INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Projet COMPOSE

Conception de programmes et systèmes adaptatifs

Rocquencourt

THÈME 2A

Table des matières

Le projet Compose a été localisé à Rennes jusqu'au 30 juin 2001. Depuis le 1er juillet 2001, Compose est localisé à Bordeaux. Compose est un projet commun avec le CNRS, l'Université de Bordeaux 1 (dans le cadre du LaBRI, laboratoire de recherche en informatique de Bordeaux) et l'ENSEIRB (école nationale supérieure en électronique, informatique et radiocommunications de Bordeaux). Compose est actuellement rattaché à l'unité de recherche de Rocquencourt.

1 Composition de l'équipe

Responsable scientifique
Charles Consel [Professeur à l'ENSEIRB]
Assistante de projet
Catherine Godest [TR CNRS, jusqu'au 30 juin 2001]
Personnel Inria
Gilles Muller [CR, responsable du projet jusqu'au 30 juin 2001]
Personnel ENSEIRB
Xavier Delord [Maître de conférences, depuis le 1er septembre 2001]
Benoit Escrig [Maître de conférences, depuis le 1er septembre 2001]
Patrice Kadionik [Maître de conférences, depuis le 1er septembre 2001]
Chercheurs post-doctorants
Dan He [post-doctorant]
Yaiyan Yu [post-doctorant, depuis le 1er juin 2001]
Ingénieur expert
Jocelyn Fréchot [jusqu'au 1er septembre 2001]
Doctorants
Hédi Hamdi [boursier INRIA, depuis le 1er novembre 2001]
Anne-Françoise Le Meur [boursière Inria]
Fabrice Mérillon [boursier Menrt]
Luciano Porto Barreto [boursier Inria]
Laurent Réveillère [boursier Menrt jusqu'au 1er septembre 2001, puis ATER à l'ENSEIRB]
Tarek Sahli [boursier INRIA, depuis le 1er novembre 2001]

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.

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.

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.*

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_n^m 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].

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

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

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 :

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

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 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

[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.

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 [4]. 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 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.

[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).

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,DS00] [2] 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).

4 Domaines d'applications

4.1 Panorama

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, 6], le génie logiciel [5], 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 besoins de ces secteurs de l'industrie informatique sont particulièrement représentatifs de notre

[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.

[DS00] O. Danvy, U. P. Schultz, « Lambda-Dropping: Transforming Recursive Equations into Programs with Block Structure », *Theoretical Computer Science 248*, 1–2, 2000, p. 243–287.

[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.

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 7.3 et 7.4), conception de services génériques (voir module 7.1).

5 Logiciels

5.1 Tempo, un évaluateur partiel pour C

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

Participants : Charles Consel [correspondant], Anne-Françoise Le Meur.

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 [HN00,MMV00]. 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.

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]. 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. Une quarantaine d'utilisateurs en disposent à ce jour, dont Bull, France Telecom et Thomson Multimédia.

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

Mots clés : compilation, Java.

Participant : Gilles Muller [correspondant].

Harissa est un environnement d'exécution du langage Java qui intègre un interprète et un compilateur de code intermédiaire vers C [MS99]. Harissa permet de mélanger au sein d'une

[HN00] L. Hornof, J. Noyé, « Accurate Binding-Time Analysis for Imperative Languages: Flow, Context, and Return Sensitivity », *Theoretical Computer Science*, 2000.

[MMV00] G. Muller, R. Marlet, E. Volanschi, « Accurate Program Analyses for Successful Specialization of Legacy System Software », *Theoretical Computer Science 248*, 1–2, 2000.

[MS99] G. Muller, U. Schultz, « Harissa: A Hybrid Approach to Java Execution », *IEEE Software*, mars 1999, p. 44–51.

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://compose.labri.fr/prototypes/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 : Charles Consel [correspondant].

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.

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://compose.labri.fr/prototypes/jscc`.

5.4 JSpec, un spécialiseur pour Java

Mots clés : évaluation partielle, Java, spécialisation.

Participant : Charles Consel [correspondant].

Nous avons développé un prototype de spécialiseur Java, nommé JSpec (voir module 5.2), par intégration d'outils que nous avons antérieurement réalisés : 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 JSpec pour optimiser une application de filtrage d'images, avec un gain d'un facteur 4 en temps d'exécution [Sch00], et une mise en œuvre incrémentale des points de

[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.

[Sch00] U. Schultz, *Object-Oriented Software Engineering using Partial Evaluation*, Thèse de doctorat, Université de Rennes I, décembre 2000.

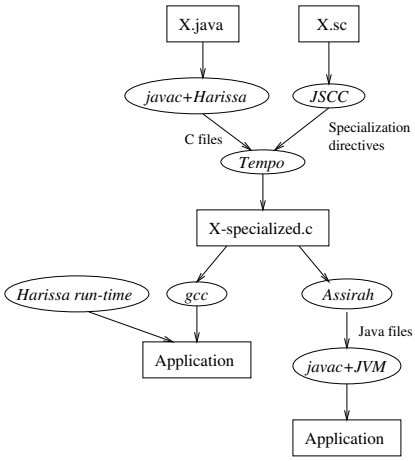

FIG. 1 – Spécialisation de programmes Java

reprise [LM00]. 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.

JSpec est disponible via le Web à l'adresse `http://compose.labri.fr/prototypes/jspec`.

5.5 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 [correspondant].

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 6.2). GAL (Graphic Adaptor Language) est un langage qui permet 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://compose.labri.fr/prototypes/gal`.

[LM00] J. LAWALL, G. MULLER, « Efficient Incremental Checkpointing of Java Programs », *in : Proceedings of the International Conference on Dependable Systems and Networks*, IEEE, p. 61–70, New York, NY, USA, juin 2000.

5.6 PLAN-P

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

Participants : Gilles Muller [correspondant], Dan He.

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 [9], un langage permettant une programmation sûre et efficace des réseaux actifs. Comme pour le langage GAL [10], 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 6.2).

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 [TCM+00]. 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 multipoint adaptative de flux audio et la construction de serveurs HTTP extensibles. PLAN-P et son système d'exécution pour Solaris sont disponibles via le Web à l'adresse `http://compose.labri.fr/prototypes/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.

Résumé : *En dépit des nombreux succès de la spécialisation de programmes, cette technique n'est pas encore accessible à des programmeurs non experts. Cette limitation est principalement due au fait que les chercheurs dans ce domaine se sont uniquement concentrés sur l'outil de transformation et ne se sont pas intéressés à l'intégration de la spécialisation de programmes dans le processus de développement logiciel. Nos travaux récents ont visé à corriger ce manque d'outils.*

[TCM+00] S. Thibault, C. Consel, R. Marlet, G. Muller, J. Lawall, « Static and Dynamic Program Compilation by Interpreter Specialization », *Higher-Order and Symbolic Computation 13*, 3, 2000, p. 161–178.

Outils pour la spécialisation de systèmes d'exploitation

Participants : Charles Consel, Gilles Muller.

Dans un premier temps, nous avons développé, en collaboration avec nos collègues de l'Oregon Graduate Institute (Portland), une boîte à outils qui permet d'assister le processus de spécialisation dans le domaine des systèmes d'exploitation. Nous avons démontré l'intérêt et l'efficacité de ces outils sur trois composants système différents. Ces outils nous ont permis d'obtenir des gains significatifs de performance sans avoir à recourir à des modifications manuelles du code qui sont toujours sources d'erreurs. Enfin, notre expérience d'utilisation de ces outils suggère de nouvelles façons de concevoir les systèmes d'exploitation permettant de concilier performance et robustesse de l'architecture logicielle [12].

Modules de spécialisation

Participants : Charles Consel, Hédi Hamdi, Anne-Françoise Le Meur, Tarek Sahli.

Nous avons travaillé sur une approche déclarative qui vise à intégrer la notion de spécialisation de programmes dans le processus de développement logiciel. Cette approche repose sur la création de *modules de spécialisation*. Un module de spécialisation prend la forme de déclarations qui spécifient les scénarios de spécialisation d'un programme, c'est-à-dire les opportunités de spécialisation de ce programme. Ces déclarations sont écrites par le programmeur lors du développement et sont distinctes du code. Non seulement ces déclarations documentent les opportunités de spécialisation mais elles correspondent également aux conditions à remplir afin de garantir que toutes les opportunités de spécialisation sont bien prises en compte pendant le processus de spécialisation.

Nous avons développé un compilateur pour le langage de déclaration des modules de spécialisation. La compilation de modules de spécialisation permet d'extraire des informations qui servent à :

- configurer automatiquement notre spécialiseur Tempo,
- visualiser graphiquement comment les scénarios de spécialisation de plusieurs programmes se composent. Grâce à cette représentation, le programmeur possède une vue globale des déclarations à tous les moments du développement,
- effectuer des vérifications relatives à la cohérence des scénarios de spécialisation vis à vis des résultats des analyses du spécialiseur. Ces vérifications sont primordiales pour garantir que le spécialiseur génère un programme dont le degré de spécialisation satisfait les déclarations du programmeur.

Nous avons illustré notre approche avec le développement d'encodeurs de données pour les codes correcteurs. Ceci nous a permis de montrer qu'il est possible de spécialiser efficacement et rapidement un ensemble de programmes génériques, si leur développement est couplé avec notre approche déclarative [13].

6.2 Langages dédiés et leur application aux systèmes d'exploitation

Participants : Charles Consel, Gilles Muller, Patrice Kadionik, Benoit Escrig, Xavier

Delord, Dan He, Yaiyan Yu, Luciano Porto Barreto, Fabrice Mérillon, Laurent Réveillère.

Mots clés : langage dédié, pilote de périphériques, réseaux actifs.

Glossaire :

Langage dédié langage qui permet d'exprimer des variations à l'intérieur d'un domaine ou d'une famille d'applications.

Réseau actif architecture de réseau ouverte permettant une adaptation du comportement du réseau par téléchargement dynamique de nouveaux protocoles au sein des routeurs.

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.*

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.

Nous avons proposé une méthodologie pour la conception de langages dédiés reposant sur une approche à deux niveaux [3]. 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 [TCM+00].

[TCM+00] S. Thibault, C. Consel, R. Marlet, G. Muller, J. Lawall, « Static and Dynamic Program Compilation by Interpreter Specialization », *Higher-Order and Symbolic Computation 13*, 3, 2000, p. 161–178.

Notre méthodologie a été validée en premier lieu sur GAL [10], un langage dédié pour l'écriture de pilotes de cartes graphiques (voir module 5.5), et PLAN-P [9], un langage dédié pour l'écriture de protocoles dans les réseaux actifs (voir module 5.6). 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.

Distribution de flux vidéo au moyen de réseaux actifs

La distribution de flux vidéo est une classe d'applications de plus en plus importante (vidéo à la demande, enseignement à distance, jeux...). Toutefois, son implémentation reste un challenge en raison du besoin en bande passante, associé aux contraintes temps réel. Les solutions actuelles reposent sur des approches adaptatives qui permettent de répondre aux variations en bande passante tout en limitant les pertes en qualité visuelle. Cependant, la plupart des techniques d'adaptation actuelles sont limitées en extensibilité et ne tiennent pas compte de l'hétérogénéité des réseaux qui composent actuellement l'Internet.

Dans une architecture réseau reposant sur des routeurs programmables, ces derniers peuvent prendre des décisions à partir de l'observation de l'environnement. Comme la décision est locale, les routeurs peuvent réagir très rapidement à un changement de bande passante. De plus, une décision d'adaptation peut être spécifique à un segment réseau. Enfin, il est possible d'optimiser la politique d'adaptation en fonction de la structure du flux MPEG ou des caractéristiques du routeur. De ce fait, cette architecture est extensible et peut prendre en compte l'hétérogénéité du réseau.

Nous avons implémenté une politique d'adaptation consistant à dégrader le flux MPEG en supprimant prioritairement les trames les moins importantes (c.à.d., les trames B et P). Par comparaison avec une architecture reposant sur des routeurs standard, nos expérimentations ont montré que pour un réseau sans fil IEE802.11 très chargé, notre politique d'adaptation permet aux clients de recevoir et décoder jusqu'à 9 fois plus de trames MPEG [16].

Le domaine des pilotes de périphériques

Pour suivre la cadence effrénée à laquelle les périphériques sortent sur le marché, les pilotes doivent être rapidement développés, mis au point et testés. Nous avons récemment présenté une nouvelle approche pour améliorer la robustesse des pilotes de périphériques reposant sur un langage de définition d'interface, nommé Devil. 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 [6]. 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 périphériques utilisés dans un PC. Plus récemment, nous avons conçu une extension de Devil, appelée Trident, qui permet de décrire les configurations de bus matériel utilisés dans les systèmes embarqués [14].

Afin d'évaluer l'amélioration de la robustesse des pilotes de périphériques offerte par Devil, nous utilisons une analyse de mutations qui nous permet d'injecter des erreurs de programmation dans des pilotes de périphériques Linux. Nous avons effectué ces tests pour des pilotes reposant sur Devil et pour les pilotes originaux écrits en C. Nous évaluons dans quelle mesure les erreurs peuvent être décelées plus tôt dans le processus de développement, en rapportant celles qui sont soit détectées à la compilation, soit à l'exécution. Les résultats de nos expériences sur le pilote de disque IDE Linux prouvent que presque 3 fois plus d'erreurs sont détectées dans le pilote reposant sur Devil que dans le pilote original écrit en C [11, 15].

Perspectives

Dans le cadre de deux collaborations avec France Télécom (voir modules 7.3 et 7.4), nous étudions l'utilisation des langages dédiés pour réaliser le processus d'adaptation au sein de systèmes d'exploitation flexibles. Dans ce cadre, nous étudions un langage pour la conception d'ordonnanceurs de processus pour les applications multimédia.

7 Contrats industriels (nationaux, européens et internationaux)

7.1 ESAPS, projet européen ITEA

Participants : Charles Consel, Anne-Françoise Le Meur.

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 est validée sur deux domaines d'applications : la supervision aérienne et la gestion de commutateurs.

Notre contribution au projet ESAPS est 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. Dans le cadre de ce projet, nous avons contribué au développement de méthodes et d'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 [13].

7.2 DESS, projet européen ITEA

Participants : Charles Consel, Gilles Muller, Anne-Françoise Le Meur, Fabrice Mérillon, Luciano Porto Barreto, Laurent Réveillère, Jocelyn Fréchot.

Ce projet vise à définir une méthodologie de développement de logiciels adaptée aux systèmes embarqués, reposant sur des composants réutilisables, ainsi qu'à développer des outils permettant une mise en œuvre aisée de cette méthodologie.

La contribution du projet Compose vise à évaluer l'utilisation du langage dédié Devil pour écrire des pilotes de périphériques pour les systèmes embarqués. Dans le cadre de cette coopération, nous avons conçu une extension de Devil, appelée Trident, qui permet de décrire les configurations de bus matériel utilisés dans les systèmes embarqués [14].

7.3 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 via l'utilisation de langages dédiés. Dans ce cadre de cette action, nous développons Bossa, un langage dédié et son système d'exécution, qui permet un développement aisé de politiques d'ordonnancement de processus. Une première implémentation de Bossa dans le noyau Linux est en cours de réalisation.

7.4 PHENIX : Noyau d'infrastructure répartie adaptable, contrat RNRT

Participants : Gilles Muller, Charles Consel, Dan He.

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.

8 Actions régionales, nationales et internationales

8.1 Actions internationales

Nous entretenons des relations étroites avec l'Oregon Graduate Institute à Portland (professeur Jonathan Walpole) et le Georgia Institute of Technology (professeur Calton Pu). Notre collaboration porte actuellement sur l'adaptation de composants pour les systèmes distribués.

8.2 Visites et invitations de chercheurs

Julia Lawall, professeur à l'université de Copenhague (DIKU) a effectué deux séjours dans notre projet du 6 au 9 mai et du 24 mai au 18 août.

À notre invitation, Craig Chambers, professeur à l'Université de Washington à Seattle, Calton Pu, professeur au Georgia Institute of Technology, Jonathan Walpole, professeur à l'Oregon Graduate Institute, Marc Shapiro, chercheur à Microsoft Cambridge (RA) et Kenji Kono, professeur à l'université d'electro-communication de Tokyo, ont présenté des séminaires au LaBRI et à l'ENSEIRB.

9 Diffusion de résultats

9.1 Animation de la communauté scientifique

Charles Consel a participé aux comités de programme des conférences suivantes : *European Symposium on Programming* (ESOP 2002), *ACM SIGPLAN ASIAN Symposium on Partial Evaluation and Semantics-Based Program Manipulation* (ASIA-PEPM'02), *Third International Conference on Metalevel Architectures and Reflection* (Reflection'2001), *Second Symposium on Programs as Data Objects* (PADO II).

Gilles Muller a participé aux comités de programme des conférence suivantes : *IEEE Symposium on Reliable Distributed Systems* (SRDS 2001), *IEEE International Conference on Dependable Systems and Networks* (DSN 2001), *2ème Conférence Française sur les Systèmes d'Exploitation* (CFSE-2). Il a également été membre du Jury de la meilleure thèse en système d'exploitation ; ce prix a été décerné à l'occasion de CFSE-2.

9.2 Enseignement

Charles Consel a encadré un stagiaire de DEA : Eric Hennequin.

Charles Consel enseigne en DEA de l'Université de Bordeaux 1, un cours intitulé : « Approche langage à la conception de systèmes adaptatifs ».

9.3 Participation à des colloques, séminaires, invitations

Charles Consel a donné une conférence invitée lors de l'école d'été 2001 de l'Université de Washington et du Microsoft Summer Research Institute à Sleeping Lady, état de Washington. Il a également présenté des séminaires à l'INRIA Rocquencourt et au Georgia Institute of Technology.

Gilles Muller a présenté des séminaires à l'INRIA Sophia-Antipolis (septembre 2001), à VERIMAG (décembre 2001), au Georgia Institute of Technology, à l'Université de Washington à Seattle et au laboratoire de recherche d'ATT dans le New Jersey (novembre 2001).

10 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, 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.

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

[5] 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.

[6] F. Mérillon, L. Réveillère, C. Consel, R. Marlet, G. Muller, « Devil : An IDL for Hardware Programming », *in : 4th Symposium on Operating Systems Design and Implementation (OSDI 2000)*, USENIX Association, p. 17–30, octobre 2000.

[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.

Thèses et habilitations à diriger des recherches

[11] L. Réveillère, *Approche Langage au développement de pilotes de périphériques robustes*, Thèse de doctorat, Université de Rennes 1, France, décembre 2001.

Articles et chapitres de livre

[12] D. McNamee, J. Walpole, C. Pu, C. Cowan, C. Krasic, C. Goel, C. Consel, G. Muller, R. Marlet, « Specialization Tools and Techniques for Systematic Optimization of System Software », *ACM Transactions on Computer Systems 19*, mai 2001, p. 217–251.

Communications à des congrès, colloques, etc.

[13] A.F. Le Meur, C. Consel, « Configurabilité garantie des composants par les modules de spécialisation », *in : Journées composants : flexibilité du système au langage*, Besançon, France, octobre 2001.

[14] F. Mérillon, G. Muller, « Dealing with Hardware in Embedded Software : A General Framework Based on the Devil Language », *in : The Workshop on Languages, Compilers and Tools for Embedded Systems (LCTES 2001) - June 22-23, 2001, Snowbird, Utah*, *36*, 8, p. 121–127, août 2001.

[15] L. Réveillère, G. Muller, « Improving Driver Robustness : an Evaluation of the Devil Approach », *in : The International Conference on Dependable Systems and Networks*, IEEE Computer Society, p. 131–140, Göteborg, Sweden, juillet 2001.

Rapports de recherche et publications internes

[16] D. He, J.L. Lawall, G. Muller, « Distributing MPEG movies over the Internet using programmable networks », *Research Report n° 1263-01*, LaBRI, Bordeaux, France, novembre 2001.