a donné deux exposés sur les FairThreads, le premier à Northeastern University de Boston où il était invité par le Pr. M. Felleisen, le second lors d'un exposé invité à la conférence *International Lisp Conference 2002*, à San Francisco, USA. Manuel Serrano a, par ailleurs, présenté ses travaux sur la programmation des interfaces graphiques à la conférence *Usenix 2002 Technical Annual Conference, Freenix Track* à Monterey, USA, ses travaux sur Scribe au *3e Scheme Workshop* et ses travaux sur la compilation des langages fonctionnels vers JVM à la conférence *International Conference on Functional Programming 2002* à Pittsburgh, USA. Il poursuit ses travaux auprès de la communauté Scheme, d'une part en étant membre du comité de direction du Workshop Scheme et d'autre part en étant membre du nouveau comité de réflexion sur la nouvelle norme de Scheme (R^6Rs).
• Jean-Ferdy Susini a présenté ses travaux à l'Ecole des Mines de Nantes (équipes de P. Cointe et J.-M. Muller).
• Vincent Vanackère a participé à l'Ecole de Printemps « Sémantique des Langages de Programmation », et à la conférence FLOC'02, où il a donné un exposé sur [23], dans le cadre du workshop VERIFY.
• Pascal Zimmer a participé aux conférences ETAPS'02, FLOC'02, où il a donné un exposé sur [15] (dans le cadre du workshop FICS), et CONCUR'02. Il a également participé à l'Ecole de Printemps « Sémantique des Langages de Programmation », et donné un exposé au séminaire croisé des doctorants de l'INRIA Sophia sur [22].

9.2. Enseignements

Raul Acosta-Bermejo, Frédéric Boussinot et Jean-Ferdy Susini (avec la participation de Laurent Hazard) ont donné un cours intitulé *Objets réactifs en JAVA*. Ce cours regroupait des étudiants des DEA RSD et Informatique de l'UNSA (Université de Nice-Sophia Antipolis) et des étudiants de 3ème année de l'ESSI. Frédéric Boussinot et Jean-Ferdy Susini ont co-encadré le stage de DEA de Alexander Samarin.

Frédéric Boussinot et Manuel Serrano ont co-encadré le stage de DEA de Pascal Fauconnier sur la programmation par icobjs en Senior. Bernard Serpette et Manuel Serrano ont co-encadré le stage de DEA de Damien Ciabrini sur la mise au point dans les langages évolués.

Frédéric Boussinot et Pascal Zimmer ont co-encadré le stage de licence de Damien Pous sur l'implémentation des Ambients en Icobj. Gérard Boudol et Ilaria Castellani ont co-encadré le stage de licence de Thomas Gazagnaire sur l'inférence de types pour la non-interférence.

Pascal Zimmer enseigne à l'Université de Nice, en TP de « Algorithmique et Programmation », DEUG MASS 1ère année, en TD de « Informatique Théorique », DEUG MI 2ème année.

Silvano Dal-Zilio est intervenu dans le cours « Typage et Vérification » du DEA d'Informatique de Marseille et dans le cours de programmation Java du DESS Génie Informatique et Statistique de l'université de Provence (Aix Marseille I). Il a encadré le stage de Sofiane Bensadi, étudiant dans le DEA d'Informatique de Marseille.

10. Bibliographie

Bibliographie de référence

[1] R. AMADIO, P.-L. CURIEN. *Domains and Lambda-Calculi.* Cambridge University Press, 1998.

[2] G. BERRY, G. BOUDOL. *The chemical abstract machine.* in « Theoretical Computer Science », volume 96, 1992.

[3] G. BOUDOL. *The π-calculus in direct style.* in « Higher-Order and Symbolic Computation », volume 11, 1998.

[4] F. BOUSSINOT. *Objets réactifs en Java.* Collection Scientifique et Technique des Telecommunications, PPUR, 2000.

[5] F. BOUSSINOT. *La programmation réactive.* Masson, 1996.

[6] F. BOUSSINOT, J.-F. SUSINI. *Java threads and SugarCubes.* in « Software Practice & Experience », volume 30, 2000.

[7] I. CASTELLANI. *Process Algebras with Localities.* éditeurs J. BERGSTRA, A. PONSE, S. SMOLKA., in « Handbook of Process Algebra », North-Holland, Amsterdam, 2001, pages 945-1045.

[8] D. SANGIORGI, D. WALKER. *The π-Calculus : a Theory of Mobile Processes.* Cambridge University Press, 2001.

[9] M. SERRANO. *Bee : an Integrated Development Environment for the Scheme Programming Language.* in « Journal of Functional Programming », numéro 2, volume 10, mai, 2000, pages 1-43.

Thèses et habilitations à diriger des recherche

[10] C. LHOUSSAINE. *Réceptivité, Mobilité et π-Calcul.* thèse de doctorat, Université d'Aix-Marseille 1, juin, 2002.

Articles et chapitres de livre

[11] R. ACOSTA-BERMEJO. *La programmation en Rejo.* in « Les Intergiciels, Développements Récents dans CORBA, Java RMI et les Agents Mobiles (Eds. I. Demeure et E. Najm), Chap. 5, Hermes », 2002.

[12] R. AMADIO, C. MEYSSONNIER. *On decidability of the control reachability problem in the asynchronous π-calculus.* in « Nordic Journal of Computing », volume 9(2), 2002, pages 70-101.

[13] G. BOUDOL, I. CASTELLANI. *Non-interference for concurrent programs and thread systems.* in « in « Merci, Maurice, A mosaic in honour of Maurice Nivat » (P.-L. Curien, Ed.), Theoretical Computer Science », volume 281(1), 2002, pages 109-130.

Communications à des congrès, colloques, etc.

[14] R. AMADIO, W. CHARATONIK. *On name generation and set-based analysis in Dolev-Yao model.* in « CONCUR'02 », série Lecture Notes in Computer Science, numéro 2421, pages 499-514, 2002.

[15] G. BOUDOL, P. ZIMMER. *Recursion in the call-by-value λ-calculus.* in « Fixed Points in Computer Science (FICS'02) », 2002.

[16] C. BRUNETTE. *A visual reactive framework for dynamic behavior creation.* in « DSVL, OOPSLA'02 », 2002.

[17] E. GALLESIO, M. SERRANO. *Biglook : a Widget Library for the Scheme Programming Language.* in « 2002 Usenix annual technical conference, Freenix track », pages 85-97, Monterey, Californie, USA, juin, 2002.

[18] D. HIRSCHKOFF, E. LOZES, D. SANGIORGI. *Separability, expressiveness and decidability in the Ambient logic.* in « LICS'02 », pages 423-432, 2002.

[19] D. SANGIORGI. *Types, or : where is the difference between CCS and π ?.* in « CONCUR'02 », série Lecture Notes in Computer Science, numéro 2421, pages 76-97, 2002.

[20] B. SERPETTE, M. SERRANO. *Compiling Scheme to JVM bytecode : a performance study.* in « 7th ICFP », Pittsburgh, Pensylvanie, USA, octobre, 2002.

[21] M. SERRANO, E. GALLESIO. *This is Scribe !.* in « 3th ACM SIGPLAN Workshop on Scheme and Functional Programming », Pittsburgh, Pennsylvania, USA, juin, 2002.

[22] D. TELLER, P. ZIMMER, D. HIRSCHKOFF. *Using Ambients to control resources.* in « CONCUR'02 », série Lecture Notes in Computer Science, numéro 2421, pages 288-303, 2002.

[23] V. VANACKERE. *The TRUST protocol analyser, automatic and efficient verification of cryptographic protocols.* in « VERIFY'02 », 2002.

Rapports de recherche et publications internes

[24] S. DAL-ZILIO, D. LUGIEZ. *XML Schema, Tree Logic and Sheaves Automata.* rapport technique, INRIA, 2002, http://www.inria.fr/rrrt/rr-4631.html.

[25] C. LHOUSSAINE. *Type Inference for the receptive distributed π-calculus.* rapport technique, numéro 4373, INRIA, 2002, http://www.inria.fr/rrrt/rr-4373.html.

Divers

[26] R. AMADIO, G. BOUDOL, C. LHOUSSAINE. *On message deliverability and non-uniform receptivity.* to appear in Fundamenta Informaticae.

[27] R. AMADIO, D. LUGIEZ, V. VANACKÈRE. *On the symbolic reduction of processes with cryptographic functions.* to appear in Theoretical Computer Science.

[28] G. BOUDOL. *The recursive record semantics of objects revisited.* to appear in the Journal of Functional Programming.

[29] G. BOUDOL, I. CASTELLANI, F. GERMAIN, M. LACOSTE. *Models of distribution and mobility : state of the art.* delivrable D.1.1.1, projet MIKADO (I. Castellani, Ed.).

[30] F. BOUSSINOT. *FairThreads : mixing cooperative and preemptive threads in C.* submitted for publication.

[31] E. GALLESIO, M. SERRANO. *Programming Graphical User Interfaces with Scheme.* to appear in the Journal of Functional Programming.

[32] X. GUAN, J. YOU. *Encoding channels in typed Ambients.* submitted for publication.

[33] M. SERRANO, F. BOUSSINOT, B. SERPETTE. *Event-driven Architectures meet Threads, a programming language perspective.* submitted for publication.