Modèles de programmation non conventionnels

Les paradigmes de programmation fonctionnel, objet ou logique n'ont toujours pas complètement répondu aux attentes formulées dans les années 1980 et 1990 : réutilisation, modularité, correction, expressivité, évolution, flexibilité, distribution, portabilité, parallélisme restent toujours au centre des préoccupations des concepteurs de logiciels.

Dans le même temps, de nouveaux besoins, de nouvelles applications et de nouvelles opportunités de support matériel remettent en cause la notion traditionnelle de programme monolithique, conçu, produit et finalisé avant son utilisation finale. Les logiciels sont de plus en plus perçus comme des entités autonomes, ouvertes qui doivent faire face à la prolifération des environnements d'exécution, qui doivent s'adapter et intégrer dynamiquement de nouvelles fonctions au sein d'une interface commune.

Ces constatations ont motivé la tenue d'un séminaire international sur les paradigmes de programmation non conventionnels (Unconventional Programming Paradigms, UPP04 : http://upp.lami.univ-evry.fr) au Mont Saint-Michel du 15 au 17 septembre 2004.

Ce séminaire, sur invitation, était organisé dans le cadre d'une collaboration entre l'Union européenne (IST[1], Future and Emerging Technologies) et la NSF[2]. L'organisation a été prise en charge par l'ERCIM[3] . Il a bénéficié du concours de l'INRIA[4], de l'université d'Evry[5], de l'université de Rennes[6], de Microsoft Research[7], du LaMI[8], de l'IRISA[9], de Genopole-Evry[10] et du CNRS[11].

---

1. Information Society Technology : http://www.cordis.lu/ist/
2. National Science Foundation : http://www.nsf.gov/
3. European Research Consortium for Informatics and Mathematics : http://www.ercim.org/
4. http://www.inria.fr/
5. http://www.univ-evry.fr/
6. http://www.univ-rennes1.fr/
7. http://research.microsoft.com/
8. http://www.lami.univ-evry.fr/
9. http://www.irisa.fr/
10. http://www.genopole.org/
11. http://www.cnrs.fr/

L'objectif du séminaire était donc de faire le point sur les résultats récents concernant les nouveaux modèles de programmation. Ces nouveaux paradigmes sont suscités par des technologies nouvelles de calcul (bio-molécules, MEMS, grilles) et de nouvelles problématiques (modélisation en biologie, systèmes distribués auto-réparateurs, calcul robuste, programmation générique) qui renouvellent l'idée conventionnelle de programme.

Cinq grand domaines thématiques ont été identifiés :

– modèles amorphes (*amorphous computing*) ;

– modèles autonomes (*autonomic computing*) ;

– programmation générative (*generative programming*) ;

– modèles biologiques (*biocomputing*) ;

– modèles chimiques (*chemical computing*).

Le principe de ces rencontres, repris d'un fructueux séminaire international sur les nouvelles directions dans les langages parallèles de haut niveau[12], est d'inviter l'un des acteurs reconnus au niveau international dans chacun de ces domaines à organiser une demi-journée en sollicitant quatre à cinq contributions.

Les participants et la liste des exposés présentés sont accessibles à partir de la page du colloque[13]. Un volume rassemblant les contributions paraîtra aux LNCS courant 2005. Aussi, ne présenterons-nous pas ici un résumé des travaux de recherche abordés lors de ce séminaire ; nous préférons détailler, sous la forme d'une liste les motivations, les différentes sources d'inspiration et les problèmes qui justifient le développement de ces nouvelles idées dans le domaine des langages de programmation.

Les métaphores du calcul

Les paradigmes de programmation sont contraints par les particularités d'un modèle matériel[14] ou bien inspirés par une métaphore de ce que doit être un calcul. Aujourd'hui, de nouveaux supports matériels du calcul sont étudiés. On connaît l'expérience d'Adleman, en 1994, qui démontre que l'on peut résoudre un problème d'optimisation combinatoire (le voyageur de commerce) dans une éprouvette avec des molécules d'ADN. Mais d'autres possibilités font l'objet de recherches très actives : on espère ainsi utiliser la croissance de colonie de bactéries, la diffusion de

12. Lecture Notes in Computer Science, Vol. 574, 1991.

13. http://upp.lami.univ-evry.fr

14. « While it is perhaps natural and inevitable that languages like Fortran and its successors should have developed out of the concept of the von Neumann computer as they did, the fact that such languages have dominated our thinking for twenty years is unfortunate. It is unfortunate because their long-standing familiarity will make it hard for us to understand and adopt new programming styles which one day will offer far greater intellectual and computational power. » John Backus (1981).

réactifs chimiques, l'auto-assemblage de biomolécules… pour calculer. La programmation de ces nouveaux supports d'exécution pose des problèmes certains et motive le développement de langages et d'une algorithmique adaptés.

Mais par ailleurs, les mécanismes d'un langage de programmation, ou de nouveaux algorithmes, peuvent être directement inspiré par une métaphore, sans faire référence à un support d'exécution. Par exemple, les algorithmes évolutionnistes s'inspirent des mécanismes étudiés par la théorie de l'évolution même si nous ne disposons pas de calculateur pouvant réaliser directement ces mécanismes.

On doit donc distinguer les langages de programmation inspirés par *xxx* et les langages de programmation qui serviront à programmer les calculateurs fondés sur *xxx* (remplacer *xxx* par : la chimie, la biologie cellulaire, la théorie des quanta, etc.). Certes, ces deux axes de recherche partagent des problématiques communes et sont en dialogue constant[15], mais chacun s'attaque aussi à des problèmes spécifiques. C'est un point à ne pas oublier dans les considérations qui suivent.

Revenons sur les métaphores qui ont inspiré le développement de langages de programmation, indépendamment d'un support matériel. Quelques exemples sont donnés par la machine à écrire inspirant la machine de Turing ; le bureau, les ciseaux et la corbeille à papier pour les interfaces-utilisateurs ; les notions de classifications et d'ontologies pour les langages à objets, l'architecture des bâtiments pour les patrons de conceptions ; les théories métamathématiques (par exemple le lambda-calcul) pour les langages fonctionnels, etc.

Si l'on considère l'histoire des langages de programmation, il semble que les métaphores qui ont « réussi » dans le logiciel sont majoritairement fondées sur des activités humaines ou sociales, aussi bien concrètes comme l'architecture, qu'abstraites comme les mathématiques ou les activités de bureau.

Aujourd'hui, nous pouvons constater une grande effervescence autour de modèles de programmation inspirés par les « métaphores naturelles » issues de la biologie, de la physique et de la chimie. Citons comme exemple la chimie artificielle, le calcul par ADN, le calcul quantique, les systèmes de Păun, les algorithmes évolutionnistes, le calcul cellulaire, le recuit simulé, l'optimisation par essaim ou par colonie de fourmis, les algorithmes mémétiques, etc. Si ces domaines sont relativement nouveaux, cela ne veut pas dire que la métaphore naturelle ait été absente de l'histoire de l'informatique. Bien au contraire, les réseaux de neurones

15. Par exemple, les automates cellulaires ont été inspirés initialement par une abstraction des cellules biologiques dans un tissu avec la préoccupation de développer un modèle formel de l'auto-réplication. Plus tard, le modèle formel des automates cellulaires a inspiré le développement d'architectures matérielles parallèles, en partie avec la motivation d'offrir un support d'exécution efficace pour ce type de programme. Et ces architectures se sont révélées efficaces pour résoudre des systèmes d'équations aux dérivées partielles qui pouvaient, par exemple, décrire la diffusion d'un morphogène à travers les cellules d'un tissu biologique.

formels ou bien les automates cellulaires, inspirés par des processus biologiques, sont des notions élaborées de longue date. Cependant, ce hiatus entre le faible impact jusqu'à présent des métaphores naturelles dans les langages de programmation et leur soudain développement soulève plusieurs questions :

– Qu'attendons-nous des métaphores naturelles que nous ne trouvons pas dans les métaphores des activités humaines ? À quels besoins répondent-elles ?

– Quels sont les liens entre la physique et la notion de calcul ? La physique détermine évidemment les phénomènes que nous pouvons utiliser pour « faire du calcul » (les calculateurs et leurs architectures matérielles). Cependant, en quoi la physique peut inspirer de nouvelles notions de *programmation* ? Par exemple, quel est l'impact sur la programmation des célèbres cours de Feynman sur la physique du calcul ? Quelles sont les leçons que l'on peut tirer des calculateurs analogiques développés dans les années 1950 ?

– Quels sont les liens entre biologie et calcul ? La biologie est à l'évidence une grande source d'inspiration pour de nouveaux modèles de calcul. Les informaticiens recherchent activement des principes de conception leur permettant de développer des systèmes qui exhiberaient des propriétés habituellement attribués au vivant : l'autonomie, l'adaptabilité, l'auto-guérison, la robustesse, l'auto-organisation, etc. Cependant, sommes nous certain que ces termes employés à la fois en biologie et en Informatique recouvrent les mêmes propriétés ? Par exemple, en biologie, ces propriétés sont souvent exhibées au niveau d'une population sur le long terme et sur une grande échelle, *i.e.* au niveau d'une espèce et non pas au niveau d'un individu. Qu'une espèce soit robuste vis-à-vis des variations de son environnement n'implique pas que l'individu soit lui robuste ou adaptable.

– Avons-nous épuisé les métaphores des activités humaines (ingénieries, économie, organisation sociale, littérature, mathématique...) ? Par exemple, dans le domaine des inspirations de nature mathématique, la logique et les métamathématiques sont fortement couplées à l'informatique. L'apport de la géométrie ou de la topologie semble faible alors même que ces domaines ont très largement inspirés d'autres domaines (tout le vingtième siècle montre par exemple la géométrisation progressive de la physique).

– Le monde physique est-il une source pertinente d'inspiration ? Ou pour reformuler cette question, est-ce que les relations entre objets physiques constituent un cadre adapté à l'expression des relations entre des entités immatérielles comme le logiciel ou les calculs ? Par exemple, les langages synchrones font l'hypothèse que les réactions aux événements sont instantanées. Malgré la violation apparente des lois de la physique, ce modèle est particulièrement utile pour concevoir et raisonner sur les logiciels temps réel.

Programmation algorithmique et programmation des systèmes

La « programmation algorithmique » est le terme que nous avons retenu pour la traduction de l'expression anglaise « programming in the small », activité qui

s'oppose à « programming in the large » que nous traduirons par « programmation des systèmes ». La distinction entre les deux types de programmation a été introduite par Deremer et Kron dans un article célèbre[16]. La programmation algorithmique est celle qui est enseignée à nos étudiants et le slogan

programme = structure de données + algorithmes

a forgé notre conception de ce que doit être un programme. Cependant, à la lumière des changements actuels, il est permis de se poser quelques questions :

– Ce slogan est-il toujours pertinent pour les nouveaux modèles de programmation (face aux nouveaux problèmes et pour répondre aux nouvelles applications) ?

– Quelles sont les nouvelles structures de données offertes par la programmation chimique, le calcul cellulaire, et les autres nouveaux modèles de programmation ?

– Les nouveaux modèles de calculs apportent-ils de nouveaux algorithmes ou ne proposent-ils qu'une accélération de mécanismes déjà existants ?

Les structures de contrôle sont les mécanismes par lesquels nous organisons un ensemble de calculs élémentaires. La structuration des calculs naturels « in vivo » semble difficile : comment par exemple réaliser le séquencement de deux réactions chimiques dans un tube à essai (comment faire démarrer une réaction chimique seulement après que l'état d'équilibre d'une première réaction a été atteint) ? Comment réaliser une boucle dans un calcul quantique ? Remarquons que la question de la structuration des calculs ne se pose pas vraiment dans les langages de programmation qui s'inspirent d'un phénomène naturel sans reposer sur un support dédié, sans doute parce que ces nouveaux mécanismes de programmation sont alors le plus souvent intégrés à un langage classique.

Mais la distinction introduite par P. J. Landin[17] reste utile. Landin distingue d'une part les mécanismes permettant de spécifier de nouvelles structures de données et les opérations primitives afférentes et, d'autre part, les mécanismes et les notions permettant de combiner ces opérations et d'exprimer des relations fonctionnelles entre ces données. Dans cette deuxième catégorie, dédiée (pour reprendre les termes mêmes de Landin) à « l'expression de choses en fonction d'autres choses indépendamment de la nature de ces choses », on peut trouver par exemple la notion d'*identificateur* et les *mécanismes de liaison* qui permettent de préciser comment ces noms sont introduits, définis puis utilisés. Le choix de structures de données et des opérations afférentes permet de définir de nouveaux supports de calcul (si ces opérations correspondent directement à un phénomène physique que l'on va utiliser « in vivo »), des API (Application Programming Interface) appropriés à un domaine d'application ou encore des langages de

16. *Programming In the Large Versus Programming In the Small*, IEEE Trans. On Software Engineering, juin 1976.
17. "The Next 700 Programming Languages", *Communications of the ACM*, 9(3), 1966.

programmation spécifiques (appelés encore langages dédiés ou « orientés problèmes »). Les mécanismes puissants de composition permettent eux d'obtenir des langages de programmation souples, concis, expressifs, adaptables, facilitant la réutilisation, etc. Aussi :

– Quelles sont les nouvelles structures de contrôle apportées par les nouveaux paradigmes de programmation ?

– Est-ce que les nouveaux modèles de calcul correspondent à des structures de données dédiées à un domaine d'application spécialisé ou bien offrent-ils de nouvelles manières d'exprimer des choses en fonction d'autres choses ?

La recherche sur le calcul biologique et le calcul quantique se concentre actuellement majoritairement sur des questions de complexité (quel est le coût en temps et en espace d'un tri, d'une factorisation en nombres premiers, …, si l'on suppose que telle opération peut se faire avec tel coût). Ils visent donc *in fine* à l'utilisation d'un support matériel dédié permettant d'abaisser la complexité d'un algorithme.

Cependant, les métaphores naturelles peuvent aussi servir à renouveler la programmation des systèmes en suggérant de nouvelles approches dans le développement de très grands logiciels, pour la spécification de leur architecture, l'interconnexion de leurs parties, en offrant de nouveaux mécanismes pour cacher l'information inutile, abstraire les détails ou capitaliser et réutiliser le code. Par exemple, un langage comme Gamma, inspiré par une métaphore chimique, a été utilisé comme langage de coordination et la CHAM (chemical abstract machine) comme cadre théorique pour exprimer la sémantique de processus non déterministes. La programmation des systèmes est très certainement un des principaux challenges auxquels les langages de programmation doivent faire face. Et toute nouvelle source d'inspiration est précieuse car, selon certains, les avancées de la programmation ces deux dernières décennies n'ont finalement pas permis de progresser sur ce point :

Computer Science is in deep trouble. Structured design is a failure. Systems, as currently engineered, are brittle and fragile. They cannot be easily adapted to new situations. Small changes in requirements entail large changes in the structure and configuration. Small errors in the programs that prescribe the behavior of the system can lead to large errors in the desired behavior. Indeed, current computational systems are unreasonably dependent on the correctness of the implementation, and they cannot be easily modified to account for errors in the design, errors in the specifications, or the inevitable evolution of the requirements for which the design was commissioned. (Just imagine what happens if you cut a random wire in your computer!) This problem is structural. This is not a complexity problem. It will not be solved by some form of modularity. We need new ideas. We need a new set of engineering principles that can be applied to effectively build flexible, robust, evolvable, and efficient systems. (Gerald Jay Sussman, 1999.)

Cet échec, qu'il conviendrait tout de même de relativiser, pointé par G. J. Sussman, mais dénoncé aussi de façon trop provocatrice, par P. Gabriel dans un débat célèbre[18], amène les questions suivantes :

– Quelle est la cause de cet échec relatif : nos langages de programmation, nos méthodes de développement, nos outils de génie logiciel ?

– Pourquoi les nouveaux paradigmes de programmation, et en particulier ceux inspirés par la biologie, permettraient de répondre aux problèmes de fragilité et d'inflexibilité des logiciels actuels ?

– Quels sont les concepts et les mécanismes dans ces nouveaux langages de programmation qui permettent de (mieux) découvrir/localiser/corriger les erreurs de programmation ou bien d'atténuer leur portée et de « vivre avec » ?

Sémantique et construction des programmes

L'influence de la logique a été primordiale dans le développement de la sémantique des langages de programmation. Cependant, les nouveaux modèles de programmation mettent tous en avant la notion de *systèmes dynamiques*.

– Quel est « le bon » cadre mathématique permettant de construire et manipuler la notion de système dynamique en accord avec nos conceptions de ce que doit être une architecture logicielle ?

– Que peut-on attendre d'un mariage entre la théorie du contrôle des systèmes et de l'informatique théorique ?

– Si l'on considère la nature distribuée des nouvelles ressources de calculs et des nouvelles applications, pourra-t-on développer une théorie utile et éclairante des systèmes dynamiques sans temps ou sans état global ?

– Est-ce que les nouveaux modèles de programmation et les concepts de la théorie des systèmes dynamiques nous permettront de développer une notion de calcul approché, de calcul probabiliste, de calcul flou ou de calcul non déterministe qui permettent d'aller plus loin que le traitement actuel de ces caractéristiques ? Sont-ils plus à même de traiter les informations incertaines et incomplètes ?

– Est-il possible de définir une notion utile et pratique de *systèmes ouverts* ? Quels sont les mécanismes de « l'ouverture » ? Comment maintenir la cohérence et l'adéquation d'un système ouvert ?

L'objectif d'un programme est de résoudre un problème et, partant, de réaliser un certain calcul. Pour s'assurer qu'un programme réalise bien la tâche assignée, nous avons développé des méthodes et outils variés : le typage, l'analyse statique, l'interprétation abstraite, la bisimulation, le model-checking, etc.

---

18. Voir le débat « Object have failed » organisé par R. Gabriel à OOPSLA 2002. Des commentaires sont disponibles sur le site http://www.dreamsongs.org.

– Est-ce que ces techniques et outils sont applicables aux nouveaux programmes ? Quel est le type d'un brin d'ADN dans un tube à essai ? Quelle est la « correction par construction » d'un programme amorphe[19] ? Comment réaliser le model-checking d'un P-système[20] ?

– Les techniques évoquées ci-dessus partagent la même approche : établir (aussi efficacement et automatiquement que possible) certaines assertions logiques sur (l'exécution d') un programme. Cette approche impose des limites sur le type d'assertions qui peuvent être prouvées. De plus, nous ne voulons pas que l'expression de ces assertions soit plus complexe que le programme lui-même ou que les établir soit plus difficile que de développer le programme lui-même. Les nouveaux paradigmes de programmation offrent-ils des alternatives à cette approche ? Un exemple pourra éclairer cette question : plutôt que d'assurer la correction « statique » et *a priori* du programme, on peut envisager de corriger (ou mieux, que le programme se corrige lui-même) incrémentalement jusqu'à obtenir *in fine* l'accomplissement de la tâche prévue. Cette approche est clairement reliée à des notions comme l'apprentissage, l'évolution, l'auto-organisation, l'auto-maintenance. En quoi les nouveaux modèles de programmation permettent-ils de mettre en œuvre ces nouvelles approches ?

– Plus généralement, comment ces nouveaux modèles permettent-ils d'assister le programmeur dans la construction, la compréhension, la correction, l'amélioration, le test et la réutilisation de logiciels ?

Pour de nombreuses raisons, la notion de programme monolithique, fonctionnant de manière isolée, écrit par un seul programmeur ou bien par un groupe de programmeur fortement couplé, est de moins en moins adaptée à la réalité des développements logiciels et aux demandes des applications. Après l'utilisation d'outils de préprocessing et de générateurs de code, nous avons inventé des concepts comme l'édition de liens dynamiques, les patrons, la compilation multiphase (*multi-stage compilation*), la programmation par aspects et le tissage, la compilation juste à temps, les mises à jour automatiques, les technologie push et pull, les supports de déploiement et « SourceForge »[21] pour le développement coopératif. Aussi, en quoi les nouveaux modèles de programmation vont-ils dans le sens de ces nouveaux modèles de développement, ces nouvelles approches du cycle de vie du logiciel ?

Nouvelles applications et nouvelles ressources de calcul

Nos langages de programmation reflètent majoritairement le paradigme séquentiel « Von Neumann » à travers la modification d'un état global. Cela est vrai aussi au niveau du matériel, même dans nos machines parallèles : nous partitionnons nos unités de calcul en une unité de traitement, toujours plus rapide et une unité de

19. http://www.swiss.ai.mit.edu/projects/amorphous/
20. http://psystems.disco.unimib.it
21. http://sourceforge.net

stockage, une mémoire toujours plus grande mais toujours accédée séquentiellement.

Si cette partition était adaptée aux débuts de l'informatique, cette architecture risque fort d'être incapable d'utiliser la capacité toujours croissante des ressources matérielles : 10^9 transistors seront accessibles sur un chip en 2007. Les développements de circuits tridimensionnels, les nouvelles ressources offertes par les avancées des nanotechnologies ou du bio-computing peuvent potentiellement multiplier par mille ces chiffres.

– Est-ce que les paradigmes de programmation discutés ici peuvent tirer profit de ces nouvelles puissances de calcul ? Suggèrent-ils de nouvelles architectures matérielles ? Même en restant dans le domaine des circuits en silicium, pourrons-nous exploiter ces « océans de transistors » ?

– L'avancée des nanotechnologies et de l'ingénierie biologique peut permettre le développement de nouveaux media de calculs fondés sur les molécules. L'utilisation de l'ADN et de l'ARN pour implanter des calculs massivement parallèles et résoudre des problèmes NP-difficile est un domaine de recherche très actif. Il est aussi possible de construire des formes bi- et tridimensionnelles à l'échelle de la molécule, ouvrant de nombreuses perspectives dans l'assemblage des nanostructures moléculaires. Cependant, l'informatique a déjà été confrontée à des situations aussi prometteuses par le passé, sans que les résultats tangibles ne se présentent réellement. Par exemple, en utilisant les capacités de l'optoélectronique, il est possible d'implanter des opérations complexes comme la transformée de Fourrier (FFT) ou bien le routage de signaux, à un coût dérisoire. Mais jusqu'à présent, l'impact de l'optoélectronique sur nos calculateurs a été marginal. Une explication possible vient du fait que les opérations offertes sont trop « rigides » et ne peuvent servir de fondement au développement d'un calculateur *générique*. Il est donc légitime de se poser la question de « l'universalité » des nouveaux media de calculs et des nouveaux paradigmes de programmation. Il ne s'agit pas ici de l'universalité au sens de la machine de Turing, mais de la capacité à s'intégrer facilement à un cadre permettant de répondre de manière souple à des applications diverses et des besoins variés.

De nouvelles applications motivent le développement des nouveaux modèles de programmation. La « toile » et internet fournissent de nombreux problèmes à résoudre. Un des challenges est d'obtenir un comportement global cohérent à partir de la programmation du comportement local des entités. Ce challenge est aussi celui rencontré par la programmation de nano-robots, l'utilisation de poussières intelligentes ou l'exploitation d'ordinateurs amorphes : « *How do we obtain coherent behavior from the cooperation of large numbers of unreliable parts that are interconnected in unknown, irregular, and time varying ways?* »[22].

22. Gerald L. Sussman à propos de la programmation des ordinateurs amorphes.

– Quel est le cadre théorique et pratique permettant de raisonner sur le comportement collectif d'une population aussi bien à une très grande échelle (le web) qu'à une échelle microscopique (nano-composant) ?

– Pourquoi notre approche actuelle de l'algorithmique, de la conception logicielle et des méthodes formelles ne permet-elle pas de faire face de manière satisfaisante aux problèmes de tolérance aux pannes, de sûreté de fonctionnement, de coopération, de régulation, etc. ?

En guise de conclusion

Toutes ces questions ont alimenté en arrière-plan les discussions passionnées et souvent surprenantes qui ont accompagné les exposés à UPP. Ceux-ci ont montré tout le potentiel de ces approches émergentes, à partir d'exemples de systèmes matériels et logiciels et de propositions plus spéculatives.

La prise en compte des nouvelles applications et des nouvelles ressources matérielles que nous avons évoquées ci-dessus, ne constituent qu'une partie des motivations qui poussent le domaine des langages de programmation à se renouveler et à rechercher de nouvelles sources d'inspiration. La notion de programme, de langage de programmation, de logiciel, s'est élaborée progressivement au cours des cinquante dernières années. Aujourd'hui, à travers ces nouvelles métaphores, se dessinent une critique de nos conceptions et de nos outils actuels, mais aussi des propositions concrètes. Nul doute que nous assistons à l'émergence d'une nouvelle approche de ce que doit être un programme et son cycle de vie. Le calcul chimique, amorphe, biologique, quantique sont des domaines de recherche en pleine effervescence et qui réunissent des chercheurs fertiles et enthousiastes. C'est au sein de ces communautés de recherche que s'élaborent les idées, les concepts et les outils qui nous permettront de repenser nos approches algorithmiques, nos architectures et nos méthodes formelles, afin que nos programmes et nos systèmes informatiques deviennent plus sûrs, plus robustes, plus autonomes, plus adaptables et plus efficaces.

Nous invitons tous les lecteurs curieux de mieux connaître ces domaines prometteurs à se reporter au volume qui sera publié en 2005.

Jean-Louis Giavitto, Olivier Michel,
LaMI, Evry
Jean-Pierre Banâtre,
IRISA, Rennes
Pascal Fradet,
INRIA Rhône-Alpes, Montbonnot