INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Projet COMPOSE

Conception de programmes et systèmes adaptatifs

Rennes

THÈME 2A

Rapport d'Activité

1999

Table des matières

1 Composition de l'équipe

Responsable scientifique

Charles Consel [professeur, université de Rennes 1]

assistantes de projet

Laetitia Brenugat [vacataire Inria, jusqu'au 15 juin 1999]

Catherine Godest [TR CNRS depuis le 1er juin 1999]

Personnel Inria

Renaud Marlet [CR]

Gilles Muller [CR]

Ingénieurs experts

Peter Chang [à partir du 15 octobre 1999]

Miguel de Miguel [du 1er janvier 1999 au 31 mai 1999]

Ronan Gaugne [jusqu'au 24 mars 1999]

Robin Hansen [à partir du 6 avril 1999]

Neetu Nangia [à partir du 15 novembre 1999]

Chercheurs doctorants

Luciano Porto Barreto [stagiaire du 1er avril 1999 au 31 octobre 1999, puis boursier Inria, à partir du 1er novembre 1999]

Philippe Boinot [boursier Inria]

Sandrine Chirokoff [boursier MENRT]

Anne-Françoise Lemeur [boursière Inria, à partir du 1er octobre 1999]

Fabrice Mérillon [boursier MENRT]

Ulrik Pagh Schultz [boursier Inria]

Laurent Réveillère [boursier MENRT]

Collaborateur extérieur

Jacques Noyé [maître-assistant à l'Ecole des Mines de Nantes]

2 Présentation et objectifs généraux

Mots clés : évaluation partielle, spécialisation, compilation, transformation de programmes, génie logiciel, optimisation de systèmes d'exploitation, systèmes adaptatifs, systèmes embarqués, systèmes d'exploitation.

Résumé : *Le projet s'intéresse à la conception de systèmes adaptatifs. Notre démarche consiste à rendre performant un programme générique en le spécialisant en fonction d'un contexte donné d'utilisation. Plus précisément, notre objectif est d'étudier les techniques de spécialisation et leur utilisation pour des applications de taille réelle. En particulier, l'évaluation partielle est à la base de notre approche de conception de programmes et de systèmes adaptatifs.*

Le développement des logiciels et des systèmes informatiques modernes est soumis à des objectifs importants mais contradictoires de généralité et de performance. La généralité dans la conception est souvent recherchée dans l'ingénierie du logiciel afin de réduire le cycle de développement et les coûts de production et de maintenance. Pour ce faire, on essaie de rendre les logiciels aisément adaptables, réutilisables et maintenables. Ces besoins ont été moteurs dans de nombreuses recherches en langage de programmation et en ingénierie du logiciel telles que les langages objets ou les bus logiciels (par exemple Corba). Toutefois, le gain en généralité entraîne fréquemment une perte en efficacité à l'exécution, ce qui limite le domaine d'application des approches précédentes. Par exemple, dans le domaine du calcul scientifique, on préfère souvent réécrire des bibliothèques trop génériques car l'explosion des paramètres les rend généralement inefficaces.

Notre projet vise à concilier des impératifs de généricité, lors de la conception d'un logiciel, et de performance, lors de son implémentation. Plus précisément, notre démarche consiste à rendre performant un programme générique en l'*adaptant* à un contexte donné d'utilisation. Le contexte est défini par un ensemble de paramètres qui peuvent être relatifs à la taille du problème traité, à des propriétés sur les valeurs d'entrée, à la configuration du matériel, etc. Ce contexte peut être déterminé avant l'exécution du programme ou peut varier à différents stades de son exécution. En conséquence, le processus d'adaptation doit pouvoir être effectué à la fois statiquement, à la compilation, et dynamiquement, lors de l'exécution.

Promouvoir la conception de logiciels adaptatifs en tant que technique réaliste d'ingénierie logicielle suppose de couvrir tous les aspects du processus de développement de logiciels, allant de la méthodologie de conception de logiciels adaptatifs jusqu'à leur instanciation effective, dans le contexte d'applications de taille réelle. En fait, à ces différents aspects correspondent des questions fondamentales qu'il est important d'énoncer pour comprendre les enjeux de cette problématique. Comment concevoir un logiciel adaptatif ? Comment rendre un logiciel existant adaptatif ? Comment instancier un logiciel adaptatif ? Comment mesurer les bénéfices de l'approche ?

Ces questions nous amènent à adopter une *démarche verticale* dans le choix de nos objectifs de recherche depuis l'étude des principes de l'adaptation de programmes jusqu'au développement d'outils effectuant cette adaptation dans le cas d'applications de taille réelle.

Conception de logiciels adaptatifs. Notre objectif est de développer des méthodologies de conception de logiciels dont la généricité permet de traiter un problème général, et dont l'instanciation permet de se focaliser sur un sous-problème donné. Afin d'atteindre cet objectif, nous étudions la notion de langage dédié permettant de programmer des familles d'applications. Nous étudions également différents types d'architectures logicielles permettant de rendre explicites les aspects génériques du logiciel.

Principes et techniques. Nous étudions les principes sur lesquels repose le processus d'adaptation de programmes. L'étude des aspects fondamentaux de ce processus nous conduit à formaliser certaines de ses phases, telles que des analyses et des transformations de programmes. Ce travail nous permet un développement rigoureux de techniques de mise en œuvre du processus d'adaptation de programmes.

Développement d'outils. Pour compléter notre approche de conception de logiciels adaptatifs nous développons des outils permettant de spécialiser un logiciel générique en fonction d'un certain contexte d'utilisation.

Applications de taille réelle. La validation de notre approche passe inévitablement par son utilisation dans des applications industrielles. Nos outils doivent ainsi traiter des langages de programmation utilisés dans l'industrie tels que C. Nous visons en premier lieu, les domaines des télécommunications et des systèmes embarqués grand public dans lesquels nous collaborons déjà avec des industriels, et où des besoins d'adaptabilité ont été clairement identifiés.

3 Fondements scientifiques

Mots clés : évaluation partielle, spécialisation, transformation de programmes, systèmes adaptatifs, génie logiciel.

Glossaire :
Évaluation partielle transformation de programmes qui a pour but de spécialiser un programme en fonction de certaines de ses données d'entrée.

L'adaptabilité est devenue une caractéristique incontournable dans la conception des nouveaux logiciels pour leur permettre de répondre à des besoins fondamentaux tels que :

- l'évolution et l'hétérogénéité des matériels informatiques, ainsi que la prise en compte de leurs caractéristiques de bas niveau ;
- la généralité sans cesse croissante des problèmes que doivent traiter les logiciels pour contrebalancer leur coût de développement ;
- le besoin d'intégration avec d'autres composants logiciels pour constituer des systèmes informatiques complets.

Des techniques de conception de logiciels adaptatifs existent déjà ; elles consistent généralement à structurer un logiciel de telle sorte qu'il puisse évoluer en fonction de son contexte d'utilisation. Cette structuration prend, traditionnellement, la forme de *modules* et de *couches logicielles*. L'évolution de ces techniques s'est traduite par un réel engouement pour les langages à objets dont l'un des objectifs principaux est d'offrir des mécanismes d'organisation et de généralisation de composants logiciels. Plus récemment, des approches reposant sur la notion de bus logiciel ont été proposées pour permettre la composition de composants logiciels indépendants, mais dont l'interface est spécifiée.

Toutefois, les approches existantes sont incomplètes car, bien qu'elles prennent en compte les aspects conceptuels d'un logiciel, elles négligent ses aspects relatifs à l'implémentation. Par faute de méthodes et d'outils adéquats, l'adaptabilité se traduit souvent par l'introduction, dans la mise en œuvre, de mécanismes tels que l'interprétation (de paramètres ou d'un état global), la protection des données et du code, et la copie de données entre différentes couches logicielles. Ces mécanismes complexifient les algorithmes et entraînent une inefficacité importante qui conduit bien souvent à spécialiser manuellement un logiciel pour un contexte d'utilisation donné afin d'obtenir des performances acceptables. Cette spécialisation manuelle est bien évidemment fastidieuse et source d'erreurs. De plus, elle multiplie les versions d'un même logiciel entraînant du même coup des problèmes de maintenance. Plus généralement, elle annule les efforts d'adaptabilité du logiciel déployés lors de sa conception.

Notre objectif est d'étudier la généralisation et la systématisation des techniques de spécialisation et leur utilisation pour des applications de taille réelle.

L'évaluation partielle est à la base de notre approche de conception de programmes et systèmes adaptatifs, nous en présentons maintenant ses aspects fondamentaux.

L'évaluation partielle

L'évaluation partielle a pour but de spécialiser un programme en fonction de certaines de ses données d'entrée. Cette transformation de programmes préserve la sémantique initiale dans la mesure où le *programme spécialisé*, appliqué aux données manquantes, produit le même résultat que le programme original appliqué à toutes les données. Comme on aime à le souligner, la calculabilité de l'évaluation partielle repose sur le théorème S^m_n de Kleene [JSS89,Kle52].

À la différence d'une stratégie de transformation de programmes générale à la Burstall et Darlington [BD77], l'évaluation partielle a pour unique objectif la spécialisation de programmes. Un évaluateur partiel consiste en un ensemble réduit de règles de transformation de programmes visant à évaluer les expressions qui manipulent des données disponibles et à reconstruire les expressions dépendant des données manquantes.

Bien que simple dans son principe, la notion de spécialisation s'applique à une vaste classe de problèmes. En effet, l'évaluation partielle a été utilisée pour des applications aussi variées que la génération de compilateurs à partir d'interprètes [JSS89], l'optimisation de programmes numériques [Ber90], le filtrage [CD89] et l'instrumentation de programmes [KHC91].

[JSS89] N. Jones, P. Sestoft, H. Søndergaard, « Mix: a Self-Applicable Partial Evaluator for Experiments in Compiler Generation », *Lisp and Symbolic Computation 2*, 1, 1989, p. 9–50.

[Kle52] S. C. Kleene, *Introduction to Metamathematics*, Van Nostrand, 1952.

[BD77] R. M. Burstall, J. Darlington, « A Transformational System for Developing Recursive Programs », *Journal of ACM 24*, 1, 1977, p. 44–67.

[Ber90] A. Berlin, « Partial Evaluation Applied to Numerical Computation », *in : ACM Conference on Lisp and Functional Programming*, ACM Press, p. 139–150, Nice, France, 1990.

[CD89] C. Consel, O. Danvy, « Partial Evaluation of Pattern Matching in Strings », *Information Processing Letters 30*, 2, 1989, p. 79–86.

[KHC91] A. Kishon, P. Hudak, C. Consel, « Monitoring Semantics: a Formal Framework for Specifying, Implementing and Reasoning about Execution Monitors », *in : Proceedings of the ACM SIGPLAN*

Aspects extensionnels

Pour présenter l'évaluation partielle, il est important d'établir une distinction entre un programme et la fonction (c'est-à-dire l'objet mathématique) que ce programme dénote. Pour ce faire nous utilisons la convention suivante : lorsque le nom d'une variable apparaît en majuscule, cette variable dénote un programme, sinon elle dénote une fonction. Nous ne définissons pas la correspondance entre un programme et la fonction qu'il dénote ; nous supposons que cette correspondance est donnée par la sémantique formelle du langage. Nous supposons de plus qu'il existe un évaluateur *eval* tel que :

$$eval(P,d) = p\ d$$

Etant donné un programme P à deux entrées et une valeur v, un évaluateur partiel est un programme, noté PE, calculant le *programme résiduel* P_v

$$P_v = eval(PE,(P,v))$$

tel que, lorsque le programme résiduel est appliqué à la donnée manquante w

$$eval(P_v,w) \equiv eval(P,(v,w))$$

le programme original et le programme spécialisé produisent le même résultat. D'un point de vue fonctionnel, ceci peut être écrit comme suit :

$$p_v(w) \equiv p(v,w) \ \textit{où}\ P_v = pe(P,v)$$

Les données disponibles pendant l'évaluation partielle sont dites *statiques* (la donnée v). Les données manquantes sont dites *dynamiques* (la donnée w) [JSS89]. On dit que P_v est la version spécialisée de P en fonction de v.

Il est également possible d'optimiser le processus même d'évaluation partielle par auto-application de l'évaluateur partiel.

Les stratégies d'évaluation partielle

On distingue communément deux stratégies d'évaluation partielle : *en ligne* et *hors ligne*. La première stratégie consiste à déterminer le traitement du programme au fur et à mesure de la phase d'évaluation partielle. Les évaluateurs partiels basés sur ce principe ont l'avantage de manipuler des valeurs concrètes, et donc, peuvent déterminer précisément le traitement de chaque expression d'un programme. Toutefois, ce processus est coûteux : l'évaluateur partiel doit analyser le contexte du calcul (c'est-à-dire les données disponibles) pour sélectionner la transformation de programmes appropriée. Cette opération est effectuée de façon répétitive dans le cas de fonctions récursives, par exemple. Il n'est donc pas surprenant de constater que

'91 Conference on Programming Language Design and Implementation, ACM SIGPLAN Notices, 26(6), p. 338–352, Toronto, Ontario, Canada, juin 1991.

[JSS89] N. Jones, P. Sestoft, H. Søndergaard, « Mix: a Self-Applicable Partial Evaluator for Experiments in Compiler Generation », *Lisp and Symbolic Computation 2*, 1, 1989, p. 9–50.

les performances d'un évaluateur partiel en ligne se dégradent rapidement lorsque de nouvelles transformations de programmes sont introduites.

La deuxième stratégie d'évaluation partielle comporte deux phases : une *analyse de temps de liaison* et une phase de *spécialisation* [JSS89]. Étant donné un programme et une description de ses entrées (statique/dynamique), l'analyse de temps de liaison détermine les expressions qui peuvent être évaluées lors de la phase d'évaluation partielle et celles qui doivent être reconstruites : les premières sont dites statiques, les autres dynamiques. Les informations de temps de liaison sont valides tant que la description des entrées du programme reste inchangée. Les expressions statiques et dynamiques étant connues à l'avance, la phase de spécialisation est plus efficace. Toutefois, l'analyse de temps de liaison, manipulant des valeurs abstraites, permet d'effectuer certaines approximations. Ainsi, le degré de spécialisation d'une stratégie hors ligne peut être moindre que celui d'une stratégie en ligne.

L'activité importante qui s'est développée dans le domaine de l'évaluation partielle a conduit à la réalisation de nombreux prototypes pour une variété de langages de programmation tels que Scheme [Bon90,Con93b], C [And94] et Pascal [Mey91].

Évaluation partielle à l'exécution

Traditionnellement, l'évaluation partielle est une transformation effectuée sur le texte d'un programme. De ce fait, elle intervient à la compilation et ne peut exploiter que les valeurs disponibles à ce stade.

Nous avons développé un cadre de travail général permettant de réaliser la spécialisation de programmes impératifs à l'exécution [5]. Le processus de spécialisation que nous avons conçu a pour point de départ un programme annoté d'actions. Ces actions décrivent les transformations à effectuer sur chaque construction du programme à spécialiser et, de fait, peuvent être vues comme une description de l'ensemble des programmes qui peuvent être produits par spécialisation. Notons que ce point de départ n'est pas spécifique à la spécialisation à l'exécution ; dans notre approche, les actions sont également utilisées pour guider la spécialisation à la compilation.

À partir d'un programme annoté d'actions, nous produisons automatiquement à la compilation des fragments de code source. Ces fragments de code source sont incomplets dans la mesure où les invariants ne sont connus qu'à l'exécution. Ils sont transformés de manière à pouvoir être traités par un compilateur standard. À l'exécution, il ne reste plus qu'à sélectionner et copier certains de ces fragments de code, insérer les valeurs dynamiques et reloger

[Bon90] A. Bondorf, « Automatic Autoprojection of Higher Order Recursive Equations », *in : ESOP'90, 3rd European Symposium on Programming*, N. D. Jones (éditeur), *Lecture Notes in Computer Science*, *432*, Springer-Verlag, p. 70–87, 1990.

[Con93b] C. Consel, « A Tour of Schism », *in : Partial Evaluation and Semantics-Based Program Manipulation*, ACM Press, p. 66–77, Copenhagen, Denmark, juin 1993.

[And94] L. Andersen, *Program Analysis and Specialization for the C Programming Language*, thèse de doctorat, Computer Science Department, University of Copenhagen, mai 1994, DIKU Technical Report 94/19.

[Mey91] U. Meyer, « Techniques for Partial Evaluation of Imperative Languages », *in : Partial Evaluation and Semantics-Based Program Manipulation*, p. 94–105, New Haven, CT, USA, septembre 1991. ACM SIGPLAN Notices, 26(9).

certains sauts pour obtenir une spécialisation donnée.

Ces opérations sont simples et permettent donc un processus de spécialisation très efficace qui ne nécessite qu'un nombre limité d'exécutions du programme spécialisé avant d'être amorti.

Analyses et outils

Les principes d'évaluation partielle décrits plus haut permettent le développement rigoureux d'analyses de programmes performantes [AC94,Con93a] et la conception de transformations de programmes [CD90,CD91,Con93b] [2, 12] pour des langages fonctionnels et impératifs. Nos études conduisent à l'élaboration d'outils de spécialisation. Ces outils ont un rôle crucial dans la validation de la technologie que nous développons. Nos efforts actuels portent sur un système d'évaluation partielle pour le langage C, Tempo (voir module 5.1). Par ailleurs, nous avons entrepris le développement d'un frontal à Tempo, permettant la spécialisation de programmes Java (voir modules 5.2, 5.3 et 6.1).

4 Domaines d'applications

Mots clés : télécommunications, génie logiciel, calcul numérique, graphisme, systèmes embarqués, systèmes d'exploitation.

L'adaptabilité des logiciels est un besoin très général qui a été clairement identifié dans des domaines aussi variés que les télécommunications [9], les systèmes d'exploitation [8, 7, 16], le génie logiciel [6], le calcul numérique [Ber90] et le graphisme [GKR95]. Divers travaux dans ces domaines ont démontré que l'adaptabilité permettait, entre autres choses, de rendre un logiciel plus facilement configurable, dimensionnable et évolutif.

Nous avons plus particulièrement choisi d'appliquer nos outils aux domaines des systèmes de télécommunications et des systèmes embarqués grand public, comme en témoignent nos collaborations industrielles avec Alcatel, Bull, France Télécom et Thomson Multimédia. Les

[AC94] J. M. Ashley, C. Consel, « Fixpoint Computation for Polyvariant Static Analyses of Higher-Order Applicative Programs », *ACM Transactions on Programming Languages and Systems 16*, 5, 1994, p. 1431–1448.

[Con93a] C. Consel, « Polyvariant Binding-Time Analysis for Applicative Languages », *in : Partial Evaluation and Semantics-Based Program Manipulation*, ACM Press, p. 145–154, Copenhagen, Denmark, juin 1993.

[CD90] C. Consel, O. Danvy, « From Interpreting to Compiling Binding Times », *in : ESOP'90, 3rd European Symposium on Programming*, N. Jones (éditeur), *Lecture Notes in Computer Science, 432*, Springer-Verlag, p. 88–105, 1990.

[CD91] C. Consel, O. Danvy, « For a Better Support of Static Data Flow », *in : Functional Programming Languages and Computer Architecture*, J. Hughes (éditeur), *Lecture Notes in Computer Science, 523*, Springer-Verlag, p. 496–519, Cambridge, MA, USA, août 1991.

[Con93b] C. Consel, « A Tour of Schism », *in : Partial Evaluation and Semantics-Based Program Manipulation*, ACM Press, p. 66–77, Copenhagen, Denmark, juin 1993.

[Ber90] A. Berlin, « Partial Evaluation Applied to Numerical Computation », *in : ACM Conference on Lisp and Functional Programming*, ACM Press, p. 139–150, Nice, France, 1990.

[GKR95] B. Guenter, T. Knoblock, E. Ruf, « Specializing Shaders », *in : Computer Graphics Proceedings, Annual Conference Series*, ACM Press, p. 343–350, 1995.

besoins de ces secteurs de l'industrie informatique sont particulièrement représentatifs de notre problématique. En effet, les applications visées sont amenées à s'exécuter sur des configurations matérielles variées et destinées à évoluer dans le temps ; leur cycle de développement doit être très court ; enfin, la contrainte de performance est importante pour réduire le coût du matériel, notamment dans le cas des systèmes embarqués. Nos collaborations concernent ces besoins au travers de différents thèmes : optimisation de systèmes d'exploitation (voir modules 7.1, 7.5 et 7.6), conception de services génériques (voir module 7.2 et 7.7) et optimisation de systèmes embarqués (voir modules 7.3, 7.4 et 7.8).

5 Logiciels

5.1 Tempo, un évaluateur partiel pour C

Mots clés : évaluation partielle, langage C, spécialisation à l'exécution.

Participant : Renaud Marlet.

Nous avons conçu et développé un évaluateur partiel pour des programmes C, nommé Tempo [2, 1]. Une innovation importante apportée par ce système est qu'il permet la spécialisation de programmes à la compilation et à l'exécution [2]. Diverses analyses dont le but est de préparer la phase de spécialisation ont été conçues pour ce système [13]. Etant donnée la richesse du langage C et le fait qu'il ait été peu étudié dans le contexte de l'évaluation partielle, le développement de ces analyses a constitué une partie importante de notre travail. Les principales analyses de programmes sont les suivantes :

- analyse d'alias,
- analyse d'effets de bord,
- analyse de temps de liaison,
- analyse d'actions (transformations de programmes).

Pour s'assurer que les transformations de programmes offertes par Tempo produisent un programme très spécialisé, nous avons ciblé notre travail sur les programmes système qui sont très propices à la spécialisation. Nous avons ainsi pu recenser les besoins principaux de spécialisation existants dans ce domaine et introduire les analyses et transformations correspondantes. Tempo a été notamment validé par la spécialisation d'un code système faisant partie d'un produit commercial, en l'occurence l'implémentation de l'appel de procédure à distance (RPC) développé par Sun en 1984 [7, 15]. Les gains en vitesse obtenus par spécialisation de ce code vont jusqu'à un facteur de 3,7 sur l'encodage des données.

Tempo est actuellement disponible via une licence d'évaluation. Trente six utilisateurs en disposent à ce jour, dont Bull, France Telecom et Thomson Multimédia.

Nos travaux actuels visent à faire évoluer Tempo vers un moteur de spécialisation multi-langages. Pour ce faire, nous développons un certain nombre de frontaux à Tempo pour nous permettre de traiter d'autres langages que C. Cette approche est actuellement expérimentée pour le langage Java. langage Java

5.2 Harissa, un environnement d'exécution pour le langage Java

Mots clés : compilation, Java.

Participant : Gilles Muller.

Harissa est un environnement d'exécution du langage Java qui intègre un interprète et un compilateur de code intermédiaire vers C [17]. Harissa permet de mélanger au sein d'une même application du code compilé et du code interprété. Il conjugue ainsi les avantages de performance et de flexibilité. Harissa a été développé pour permettre la spécialisation du langage Java ; son compilateur est utilisable en tant que frontal de Tempo.

Le code C produit par le compilateur d'Harissa est de 5 à 40 fois plus rapide que le code interprété par le JDK 1.0.2 de Sun. Une version binaire d'Harissa est disponible via le Web à l'adresse `http://www.irisa.fr/compose/harissa` ; plus de 2100 utilisateurs d'Harissa sont actuellement recensés dans le monde entier.

5.3 JSCC, un compilateur de classes de spécialisation

Mots clés : compilation, Java, spécialisation.

Participant : Gilles Muller.

L'utilisation directe d'un moteur de spécialisation comme Tempo nécessite la compréhension des concepts de base de l'évaluation partielle tels que l'analyse de temps de liaison. Afin de simplifier l'utilisation d'un spécialiseur, nous avons introduit une approche déclarative à la spécialisation dans le contexte de la programmation orientée objet (voir la section 6.1).

Dans notre approche, l'unité de déclaration est la *classe de spécialisation* [VCMC97]. Elle enrichit l'information concernant une classe existante en décrivant comment et quand la spécialisation doit être réalisée. À partir de ces informations, un compilateur produit des fichiers de contexte permettant de guider le spécialiseur. Une implémentation d'un compilateur des classes de spécialisation a été réalisée pour le langage Java. Ce compilateur, nommé JSCC, prend en entrée du source Java étendu avec les classes de spécialisation et produit du Java standard. JSCC est disponible via le Web à l'adresse `http://www.irisa.fr/compose/jscc`.
graphiques

5.4 Gal, un langage et un générateur de pilotes de cartes graphiques

Mots clés : génie logiciel, langage dédié, XFree86, pilote de cartes graphiques.

Participant : Charles Consel.

Le langage Gal est le résultat d'une expérience grandeur réelle visant à valider notre schéma général de conception et d'implémentation de générateurs d'applications basé sur la notion de langage dédié (voir module 6.3). Gal (Graphic Adaptor Language) est un langage qui permet

[VCMC97] E. Volanschi, C. Consel, G. Muller, C. Cowan, « Declarative Specialization of Object-Oriented Programs », *in: OOPSLA'97 Conference Proceedings*, ACM Press, p. 286–300, Atlanta, USA, octobre 1997.

la description de pilotes de cartes graphiques [10]. Gal a été implémenté pour le serveur X Window XFree86 en suivant chaque étape de la démarche que nous avons proposée. L'implémentation finale contient notamment plusieurs analyses (qui seraient impossibles à mettre en œuvre sur les pilotes existants écrits en C) et un générateur automatique de documentation. Cette implémentation est disponible via le Web à l'adresse
`http://www.irisa.fr/compose/gal`

5.5 Plan-P

Mots clés : génie logiciel, langage dédié, réseaux actifs.

Participant : Gilles Muller.

Les protocoles internet actuels ont une limitation fondamentale : ils forcent l'uniformité des fonctionnalités à travers tout le réseau et pour toutes les applications. Une nouvelle approche, appelée *réseaux actifs*, consiste à programmer dynamiquement les routeurs pour définir des comportements spécifiques à une application ou à un paquet. Dans ce cadre, nous avons conçu Plan-P [26], un langage permettant une programmation sûre et efficace des réseaux actifs. Comme pour le langage Gal [18], Plan-P a été conçu suivant notre schéma général de conception et d'implémentation de générateurs d'applications basé sur la notion de langage dédié (voir module 6.3).

Comme le réseau est une ressource partagée, la programmation des routeurs doit s'accompagner d'un contrôle strict des programmes qui sont téléchargés. Pour ce faire, nous proposons une approche visant à envoyer le code source des programmes Plan-P sur le réseau pour permettre des vérifications adaptées en fonction de chaque routeur. Afin de résoudre le problème d'efficacité que pose une telle approche, nous utilisons un compilateur Plan-P *Just In Time* (JIT) qui est automatiquement généré à partir d'un interprète, au moyen de Tempo [TBC+98]. Par ailleurs, il est à noter que la performance d'un programme Plan-P compilé par ce JIT est supérieure à celle d'un programme Java équivalent compilé statiquement par Harissa.

Plan-P a été utilisé pour implémenter plusieurs applications dont la distribution multi-point adaptative de flux audio et la construction de serveurs HTTP extensibles [26]. Plan-P et son système d'exécution pour Solaris sont disponibles via le Web à l'adresse
`http://www.irisa.fr/compose/plan-p`.

Plan-P est actuellement évalué par plusieurs laboratoires de recherche dans le monde. En particulier, le groupe Synthetix de l'Oregon Graduate Institute est en train d'effectuer un portage de notre prototype sur Linux.

6 Résultats nouveaux

6.1 Principes, techniques et outils de spécialisation

Mots clés : évaluation partielle, spécialisation, outils, analyses de programmes.

[TBC+98] S. Thibault, L. Bercot, C. Consel, R. Marlet, G. Muller, J. Lawall, « Experiments in Program Compilation by Interpreter Specialization », *Publication interne n° 1212*, IRISA, Rennes, France, novembre 1998.

Résumé : *L'étude des aspects fondamentaux du processus de la spécialisation de programmes nous conduit à formaliser certaines de ses phases, telles que des analyses et des transformations de programmes, afin notamment de garantir leur correction. Ce travail permet un développement rigoureux de techniques de mise en œuvre du processus d'adaptation de programmes. Pour compléter notre approche de conception de logiciels adaptatifs, nous développons des outils permettant de spécialiser un logiciel générique en fonction d'un certain contexte d'utilisation. Enfin, pour valider notre approche, il est essentiel d'utiliser nos outils pour traiter des applications industrielles. Ceci nous amène à développer des outils pour des langages de programmation utilisés dans l'industrie tels que C et Java.*

Spécialisation de données

Participants : Charles Consel, Sandrine Chirokoff, Renaud Marlet, Gilles Muller.

De manière générale, la spécialisation repose sur le découpage en deux parties d'un calcul répété dont une partie du contexte d'entrée reste invariante pour toutes les itérations. Une phase préliminaire, qui n'est exécutée qu'une seule fois, effectue les calculs qui ne dépendent que des invariants. Une phase ultérieure effectue les calculs restants de manière répétée en exploitant les résultats de la phase préliminaire.

Dans la *spécialisation de programmes*, le résultat de la phase préliminaire est le code même de la phase ultérieure, qui incorpore la connaissance des invariants. Il existe une alternative, appelée *spécialisation de données*, qui consiste à encoder les résultats de la phase préliminaire dans une structure de données (un cache) qui est exploitée par la phase ultérieure. C'est une notion « orthogonale » à celle de la spécialisation à la compilation ou à l'exécution.

La spécialisation de données offre un compromis espace/temps très intéressant par rapport à la spécialisation de programmes. En effet, elle génère des programmes compacts (en particulier dans le cas de dépliages de boucles). De plus, la phase préliminaire de spécialisation est très rapide. En revanche, la spécialisation de données est un peu moins performante que la spécialisation de programmes.

Nous avons implémenté un prototype de spécialiseur de données par extension de Tempo. Ce prototype nous a permis de procéder à une première évaluation de performance sur des programmes de calcul scientifique et des interprètes [11, 21].

Par ailleurs, nous avons utilisé la spécialisation de données pour optimiser la spécialisation à l'exécution. Plus précisément, la spécialisation de données permet de programmer de manière simple et générique des optimisations telles que le regoupement des canevas binaires ou l'expansion de procédure. Ces différentes optimisations nous ont permis d'améliorer la performance des programmes spécialisés à l'exécution d'un facteur allant jusqu'à 4 [28].

Spécialisation incrémentale

Participants : Charles Consel, Renaud Marlet, Philippe Boinot.

La spécialisation incrémentale est une technique qui permet d'étager la spécialisation d'un programme au fur et à mesure de la disponibilité des invariants. L'avantage majeur de cette technique est de factoriser le processus de spécialisation et de réduire ainsi le surcoût en temps d'exécution de la spécialisation.

Nous avons montré que la spécialisation incrémentale pouvait être implémentée par itération au moyen d'un spécialiseur à l'exécution normal [22], en évitant de ce fait la réalisation d'un spécialiseur ad-hoc. Nous avons expérimenté cette technique avec le spécialiseur à l'exécution de Tempo. Outre une réduction du temps nécessaire à la spécialisation, les résultats de cette étude ont montré que les opportunités de spécialisation présentes au sein d'un programme générique étaient conservées.

Spécialisation du langage Java

Participants : Charles Consel, Gilles Muller, Jacques Noyé, Philippe Boinot, Ulrik Pagh Schultz.

Le développement d'un nouveau spécialiseur à partir du néant est une tâche longue et complexe demandant d'importants moyens de développement. Afin de développer un spécialiseur pour le langage Java, nous expérimentons une nouvelle approche reposant sur l'utilisation de Tempo en tant que moteur de spécialisation commun à plusieurs langages et de traducteurs spécifiques à chaque langage. Pour pouvoir traiter efficacement des programmes à objets, cette approche demande l'enrichissement des analyses de Tempo. En particulier, nous développons une analyse de temps de liaison plus précise en ce qui concerne les instances de structures de données.

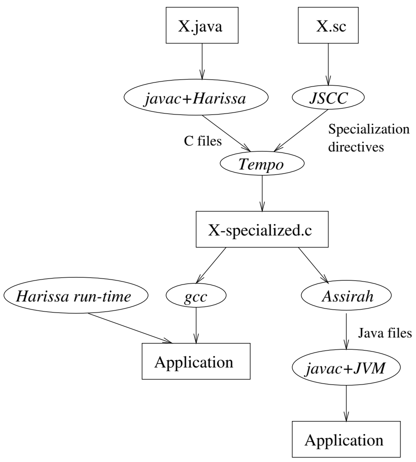

FIG. 1 – *Spécialisation de programmes Java*

Nous construisons actuellement un prototype de spécialiseur Java par intégration d'outils que nous avons antérieurement réalisés (voir figure 1): JSCC, le compilateur des classes de spécialisation (voir module 5.3), Harissa notre traducteur de Java vers C (voir module 5.2), Tempo (voir module 5.1) et Assirah, un traducteur arrière de C vers Java. De ce fait, un programme Java spécialisé peut être soit exécuté au sein du système d'exécution d'Harissa, soit être re-traduit en Java pour être exécuté par tout interprète ou compilateur à la volée Java standard.

Nous avons utilisé ce prototype de spécialiseur Java pour optimiser une application de filtrage d'images, avec un gain d'un facteur 4 en temps d'exécution [24], et une mise en œuvre incrémentale des points de reprise [27]. Pour cette dernière application, le gain est proportionnel à la complexité de la structure objet du programme et au schéma de modification des objets. Sur nos expériences, nous avons mesuré un gain d'un facteur allant jusqu'à 15.

La disponibilité d'un spécialiseur Java engendre de nombreuses perspectives de recherche relatives à l'optimisation des schémas de conception (*design pattern*) dans les programmes à objets, à la prise en compte de fonctionnalités de Java telles que la réflexion, et à la spécialisation de composants génériques objets [25].

Les schémas de conception sont reconnus pour faciliter le développement de programmes à objets. Cependant, ils introduisent souvent une certaine inefficacité au sein du logiciel final. En associant spécialisation de programmes et schémas de conception, il est possible d'extraire des informations sur la structure du programme qui permettent de guider le processus de spécialisation. Plus précisément, nous avons introduit la notion de *schémas de spécialisation* qui permet de capturer la connaissance sur l'optimisation d'un schéma de conception [30].

Par ailleurs, nous avons étudié comment modéliser l'implémentation des classes de spécialisation (voir module 5.3) à l'aide de schémas de conception. Cette modélisation, utilisant les schémas de conception bien connus *Stratégie*, *Observateur* et *Façade*, peut être utilisée plus généralement pour mettre en place un changement dynamique et concerté du comportement d'un ensemble d'objets [19].

La réflexion joue un rôle important dans la programmation d'applications génériques. Elle a toutefois l'inconvénient d'ajouter une couche d'interprétation qui nuit à la performance. Une solution consiste à utiliser l'évaluation partielle pour éliminer cette couche d'interprétation. Nous nous sommes intéressés aux modifications à apporter à un évaluateur partiel pour Java afin de traiter l'Api de réflexion [20]. Ces modifications consistent essentiellement à prendre en compte les informations de type dans la division entre données statiques et dynamiques et à introduire, lors de la spécialisation, une action de réflexion. Un banc d'essai utilisant le canevas de sérialisation nous a permis d'illustrer les bénéfices de l'approche.

6.2 Architectures logicielles génériques

Participants : Charles Consel, Renaud Marlet.

Mots clés : génie logiciel, architecture logicielle, interprète, machine abstraite.

Glossaire :

Architecture logicielle Moyen de structurer un logiciel complexe. Parmi les architecture logicielles les plus courantes on peut citer les bus logiciels, les architectures en couches, les systèmes de communication par évènements ou messages, les langages dédiés et les langages de coordination.

Résumé : *L'évaluation partielle est un outil qui permet de traiter le compromis flexibilité/efficacité dans les logiciel. De ce fait, nous avons montré qu'il était possible d'optimiser l'implémentation de nombreuses architectures logicielles au moyen de Tempo.*

Pour de nombreuses approches de génie logiciel, l'évaluation partielle s'avère être un outil tout à fait général pour traiter le compromis flexibilité/efficacité qui existe dans ce domaine [4, 6]. Après recherche de la source de l'inefficacité au sein de plusieurs architectures logicielles, nous avons observé que celle-ci était en partie due à l'intégration des données et du contrôle lors de l'assemblage des divers composants d'un système. La raison tient à ce que la flexibilité n'apparaît pas uniquement au stade de la conception, mais se matérialise également dans l'implémentation. Nous proposons d'utiliser la spécialisation comme moyen systématique pour améliorer la performance et, dans certains cas, réduire la taille des programmes. Nous avons étudié pour cela plusieurs implémentations représentatives de mécanismes flexibles employés dans diverses architectures logicielles : diffusion sélective, recherche de motifs, interprètes, couches logicielles, bibliothèques génériques. Nous avons montré comment une application systématique de l'évaluation partielle conduisait à l'optimisation de ces mécanismes [14].

6.3 Langages dédiés et leur application aux systèmes d'exploitation

Participants : Charles Consel, Renaud Marlet, Gilles Muller, Luciano Porto Barreto, Fabrice Mérillon, Laurent Réveillère.

Glossaire :
Langage dédié langage qui permet d'exprimer des variations à l'intérieur d'un domaine ou d'une famille d'applications.
Glossaire :
Réseau actif architecture de réseau ouverte permettant une adaptation du comportement du réseau par téléchargement dynamique de nouveaux protocoles au sein des routeurs.
Mots clés : langage dédié, pilote de périphériques, réseaux actifs.

Résumé : *Les langages dédiés sont des langages restreints à un domaine ou une famille d'applications. Ils offrent un haut niveau d'abstraction sur le domaine considéré. Leurs avantages sont une programmation simplifiée, plus concise et plus rapide. Par ailleurs, les langages dédiés permettent la vérification statique de propriétés spécifiques au domaine considéré. Les langages dédiés permettent ainsi d'augmenter la qualité des programmes et la productivité des développements. Pour ces différentes raisons, les langages dédiés ouvrent des perspectives intéressantes pour le développement de systèmes d'exploitation. Au travers de plusieurs expérimentations, nous évaluons le bénéfice des langages dédiés dans le développement de systèmes [23].*

Le développement de systèmes d'exploitation est reconnu comme une tâche complexe et sujette aux erreurs. Ainsi, il est fréquent de rencontrer de nombreux bugs dans les systèmes d'utilisation générale. Par ailleurs, dans des domaines comme les systèmes embarqués grand public, un court temps de développement est un facteur essentiel de succès commercial. En conséquence, développer du logiciel système fiable rapidement sans pour autant sacrifier la performance est un enjeu majeur pour de nombreux industriels.

Les langages dédiés sont des langages restreints à un domaine ou une famille d'applications. Ils offrent un haut niveau d'abstraction sur le domaine considéré. Leurs avantages sont une programmation simplifiée, plus concise et plus rapide. Par ailleurs, les langages dédiés permettent la vérification statique de propriétés spécifiques au domaine considéré. Les langages dédiés permettent ainsi d'augmenter la qualité des programmes et la productivité des développements. Pour ces différentes raisons, les langages dédiés ouvrent des perspectives intéressantes pour le développement de systèmes d'exploitation [23].

Nous avons proposé une méthodologie pour la conception de langages dédiés reposant sur une approche à deux niveaux [4]. Le premier niveau consiste à identifier les objets et opérations fondamentales du domaine d'applications, de manière à former une machine abstraite. Le second niveau consiste en la conception du langage même. Un programme dans ce langage dédié est implémenté via un interprète à l'aide des opérations de la machine abstraite définie au premier niveau. Cette structure en couche est très flexible, mais pose a priori des problèmes de performance. Grâce à l'utilisation systématique de l'évaluation partielle, il est possible de supprimer le surcoût de l'interprétation [TBC+98].

Notre méthodologie a été validée en premier lieu sur Gal [18], un langage dédié pour l'écriture de pilotes de cartes graphiques (voir module 5.4), et Plan-P [26], un langage dédié pour l'écriture de protocoles dans les réseaux actifs (voir module 5.5). Ces deux langages nous ont permis de mettre en évidence les avantages que l'on attribue généralement aux langages dédiés : productivité accrue, programmation de haut niveau grâce à une plus grande abstraction, vérifications formelles facilitées. À titre d'exemple, le facteur de réduction en taille par rapport à un programme équivalent écrit en C est de 10 pour Gal et de 3 pour Plan-P. Gal permet la vérification de propriétés sur les drivers comme le non-recouvrement de registres d'entrées/sorties. Plan-P permet la vérification de propriétés telles que la terminaison des protocoles, la livraison et la non-duplication exponentielle des paquets. Enfin, les programmes écrits dans ces deux langages sont aussi performants que des programmes équivalents écrits en C [9, 18, 26].

Plus récemment nous avons travaillé à la conception d'un nouveau langage généralisant les bénéfices de Gal à tout type de dispositif matériel. Le langage Devil permet une description en couches de l'interface d'un périphérique en séparant les différents niveaux d'abstractions (adresses, registres, fonctions logiques) rencontrés dans les circuits périphériques [29]. Grâce au typage fort du langage, il est possible de vérifier la correction de l'utilisation de chaque niveau d'abstraction par le niveau supérieur. En particulier, il est possible de vérifier la correction de l'utilisation de l'interface du périphérique par le programmeur d'un pilote. Nous avons validé la puissance d'expression du langage Devil en réussissant à spécifier la plupart des circuits

[TBC+98] S. Thibault, L. Bercot, C. Consel, R. Marlet, G. Muller, J. Lawall, « Experiments in Program Compilation by Interpreter Specialization », *Publication interne n° 1212*, IRISA, Rennes, France, novembre 1998.

périphériques utilisés dans un PC.

Enfin, dans le cadre de deux collaborations avec France Télécom (voir modules 7.5 et 7.6), nous étudions l'utilisation des langages dédiés pour réaliser le processus d'adaptation au sein de systèmes d'exploitation réflexifs.

7 Contrats industriels (nationaux, européens et internationaux)

télécommunication, contrat Alcatel ref 197 A 936 000 MC 012

7.1 Approche déclarative à l'adaptation d'applications de télécommunication, contrat Alcatel ref 197 A 936 000 MC 012

Participants : Gilles Muller, Charles Consel, Philippe Boinot, Miguel de Miguel, Peter Chang.

Cette action de recherche s'effectue dans le cadre du projet Reutel 2000 entre Alcatel et l'Inria. Notre participation vise à développer une approche déclarative permettant de spécifier les différentes étapes du processus d'adaptation et de spécialisation d'applications de télécommunication.

Dans le cadre de cette action, nous développons un spécialiseur de programmes Java reposant sur Tempo (voir modules 6.1). Un premier prototype de ce spécialiseur a fait l'objet d'une diffusion restreinte en décembre 1999.

7.2 ESAPS, projet européen ITEA

Participants : Charles Consel, Renaud Marlet, Neetu Nangia.

L'objectif du projet ESAPS est d'établir la technologie nécessaire au développement de lignes de produits pour des familles de systèmes. La technologie proposée sera validée sur deux domaines d'applications : la supervision aérienne et la gestion de commutateurs.

Notre contribution au projet ESAPS sera centrée sur les composants logiciels adaptables. Cet axe inclut différents aspects tels que la structuration d'un composant adaptable et le support déclaratif permettant d'exprimer les possibilités d'adaptation d'un composant. De plus, nous fournirons la technologie et les outils permettant d'automatiser le processus d'adaptation. Ceci inclut un outil pour effectuer la spécialisation de programme, ainsi qu'un compilateur de déclaration d'adaptation.

7.3 Contrat Thomson Multimédia référence 199C031

Participants : Charles Consel, Gilles Muller, Renaud Marlet, Robin Hansen.

Cette action de recherche est la continuation d'une première collaboration avec Thomson Multimédia qui a démontré l'intérêt industriel de notre technique de spécialisation de programmes.

Cette action vise à appliquer notre technologie de spécialisation pour générer automatiquement des composants logiciels à partir de descriptions de haut niveau. Le domaine d'application visé est celui des systèmes embarqués grand public pour la télévison numérique. En complémentarité directe avec cette action, nous participons conjointement avec Thomson Multimédia au projet européen DESS (voir module 7.4).

7.4 DESS, projet européen ITEA

Participants : Charles Consel, Renaud Marlet, Robin Hansen.

Ce projet vise à définir une méthodologie de développement de logiciels adaptée aux systèmes embarqués, reposant sur des composants orientés objet réutilisables, ainsi qu'à développer des outils permettant une mise en œuvre aisée de cette méthodologie. Afin de procéder à la validation de notre approche, nous réaliserons des démonstrations de la méthodologie sur des développements réalistes.

La contribution du projet Compose vise à résoudre par spécialisation de programmes les problèmes de performance introduits par la paramétrisation des composants et par les concepts de haut niveau manipulés par la méthodologie. Nous participerons à l'analyse des expériences menées sur ce sujet.

langages dédiés, contrat Cnet-CTI

7.5 Adaptation de systèmes réflexifs au moyen de langages dédiés, contrat Cnet-CTI

Participants : Gilles Muller, Charles Consel, Luciano Porto Barreto.

Les systèmes d'exploitation réflexifs sont des systèmes dont l'état interne est observable par introspection et dont le comportement est dynamiquement adaptable. Dans le domaine des télécommunications, la capacité des systèmes d'exploitation réflexifs à supporter des applications variées permet de satisfaire des besoins non anticipés lors de la conception.

Cette action de recherche vise à permettre l'adaptation et l'optimisation de systèmes de télécommunication réflexifs via l'utilisation de langages dédiés. Notre objectif est de concevoir un composant d'un système de télécommunication pouvant être paramétré au moyen d'un programme écrit dans un langage dédié. Le but recherché est de concilier performance, adaptabilité et sûreté du processus d'adaptation.

7.6 Phenix : Noyau d'infrastructure répartie adaptable, contrat RNRT

Participants : Gilles Muller, Charles Consel.

Cette action de recherche s'inscrit dans le prolongement de notre collaboration avec le Cnet sur l'adaptation de systèmes réflexifs au moyen de langages dédiés. Le Cnet, le projet Sor de l'Inria-Rocquencourt et le Lip6 sont nos partenaires au sein de cette action.

contrat Cnet-CTI ref 196C198

7.7 Génération automatique de services réseaux intelligents ; contrat Cnet-CTI ref 196C198

Participants : Charles Consel, Renaud Marlet, Ronan Gaugne.

Cette action de recherche s'est terminée en avril 1999. Son objectif était le développement d'une approche permettant la conception de services réseaux intelligents génériques. En l'occurence, nous avons orienté nos travaux vers l'utilisation de langages dédiés (voir module 6.3). Dans le cadre de cette action, nous avons développé le langage Gal, destiné au domaine des pilotes pour cartes graphiques (voir module 5.4), puis le langage Plan-P, destiné à la programmation des réseaux actifs (voir module 5.5). Les expérimentations réalisées en utilisant Plan-P ont montré qu'il était possible de programmer dynamiquement des routeurs pour offrir des comportements spécifiques aux applications réseaux.

systèmes embarqués ; contrat Bull ref 198C3620031325012

7.8 Optimisation de programmes Java pour systèmes embarqués ; contrat Bull ref 198C3620031325012

Participants : Charles Consel, Gilles Muller, Ulrik Pagh Schultz.

Le but de cette action est l'optimisation de programmes Java pour des systèmes embarqués possédant des ressources très limitées. L'implémentation de tels systèmes requiert un compromis entre flexibilité, performance et occupation mémoire. Les nouvelles techniques d'optimisation que nous développons tiennent compte de toutes ces contraintes.

Les études réalisées dans le cadre de cette action nous ont conduit à développer une technique de compaction du bytecode permettant d'économiser jusqu'à 30% de la taille des programmes. Par ailleurs, nous nous intéressons au développement et à la spécialisation d'applets Java génériques pour cartes à puces.

7.9 Actions internationales

Nous entretenons des relations étroites avec l'Oregon Graduate Institute à Portland et nous collaborons notamment avec le groupe Synthetix dirigé par le professeur Calton Pu. Notre collaboration porte actuellement sur l'adaptation d'applications distribuées via l'utilisation de réseaux actifs. En particulier, ce groupe est en train d'effectuer un portage sur Linux de Plan-P (voir module 5.5), notre langage de programmation pour réseaux actifs.

7.10 Visites, et invitations de chercheurs

7.10.1 Amérique du nord

Julia Lawall, chercheur à l'université de Brandeis a effectué un séjour dans notre groupe du 15 mai au 30 septembre 1999.

À notre invitation, Barry Redmond, chercheur au Trinity College de Dublin, Chiba Shigeru, professeur à l'Université de Tsukuba, Andrew Tolmach, professeur à l'Oregon Graduate Institute, Jonathan Walpole, professeur à l'Oregon Graduate Institute ont présenté des séminaires à l'Irisa.

8 Diffusion de résultats

8.1 Animation de la Communauté scientifique

Charles Consel a participé au comité de programme des conférences POPL'00, ICFP'00, DSL'99 et Reflection'99. Par ailleurs, il est éditeur invité de TCS pour le numéro spécial sur "Partial Evaluation and Semantics Based Program Manipulation".

Renaud Marlet a participé au comité de programme des conférences et workshops PEPM'99, ASE'99 et Dynamo'00.

Gilles Muller a participé au comité de programme des conférences et workshop EDCC-3, SRDS'99 et PEPM'00. Par ailleurs, il a organisé une journée “Thèmes Émergents” de l'ASF (Association ACM-Sigops de France), le 27 octobre 1999 à l'Irisa.

Gilles Muller a fait partie d'un comité d'experts présidé par Jean-Jacques Levy dont la mission a consisté en l'évaluation du système informatique embarqué du module européen Columbus de la future station spatiale. À la suite de cette expertise, il a effectué une mission de conseil auprès de Matra Marconi Space sur la conception d'un serveur de fichiers NFS tolérant les fautes.

8.2 Enseignement

Nous avons encadré deux stagiaires de DEA : Mikael Peltier, Laurent Bertrand

Charles Consel enseigne, à l'université de Rennes 1, l'évaluation partielle en DEA d'informatique, l'analyse et la transformation de programmes en DESS Isa et la compilation en maîtrise d'informatique.

Gilles Muller et Renaud Marlet interviennent en DEA filière architectures et systèmes d'exploitation, dans le cours sur les approches langage à la conception de systèmes d'exploitation.

8.3 Participation à des colloques, séminaires, invitations

Charles Consel a donné deux exposés invités dans les conférences PEPM'99, en janvier 1999, et DSL'99, en octobre 1999. Il a présenté des séminaires à l'Inria Rocquencourt, en mai 1999, à l'École Normale Supérieure (Paris), en mai 1999, au Laas, en juin 1999 et au Labri en octobre 1999. En juin 1999, il a donné un exposé devant le conseil scientifique de l'Irisa et un exposé invité à la commission d'évaluation de l'Inria.

Renaud Marlet a présenté un exposé à l'École Normale Supérieure (Paris), en mai 1999. En octobre, il a présenté un exposé au laboratoire de recherche d'IBM Watson. Charles Consel et Renaud Marlet ont présenté un exposé à la journée “Thèmes Émergents” de l'ASF organisée à Rennes.

Gilles Muller a présenté un séminaire à l'université de Paris VI en mars 1999. En octobre, il a présenté un exposé lors de la journée du club SEE “Systèmes Informatiques de Confiance” sur le thème des “Cots et de la réutilisation”. En novembre, il donné un exposé invité au workshop sur les réseaux actifs associé à la conférence IST'99 à Helsinki.

9 Bibliographie

Ouvrages et articles de référence de l'équipe

[1] C. Consel, L. Hornof, J. Lawall, R. Marlet, G. Muller, J. Noyé, S. Thibault, N. Volanschi, « Tempo: Specializing Systems Applications and Beyond », *ACM Computing Surveys, Symposium on Partial Evaluation 30*, 3, 1998.

[2] C. Consel, L. Hornof, F. Noël, J. Noyé, E. Volanschi, « A Uniform Approach for Compile-Time and Run-Time Specialization », *in: Partial Evaluation, International Seminar, Dagstuhl Castle*, O. Danvy, R. Glück, P. Thiemann (éditeurs), *Lecture Notes in Computer Science*, 1110, p. 54–72, février 1996.

[3] C. Consel, S. C. Khoo, « Parameterized Partial Evaluation », *ACM Transactions on Programming Languages and Systems 15*, 3, 1993, p. 463–493.

[4] C. Consel, R. Marlet, « Architecturing software using a methodology for language development », *in: Proceedings of the* 10^{th} *International Symposium on Programming Language Implementation and Logic Programming*, C. Palamidessi, H. Glaser, K. Meinke (éditeurs), *Lecture Notes in Computer Science*, 1490, p. 170–194, Pisa, Italy, septembre 1998. Article invité.

[5] C. Consel, F. Noël, « A General Approach for Run-Time Specialization and its Application to C », *in: Conference Record of the* 23^{rd} *Annual ACM SIGPLAN-SIGACT Symposium on Principles Of Programming Languages*, ACM Press, p. 145–156, St. Petersburg Beach, FL, USA, janvier 1996.

[6] R. Marlet, S. Thibault, C. Consel, « Efficient Implementations of Software Architectures via Partial Evaluation », *Journal of Automated Software Engineering 6*, 4, octobre 1999, p. 411–440.

[7] G. Muller, R. Marlet, E. Volanschi, C. Consel, C. Pu, A. Goel, « Fast, Optimized Sun RPC Using Automatic Program Specialization », *in: Proceedings of the 18th International Conference on Distributed Computing Systems*, IEEE Computer Society Press, p. 240–249, Amsterdam, The Netherlands, mai 1998.

[8] C. Pu, T. Autrey, A. Black, C. Consel, C. Cowan, J. Inouye, L. Kethana, J. Walpole, K. Zhang, « Optimistic Incremental Specialization: Streamlining a Commercial Operating System », *in: Proceedings of the 1995 ACM Symposium on Operating Systems Principles*, ACM Operating Systems Reviews, 29(5), ACM Press, p. 314–324, Copper Mountain Resort, CO, USA, décembre 1995.

[9] S. Thibault, C. Consel, G. Muller, « Safe and Efficient Active Network Programming », *in: 17th IEEE Symposium on Reliable Distributed Systems*, p. 135–143, West Lafayette, Indiana, octobre 1998.

[10] S. Thibault, R. Marlet, C. Consel, « Domain-Specific Languages: from Design to Implementation – Application to Video Device Drivers Generation », *IEEE Transactions on Software Engineering 25*, 3, mai–juin 1999, p. 363–377.

Articles et chapitres de livre

[11] S. Chirokoff, C. Consel, R. Marlet, « Combining Program and Data Specialization », *Higher-Order and Symbolic Computation 12*, 4, 1999.

[12] O. Danvy, U. P. Schultz, « Lambda-Dropping: Transforming Recursive Equations into Programs with Block Structure », *Theoretical Computer Science*, 1999, À paraître.

[13] L. Hornof, J. Noyé, « Accurate Binding-Time Analysis for Imperative Languages: Flow, Context, and Return Sensitivity », *Theoretical Computer Science*, 1999, À paraître.

[14] R. Marlet, S. Thibault, C. Consel, « Efficient Implementations of Software Architectures via Partial Evaluation », *Journal of Automated Software Engineering 6*, 4, octobre 1999, p. 411–440.

[15] G. Muller, R. Marlet, E. Volanschi, « Scaling up Partial Evaluation for Optimizing the Sun Commercial RPC Protocol », *Theoretical Computer Science*, 1999, À paraître.

[16] G. Muller, C. Pu, C. Consel, *Encyclopedia of Distributed Computing*, Kluwer Academic Publisher, 1999, ch. Specialization, À paraître.

[17] G. Muller, U. Schultz, « Harissa: A Hybrid Approach to Java Execution », *IEEE Software*, mars 1999, p. 44–51.

[18] S. Thibault, R. Marlet, C. Consel, « Domain-Specific Languages: from Design to Implementation – Application to Video Device Drivers Generation », *IEEE Transactions on Software Engineering 25*, 3, mai–juin 1999, p. 363–377.

Communications à des congrès, colloques, etc.

[19] M. Braux, J. Noyé, « Changement dynamique de comportement par composition de schémas de conception », *in: Langages et Modèles à Objets (LMO'99)*, p. 147–162, Villefranche sur Mer, France, janvier 1999.

[20] M. Braux, J. Noyé, « Towards partial evaluating reflection in Java », *in: ACM SIGPLAN Workshop on Partial Evaluation and Semantics-Based Program Manipulation*, ACM Press, Boston, MA, USA, janvier 2000. to appear.

[21] S. Chirokoff, C. Consel, « Combining Program and Data Specialization », *in: ACM SIGPLAN Workshop on Partial Evaluation and Semantics-Based Program Manipulation*, ACM Press, p. 45–59, San Antonio, TX, USA, janvier 1999.

[22] R. Marlet, C. Consel, P. Boinot, « Efficient Incremental Run-Time Specialization for Free », *in: Proceedings of the ACM SIGPLAN'99 Conference on Programming Language Design and Implementation (PLDI'99)*, p. 281–292, Atlanta, Georgia, USA, 1–4 mai 1999.

[23] G. Muller, « Les langages dédiés : une approche sûre et efficace à la conception de systèmes d'exploitation », *in: Première conférence française sur les systèmes d'exploitation (CFSE1)*, p. 44–51, Rennes, juin 1999.

[24] U. Schultz, J. Lawall, C. Consel, G. Muller, « Towards Automatic Specialization of Java Programs », *in: Proceedings of the European Conference on Object-oriented Programming (ECOOP'99)*, *Lecture Notes in Computer Science*, *1628*, p. 367–390, Lisbon, Portugal, juin 1999.

[25] U. Schultz, « Black-Box Program Specialization », *in: Proceedings of the Fourth International Workshop on Component-Oriented Programming (WCOP'99)*, W. Weck, J. Bosch, C. Szyperski (éditeurs), Lisbon, Portugal, 1999. To appear as a technical report at Department of Software Engineering and Computer Science, University of Karlskrona/Ronneby.

[26] S. Thibault, J. Marant, G. Muller, « Adapting Distributed Applications Using Extensible Networks », *in: Proceedings of the 19th International Conference on Distributed Computing Systems*, IEEE Computer Society Press, p. 234–243, Austin, Texas, mai 1999.

Rapports de recherche et publications internes

[27] J. Lawall, G. Muller, « Efficient Incremental Checkpointing of Java Programs », *Publication interne n° 1264*, IRISA, Rennes, France, novembre 1999.

[28] J. Lawall, G. Muller, « Optimization of Run-Time Specialized Code Using Data Specialization », *Publication interne n° 1263*, IRISA, Rennes, France, décembre 1999.

[29] F. Mérillon, L. Réveillère, C. Consel, R. Marlet, G. Muller, « Towards Verifiable Device Drivers: An Approach Based on Domain-Specific Languages », *Publication interne n° 1270*, IRISA, Rennes, France, novembre 1999.

[30] U. Schultz, J. Lawall, C. Consel, G. Muller, « Specialization Patterns », *Research Report n° 1242*, IRISA, Rennes, France, mars 1999.