INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Projet COMPOSE

Conception de programmes et systèmes adaptatifs

Futurs

THÈME 2A

Table des matières

1. Composition de l'équipe

Le projet 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).

Responsable scientifique

Charles Consel [Professeur à l'ENSEIRB]

Assistante de projet

Simone Dang Van [1/2 poste secrétariat ENSEIRB, depuis le 1er septembre 2002]

Personnel ENSEIRB

Benoit Escrig [Maître de conférences]
Patrice Kadionik [Maître de conférences]
Laurent Réveillère [Maître de conférences, depuis le 1er septembre 2002]

Chercheurs post-doctorants

Dan He [post-doctorant, jusqu'au 19 mai 2002]
Yaiyan Yu [post-doctorant, jusqu'au 17 septembre 2002]

Ingénieur associé

Abdelaaziz El khaoulany [depuis le 1er septembre 2002]

Doctorants

Hédi Hamdi [boursier INRIA]
Anne-Françoise Le Meur [boursière Inria jusqu'au 1er septembre 2002, puis ATER à l'ENSEIRB]
Luciano Porto Barreto [boursier Inria, jusqu'au 31 décembre 2002]

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.

2.1.1. 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.

2.1.2. 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.

2.1.3. 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.

2.1.4. 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.

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.

3.1.1. 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 [36][38].

À la différence d'une stratégie de transformation de programmes générale à la Burstall et Darlington [27], 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 [36], l'optimisation de programmes numériques [25], le filtrage [30] et l'instrumentation de programmes [37].

3.1.2. 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 :

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

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 :

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

Les données disponibles pendant l'évaluation partielle sont dites *statiques* (la donnée v). Les données manquantes sont dites *dynamiques* (la donnée w) [36]. 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.

3.1.3. 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* [36]. 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 [26][28], C [23] et Pascal [40].

3.1.4. 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.

3.1.5. Analyses et outils

Les principes d'évaluation partielle décrits plus haut permettent le développement rigoureux d'analyses de programmes performantes [24][29] et la conception de transformations de programmes [31][32][28][33] [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).

4. Domaines d'application

4.1. Panorama

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

L'adaptabilité des logiciels est un besoin très général qui a été clairement identifié dans des domaines aussi variés que les télécommunications [14], les systèmes d'exploitation [10][8][9], le génie logiciel [6], le calcul numérique [25] et le graphisme [34]. 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 France Télécom. Les 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 l'optimisation de systèmes d'exploitation (voir modules 7.1 et 7.2).

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 [35][41]. 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 [8]. 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. JSpec, un spécialiseur pour Java

Mots clés : *évaluation partielle*, *Java*, *spécialisation*.

Participant : Charles Consel [correspondant].

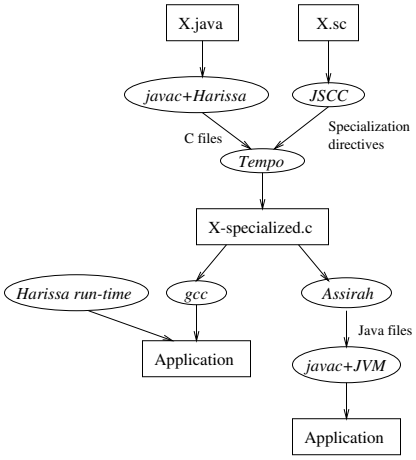

Figure 1. Spécialisation de programmes Java

Nous avons développé un prototype de spécialiseur Java, nommé JSpec, par intégration d'outils que nous avons antérieurement réalisés : JSCC, le compilateur des classes de spécialisation, Harissa notre traducteur de Java vers C, 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 [42], et une mise en œuvre incrémentale des points de reprise [39]. 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.

5.3. PLAN-P, un langage pour les routeurs programmables

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

Participants : Charles Consel [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 [14], un langage permettant une programmation sûre

et efficace des réseaux actifs. Comme pour le langage GAL [15], 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 [43]. 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. 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.

5.4. Devil, un langage pour les pilotes de périphériques

Mots clés : *génie logiciel*, *langage dédié*, *pilotes de périphériques*.

Participants : Laurent Réveillère [correspondant], Charles Consel.

Nous avons développé 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 [11].

Nous avons validé la puissance d'expression du langage Devil en réussissant à spécifier la plupart des circuits périphériques utilisés dans un PC. Nous avons conçu une extension de Devil, appelée Trident, qui permet de décrire les configurations de bus matériel utilisés dans les systèmes embarqués [5]. Nous avons développé un compilateur, nommé Taz, pour le langage Devil générant du code C pour le système d'exploitation Linux. Nous avons évalué la performance à l'exécution de différentes pilotes et nous n'avons constaté aucune perte de performance significative comparé aux pilotes originaux écrits en C [9]. Le langage Devil permet d'améliorer la robustesse des pilotes de périphériques en rendant possible un grand nombre de vérifications statiques sur les spécifications et en incluant du code dans les pilotes pour réaliserr de vérifications dynamiques. Les résultats de nos expériences, reposant sur une analyse de mutations, prouvent que jusqu'à près de 6 fois plus d'erreurs sont détectées dans le pilote reposant sur Devil que dans le pilote original écrit en C [13][12].

Devil est la continuation d'une étude sur la description de cartes graphiques pour le serveur X Windows XFree86. Nous avons développé un langage, nommé GAL, permettant de décrire des pilotes de périphériques dans ce contexte [15]. Bien que couvrant un domaine très restreint, ce langage a permis de valider l'approche avec succès.

L'implémentation du compilateur Taz est disponible via le web à l'adresse http://compose.labri.fr/prototypes/devil.

5.5. Modules de spécialisation

Mots clés : *outils de génie logiciel*, *langage dédié*, *spécialisation*, *composants génériques*.

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

Nous avons travaillé sur une approche déclarative basée sur la définition de modules de spécialisation (voir Section 6.1.1) qui permet d'une part de rendre le processus de spécialisation prévisible par rapport aux déclarations et d'autre part de faciliter l'utilisation d'un spécialiseur [22].

Nous avons conçu et implémenté un compilateur pour le langage des modules de spécialisation. Une partie de ce compilateur a été spécialement conçu pour s'interfacer avec le spécialiseur Tempo. Ainsi la compilation des modules de spécialisation permet notamment de générer automatiquement des fichiers de paramétrisation du spécialiseur Tempo, ce qui simplifie son utilisation. De plus, nous avons rajouté des vérifications dans les analyses de Tempo pour rendre le comportement du spéciliseur prévisible. Enfin, nous avons développé un environnement graphique [21]. Cet environnement offre aux programmeurs des outils d'aide au développement de composants spécialisables et permet aux utilisateurs de ces composants de les spécialiser en fonction de leur besoin.

6. Résultats nouveaux

6.1. Principes, techniques et outils de spécialisation

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

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.

6.1.1. Modules de spécialisation

Participants : Charles Consel, Anne-Françoise Le Meur.

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 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 [21][22].

6.2. Langages dédiés et leur application aux systèmes d'exploitation

Participants : Charles Consel, Laurent Réveillère, Patrice Kadionik, Benoit Escrig, Dan He, Yaiyan Yu, Luciano Porto Barreto.

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.

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 [43].

Notre méthodologie a été principalement validée sur PLAN-P [14], un langage dédié pour l'écriture de protocoles dans les réseaux actifs (voir module 5.3), et Devil [13], un langage dédié pour l'écriture de pilotes de périphériques 5.4. 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, jusqu'à 6 fois plus d'erreurs sont détectées dans un pilote reposant sur Devil que dans un pilote équivalent écrit en C [12]. Enfin, les programmes écrits dans ces deux langages sont aussi performants que des programmes équivalents écrits en C.

6.2.1. 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 [19].

6.2.2. Spécification et implémentation de politiques d'ordonnancement de processus

L'apparition constante de nouvelles applications couplée à l'utilisation de l'ordinateur sur des domaines spécifiques requièrent l'utilisation de politiques d'ordonnancement spécialisées. Par exemple, les applications multimédia ou celles employées dans le domaine des systèmes embarqués possèdent des contraintes de temps d'exécution et ont des besoins très spécifiques en termes de performance, temps de réponse, extensibilité et disponibilité. Toutefois, le développement d'un ordonnanceur soulève encore de nombreux problèmes. D'abord, le soucis de performance se traduit par un code fortement optimisé, ce qui augmente le risque d'introduire une erreur dans l'implémentation d'une politique d'ordonnancement. Or une telle faute peut avoir de sévères conséquences sur le fonctionnement du système tout entier. En outre, le manque d'outils de mise au point rend le processus de développement difficile et décourage l'expérimentation de nouvelles politiques dans des systèmes réels.

Pour faciliter la spécification et l'implémentation de politiques d'ordonnancement dans le noyau d'un système d'exploitation, nous proposons un cadre de développement nommé Bossa [18]. Ce dernier s'articule autour de deux principaux axes : (i) un système d'exécution extensible pour l'implémentation d'ordonnanceurs et leurs intégrations dans le noyau du système cible, (ii) un langage dédié permettant un développement aisé de politiques d'ordonnancement.

Le support d'exécution propose une plate-forme logiciel reposant sur un modèle à événements permettant la spécification d'un ordonnanceur comme un ensemble de traitants d'événement. Ceci rend le code d'ordonnancement modulaire en séparant la politique du reste du noyau. Le langage dédié a pour but de faciliter l'implémentation de politiques d'ordonnancement en fournissant d'une part des abstractions de haut niveau plus proches du programmeur du domaine et d'autre part en rendant possible la vérification des politiques [20].

Nous avons utilisé Bossa pour implémenter des politiques telles que EDF (Earliest Deadline First), la politique d'ordonnancement de Linux, Stride et des extensions basées sur la notion de progrès d'une application. Bossa est disponible via le web à l'adresse http://compose.labri.fr/prototypes/bossa.

7. Contrats industriels

7.1. Adaptation de systèmes réflexifs au moyen de langages dédiés, contrat CNET-CTI

Participants : 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.

7.2. PHENIX : Noyau d'infrastructure répartie adaptable, contrat RNRT

Participants : 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 21 au 26 avril et du 1er juillet au 19 août.

À notre invitation, Krzysztof Czarnecki, Chercheur à DaimlerChrysler (Allemagne), Christian Lengauer, Professeur à l'Université de Passau (Allemagne), Laurent Hascoet, Chercheur à l'INRIA, Pierre Cointe, professeur à l'école des mines de Nantes, Wu-Chang Feng, professeur à l'Oregon Graduate Institute (USA), Olivier Danny, professeur à l'université de Aarhus (Danemark) ont présenté des séminaires au LaBRI et à l'ENSEIRB.

9. Diffusion des résultats

9.1. Animation de la communauté scientifique

Charles Consel a participé aux comités de programme des conférences suivantes : ACM SIGPLAN *Conference on Programming Language Design and Implementation* (PLDI 2002), *European Symposium on Programming* (ESOP 2002), ACM SIGPLAN ASIAN *Symposium on Partial Evaluation and Semantics-Based Program Manipulation* (ASIA-PEPM 2002).

Charles Consel a été co-président du comité de programme de la conférence ACM SIGPLAN/SIGSOFT Conference on *Generative Programming and Component Engineering* (GPCE 2002). Il est co-organisateur de l'école de Dagstulh en 2003 sur les langages dédiés.

9.2. Enseignement

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 LDTA 2002. *2nd Workshop on Language Descriptions, Tools and Applications*. Il a également présenté des séminaires à l'INRIA Rocquencourt et au Georgia Institute of Technology (USA, Atlanta).

Laurent Réveillère a présenté ses travaux de thèse sur Devil à l'Université de Berkeley, à Microsoft Research (Redmond) ainsi qu'a l'Université de Carnegie Mellon.

10. Bibliographie

Bibliographie de référence

[1] C. CONSEL, L. HORNOF, J. LAWALL, R. MARLET, G. MULLER, J. NOYÉ, S. THIBAULT, N. VOLANSCHI. *Tempo : Specializing Systems Applications and Beyond.* in « ACM Computing Surveys, Symposium on Partial Evaluation », numéro 3, volume 30, 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 », série Lecture Notes in Computer Science, numéro 1110, éditeurs O. DANVY, R. GLÜCK, P. THIEMANN., pages 54-72, février, 1996.

[3] C. CONSEL, R. MARLET. *Architecturing software using a methodology for language development.* in « Proceedings of the 10th International Symposium on Programming Language Implementation and Logic Programming », série Lecture Notes in Computer Science, numéro 1490, éditeurs C. PALAMIDESSI, H. GLASER, K. MEINKE., pages 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, pages 145-156, St. Petersburg Beach, FL, USA, janvier, 1996.

[5] 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 », numéro 8, volume 36, pages 121-127, août, 2001.

[6] R. MARLET, S. THIBAULT, C. CONSEL. *Efficient Implementations of Software Architectures via Partial Evaluation.* in « Journal of Automated Software Engineering », numéro 4, volume 6, octobre, 1999, pages 411-440.

[7] 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.* in « ACM Transactions on Computer Systems », volume 19, mai, 2001, pages 217-251.

[8] 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, pages 240-249, Amsterdam, The Netherlands, mai, 1998.

[9] 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, pages 17-30, octobre, 2000.

[10] 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, pages 314-324, Copper Mountain Resort, CO, USA, décembre, 1995.

[11] L. RÉVEILLÈRE, F. MÉRILLON, C. CONSEL, R. MARLET, G. MULLER. *A DSL Approach to Improve Productivity and Safety in Device Drivers Development.* in « Proceedings of the 15th IEEE International Conference on Automated Software Engineering (ASE 2000) », IEEE Computer Society Press, pages 101-109, Grenoble, France, septembre, 2000.

[12] 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, pages 131-140,

Göteborg, Sweden, juillet, 2001.

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

[14] S. THIBAULT, C. CONSEL, G. MULLER. *Safe and Efficient Active Network Programming.* in « 17th IEEE Symposium on Reliable Distributed Systems », pages 135-143, West Lafayette, Indiana, octobre, 1998.

[15] S. THIBAULT, R. MARLET, C. CONSEL. *Domain-Specific Languages : from Design to Implementation - Application to Video Device Drivers Generation.* in « IEEE Transactions on Software Engineering », numéro 3, volume 25, mai-juin, 1999, pages 363-377.

[16] 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.

Thèses et habilitations à diriger des recherche

[17] A. L. MEUR. *Approche déclarative à la spécialisation de programmes C.* Thèse de doctorat, Université de Rennes 1, France, décembre, 2002, à paraître.

Communications à des congrès, colloques, etc.

[18] L. P. BARRETO, G. MULLER. *Bossa : a Language-based Approach to the Design of Real-time Schedulers.* in « 10th International Conference on Real-Time Systems (RTS'2002) », Paris, France, mars 26-28, 2002.

[19] D. HE, G. MULLER, J. LAWALL. *Distributing MPEG movies over the Internet using programmable networks.* in « The 22nd International Conference on Distributed Computing Systems (ICDCS'02) », pages 161-170, Vienna, Austria, juillet, 2002.

[20] J. LAWALL, G. MULLER, L. P. BARRETO. *Capturing OS expertise in a modular type system : the Bossa experience.* in « Proceedings of the ACM SIGOPS European Workshop 2002 (EW2002) », pages 54-62, Saint-Emillion, France, septembre, 2002.

[21] A.-F. LE MEUR, C. CONSEL, B. ESCRIG. *An Environment for Building Customizable Software Components.* in « IFIP/ACM Conference on Component Deployment », Berlin, Germany, juin, 2002.

[22] A.-F. LE MEUR, J. LAWALL, C. CONSEL. *Towards Bridging the Gap Between Programming Language and Partial Evaluation.* in « ACM SIGPLAN Workshop on Partial Evaluation and Semantics-Based Program Manipulation », ACM Press, Portland, OR, USA, janvier, 2002.

Bibliographie générale

[23] 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.

[24] J. M. ASHLEY, C. CONSEL. *Fixpoint Computation for Polyvariant Static Analyses of Higher-Order Applicative Programs.* in « ACM Transactions on Programming Languages and Systems », numéro 5, volume 16,

1994, pages 1431-1448.

[25] A. BERLIN. *Partial Evaluation Applied to Numerical Computation.* in « ACM Conference on Lisp and Functional Programming », ACM Press, pages 139-150, Nice, France, 1990.

[26] A. BONDORF. *Automatic Autoprojection of Higher Order Recursive Equations.* in « ESOP'90, 3rd European Symposium on Programming », série Lecture Notes in Computer Science, volume 432, Springer-Verlag, éditeurs N. D. JONES., pages 70-87, 1990.

[27] R. M. BURSTALL, J. DARLINGTON. *A Transformational System for Developing Recursive Programs.* in « Journal of ACM », numéro 1, volume 24, 1977, pages 44-67.

[28] C. CONSEL. *A Tour of Schism.* in « Partial Evaluation and Semantics-Based Program Manipulation », ACM Press, pages 66-77, Copenhagen, Denmark, juin, 1993.

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

[30] C. CONSEL, O. DANVY. *Partial Evaluation of Pattern Matching in Strings.* in « Information Processing Letters », numéro 2, volume 30, 1989, pages 79-86.

[31] C. CONSEL, O. DANVY. *From Interpreting to Compiling Binding Times.* in « ESOP'90, 3rd European Symposium on Programming », série Lecture Notes in Computer Science, volume 432, Springer-Verlag, éditeurs N. JONES., pages 88-105, 1990.

[32] C. CONSEL, O. DANVY. *For a Better Support of Static Data Flow.* in « Functional Programming Languages and Computer Architecture », série Lecture Notes in Computer Science, volume 523, Springer-Verlag, éditeurs J. HUGHES., pages 496-519, Cambridge, MA, USA, août, 1991.

[33] O. DANVY, U. P. SCHULTZ. *Lambda-Dropping : Transforming Recursive Equations into Programs with Block Structure.* in « Theoretical Computer Science », numéro 1-2, volume 248, 2000, pages 243-287.

[34] B. GUENTER, T. KNOBLOCK, E. RUF. *Specializing Shaders.* in « Computer Graphics Proceedings », série Annual Conference Series, ACM Press, pages 343-350, 1995.

[35] L. HORNOF, J. NOYÉ. *Accurate Binding-Time Analysis for Imperative Languages : Flow, Context, and Return Sensitivity.* in « Theoretical Computer Science », 2000.

[36] N. JONES, P. SESTOFT, H. SØNDERGAARD. *Mix : a Self-Applicable Partial Evaluator for Experiments in Compiler Generation.* in « Lisp and Symbolic Computation », numéro 1, volume 2, 1989, pages 9-50.

[37] 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), pages 338-352, Toronto, Ontario, Canada, juin, 1991.

[38] S. C. KLEENE. *Introduction to Metamathematics.* Van Nostrand, 1952.

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

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

[41] G. MULLER, R. MARLET, E. VOLANSCHI. *Accurate Program Analyses for Successful Specialization of Legacy System Software.* in « Theoretical Computer Science », numéro 1-2, volume 248, 2000.

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

[43] S. THIBAULT, C. CONSEL, R. MARLET, G. MULLER, J. LAWALL. *Static and Dynamic Program Compilation by Interpreter Specialization.* in « Higher-Order and Symbolic Computation », numéro 3, volume 13, 2000, pages 161-178.