Habilitation à Diriger des Recherches

de l'Université Lille 1 – Sciences et Technologies
Spécialité « Informatique »

Langages réflexifs pour le développement d'applications de robotique mobile

par

Luc Fabresse

Soutenue le 8 décembre 2015, devant le jury composé de :

Président :
- **Alain Plantec** — Professeur, Université de Bretagne Occidentale, Membre de l'équipe MOCS du Lab-STICC

Rapporteurs :
- **Pierre Cointe** — Professeur, Mines Nantes, Directeur du Laboratoire d'Informatique de Nantes Atlantique
- **Théo D'Hondt** — Professeur, Vrije Universiteit Brussel, Responsable du Software Languages Lab
- **Jean-Bernard Stefani** — Directeur de Recherche, Délégué Scientifique du centre de recherche INRIA Grenoble

Examinateurs :
- **David Andreu** — Maître de Conférences, HDR, Univ. Montpellier 2, LIRMM, Membre de l'équipe DEMAR
- **Roel Wuyts** — Professeur, IMEC, Université KU Leuven, Membre de l'équipe DistriNet

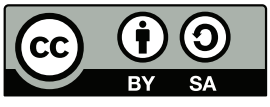
Garant :
- **Stéphane Ducasse** — Directeur de Recherche, INRIA Lille - Nord Europe, Responsable de l'équipe RMoD

Invité :
- **Noury Bouraqadi** — Professeur, Mines Douai, Responsable du thème CAR (Composants, Agents et Robots)

Numéro d'ordre : 41896

Luc Fabresse, 2015.

Département Informatique et Automatique
941 rue Charles Bourseul
CS 10838
59508 Douai Cedex
France



RÉSUMÉ

Dans un avenir proche, nous serons confrontés au besoin de développer rapidement et massivement des logiciels pour répondre aux nouveaux usages liés à l'essor de la robotique mobile et autonome. Or, le cycle de développement actuel de ces applications est généralement long : développement, compilation, déploiement et exécution. Dans ce contexte, mon objectif de raccourcir ce cycle et permettre un retour immédiat (dans une logique *live programming*) en proposant de nouveaux langages, outils et infrastructures aux développeurs. Mes travaux de recherche se concentrent sur les couches hautes des applications robotiques (avec des contraintes matérielles et temporelles plus faibles) et se fondent sur l'idée que les *langages réflexifs* sont de meilleurs candidats pour répondre à cet objectif, car ils offrent un haut niveau d'abstraction et intègrent déjà des mécanismes permettant l'adaptation et l'extension des programmes. Dans ce contexte, ce mémoire décrit les travaux que j'ai menés ou auxquels j'ai participé selon quatre thèmes principaux.

Dans le premier thème, nous avons proposé des *langages à composants* permettant d'exprimer l'architecture d'un logiciel directement dans son code source comme un assemblage de composants pouvant être distants ou pervasifs. Cette explicitation de l'architecture facilite ainsi l'adaptation des programmes.
Le deuxième thème est celui des langages à objets réflexifs qui offrent une représentation de leur propre infrastructure grâce à des méta-objets ainsi que des méta-opérations pour la modifier. Étendre le noyau d'un langage réflexif est toutefois complexe. Cela nécessite de modifier son amorçage qui est généralement en partie enfoui dans le code de sa machine virtuelle. Nous avons donc proposé une approche de plus haut niveau basée sur la co-exécution d'environnements par une même machine virtuelle pour amorcer un langage réflexif à partir d'une définition circulaire de lui-même.
Le troisième thème est lié aux ressources limitées disponibles sur un robot mobile (e. g. mémoire, capacité de calcul). Pour réduire la mémoire consommée par une application, nous avons proposé deux nouveaux mécanismes basés sur la réflexion : le premier élimine le code non-utilisé avant son déploiement et le second est une mémoire virtuelle au niveau applicatif permettant de gérer les objets non-utilisés mais référencés durant son exécution. Nous travaillons également sur la possibilité de décharger le processeur central du robot en déportant automatiquement certains calculs coûteux vers des architectures matérielles spécialisées comme des FPGAs.
Le quatrième thème présente des outils et des infrastructures pour améliorer le développement d'applications mono et multi-robots. Nous décrivons ainsi une infrastructure utilisant la réflexion pour déboguer à distance une application s'exécutant sur un robot mobile. Notre application cible est la cartographie d'un territoire inconnu par une flotte de robots. Pour faciliter la comparaison objective de solutions de cartographie multi-robots, nous avons proposé une infrastructure permettant de simuler le même code que celui qui serait déployé sur des robots réels ainsi que des métriques quantifiables (e. g. temps d'exploration, mémoire et CPU consommés).

À travers ces quatre thèmes, ces travaux proposent des solutions originales pour développer, tester, déployer, exécuter et évaluer des applications contraintes et mobiles comme celles pour robots autonomes. Ces travaux ouvrent de nouvelles pistes de recherche comme le développement agile d'applications robotiques ou encore l'adaptation dynamique de ces applications pour une meilleure prise en compte du contexte.

Mots-clés : langage réflexif, langage à composants, mémoire virtuelle applicative, débogage à distance, robotique mobile et autonome

ABSTRACT

Reflective Languages for Mobile Robotics Applications Development

Mobile autonomous robots are progressively entering the mass market. This creates a growing demand to develop new service robotics applications. However, the typical development cycle of a mobile robotics application is long and involves several steps: development, (cross-)compilation, deployment and execution. My main objective is to shorten this cycle and support a live programming style by proposing new models, languages, tools and infrastructures to the developers. My researches focus on high-level parts of robotics applications (weaker hardware and time constraints) and advocate the use of dynamic and reflective languages. Indeed, those languages provide a high-level of abstraction and mechanisms to adapt and extend programs. In this context, this manuscript describes the four themes of my research.

In the first theme, we propose *component-oriented languages* allowing developers to express the architecture of an application directly in its source code as an assembly of components that may be remote or pervasive. Making architectures explicit ease program adaptation.
The second theme is about extending the kernel of a reflective language. Such extensions may require modifying the bootstrap of the language that is often mixed and hidden in its virtual machine code. In this regard, we proposed a higher-level approach based on the co-execution of multiple language runtime environments on the top of the same virtual machine. This infrastructure has been successfully used to bootstrap multiple language kernels from their self-definitions.
The third theme is about limited resources available on a mobile robot (e. g. memory, computing capabilities). We proposed new mechanisms based on reflection to reduce the memory consumption of an object-based application: the first one is a dynamic dead code elimination technique and the second one is an application-level virtual memory that can swap out and in, referenced but unused objects during the execution. We also work on pushing part of the heavy computations to specialized hardware components such as FPGAs.
The fourth and last theme is about tools and infrastructures to improve the development of single and multi-robots applications. We proposed a debugging solution for a remote applications executed on a mobile robot. This solution relies on a mirror-based reflective model and allows developers to interactively program in the context of errors when they occur. Our main application domain is the exploration and mapping of unknown environments using multi-robots systems. To ease the development but also comparisons of different solutions in this domain, we proposed a benchmarking infrastructure that can simulate exactly the same code as the one that would be deployed on real robots. We also worked on metrics (e. g. exploration time, memory and CPU consumed) to objectively compare different solutions.

Throughout these four themes, our research introduce new solutions with many perspectives to develop, test, deploy, execute and benchmark mobile and autonomous robotics applications.

Keywords: Mobile robotics applications, reflective language, components, remote debugging, application-level virtual memory

REMERCIEMENTS

Le meilleur ami de « merci » est « beaucoup ».

Michel Bouthot

Je commence par remercier chaleureusement Pierre Cointe, Theo D'Hondt et Jean-Bernard Stefani pour avoir accepté d'être rapporteurs de cette HDR malgré leurs nombreuses responsabilités et leurs agendas particulièrement chargés en cette période de l'année. Je formule également les mêmes remerciements aux examinateurs David Andreu, Alain Plantec et Roel Wuyts. Merci à tous pour avoir pris le temps d'analyser mon travail de recherche et pour toutes vos remarques constructives.

Je souhaite ensuite remercier deux figures importantes dans ma vie professionnelle. Tout d'abord, Stéphane Ducasse qui a accepté d'être le garant de cette HDR. Mais, je le remercie également pour toute l'énergie positive qu'il insuffle autour de lui et pousse ceux qui le côtoie à se dépasser et s'investir dans des travaux de recherche passionnants autour des langages dynamiques et de la réflexion ou même dans l'enseignement à travers le montage d'un MOOC (*Massive Open Online Course*) sur le développement par objets à travers Pharo. Je remercie ensuite Noury Bouraqadi qui est sans conteste l'une des principales raisons de ma présence aux Mines de Douai. Je le remercie de m'avoir fait découvrir le domaine de la robotique mobile ainsi que pour tous ses conseils, nos discussions scientifiques et nos séances de pair-programming. Noury, collaborer au quotidien avec toi est un réel plaisir. Merci à vous deux donc, ne changez rien !

Je souhaite ensuite remercier tous les étudiants de Master, ingénieurs, doctorants et post-doctorants que j'ai co-encadrés ces dernières années : Mariano Martinez-Peck, Matthieu Faure, Nick Papoulias, Guillermo Polito, Santiago Bragagnolo, Zhi Yan, Pablo Estefó, Khelifa BAIZID, Xuan Sang LE et Max Mattone. J'ai appris au contact de chacun d'entre vous et ce mémoire d'HDR est également un peu le votre puisqu'il décrit et synthétise les travaux que nous avons menés ensemble.

Parmi les enseignements-chercheurs du département IA, je tiens à remercier plus particulièrement Eric Duviella et Cécile Labarre pour leurs conseils avisés quant à la rédaction de mémoire. Merci également à Arnaud Doniec,

Anthony Fleury et Guillaume Lozenguez avec qui c'est toujours un plaisir de travailler.

Je remercie également l'ensemble des membres du DIA et des Mines de Douai. Je n'en citerai que quelques uns ci-dessous, que les autres me pardonne de les passer sous silence. Merci à tous mes collègues enseignants dont Rémy Pinot et Mathieu Vermeulen avec qui l'organisation du « projet Internet » est chaque année une tâche conséquente mais plaisante grâce à eux. Merci à trois piliers incontournables qui facilitent mon quotidien administratif : Christine Delille, Muriel Morgan et Annie Gaillard. Merci aussi à David Gille qui fait de son mieux pour satisfaire mes demandes en terme de ressources informatiques que ce soit pour des expérimentations de robotique ou pour des enseignements. Certes, j'aimerais toujours moins de contraintes réseau, moins de proxies et moins de Windows mais je sais qu'il fait au mieux ;-)

Je remercie enfin tous les sportifs qui font que certaines pauses déjeuners sont de vrais moments de détente et de convivialité : François Corté (le coach), Sébastien Ambellouis, Jean-François Beguin, Noury Bouraqadi, Eric Duviella, Kathy Fronton, Frédéric Klopcic, Etienne Leblanc et Carole Portillo.

Je remercie également mon frère qui est un soutien sans faille toujours disponible pour prêter une oreille attentive. Il est également mon complice et spécialiste attitré pour me tenir informé et tester les nouveautés technologiques et ludiques indispensables.

Pour finir, je tiens à remercier Isabelle Fabresse, ma tendre moitié qui m'a suivi sans hésiter à l'autre bout de la France même si tous les jours n'ont pas été faciles depuis. Je la remercie donc ici pour sa joie de vivre au quotidien et la vie pleine de bonheur qu'elle su nous construire avec Tom et Lily dans cette région froide mais tellement chaleureuse.

Merci à tous !

Luc

TABLE DES MATIÈRES

1 INTRODUCTION 1
- 1.1 Contexte scientifique . . . 1
- 1.2 Problématiques . . . 2
- 1.3 Objectif . . . 5
- 1.4 Approche adoptée . . . 6
  - 1.4.1 Langages de programmation haut niveau . . . 6
  - 1.4.2 Langages réflexifs . . . 7
- 1.5 Thèmes de recherche . . . 8
  - 1.5.1 Langages et composants . . . 8
  - 1.5.2 Réflexion et modularité . . . 10
  - 1.5.3 Gestion des ressources . . . 11
  - 1.5.4 Infrastructures et outils . . . 12
- 1.6 Organisation du manuscrit . . . 14

2 LANGAGES ET COMPOSANTS 15
- 2.1 Langages à composants . . . 16
  - 2.1.1 Motivation . . . 16
  - 2.1.2 État de l'art . . . 18
  - 2.1.3 CLIC : un langage pour la symbiose objets/composants . 20
  - 2.1.4 SCL : un langage à composants pur . . . 24
  - 2.1.5 COMPO : un langage à composants réflexif . . . 27
  - 2.1.6 CLACS : contraintes architecturales de première classe . . 28
- 2.2 Composants et mobilité . . . 29
  - 2.2.1 Motivation . . . 29
  - 2.2.2 État de l'art . . . 30
  - 2.2.3 SDL : un langage d'assemblage pour composants pervasifs 31
- 2.3 Travaux en cours et perspectives . . . 35
- 2.4 Synthèse des contributions . . . 36

3 RÉFLEXION ET MODULARITÉ 37
- 3.1 Modularisation du méta-niveau d'un langage à objets réflexif . 38
  - 3.1.1 Motivation . . . 38
  - 3.1.2 État de l'art . . . 39
  - 3.1.3 MetaTalk : un langage supportant l'injection dynamique de la réflexion . . . 41
- 3.2 Amorçage de langages à objets réflexifs . . . 46
  - 3.2.1 Motivation . . . 46
  - 3.2.2 État de l'art . . . 47
  - 3.2.3 Espell : virtualisation d'environnements d'exécution de langages . . . 50

3.2.4 Seed : amorçage de langages réflexifs avec Espell 53
3.3 Travaux en cours et perspectives 55
3.4 Synthèse des contributions 56

4 GESTION DES RESSOURCES 57
4.1 Construction d'environnements d'exécution de taille minimale 58
4.1.1 Motivation 58
4.1.2 État de l'art 59
4.1.3 L'approche « *Run-fail-grow* » 63
4.2 Gestion de la mémoire à l'exécution 68
4.2.1 Motivation 68
4.2.2 État de l'art 69
4.2.3 Marea : une mémoire virtuelle au niveau applicatif . . . 73
4.3 Travaux en cours et perspectives 79
4.4 Synthèse des contributions 80

5 INFRASTRUCTURES ET OUTILS POUR LA ROBOTIQUE 81
5.1 Débogage à distance 82
5.1.1 Motivation 82
5.1.2 État de l'art 83
5.1.3 Mercury : une infrastructure de débogage à distance . . 88
5.2 Banc d'essai pour applications multi-robots 90
5.2.1 Motivation 90
5.2.2 État de l'art 91
5.2.3 Nos travaux 94
5.3 Travaux en cours et perspectives 101
5.4 Synthèse des contributions 103

6 SYNTHÈSE ET PERSPECTIVES 105
6.1 Synthèse 105
6.2 Perspectives 107
6.2.1 Développement « agile » d'applications de robotique de services 107
6.2.2 Adaptation et mise à jour dynamique d'applications . . 109
6.2.3 *Benchmarking* de systèmes multi-robots 110

GLOSSAIRE 111

BIBLIOGRAPHIE 113

ANNEXES 139

A CURRICULUM VITÆ DÉTAILLÉ 141
Etat civil 142
Synthèse des activités de recherche 143
Activités d'enseignement 144
Participation à des projets d'enseignement 145
Participation à des projets scientifiques 146

Co-encadrement de post-doctorants 149
Co-encadrement de doctorants 150
Encadrement d'ingénieurs et de masters 153
Co-organisation de conférences 154
Séjours invités 154
Participation à des jurys de thèse 155
Relecture d'articles 155
Implication dans la communauté scientifique 156
Activités administratives 156
Synthèse des travaux 157
Liste des publications depuis 2007 159

1 INTRODUCTION

> Ne dis pas peu de choses en beaucoup de mots,
> mais dis beaucoup de choses en peu de mots.
>
> PYTHAGORE

Ce mémoire synthétise les travaux de recherche menés depuis ma thèse effectuée au sein du Laboratoire d'Informatique de Robotique et de Microélectronique de Montpellier (LIRMM) dans l'équipe MaREL[1] et soutenue en 2007. Ces travaux ont été réalisés dans un premier temps au LIRMM en tant qu'ATER puis à l'Ecole des Mines de Douai en tant qu'enseignant-chercheur depuis 2008 au sein du département Informatique et Automatique[2] et de son unité de recherche[3] (UR IA).

1.1 CONTEXTE SCIENTIFIQUE

L'omniprésence actuelle des équipements électroniques communicants résulte de trois principaux facteurs. Le premier est la miniaturisation et l'accroissement de la puissance de calcul comme suggéré par la conjecture de Moore [Moo65]. Le deuxième est l'essor des technologies de connexion sans fil en terme de portée et de débit. Le troisième est l'augmentation de la capacité des batteries suite à des changements de technologie chimique (Ni-MH, Lipo). Ces trois facteurs font que nous sommes actuellement immergés dans un monde d'équipements connectés aussi appelés *Cyber-Physical System* (CPS).

Un *système cyber-physique* [Lee08] est un système intégrant des capacités de calcul et de communication pour contrôler des entités (ou processus) dans le monde physique. Ces systèmes peuvent être fixes comme des capteurs de température ou de présence par exemple. D'autres sont mobiles comme les téléphones portables ou les tablettes ; la mobilité étant ici subie.

Enfin, les *robots mobiles et autonomes* [SK08] constituent le sous-ensemble le plus intéressant et complexe des systèmes cyber-physiques. Ils sont gé-

1 https://www.lirmm.fr/recherche/equipes/marel
2 http://ia.mines-douai.fr
3 http://ia.ur.mines-douai.fr

néralement architecturés autour d'un réseau complexe de capteurs et d'actionneurs couplés à une ou plusieurs unités de calcul leurs permettant d'effectuer des actions dans le monde physique. L'autonomie est une caractéristique essentielle permettant à un robot d'accomplir des missions seul (sans télé-opération). Pour cela, un robot autonome possède une application de contrôle lui permettant de prendre des décisions sans intervention humaine.

La robotique mobile autonome est aujourd'hui un axe de recherche prometteur et donc stratégique. En témoignent les investissements récents et conséquents dans ce domaine à travers le monde. Le premier ministre Japonais annonçait en juin 2014 un investissement d'environ $22 milliards[4] dans la robotique sur 5 ans afin de conforter le statut de pionnier du Japon dans ce domaine et d'offrir des jeux olympiques de 2020 sous le signe de la robotique. La Corée du Sud continue également son développement de la robotique avec un plan de financement de $316 millions sur 10 ans commencé en 2012. En juin 2011, les USA annonçaient officiellement leur initiative[5] de $500 millions pour l'automatisation des entreprises notamment par la robotique. L'Europe propose également un programme d'investissement[6] jusqu'en 2020 avec le consortium d'entreprise euRobotics[7] d'environ 2 milliards d'euros. La France a aussi annoncé en 2011 qu'elle considérait la robotique comme une technologie clé[8] et s'est dotée d'un fonds d'investissement de 80 millions d'euros[9] pour dynamiser ce secteur. Tout cela témoigne du dynamisme dans ce domaine.

Les travaux présentés dans la suite de ce mémoire visent les systèmes cyber-physiques mobiles comme les smartphones et plus particulièrement les robots autonomes. Néanmoins, certains de nos travaux peuvent tout à fait être appliqués dans le cadre du développement d'applications classiques.

1.2 PROBLÉMATIQUES

L'essor de la robotique et des équipements mobiles connectés s'accompagne de nouveaux usages et donc du besoin de développer des applications pour les supporter. Toutefois, de nombreux facteurs complexifient actuellement le développement d'applications pour ces systèmes.

La figure 1 schématise le cycle de développement actuellement utilisé pour implémenter une application pour un CPS mobile. Dans un premier temps, le développeur programme sur une machine de développement et compile (il

4 http://www.meti.go.jp/english/press/2015/0123_01.html
5 http://www.nsf.gov/nri
6 http://sparc-robotics.eu
7 http://www.eu-robotics.net
8 http://www.entreprises.gouv.fr/secteurs-professionnels/plan-robotique-pour-la-nouvelle-france-industrielle
9 http://www.robolutioncapital.com

peut s'agir d'une compilation croisée) son application. Ensuite, il déploie l'application compilée sur l'équipement cible. Après exécution, le développeur peut récupérer les traces (journaux) qu'il pourra analyser sur sa machine de développement afin d'identifier d'éventuels dysfonctionnements de l'application. Ce cycle de développement pourtant classique présente de nombreux inconvénients à différents niveaux. Parmi ceux-ci, notre travail se focalise notamment sur : le développement à distance, les contraintes en ressources (capacité de calcul et mémoire), l'intégration matérielle et logicielle ou encore l'impact de la mobilité sur le logiciel.

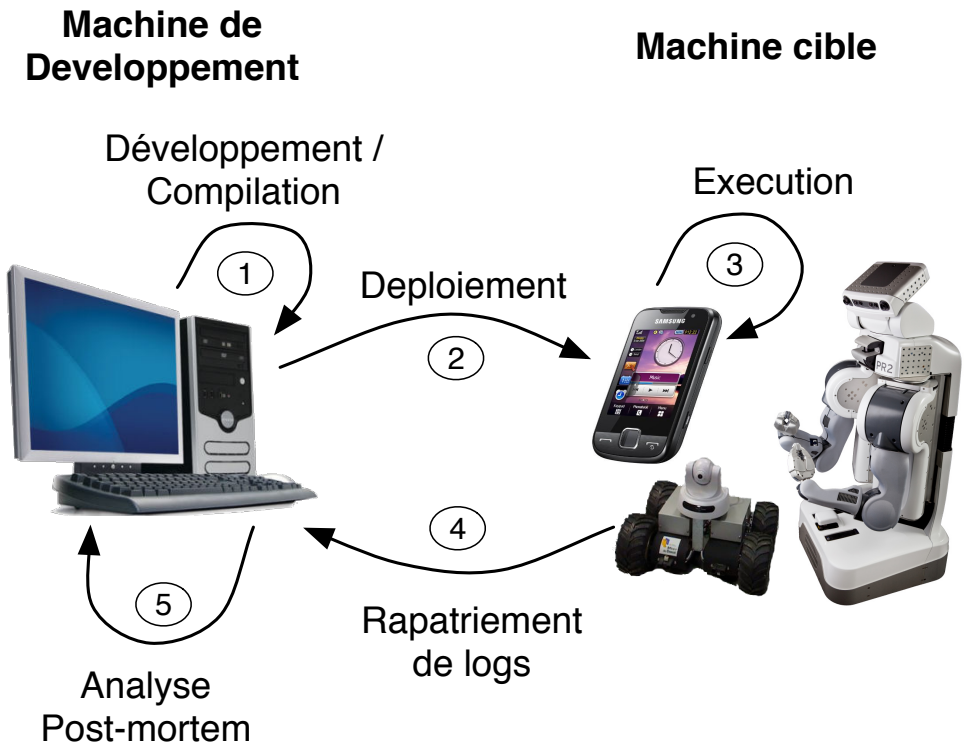


FIGURE 1 : Cycle de développement « classique » d'une application pour un CPS mobile.

DÉVELOPPEMENT DISTANT. Le développeur doit généralement programmer sur une machine dite de développement pour déployer ensuite son application sur le robot ou le smartphone cible. En effet, ces équipements ne sont généralement pas adaptés pour le développement. Ils peuvent être limités en terme de capacité de calcul ou de mémoire. Ils peuvent même être dépourvus de périphériques d'entrées/sorties comme un clavier ou une souris par exemple. Développer sur une machine tierce présente des difficultés, car elle ne dispose généralement pas des capteurs et actionneurs présents sur le robot mais indispensables au fonctionnement du programme. La mise au point du programme passe donc par un re-déploiement systématique sur le robot pour tester les nouvelles fonctionnalités implémentées. Le développeur doit également mettre en place des mécanismes spécifiques (comme la journalisation)

afin de récupérer des informations en cas d'erreur pour en déterminer les causes sur sa machine de développement. Ce cycle répétitif entre la machine de développement et l'équipement cible nuit à l'efficacité du développement.

INTÉGRATION MATÉRIELLE ET LOGICIELLE. Dans une certaine mesure, l'utilisation d'un simulateur de smartphone ou de robot sur la machine de développement permet de tester en local certaines fonctionnalités. La simulation est une étape importante car elle permet de tester une application robotique en toute sécurité pour le robot et son environnement. Bien évidemment, le code utilisé en simulation et sur le robot réel doit être le même à l'exception de l'acquisition des données par exemple. Nous excluons le cas où le code simulé est une implémentation différente et souvent simplifiée par rapport à l'architecture réelle d'un robot comme cela peut être le cas dans certaines validations sous MathLab par exemple. La simulation permet donc un premier niveau de test d'une application mais reste incomplète par définition. Le passage entre la simulation et l'exécution sur robot est une tâche complexe qui comprend l'intégration entre le logiciel et le matériel (capteurs et actionneurs). Cette intégration peut engendrer de nombreux problèmes (données incohérentes, perte de données, ...) difficiles à diagnostiquer.

CONTRAINTES EN RESSOURCES. Les ressources d'un robot sont déterminées par : la puissance de son processeur, sa quantité de mémoire, la capacité de sa batterie, la bande passante de ses interfaces réseau, *etc*. Développer sur une machine tierce peut donc induire en erreur quant aux performances futures de l'application une fois déployée car la machine de développement peut disposer de plus de ressources que le robot cible. Toutefois, aujourd'hui un smartphone ou un robot mobile peut disposer de ressources conséquentes. Par exemple, le robot PR2 est équipé de 2 processeurs i7 de 4 cœurs chacun, de 24Gio de RAM et de WiFi n. Ces caractéristiques sont à mettre en perspective avec les missions complexes que l'on souhaite réaliser avec un tel robot. En effet, les applications robotiques requièrent de traiter rapidement une grande quantité de données issues des différents capteurs (plusieurs caméras, lasers et centrales inertielles) afin de planifier les actions que le robot doit effectuer et d'envoyer les commandes correspondantes aux actionneurs. En définitive, les ressources sont toujours contraintes par rapport aux applications que l'on souhaite développer. Il est donc important d'utiliser les ressources disponibles le plus efficacement possible pour permettre le développement d'applications de robotique autonome plus ambitieuses.

MOBILITÉ ET CONTEXTE D'EXÉCUTION. Du fait de leur mobilité, l'environnement (contexte d'exécution) des applications est peu maîtrisé par le développeur. Une application peut par exemple déclencher des actions en fonction de sa localisation géographique, donnée par un GPS, ou encore en fonction des réseaux WiFi captés par exemple. Les dépendances au contexte complexi-

fient le développement et surtout les tests. Une solution consiste à simuler l'environnement par le biais de capteurs virtuels envoyant des données de test à l'application. Cela est notamment le cas en robotique lorsque le simulateur envoie des données laser en fonction de la position du robot dans un monde virtuel. Les dépendances au contexte peuvent être encore plus fortes et une application peut nécessiter l'utilisation d'autres équipements présents dans l'environnement. Par exemple, une application pour smartphone qui utilise des services ambiants dans une galerie commerciale ou encore un robot mobile qui utilise les services fournis par un autre robot mobile à portée de communication.

1.3 OBJECTIF

Face à ces problématiques liées au développement pour les CPS mobiles, l'objectif général qui guide mes travaux de recherche est le suivant :

Objectif général

Proposer des langages, modèles et outils permettant le développement interactif et haut niveau d'applications pour CPS mobiles et plus particulièrement pour le contrôle de robots mobiles et autonomes.

Cet objectif s'inscrit pleinement dans le contexte actuel d'essor de la robotique mobile et plus largement des systèmes cyber-physiques mobiles. Il vise à améliorer les pratiques actuelles en terme de développement d'applications pour ces systèmes qui présentent des contraintes spécifiques. L'idée est de simplifier la tâche du programmeur en se concentrant sur les langages et les outils qui sont à sa disposition. De façon plus concrète, dans le cadre du développement d'applications pour des CPS mobiles, notre travail est guidé par les interrogations suivantes :

- Quelles nouvelles constructions ou mécanismes langagiers permettraient aux développeurs de mieux appréhender ou s'abstraire (au moins partiellement) :
  - de la complexité des architectures logicielles ?
  - de la gestion de la mémoire ?
  - de la gestion de la capacité de calcul ?
- Comment fluidifier le traditionnel cycle de développement : programmation, déploiement, exécution/journalisation, analyse ?
- Comment mieux supporter l'intégration logicielle et matérielle ?
- Quels outils pour les inévitables phases de débogage ?

1.4 APPROCHE ADOPTÉE

Deux idées fortes ont guidé nos travaux et sont au cœur des contributions que nous proposons. La première est qu'il faut fournir au programmeur des langages de programmation permettant un haut niveau d'abstraction. La seconde est que les langages réflexifs sont de meilleurs candidats pour répondre aux besoins d'adaptation et d'extensibilité des applications. Ces deux idées sont tout à fait compatibles avec le développement des couches de contrôle hautes (décisionnelles) d'un robot mobile et autonome où les contraintes matérielles et temporelles sont plus faibles mais néanmoins présentes. De fait, nos travaux se concentrent sur ces couches hautes et non pas sur les couches basses (réactives) ni sur le développement temps réel.

1.4.1 Langages de programmation haut niveau

Un langage de programmation définit une syntaxe et une sémantique afin de permettre à un développeur d'écrire des programmes. Un programme est une séquence de caractères conforme à la syntaxe du langage dans lequel il est écrit. Il exprime un calcul que doit réaliser un ordinateur conformément à la sémantique du langage dans lequel il est écrit. Tous les langages ne fournissent pas les mêmes abstractions et constructions. Par exemple, certains langages masquent la gestion de la mémoire grâce à un ramasse-miettes (*garbage collector*) [McC60]. De nombreuses notions peuvent être abstraites par les langages, allégeant ainsi l'écriture des programmes. Dans la littérature, on parle de la capacité d'*absorption* des langages [Smi84]. Un langage est d'autant plus bas niveau qu'il impose au programmeur une vision proche du matériel (mémoires, registres). Toutefois, la plupart des langages modernes sont de haut niveau et s'abstraient de plus en plus des contraintes du matériel notamment via l'utilisation de machines virtuelles. Outre la portabilité des programmes, cela permet au langage d'offrir au développeur des abstractions de haut niveau qui seront ensuite supportées lors de l'exécution des programmes par la machine virtuelle. L'extension d'un langage passe donc par la coévolution du langage lui-même et de sa machine virtuelle.

Les langages de programmation sont à l'interface entre la machine et le développeur. Ils doivent donc être de haut niveau afin de simplifier l'écriture des programmes. Dans le cas contraire, c'est l'implémentation du langage qui est simplifiée au détriment de l'écriture de tous les programmes qui seront écrits avec ce langage. De plus, la montée en abstraction des langages permet également de mieux faire face à la taille et à la complexité toujours croissante des applications. En ajoutant de nouvelles abstractions ou mécanismes dans les langages pour gérer la distribution ou les ressources (e. g. mémoire, capacité de calcul) par exemple, on décharge le développeur de ces tâches (au moins en partie) comme cela a été le cas pour la mémoire avec l'introduction des ramasses miettes. Les langages dédiés à un domaine (*Domain-Specific*

Languages – DSL) [Bru97, MHS05] sont l'illustration de ce besoin d'offrir des abstractions adéquates aux développeurs jusqu'à les adapter spécifiquement au domaine métier de l'application développée. Ce besoin d'abstraction est également confirmé par les recherches utilisant des langages de haut niveau pour le développement d'applications bas niveau (*high-level low-level programming*) [FBC+09] i. e. au niveau système ou machine virtuelle par exemple.

En résumé, nos travaux se fondent sur l'idée que les langages de programmation de haut niveau sont une clé pour le développement (et l'exécution) des applications de demain. Cela n'empêche en rien le développement d'applications bas niveau (proches du matériel) comme cela peut être nécessaire dans le domaine des systèmes cyber-physiques.

1.4.2 Langages réflexifs

Dans le domaine des langages de programmation, la réflexion (ou réflexivité) est : « la capacité d'un programme à raisonner ou agir sur lui-même » [Smi82a, Mae87]. Un langage à objets réflexif offre donc ces possibilités à ses programmes parce qu'il possède une représentation de sa propre infrastructure grâce des méta-objets et des méta-opérations pour la modifier. On parle aussi de *protocole méta-objet* (MOP) [KdRB91].

Le processus de matérialisation d'une notion du langage (comme une classe ou une méthode) sous la forme d'un objet manipulable par ce même langage est appelé : *réification*. Par exemple, le contexte d'exécution, implicite dans la plupart des langages, est réifiée en Smalltalk sous la forme d'un objet [Riv96] permettant l'accès au receveur courant, à la méthode en cours d'exécution, aux arguments ou encore au pointeur de pile. Une propriété importante est qu'une donnée réifiée est *causalement connectée* à l'information qu'elle réifie. Dans l'exemple précédent, cela signifie que si le contexte d'exécution est modifié, l'objet qui le représente sera automatiquement modifié et inversement. En fonction de la notion réifiée, on distingue la *réflexion structurelle* de la *réflexion comportementale*. La réflexion structurelle fait référence à la représentation de l'état ou structure du langage réflexif comme les classes, les méthodes ou encore l'accès à la valeur d'une variable d'instance d'un objet. La réflexion comportementale correspond à la réification du fonctionnement du langage. Par exemple, si l'interprète est réifié, il devient possible de changer l'envoi de message. On distingue également deux utilisations de la réflexion. La première est l'*introspection*. Elle fait référence à un programme qui analyse son état en accédant en lecture uniquement aux méta-objets. La seconde est l'*intercession* qui traduit le fait qu'un programme se modifie lui-même.

Les langages réflexifs sont un atout considérable pour l'écriture de programmes dynamiques et adaptables. En effet, ces langages permettent de modifier presque n'importe quel aspect d'une application (modifier la pile ou le code à la volée, changer les objets, charger du code inconnu). Ils per-

mettent même d'adapter le langage lui-même lorsque cela est nécessaire. Un langage réflexif est même décrit dans la littérature comme une région dans l'espace de conception des langages plutôt qu'un unique langage du fait de sa capacité à se modifier [KdRB91].

En résumé, en utilisant les capacités des langages réflexifs, nos travaux proposent de nouvelles abstractions ou de nouveaux mécanismes afin de mieux répondre aux besoins des développeurs d'applications pour les systèmes cyber-physiques mobiles.

1.5 THÈMES DE RECHERCHE

L'objectif de recherche présenté dans la section 1.3 est général et large. Nos travaux se sont donc concentrés autour de quatre thèmes principaux représentés sur la figure 2 : langages et composants, réflexion et modularité, gestion des ressources et infrastructures et outils. Bien évidemment, les travaux menés dans chacun de ces thèmes ne sont pas cloisonnés et les résultats issus d'un thème sont souvent utilisés ou adaptés pour répondre à une problématique dans un autre thème. Comme mes travaux ne sont pas tous le fruit d'un travail solitaire, cette cartographie positionne également les différents co-encadrements que j'ai réalisés ces dernières années ainsi que les principales publications qui en ont découlé. Ces co-encadrements de doctorants et post-doctorants sont également la conséquence de ma participation au montage de plusieurs projets de recherche financés (3 terminés et 3 en cours) ainsi que de plusieurs collaborations scientifiques fortes notamment avec les équipes RMoD de l'INRIA Lille, ISOE des Mines d'Alès et MaREL du LIRMM (cf. Annexe A, page 146 pour plus de détails sur les projets scientifiques et les collaborations auxquelles je participe). La suite de cette section détaille plus précisément chacun de ces quatre thèmes.

1.5.1 Langages et composants

Un *composant logiciel* [McI68, BSW96, HC01, Szy02] est une unité logicielle possédant des interfaces explicites qui la rende indépendante de son contexte. Cette indépendance permet à un composant d'être assemblé avec d'autres composants sans aucune modification. Pour cela, il existe des modèles de composants spécifiant la structure des composants et de leurs interfaces ainsi que les différentes règles d'assemblage ou d'interaction.

Bien qu'il soit possible d'implémenter des composants logiciels avec n'importe quel langage de programmation (ce qui est forcément le cas s'il est Turing complet), j'ai montré dans ma thèse les avantages à utiliser un *langage à composants* [Fab07, FBDH10]. Un langage à composants propose des abstractions et des mécanismes spécifiques pour programmer des composants (*de-*

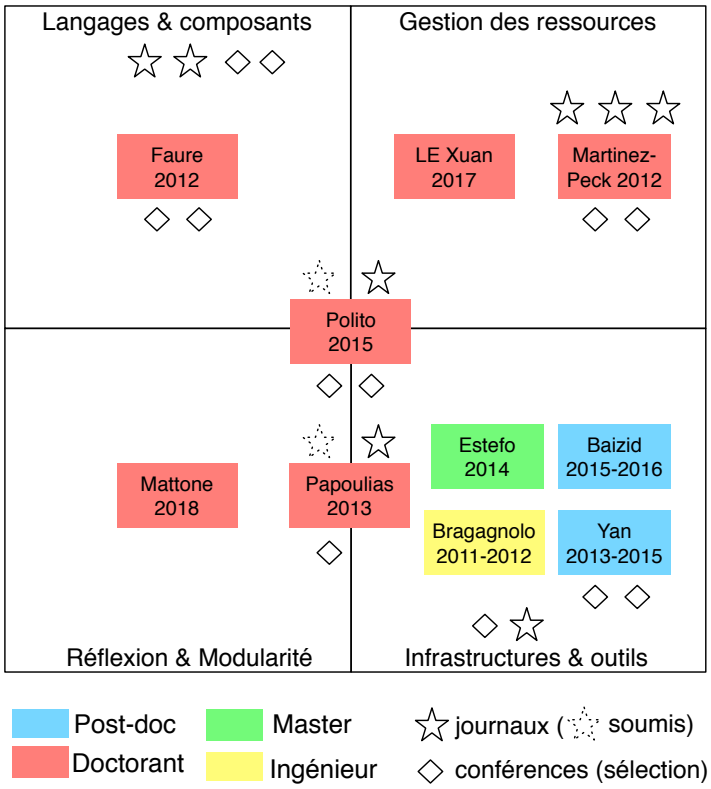


FIGURE 2 : Cartographie des encadrements et des principales publications selon 4 thèmes (cf. page 159 pour la liste complète).

sign for reuse) mais aussi pour les assembler (*design by reuse*) afin de produire des applications. Les langages à composants proposent donc de supporter le paradigme composant au niveau de l'implémentation ce qui facilite ensuite la maintenance, la compréhension et la rétro-ingénierie des programmes. Cela n'est pas le cas des approches basées sur la génération de code car elles se traduisent inévitablement par une complexification de toutes les activités se basant sur le code source généré comme le débogage. Les langages à composants offrent également la possibilité d'expliciter l'architecture des applications en terme d'assemblage de composants directement dans leur code source. Finalement, peu de travaux se sont vraiment intéressés à construire un langage à composants en se concentrant sur les problématiques au cœur de cette approche à savoir le *découplage* et l'*assemblage*.

Une application construite à base de composants distribués peut être impactée par la mobilité de l'équipement sur lequel elle s'exécute. En effet, certains composants présents dans l'environnement peuvent être nécessaires pour réaliser certaines fonctionnalités. La mobilité induit des changements d'environnement et donc un niveau de fonctionnalité possiblement différent. Certains travaux permettent de prendre en compte l'environnement lors du développement d'applications. Par exemple, les travaux sur AmbientTalk [DVCM+05] proposent même un support langagier pour développer de

telles applications. Toutefois, AmbientTalk ne permet d'expliciter l'architecture de l'application sous la forme d'un assemblage de composants.

C'est dans ce contexte que nous avons proposé et expérimenté différentes approches pour supporter le paradigme composant à l'aide de langages à composants. Nos travaux ont visé plusieurs objectifs :

- offrir de nouvelles façon de modulariser le code pour une meilleure réutilisabilité,
- permettre l'expression de l'architecture d'un logiciel directement dans son code source sous la forme d'une entité de première classe manipulable et modifiable dynamiquement afin de plus facilement supporter l'adaptabilité,
- proposer des mécanismes d'assemblage de composants spécifiques pour gérer la mobilité des équipements,
- concevoir un langage à composants réflexif.

1.5.2 Réflexion et modularité

La réflexion (cf. section 1.4.2) est l'une des pierres angulaires sur lesquelles reposent nos travaux. Comme indiqué précédemment, nous utilisons les mécanismes réflexifs pour proposer de nouvelles abstractions ou mécanismes langagiers afin de faciliter le développement d'applications pour les CPS mobiles. Malgré ses vertus dans de nombreux domaines comme l'adaptation dynamique de programmes ou l'extension de langage, la réflexion n'est pourtant pas généralisée dans tous les langages de programmation. Cela s'explique notamment par les nombreux défis qui accompagnent la réflexion : la performance, la sécurité, la modularité et l'auto-amorçage.

PERFORMANCE. Il est difficile d'implémenter efficacement les mécanismes réflexifs [KdRB91], notamment la réflexion comportementale [MJD96]. À cause de ces problèmes de performance, la réflexion est souvent incomplète et limitée à certaines parties du langage. Il existe pourtant des travaux qui proposent des mécanismes de réflexion structurelle à granularité fine (intra-méthode) afin de proposer un support partiel et non-anticipé de la réflexion de comportement avec des performances acceptables [RDT08, Den08].

SÉCURITÉ. La flexibilité est au coeur des applications modernes. Les applications Web reposent de plus en plus sur les capacités réflexives des langages comme Javascript, Ruby, Python ou encore PHP. Les applications de bureau ont également ce besoin à travers le chargement d'extensions par exemple. La réflexion permet cela via le chargement dynamique de code et sa transformation à la volée. Toutefois, les applications modernes ont également besoin d'être sécurisées ce qui est difficile à garantir en présence de

réflexion [LWL05]. De travaux comme Caja [MSL+08a] ou Singularity [HL07] visent à concilier la réflexion et la sécurité.

MODULARITÉ. Dans les implémentations actuelles de langages réflexifs, les fonctionnalités de base et méta sont souvent mélangées au sein des objets. Différentes architectures ont été proposées dans la littérature pour mieux découpler les informations ou opérations appartenant à un niveau de base ou à un méta-niveau. Par exemple, 3-KRS [Mae87] propose un découpage structurel strict où seuls les méta-objets possèdent toutes les méta-informations à propos des objets de base. Les *miroirs* [BU04] sont des méta-objets particuliers qui offrent des méta-opérations pour accéder aux méta-informations des objets de base. Les miroirs permettent un meilleur contrôle des méta-opérations qui ne sont plus directement disponibles sur les objets de base. Toutefois, contrairement aux méta-objets dans 3-KRS, un miroir ne contient pas directement les méta-informations. D'autres travaux se sont aussi intéressés à la possibilité de changer dynamiquement la couche réflexive d'un langage [LV03].

AUTO-AMORÇAGE DE LANGAGES RÉFLEXIFS. L'implémentation d'un langage réflexif pose le problème de son auto-amorçage [KdRB91]. Il y a, d'une part, les problèmes liés à la méta-circularité i. e. la classe `Class` est instance d'elle-même. D'autre part, les problèmes de méta-stabilité i. e. un changement dans le langage provoque des appels de méta-opérations successifs qui rendent le système inutilisable. Différentes techniques ont été proposées pour répondre à ces problématiques [CKL96, VBG+10, Str14].

Les travaux présentés dans ce mémoire autour de ce thème sont axés sur la modularité et l'auto-amorçage. Ils proposent des solutions nouvelles pour répondre aux deux questions suivantes :

- Comment modulariser la couche réflexive d'un langage, jusqu'à la rendre optionnelle et injectable dynamiquement ?
- Quelles abstraction et mécanismes pour une implémentation haut niveau de l'auto-amorçage de langages réflexifs ?

1.5.3 Gestion des ressources

Les applications logicielles actuelles sont de moins en moins limitées par les ressources matérielles car il est de plus en plus facile et peu coûteux de déporter des calculs. De plus, les architectures logicielles modernes facilitent la mise à l'échelle horizontale (*horizontal scaling*) pour gérer la montée en charge par exemple.

Il en est tout autrement pour les applications s'exécutant sur un équipement mobile. Dans certains cas, le déport de calculs à distance peut être envisagé mais au détriment de la bande passante et avec un temps de transfert

des données pouvant être rédhibitoire. Comme déjà évoqué, une application de robotique mobile doit optimiser l'utilisation des ressources du robot afin de pouvoir effectuer tous les traitements nécessaires à l'accomplissement de la mission. Jusqu'à présent, nos travaux ont plus particulièrement ciblé la gestion de la mémoire et de la capacité de calcul.

GESTION DE LA MÉMOIRE. La plupart des langages à objets actuels possèdent un ramasse-miettes qui offre un mécanisme de libération automatique de la mémoire non-utilisée en détectant et libérant les objets qui ne sont plus accessibles. Toutefois, certains objets non-utilisés restent accessibles et ne sont donc pas désalloués par le ramasse-miettes conduisant à une surconsommation mémoire par rapport aux besoins réels de l'application [Kae86]. Il est également important de limiter la mémoire utilisée par une application au moment de son déploiement sur un terminal mobile. Pour cela, il existe des techniques d'élimination de code mort (*tailoring*) avant le déploiement [GDDC97]. Malheureusement, une grande partie des techniques existantes à ce jour ne sont pas adaptées en présence de réflexion et en l'absence de types statiques.

GESTION DE LA CAPACITÉ DE CALCUL. Une technique classique dans les langages haut niveau est d'utiliser une interface de fonctions étrangères (FFI) pour appeler des fonctions écrites dans d'autres langages (généralement de plus bas niveau) et spécifiquement optimisées. Un robot mobile étant généralement architecturé autour de plusieurs unités de calculs, il peut également être intéressant d'effectuer une partie des calculs sur une unité secondaire spécialisée comme un processeur graphique ou un circuit logique programmable (FPGA).

Les travaux présentés dans ce mémoire autour de ce thème sont axés sur :

- un mécanisme d'élimination du code et des données non-utilisés avant déploiement,
- un mécanisme de gestion de la mémoire non-utilisée pendant l'exécution,
- une interface de fonctions étrangères efficace et de haut niveau,
- le déport de calculs vers des architectures matérielles spécialisées (e. g. FPGA).

1.5.4 Infrastructures et outils

Afin d'améliorer les pratiques de développement actuellement utilisées (cf. section 1.2), nous travaillons sur les infrastructures et les outils pour le développeur. Nos travaux se sont concentrés sur les trois axes suivants.

DÉBOGAGE À DISTANCE. La figure 1 (cf page 3) a présenté le cycle de développement d'une application pour un CPS mobile et ses inconvénients. Ce long cycle de développement est à répéter un grand nombre de fois car le débogage est une étape incontournable lors du développement d'une application mobile. Donc, plutôt que de s'efforcer à éviter les erreurs, sachant qu'elles sont inévitables, nous avons proposé une solution permettant de faciliter le débogage à distance d'applications s'exécutant sur un équipement mobile.

TESTS D'APPLICATIONS ROBOTIQUES. L'intégration matérielle et logicielle est un problème récurrent auquel doit faire face le développeur. Nous avons donc proposé une démarche basée sur les tests logiciels afin de répondre à cette problématique. Il s'agit ici de guider le développeur avec un processus de développement outillé.

BANC D'ESSAI POUR APPLICATIONS MULTI-ROBOTS. La mise au point d'une application impliquant plusieurs robots mobiles qui collaborent pour réaliser une mission est un défi. Suivant le nombre de robots dans la flotte, les conditions expérimentales peuvent être longues à mettre en œuvre. De nombreux travaux privilégient donc la simulation plutôt que l'expérimentation sur des robots réels. Pour être le plus réaliste possible, la simulation doit respecter les contraintes matérielles auxquelles sera soumise l'application une fois déployée sur chacun des robots. Par exemple, la bande passante disponible pour la communication inter-robot n'est pas la même si toute la flotte est simulée sur une même machine, plusieurs machines connectées en gigabit Ethernet ou connectés en WiFi comme sur des robots réels. Une simulation peut donc s'avérer très éloignée de la réalité si elle ne prend pas en compte ces aspects.

Les travaux présentés dans ce mémoire autour de ce thème visent à offrir au développeur les possibilités suivantes :

- le développement interactif (*immediate feedback*) d'applications robotiques,
- le débogage à chaud et à distance d'applications robotiques,
- une démarche et les outils associés permettant d'assurer l'intégration matérielle et logicielle,
- une infrastructure pour simuler, comparer et évaluer des applications multi-robots.

1.6 ORGANISATION DU MANUSCRIT

Ce manuscrit est organisé selon les quatre thèmes de recherche présentés précédemment : langages et composants (cf. chapitre 2, page 15), réflexion et modularité (cf. chapitre 3, page 37), gestion des ressources (cf. chapitre 4, page 57) puis infrastructures et outils pour le développement d'applications robotiques (cf. chapitre 5, page 81). Dans chacun de ces chapitres, plusieurs axes de recherche sont présentés afin d'illustrer les travaux menés dans ce thème. Chaque chapitre se termine par la présentation des travaux en cours et des perspectives de travail à court et moyen terme ainsi qu'une synthèse des contributions. Le chapitre 6 (page 105) conclut ce mémoire par une synthèse des recherches effectuées avant de présenter nos projets de recherche pour les prochaines années.

2 | LANGAGES ET COMPOSANTS

> Let us change our traditional attitude to the construction of programs: Instead of imagining that our main task is to instruct a computer what to do, let us concentrate rather on explaining to human beings what we want a computer to do.
>
> *Literate Programming, 1984.*
> Donald Knuth

> Humans are allergic to change. They love to say, "We've always done it this way." I try to fight that. That's why I have a clock on my wall that runs counterclockwise.
>
> Grace Hopper

Ce chapitre présente les travaux que j'ai menés ou auxquels j'ai participé sur le support dans les langages de programmation des concepts de composant et d'architecture ainsi que des mécanismes d'assemblage. Dans ce cadre, j'ai initié une première collaboration entre l'équipe MaREL du LIRMM et les Mines de Douai pour travailler sur un langage à composants réflexif qui est la suite directe de ma thèse [Fab07]. Nous avons travaillé ensemble autour de la thèse de Petr Spacek [Spa13] sur le langage COMPO. J'ai également participé à la mise en place d'une seconde collaboration scientifique avec l'équipe ISOE des Mines d'Alès à travers le projet « Applications distribuées » (cf. Annexe A, page 148). Ce projet a permis le financement de la thèse de Matthieu Faure [Fau12] dont j'ai été co-encadrant, sur la thématique de l'assemblage de composants pour le développement d'applications mobiles.

2.1 LANGAGES À COMPOSANTS

2.1.1 Motivation

Le développement à base de composants logiciels (CBSD) [McI68] est une voie prometteuse en génie logiciel. Elle est même considérée comme un paradigme post-objet [Szy02]. Inspiré de l'électronique, l'objectif de ce mode de développement est d'assembler des composants logiciels (des « morceaux » de logiciel réutilisables) pour créer des applications. De nombreux travaux [LW05, CSVC10] se sont attachés à définir les concepts sous-jacents à l'approche composant comme : *composant*, *port*, *architecture*, *composite*, *connexion* ou encore *composition*. La définition de ces concepts ou mécanismes est notamment guidée par : (i) la volonté d'augmenter les possibilités de réutilisation de code, (ii) d'exprimer l'architecture des logiciels comme un assemblage de composants et enfin (iii) de mieux supporter l'adaptation logicielle.

MODULARITÉ POUR LA RÉUTILISATION. La modularité a toujours été un axe de recherche important en génie logiciel, car elle promet une meilleure réutilisation du code existant. Il existe donc aujourd'hui de nombreuses techniques permettant d'organiser du code source de façon modulaire. Chacune de ces techniques repose sur deux éléments indissociables : une abstraction langagière et un mécanisme de réutilisation associé. L'abstraction consiste à définir sous une forme abstraite et indépendante de tout contexte ce qui peut être réutilisé. Le mécanisme de réutilisation consiste à utiliser dans un contexte donné une abstraction, ce qui peut nécessiter des adaptations et/ou du paramétrage. Par exemple : fonction et appel de fonction, module et importation de module, classe et héritage, framework et paramétrage de framework (généralement par sous-classage), aspect et tissage, *etc*. À l'instar des *traits* [SDNB03], les composants offrent la possibilité de réutiliser du code de façon orthogonale à l'héritage grâce à des mécanismes d'assemblage. Malgré cet arsenal, modulariser un logiciel reste une tâche complexe, car il est difficile de capturer tous les besoins de réutilisation.

ASSEMBLAGE ET DESCRIPTION D'ARCHITECTURES. Devant la taille croissante des applications, il devient nécessaire de partitionner un logiciel en modules de taille suffisamment importante pour posséder des interfaces clairement spécifiées et peu sujettes aux modifications. Les interactions entre les différentes « briques » d'une application peuvent ensuite être définies indépendamment de leurs implantations et de leurs évolutions. L'approche à composants, et plus particulièrement sa capacité à mieux exprimer les architectures logicielles [SDK+95] en terme de composants interconnectés, propose une vision adaptée pour la programmation à grande échelle où une « brique » est un composant.

ADAPTATION LOGICIELLE. La description des architectures logicielles est le point de départ permettant d'appréhender une architecture dans sa globalité et ainsi proposer des mécanismes de haut niveau pour l'adaptation. Par exemple, un mécanisme de remplacement automatique d'un composant par un ensemble de composants interconnectés tout en préservant la qualité de l'architecture de départ [DHU+07]. Dans le contexte d'une application robotique, le modèle MADCAR [GBV06, Gro08a] propose d'utiliser une description abstraite de l'architecture où les composants sont spécifiés à l'aide de contraintes. Ensuite, en fonction des évolutions de l'environnement du robot mobile, la sélection de l'architecture et des composants à utiliser est exprimée sous la forme d'un problème de satisfaction de contraintes qui est résolu par un solveur. L'adaptation dynamique de l'application passe par un changement des contraintes, une nouvelle résolution et une migration de l'état de l'application.

Au niveau implémentation, l'idée de l'approche composant est de rationaliser le développement de logiciel pour passer d'un mode « artisanal » (logiciel développé sur mesure) à une « industrialisation » du développement reposant sur la réutilisation de composants et de rôles bien établis dans la chaîne de production du logiciel. Pour cela, la programmation par composants (PPC) distingue au moins deux rôles (cf. figure 3) : (i) le développement de composants réutilisables réalisé par un développeur de composants et (ii) le développement d'applications par réutilisation et assemblage de composants, réalisé par un architecte d'applications [Ous05].

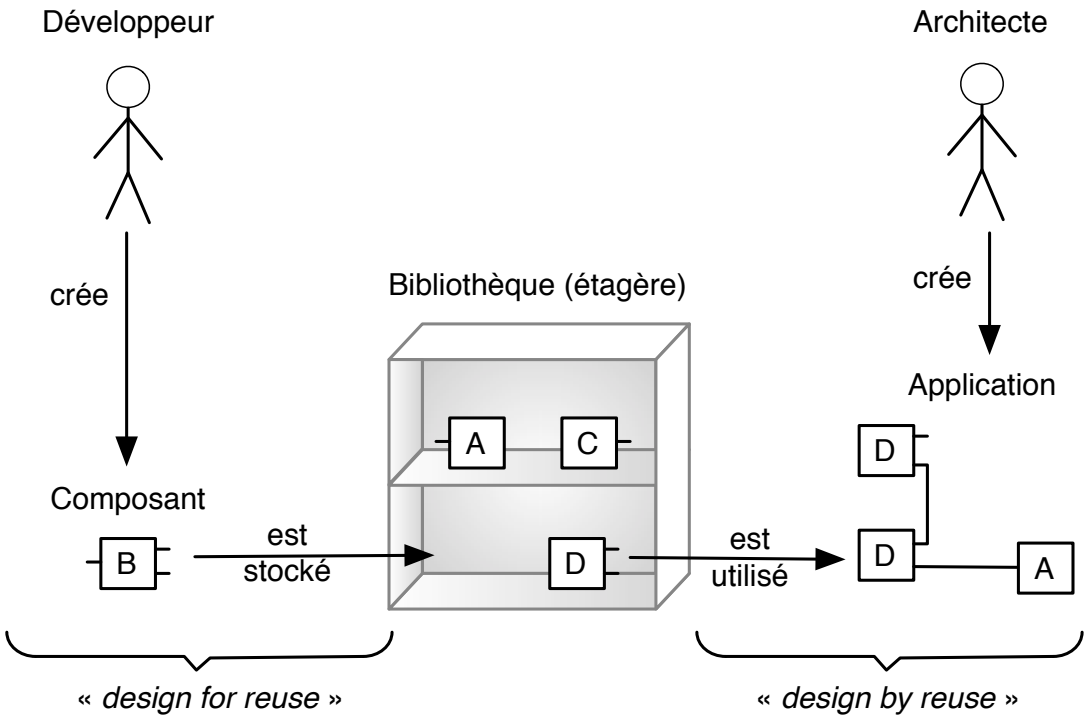


FIGURE 3 : Programmation par composants.

En pratique, l'approche composant est encore peu utilisé par rapport à l'approche objet par exemple. L'une des raisons est certainement due au fait que la plupart des travaux se sont concentrés sur la phase de conception comme c'est le cas des langages de description d'architectures (ADL) [SDK+95, MT00]) plutôt que la phase d'implémentation. De même, le modèle de composants Fractal [BCL+04] est indépendant de l'implémentation des composants grâce à un langage de description d'interfaces basé sur XML. Pourtant, le succès de l'approche objet tient certainement au fait qu'elle offre un continuum entre le mode de pensée lors de la conception et celui lors de l'implémentation grâce aux langages à objets. Par analogie, nous défendons l'idée que les *langages à composants* doivent fournir un tel continuum entre conception et implémentation. Ce n'est actuellement pas le cas avec la plupart des approches à composants lorsque l'on arrive à la phase d'implémentation comme nous allons le voir dans la section suivante.

2.1.2 État de l'art

CRITÈRES D'ÉVALUATION. Cette section présente une comparaison des approches pour faire de la programmation par composants basée sur six critères [FBDH10, FBDH12] :

R1 – DÉCOUPLAGE. Un composant ne doit pas pouvoir référencer directement un composant externe. Ce principe est fondamental pour ne pas entraver la réutilisation. Il est d'ailleurs déjà appliqué lors du développement par objets à travers l'utilisation de l'inversion de contrôle ou de l'injection de dépendances par exemple.

R2 – ADAPTABILITÉ. Connecter et déconnecter des composants doit être possible dynamiquement.

R3 – CONNEXIONS NON-ANTICIPÉES. Connecter des composants sémantiquement compatibles doit toujours être possible même si leurs développeurs respectifs n'avaient pas prévu cela.

R4 – ENCAPSULATION ET INTÉGRITÉ DES COMMUNICATIONS. L'encapsulation des composants doit être préservée pour garantir leur indépendance au contexte. Cela nécessite notamment d'assurer l'intégrité des communications entre composants i. e. éviter l'échange et le stockage de références.

R5 – UNIFORMITÉ. Les mêmes concepts liés aux composants doivent être utilisés dans tout le cycle de développement, depuis la conception jusqu'à l'implémentation et même à l'exécution.

R6 – RÉFLEXION. Le langage propose de la réflexion structurelle. Couplée à l'expression des architectures, la réflexion structurelle dans un langage

à composants ouvre la voie à des mécanismes d'adaptation dynamique avancés.

APPROCHES EXISTANTES. Il existe de nombreuses approches pour faire de la programmation par composants. Nous les avons classés en quatre familles représentatives :

LANGAGE À OBJETS ET CONVENTIONS. Utiliser un langage de programmation généraliste (généralement à objets) tout en respectant un ensemble de conventions ou idiomes comme c'est le cas en Javabeans [Ham97]. Malheureusement, les concepts liés à l'approche composant disparaissent au niveau de l'implémentation.

APPROCHES PAR EXTENSION DE *framework*. Les composants sont développés en étendant un *framework* comme c'est le cas avec JavaEE et le modèle des *Enterprise JavaBeans* (EJB) [MH99]. L'implémentation des composants est facilitée par rapport à la première famille puisque le programmeur est déchargé d'une grande quantité de code à écrire. Toutefois, l'implémentation est contrainte par le *framework* utilisé (superclasse imposée, points d'extensions prévus et non extensibles, ...).

APPROCHES GÉNÉRATIVES OU PAR TRANSFORMATION DE MODÈLES. Les approches appartenant à cette famille se veulent indépendantes de tout langage de programmation. Les interfaces et les composants sont décrits à l'aide d'un langage déclaratif standardisé tel que OMG IDL [Omg02b] par exemple. À partir de ces déclarations, des squelettes d'implémentation des composants peuvent ensuite être générés dans différents langages de programmation grâce à des outils de génération de code. Le programmeur doit ensuite compléter ces squelettes.

APPROCHES LANGAGES À COMPOSANTS. La dernière famille comprend des langages de programmation qui proposent des abstractions et/ou des mécanismes afin de programmer des composants mais aussi des applications par assemblage de composants. Par analogie avec les langages à objets, nous appelons ces approches des langages à composants. Seuls les langages à composants proposent de supporter le paradigme composant au niveau de l'implémentation. Alors que les approches génératives tentent de s'abstraire des langages d'implémentation, les langages à composants au contraire, offrent un continuum entre la conception et l'implémentation. Cela facilite ensuite le débogage mais aussi la maintenance et les évolutions du logiciel qui sont des étapes inévitables comme le signale la première loi de Lehman[1] [LB85].

1 « Un logiciel utilisé dans un environnement réel doit évoluer sinon il devient de moins en moins utile dans cet environnement ».

ÉVALUATION. Le tableau 1 synthétise notre évaluation de l'état de l'art. Le découplage est un critère particulièrement bien supporté par les différentes approches. Toutefois, Javabeans permet le découplage via l'utilisation du schéma de conception *Observateur* [GHJV95b] alors qu'ArchJava impose le découplage puisque les références directes ne sont pas possibles. L'adaptation n'est pas systématiquement supportée. Certaines approches comme les EJB, *Corba Component Model* (CCM) [OMG02a] ou ArchJava [ACN02] ne permettent pas de déconnecter des composants à l'exécution. Cette limitation est généralement due au fait que les connexions sont établies statiquement que ce soit par génération de code ou injection de dépendances à la compilation. Les trois critères restants ne sont pas supportés par la plupart des approches existantes. Pour pouvoir connecter des composants, leurs développeurs doivent avoir prévu cela et intégré le code nécessaire pour cela (R3). L'encapsulation et surtout l'intégrité des communications ne sont pas garanties (R4). Cela signifie qu'à l'exécution, les composants peuvent s'échanger des références et les stocker pour communiquer directement, même s'ils ne sont pas connectés. L'architecture logicielle décrite par l'assemblage des composants n'est alors plus le reflet de l'architecture réelle lors de l'exécution. Enfin, seul ComponentJ [SC00] propose une vision uniforme au développeur qui ne manipule que des composants (R5). ArchJava distingue les objets et les composants et chacun a ses propres mécanismes, ce qui complique la réutilisation. Concernant la réflexion, certaines approches proposent des capacités d'introspection mais pas de réflexion structurelle (R6) où il serait possible de modifier la structure des composants dynamiquement.

Cet état de l'art montre qu'aucune approche ne supporte pleinement le développement par assemblage de composants logiciels au regard des six critères établis. La suite de cette section présente différents travaux que nous avons menés pour y apporter une solution. Nous présentons d'abord CLIC qui est un langage à composants pragmatique car s'intégrant parfaitement avec un langage à objets. Toutefois, la symbiose de CLIC avec un langage à objets ne permet de satisfaire certains des critères ci-dessus, c'est pourquoi nous présentons ensuite une extension de SCL dont la conception initiale a été faite durant ma thèse et donc avant CLIC. Cette extension de SCL a été le point de départ pour la conception de COMPO, un langage à composants proposant de la réflexion structurelle. Enfin, nous présentons CLACS qui est une approche pour exprimer les contraintes architecturales sous la forme de composants SCL.

2.1.3 CLIC : un langage pour la symbiose objets/composants

Ce travail est né d'un double constat. Le premier est que les langages à objets sont actuellement les plus utilisés et qu'il serait difficile de changer complètement de paradigme en se privant des concepts et mécanismes de cette approche. Le deuxième est que les langages à composants existants sont gé-

Famille	Approche	*R1 (Découplage)*	*R2 (Adaptation)*	*R3 (Conn. non-anticipée)*	*R4 (Encapsulation)*	*R5 (Uniformité)*	*R6 (Réflexion)*
LOO+conventions	Javabeans	+	+	-	-	-	-
Frameworks	Fractal/Julia	+	+	-	-	-	-
	EJB 1&2	-	-	-	-	-	-
Génération de code	EJB 3	+	-	-	-	-	-
	CCM	+	-	-	-	-	-
	Sofa	+	+	-	-	-	-
Langages à composants	ArchJava	+	-	-	-	-	-
	ComponentJ	+	+	-	-	+	-

TABLE 1 : Évaluation des familles d'approches pour faire de la programmation par composants (+ indique que le critère est satisfait et − qu'il ne l'est pas).

néralement des extensions de langages qui cohabitent difficilement avec leur langage hôte (souvent un langage à objets) [FDH08].

Notre idée a donc été de proposer CLIC [BF09], un langage à composants spécifiquement conçu pour être parfaitement intégré avec son langage hôte en l'occurrence Pharo Smalltalk [NDP10]. Il s'agissait de proposer un modèle simple fortement couplé à celui de Smalltalk mais conforme aux concepts liés à l'approche à composants tels que définis par Szyperski [Szy98].

En CLIC, un composant est implémenté par un unique objet qui possède :

- *1 port fourni.* Un composant possède un et un seul port fourni qui est son seul point d'interaction permettant l'accès à toutes les opérations qu'il fournit. Il s'agit de la référence vers l'objet qui représente le composant.
- *0 ou plusieurs ports requis.* Un composant CLIC possède un ensemble de ports requis afin d'être connecté et d'interagir avec d'autres composants. On distingue deux types de ports requis : singleton et collection. Les premiers permettent d'interagir avec un seul autre composant. Les seconds permettent donc d'interagir avec plusieurs autres composants à travers un type de collection choisi par le développeur du composant dans la hiérarchie des collections Smalltalk (e. g. `Set`, `Array`, `Dictionary`).
- *0 ou plusieurs attributs.* Comme un objet Smalltalk, un composant CLIC possède des attributs privés afin de stocker des références vers ses

sous-composants. Toutefois, CLIC supporte le partage d'attributs par deux ou plusieurs composants comme c'est le cas dans le modèle Fractal [BCL+04] permettant à des composites de partager un même sous-composant. En CLIC, un attribut partagé est typiquement un composant global (e. g. référencé dans le dictionnaire système de Smalltalk) utilisé dans l'implémentation de différents composants. À travers ses attributs, un composant peut accéder aux opérations d'autres composants. Enfin, pour chaque attribut, un composant CLIC est aussi automatiquement doté d'un port requis d'observation de type collection permettant aux composants intéressés d'être notifiés des changements de valeurs de cet attribut (ancienne et nouvelle valeur par exemple).

- *1 architecture.* Tout composant CLIC possède une architecture qui explicite l'ensemble des connexions entre attributs. Le cas le plus simple correspond à un composant sans attributs et donc une architecture vide (i. e. aucune connexion). Une architecture permet l'initialisation des connexions d'un composant lors de sa création. Lorsqu'un sous-composant est remplacé, l'architecture permet d'identifier les connexions à détruire pour libérer l'ancien sous-composant et celles à établir pour insérer le nouveau sous-composant.

```
1  (CLComponent subclass: #Counter
2      localPrivateAttributeNames: #(count)
3      privateAttributesInitDict: { #count -> 0 }
4      sharedAttributeNames: #()
5      sharedAttributesInitDict: {}
6      localRequiredPortsDict: {}
7      category: #'ClicExamples—Clock')
8          architectureFrom: {};
9          operationsExportDictFrom: {};
10         operationsDefaultReceivers: #();
11         exportedRequiredPortsDictFrom: {}
```

Code source 1 : Définition d'une classe de composant `Counter` en CLIC.

Les descripteurs de composants CLIC sont représentés par des classes qui expriment explicitement toutes les dépendances de leurs instances. La définition des classes Smalltalk a donc été étendue pour supporter la déclaration de tous les concepts d'un composant CLIC. Le code source 1 illustre cela à travers la déclaration d'une classe de composant `Counter`. Il est ainsi possible de déclarer des attributs ainsi du code pour les initialiser (cf. lignes 2 et 3). Pour chaque déclaration d'attribut privé (e. g. `count`), CLIC fournit automatiquement : une opération de lecture (e. g. `count`) et une opération d'écriture (e. g. `count:`) de la valeur de cet attribut ainsi qu'un port requis de type collection pour signaler ses changements de valeur. Le composant `Counter` ne possède aucun port requis (cf. ligne 6). La déclaration d'un port requis né-

cessite d'indiquer son nom et son type, comme illustré dans l'extrait de code suivant :

```
localRequiredPortsDict: {
    #database -> CLSingletonPort.
    #workers -> Set }
```

La seconde partie de la déclaration de la classe de composant `Counter` (lignes 8-11) consiste en plusieurs messages Smalltalk en cascade :

- `architectureFrom:` permet d'indiquer un ensemble de connexions d'attributs à l'aide de la syntaxe suivante :

  ```
  architectureFrom:{
     (#someAttribute @ #requiredPort) => #otherAttribute.
     (#attrib1 @ #requiredPortCollection) => #(attrib2 attrib3) }
  ```

  Cette architecture décrit deux connexions. La première entre le port requis nommé `requiredPort` du composant référencé par l'attribut `someAttribute` et l'unique port fourni du composant référencé par `otherAttribute`. La deuxième connexion décrit une connexion multiple.

- `exportedPrivateAttributesDictFrom:` permet de lister les noms des attributs privés des sous-composants que le composite va exporter. Exporter un attribut revient à rendre ses accesseurs disponibles à travers le port fourni du composite mais également le port requis d'observation de cet attribut.

- `operationsExportDictFrom:` permet d'exporter des opérations offertes par les sous-composants en utilisant une syntaxe concise. L'opération exportée sera tout de même exécutée par le sous-composant.

- `operationsDefaultReceivers:` permet de déclarer une liste ordonnée de noms d'attributs. Lorsqu'un composant reçoit un message qu'il ne fournit pas et n'exporte pas, c'est la première opération fournie par l'un des sous-composants de cette liste qui est exécutée.

- `exportedRequiredPortsDictFrom:` permet de déclarer les ports requis des sous-composants à exporter au niveau du composite. Cela permet de ne pas fixer le fournisseur dans le composite et de laisser le choix à l'architecte.

Les règles d'héritage simple entre classes Smalltalk ont été étendues aux ports, attributs et architectures. Une sous-classe de composant peut redéfinir les directives d'initialisation et étendre les autres directives (ajout d'attributs, de ports requis, de connexions, ...)

En résumé, CLIC repose sur un modèle de composants simple et une implémentation de ceux-ci très proche des objets Smalltalk. Les ports fournis sont

des références sur des objets et les ports requis des variables d'instance. La conception de CLIC contraste avec les autres approches où un composant est généralement implémenté par plusieurs objets dont certains représentent explicitement les ports. La conséquence directe de ces choix est que CLIC n'introduit pas ou peu de perte en terme d'efficacité (mémoire et temps d'exécution) comparé à du Smalltalk classique.

CLIC remplit l'objectif principal d'intégration avec son langage hôte. Tout composant est utilisable comme un objet et inversement. Un objet Smalltalk peut être considéré comme un composite boîte noire avec un unique port fourni, sans ports requis et sans architecture. L'implémentation de CLIC repose sur la réflexion structurelle. Le comportement standard de Smalltalk reste inchangé ce qui permet d'utiliser la machine virtuelle et le compilateur standards de Smalltalk. En effet, l'envoi de message standard est utilisé lors de l'appel d'opérations sur des composants.

ÉVALUATION. L'évaluation de CLIC avec les critères définis dans la section 2.1.2 montre que CLIC permet le découplage (R1) et l'adaptation (R2). À l'instar de ComponentJ, CLIC offre aussi une vision uniforme (R5) au programmeur qui ne manipule que des composants. CLIC est novateur quant à l'intégration des paradigmes objets et composants puisque la forme la plus simple de composant se révèle être un objet. Néanmoins, CLIC ne satisfait pas les critères de non-anticipation des connexions (R3) et d'intégrité des communications (R4). En effet, les composants CLIC peuvent s'échanger leurs références et communiquer ensuite en dehors de toute connexion ce qui rend le mécanisme de déconnexion caduque. Concernant la réflexion, CLIC n'est pas un langage à composants réflexif (R6) où les descripteurs de composants sont eux-mêmes un assemblage de composants. Du fait de sa forte intégration avec Smalltalk, CLIC propose tout de même certaines capacités d'introspection et il est par exemple possible d'interroger une classe de composants pour retrouver la liste de ses ports et de ses dépendances.

2.1.4 SCL : un langage à composants pur

SCL (i. e. *Simple Component Language*) est le fruit de mon travail de thèse [Fab07]. Cette section présente brièvement SCL tel qu'il était à la fin de ma thèse pour ensuite détailler deux extensions de SCL [FBDH12] qui ont permis de satisfaire les critères de non-anticipation des connexions (R3) et d'intégrité des communications (R4).

PRÉSENTATION SUCCINCTE DE SCL. SCL est un langage spécifiquement construit pour faire de la programmation par assemblage de composants. L'idée première était de construire un langage de programmation en intégrant uniquement les abstractions et mécanismes nécessaires pour développer et assembler des composants. Contrairement à CLIC et à la plupart des

autres approches, SCL n'est pas une extension d'un langage à objets. En effet, à travers la conception de SCL, nous avons cherché à identifier les fondements de l'approche composant et ainsi abordé un ensemble de questions souvent oubliées dans les autres propositions comme le découplage par construction, l'uniformité ou le statut des composants de base (collections, entiers, etc). Un composant SCL est donc une instance d'un descripteur de composants équipé de ports. Comme en CLIC[2], un port est unidirectionnel (fourni ou requis), interne ou externe et simple ou multiple. Les ports sont le support pour l'invocation de services (receveur syntaxique) et l'établissement des connexions. Une connexion est un lien directif entre deux ports : le port source et le port destination. Ces deux ports peuvent être requis et/ou fournis ce qui permet à l'architecture d'établir des connexions qui contrôlent finement la résolution des invocations de services. Bien évidemment, des règles spécifiques s'appliquent lorsqu'une invocation de services transite via un port requis ou fourni. Enfin, des connexions plus complexes (n-aire notamment) peuvent être mise en place grâce à des connecteurs qui sont en fait des composants standards.

LE PORT INTERNE *self*. Dans les langages à objets, `self` (ou `this`) est une pseudo-variable contenant la référence sur l'objet courant. Cette référence peut être échangée (passée en paramètre) avec d'autres objets pour établir des communications futures. L'approche par composants vise à circonscrire les communications dans le cadre de connexions bien définies pour faciliter la déconnexion et donc le remplacement de composants. En SCL, nous avons proposé de représenter l'auto-référence par un port fourni interne nommé `Self`. Cela permet au développeur d'avoir un modèle uniforme mais aussi de contrôler les invocations de services à travers ce port. La figure 4 illustre une possible utilisation du port `Self` dans un composite. Dans le code du service `foo` de ce composite, l'invocation du service `doit` à travers le port `Self` est déléguée au sous-composant `OtherComponent` à cause des connexions établies. L'introduction de ce port `Self` permet d'éviter la fuite des références sur les composants. En effet, SCL impose que seuls les ports externes puissent être passés en tant que paramètre de service. De plus, le mécanisme d'instanciation d'un descripteur de composant SCL retourne toujours une référence sur le port fourni externe du composant nouvellement instancié nommé `Default`. Tout cela permet une encapsulation forte de l'implémentation d'un composant et par construction, le développeur ne peut pas transmettre à d'autres composants une référence sur un port interne ou un sous-composant.

PASSAGE D'ARGUMENTS PAR CONNEXION. Nous avons introduit en SCL un mécanisme de passage d'arguments spécifique permettant de garantir qu'aucune connexion ne peut être établie entre un port interne et un port reçu en paramètre ou en retour d'une invocation de service. Ce mécanisme complète

2 Comme indiqué précédemment, la conception de SCL est antérieure à celle de CLIC.

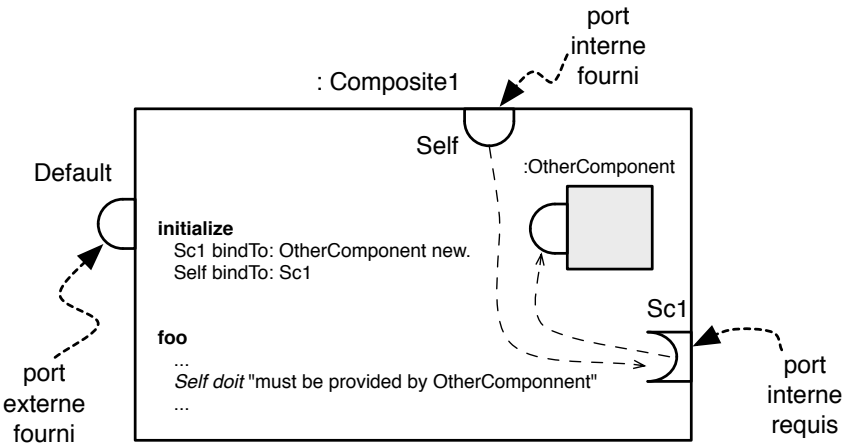


FIGURE 4 : Composite en SCL avec une connexion de délégation du port `Self`.

l'encapsulation forte des composants en SCL et assure l'intégrité des communications. La figure 5 schématise ce mécanisme de passage d'arguments par connexion. Initialement (étape 1), le composant `TCPServer` exécute son service `handle:`. Il a reçu le paramètre `aRequest` qui est un alias vers l'un de ses ports `Args` (il s'agit d'un port multiple externe) qui est connecté à un composant représentant une requête. Ce port externe `aRequest` est ensuite passé en paramètre de l'invocation service `process:`. Cette invocation de service est résolue pour identifier le composant connecté disposant de ce service (étape 2). Avant que ce composant n'exécute son service `process:`, une connexion est automatiquement établie avec l'un des ports `Args` de ce composant. Le paramètre local devient un alias vers ce port. Lorsque l'exécution de ce service est terminée, les ports `Args` sont automatiquement déconnectés. Ce mode de passage d'arguments est uniforme par rapport au modèle de SCL. De façon transparente, le programmeur ne manipule que des ports appartenant au composant qui développe. Des mécanismes supplémentaires permettent de gérer correctement les cas d'invocation de service récursifs ou ré-entrants [Fab07].

ÉVALUATION. Grâce à ces nouveaux mécanismes, SCL satisfait tous les critères d'évaluation [FBDH12] définis dans la section 2.1.2 hormis le fait qu'il n'est pas réflexif (R6). Toutefois, SCL est un langage à composants qui se différencie des langages à objets puisqu'il ne repose pas sur l'envoi de message mais sur l'invocation de services qui possède ses propres règles. De plus, SCL ne propose pas de mécanisme d'héritage entre les descripteurs de composants qui sont réutilisés par composition. Toutes ces caractéristiques font que le prototype de SCL est plus lent que son langage hôte (Smalltalk). Cela n'est pas surprenant puisque l'implémentation de ce prototype repose essentiellement sur la réflexion de Smalltalk et de la génération de code. Ce prototype a toutefois permis de valider le modèle et les algorithmes sous-jacents à la conception de SCL.

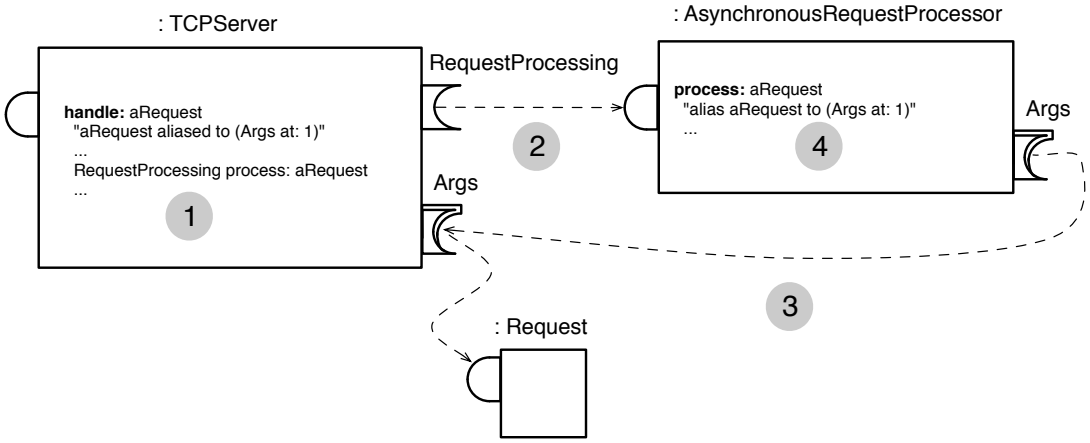


FIGURE 5 : Passage d'arguments par connexion en SCL.

2.1.5 COMPO : un langage à composants réflexif

Cette section décrit le langage COMPO qui est le fruit d'un partenariat entre Christophe Dony et Chouki Tibermacine du LIRMM et moi-même des Mines Douai. Nous avons collaboré autour de la thèse de Petr Spacek [Spa13] dont j'ai fait partie du jury. Ensembles, nous avons conçu le langage COMPO qui est la suite directe de mon travail de thèse sur SCL. Le travail autour de COMPO s'est focalisé essentiellement sur deux points : la description d'architectures et la réflexion. En SCL, l'expression des architectures est « enfouie » dans le code des connecteurs ou des services `initialize` car la conception de SCL s'est concentrée sur les aspects fonctionnels des composants plutôt que sur la spécification opérationnelle d'architectures à composants. COMPO se voulant être un langage à composants permettant la description haut niveau d'architectures à composants, la réflexion devient nécessaire pour pouvoir raisonner sur des architectures et même les transformer automatiquement comme c'est le cas dans le cadre d'une démarche de transformation de modèles ou d'adaptation dynamique.

DÉFINITION ET EXTENSION D'ARCHITECTURES À COMPOSANTS. Les descripteurs de composants sont enrichis en COMPO pour exprimer les connexions de façon déclarative (comme en CLIC). Cela permet un découplage entre la spécification d'architecture et son implémentation effective à l'aide de connexion de ports et de connecteurs. De plus, COMPO formalise des règles d'héritage simple entre descripteurs de composants afin de faciliter la réutilisation de structure et de comportement [SDTF12]. Un sous-descripteur hérite donc tous les éléments (ports, services, architecture) de son super-descripteur. COMPO permet l'ajout de nouveaux ports et services ainsi que la redéfinition de ceux hérités. Les règles de redéfinition sont spécifiques pour chacun des éléments hérités : port requis, port fourni, *etc*. Par exemple, COMPO propose une redéfinition covariante des ports requis permettant ainsi d'ajouter des

services requis à travers un port requis hérité. Cela offre plus de souplesse lors de la modélisation des composants mais au prix qu'un composant instance d'un sous-descripteur ne soit pas substituable à un composant instance d'un super-descripteur. La partie architecture d'un descripteur est également héritée et peut être redéfinie dans un sous-descripteur. Le développeur peut par exemple, ajouter de nouveaux sous-composants internes, remplacer ceux hérités ou encore remplacer les connexions héritées.

RÉFLEXION. COMPO repose sur un méta-niveau réflexif similaire à celui d'ObjVlisp [BC86] mais adapté aux composants [SDTF13]. Tous les concepts (descripteur, port, service, connexion) sont réifiés sous la forme de composants. Les régressions infinies induites par ces réifications sont stoppées à l'aide d'éléments primitifs. Ce méta-niveau offre au développeur un langage encore plus uniforme et extensible. Par exemple, il est possible d'ajouter de nouveaux types de ports si nécessaire. Toutefois, cette couche réflexive permet surtout à COMPO d'exprimer des transformations d'architectures sous la forme de composants et donc exécutable. Des composants de transformation d'architectures peuvent ainsi être écrits, assemblés et réutilisées pour transformer des architectures métiers.

ÉVALUATION. Au regard des critères d'évaluation définis dans la section 2.1.2, COMPO satisfait tous les critères. En effet, étant construit au-dessus de SCL, il satisfait aussi les critères R1 à R5 et sa couche réflexive lui permet de remplir également le critère R6. Une remarque est que sa couche réflexive concerne uniquement la partie structurelle des composants. Enfin, comme pour SCL, le prototype de COMPO est actuellement difficilement utilisable sur des exemples importants à cause de son implémentation inefficace.

2.1.6 CLACS : contraintes architecturales de première classe

CLACS [TDSF10, TSDF11] (*Constraint Language for Architecting Components in* SCL) est né d'une collaboration avec Chouki Tibermacine dont la thèse portait sur la contractualisation de l'évolution architecturale d'assemblage de composants [Tib06]. L'idée sous-jacente à ce travail est d'élever les contraintes architecturales généralement exprimées en OCL (*Object Constraint Language*) [OMG12] au rang d'entités de première classe sous la forme de composants SCL. En tant que composant, une contrainte devient ainsi réutilisable si elle est exprimée indépendamment de l'architecture métier sur laquelle elle doit s'appliquer. De plus, des contraintes de plus haut niveau peuvent être construites par assemblage. Lorsqu'une contrainte est connectée à une architecture métier, elle peut être exécutée pour vérifier que cette architecture respecte bien la contrainte qu'elle exprime. Un changement architectural dans la partie métier peut déclencher automatiquement l'exécution

des services de vérification des composants contraintes qui peuvent signaler en retour les cas de violation.

2.2 COMPOSANTS ET MOBILITÉ

2.2.1 Motivation

Les travaux présentés dans la section précédente se focalisent sur les mécanismes et les abstractions offerts au programmeur pour développer et assembler des composants. De fait, ils ne prennent pas en compte la mobilité pourtant inhérente aux applications modernes surtout lorsqu'elles s'exécutent sur un CPS mobile comme un smartphone ou un robot mobile. Les applications mobiles peuvent avoir besoin d'interagir avec des composants logiciels situés dans leur environnement local ou distant. De nombreux termes sont utilisés pour caractériser cela : informatique ubiquitaire, diffuse ou encore pervasive. Ces termes sont proches et parfois utilisés indifféremment[3]. Néanmoins, la notion d'ubiquitaire fait référence selon nous à une fonctionnalité accessible n'importe quand et depuis n'importe où. C'est le cas notamment des applications s'exécutant sur smartphone et utilisant une connexion Internet pour communiquer avec un composant ou un service distant. L'informatique pervasive peut être interprété comme une fonctionnalité accessible par les équipements proches (physiquement) de l'utilisateur ce qui implique une adaptation de l'application à son environnement local. Dans la suite de cette section, le mot service sera employé pour respecter le vocabulaire couramment utilisé dans le domaine des systèmes pervasifs. Toutefois, il s'agit bien d'un composant logiciel si ce n'est qu'il est généralement distant. C'est dans ce contexte d'informatique pervasive que cette section présente les résultats du travail de Matthieu Faure [Fau12] dont j'ai co-encadré la thèse.

Ce travail est axé sur les mécanismes et abstractions à offrir aux utilisateurs d'applications pervasives. L'exploitation des environnements pervasifs est difficile. D'une part, à cause de l'hétérogénéité et de la versatilité des équipements. Et d'autre part, à cause du besoin de représenter l'environnement qui est inconnu, dynamique et comprend une diversité d'éléments comme des équipements, des composants ou services et des utilisateurs. L'objectif principal de la thèse de Matthieu Faure a été de définir ce que pourrait être un système pervasif centré sur l'utilisateur final c'est-à-dire un système lui permettant de contrôler et d'exploiter au mieux l'environnement pervasif qui l'entoure via un langage simple d'assemblage des composants automatiquement découverts.

3 *"This magazine will treat ubiquitous computing and pervasive computing as synonyms - they mean exactly the same thing and will be used interchangeably throughout the magazine"* par Satyanarayanan Mahadev, Chief Editor of IEEE Journal of Pervasive Computing, vol. 1, 2002.

Dans le contexte de scénarios robotiques, l'objectif à long terme est de rendre possible l'utilisation de robots présents dans l'environnement. Par exemple, les clients d'une enseigne de distribution pourraient via leur smartphone utiliser les services de robots mobiles qui patrouilleraient dans la galerie marchande. Les services rendus par de tels robots pourraient aller du simple guidage (amener l'utilisateur où il souhaite aller) jusqu'à porter ses achats tout en le suivant et à terme aller chercher et ramener certains produits. Cet exemple est typiquement un système pervasif comme envisagé dans le travail présenté dans cette section.

2.2.2 État de l'art

LES BESOINS DES SYSTÈMES PERVASIFS. L'objectif des travaux menés a été explicité à travers une liste de besoins que doit remplir un système pervasif centré sur l'utilisateur.

Pour cela, nous avons recensé 10 besoins fonctionnels regroupés en 5 familles :

R1—GESTION DU CONTEXTE

- R1.a (*Représentation du contexte*). Un système pervasif doit gérer une représentation de son contexte d'exécution (présence/absence de composants/services, d'équipements, de lieu ou encore d'autres utilisateurs). L'utilisateur doit pouvoir organiser cette représentation selon ses préférences (ajouter des catégories par exemple).

- R1.b (*Découverte du contexte*). Un système pervasif doit continuellement mettre à jour son contexte en fonction des apparitions et disparitions de services ou d'équipements.

R2—DÉFINITION DE SCÉNARIO

- R2.a (*Composition de services*). L'utilisateur doit pouvoir combiner les services dynamiquement découverts dans l'environnement pour produire des scénarios répondant à ses besoins.

- R2.b (*Paramétrage de scénario*). L'utilisateur doit pouvoir définir des scénarios génériques et paramétrables afin de les réutiliser dans différents contextes. Par exemple, un scénario peut imposer l'utilisation d'un service fourni par un équipement en particulier ou non.

R3—EXÉCUTION DE SCÉNARIO

- R3.a (*Contrôle de l'exécution*). Un utilisateur doit pouvoir démarrer, mettre en pause ou stopper l'exécution d'un scénario.

- R3.b (*Exécution résiliente*). L'exécution d'un scénario doit s'adapter aux actions de l'utilisateur ainsi qu'aux changements de contexte (comme la disparition d'un service par exemple).

R4–RÉUTILISATION DE SCÉNARIO

- R4.a (*Diffusion de scénario*). La description d'un nouveau scénario doit pouvoir être diffusée dans le système pervasif pour la rendre persistante et disponible même si l'équipement qui a servi à sa définition disparait.

- R4.b (*Composition hiérarchique*). Il s'agit de pouvoir construire des scénarios composites.

R5–PARTAGE DE SCÉNARIO

- R5.a (*Mode de partage*). Un utilisateur peut choisir de partager avec d'autres utilisateurs la description d'un scénario ou bien directement une exécution de scénario.

- R5.b (*Protection*). Un utilisateur doit pouvoir préciser des droits lors du partage d'un scénario i. e. les utilisateurs avec il le partage et selon quels accès.

COMPARAISON DES APPROCHES EXISTANTES. Notre état de l'art compare les travaux suivants : Anamika [CPJ+02], DigiHome [RHT+13], MASML [WLF07], PHS) [NS04], SAASHA [HHUV09], SODAPOP [HK05] et WComp [TLR+09]. Tous ont tous été implémentés dans des prototypes opérationnels. De plus, ils constituent une sélection représentative [Fau12] des différentes approches permettant de contrôler un système pervasif. La table 2 synthétise notre étude comparative de l'état de l'art. Aucune approche ne répond pleinement à tous les besoins fonctionnels recensés, même si la plupart permettent de découvrir les services, de les composer et de les exécuter. Toutefois, la possibilité de réutiliser une composition de service sous la forme d'un scénario ou encore de composer hiérarchiquement des scénarios n'est pas ou peu supporté. De même, l'exécution résiliente des scénarios n'est qu'en partie supportée alors qu'elle est un élément intrinsèque des systèmes pervasifs.

2.2.3 SDL : un langage d'assemblage pour composants pervasifs

Cette section présente une partie de la plateforme SaS (*Scenario As Services*) qui propose des solutions pour satisfaire les exigences des systèmes pervasifs centrés sur l'utilisateur présentées dans la section précédente. La suite de cette section commence par présenter une vision générale de la plateforme SaS et discute ensuite plus spécifiquement certaines de ses originalités

Systèmes	Besoins fonctionnels									
	R1 Context management		R2 Scenario definition		R3 Scenario execution		R4 Scenario reuse		R5 Scenario sharing	
	(a)	(b)	(a)	(b)	(a)	(b)	(a)	(b)	(a)	(b)
Anamika	–	+	+	–	∼	∼	+	–	–	–
DigiHome	–	+	∼	–	∼	∼	+	–	–	–
MASML	–	+	+	–	∼	∼	∼	–	–	–
PHS	–	+	–	–	–	–	–	–	–	–
SAASHA	+	+	∼	∼	∼	∼	+	–	∼	∼
SODAPOP	∼	+	∼	–	∼	–	–	–	–	–
WComp	–	+	+	–	∼	∼	∼	+	–	∼

TABLE 2 : Comparaison des systèmes pervasifs (+ indique que le besoin est satisfait, – qu'il ne l'est pas et ∼ qu'il l'est partiellement).

comme le langage SDL (*Scenario Description Language*) de définition de scénarios par composition de services [FFH+10, FFH+11d] ou le support pour l'exécution résiliente des scénarios ainsi construits [FFH+11b].

PRÉSENTATION GÉNÉRALE DE SAS. La figure 6 illustre les étapes nécessaires à la création d'un scénario. Ces étapes sont :

1. Une fois déployé sur un terminal mobile (un smartphone par exemple), SaS découvre les services environnants. Les services découverts sont automatiquement enregistrés dans un annuaire (sans doublons).

2. Les services présents dans l'annuaire sont présentés à l'utilisateur en fonction de leur nom, de leur provenance (appareil fournisseur), ...

3. L'utilisateur peut définir un scénario en sélectionnant des services présents dans l'annuaire et en les composant grâce à une interface graphique simplifiée. La composition des services repose sur notre langage de composition de services (SDL).

4. Un scénario ainsi créé est transcrit sous la forme d'un fichier descripteur ce qui permet de le sauvegarder et de le partager facilement.

5. Le fichier descripteur est analysé afin d'extraire les différents services utilisés et les connexions à mettre en place. Un graphe d'exécution (comparable à un réseau de Petri ou une machine à états finis) du scénario est ensuite construit. Ce graphe décompose en plusieurs étapes l'exécution du scénario et établit des conditions d'entrées et de sorties pour chaque étape telles que : la présence d'un service ou d'un équipement, l'exécution réussie d'une étape précédente, ...

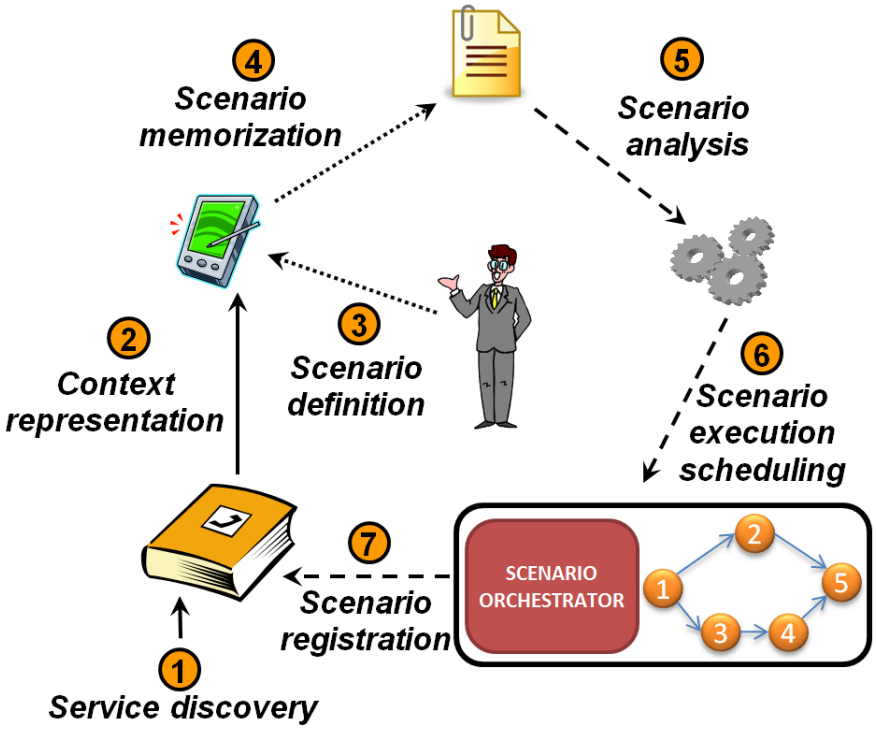


FIGURE 6 : Aperçu du cycle de création de scénarios avec SaS.

6. SaS crée dynamiquement un orchestrateur de scénario en charge de l'exécution du graphe précédemment construit et du cycle de vie du scénario (installé, en cours d'exécution, stoppé, en attente d'une condition, ...). Ce gestionnaire est intégré dans un composite conçu à la volée et déployé au sein de la plateforme d'exécution.

7. Enfin, le scénario est enregistré dans l'annuaire en tant que nouveau service. Ce nouveau scénario (service) devient donc accessible à tous les utilisateurs (suivant le mode de partage), utilisable et recomposable dans un nouveau scénario.

DÉFINITION DE SCÉNARIOS PAR COMPOSITION DE SERVICES. La définition d'un scénario est similaire à la définition de *workflows* en BEPL [JE07]. Toutefois, SDL se veut plus simple et se concentre sur les mécanismes de composition de services. Le code source 2 montre un exemple de définition de scénario (NightScenario) en SDL dans le contexte domotique. La ligne 5 montre l'invocation de l'opération `close` du service DoorService qui est fourni par l'équipement `MainDoor`. Il n'est pas toujours possible de désigner les équipements qui doivent répondre à une opération. Pour cela, SDL propose par exemple le mot clé `all` (cf. ligne 8) qui permet de sélectionner à l'exécution tous les équipements qui fournissent le service `ShutterService`. Ou encore à la ligne 18, le mot clé `any` permet d'indiquer que n'importe quel équipement fournissant le service désiré peut répondre. SDL propose également des règles de filtrage avancées permettant à l'utilisateur de sélectionner fine-

ment des équipements et des services. SDL propose également la possibilité d'exécuter en parallèle (cf. ligne 6) un ensemble d'actions. Cela signifie que l'ordre des actions n'est pas séquentiel et que dès qu'une action devient disponible, elle peut être exécutée. Enfin, SDL propose également la possibilité de déclencher des actions à une date ou une heure donnée (cf. ligne 21) ou lorsque certains événements surviennent.

```
1  scenario NightScenario ( description: close the main door, turn off shutters,
       adjust luminosity,
2                                        change thermostat value and set up the
       alarm clock;
3                          creator: Antoine)
4  [
5     MainDoor.DoorService.close();
6     [ parallel:
7        [
8           all.ShutterService.close();
9           while (LuminosityCaptor.LuminosityService.getValue() > ?) [
10             RoomLight.DimmableLightService.decrease();
11          ]
12       ]
13       if ( LivingRoomThermomether.ThermometerService.getTemperature() <= 18
14            and HouseThermostat.ThermostatService.getValue() < 5 ) [
15          HouseThermostat.ThermostatService.setValue(5);
16       ]
17       else [
18          any.AdjustTemperatureScenario.start(18);
19       ]
20    ]
21    at (8pm) [
22       PC.Player.play(WebRadio.RadioService.getChannel(InfoChannel));
23    ]
24 ]
```

Code source 2 : Exemple de définition d'un scénarion en SDL.

EXÉCUTION DE SCÉNARIOS. Un scénario doit pouvoir être exécuté même si tous les services qu'il requiert ne sont actuellement pas présents dans l'environnement. Cela offre la possibilité de construire des scénarios impliquant des services qui ne sont jamais disponibles simultanément. Comme déjà évoqué, un scénario est transformé sous la forme d'un graphe d'exécution dont les nœuds sont des blocs (ensemble d'instructions) correspondants à des étapes de l'exécution. Chaque étape respecte des propriétés d'atomicité, de cohérence, d'isolation et de durabilité (règles ACID). Toutes les constructions de SDL (`if`, `while`, `for`, blocs parallèles ou séquentiels, ...) sont projetées dans le graphe d'exécution sous la forme de nœuds et de transitions. Cette représentation permet de mieux maîtriser le cycle de vie d'un scénario. En

effet, il devient facile de détecter qu'un scénario est bloqué sur une condition (garde), car il est en attente de l'apparition d'un service. On peut même facilement ajouter des transitions par défaut spécifiant des dates limites d'attente qui déclencheront à leur tour l'exécution d'autres blocs comme la recherche d'un service similaire dans l'annuaire par exemple. L'ajout de ce type de règles par défaut procure un confort en qui concerne la gestion de la dynamicité de l'environnement où des équipements, des services ou des utilisateurs apparaissent et disparaissent en permanence.

ÉVALUATION. Cette section n'a pas présenté toutes les facettes de la plateforme qui font que SaS satisfait tous les besoins exprimés dans la section 2.2.2 [Fau12]. Le langage SDL offre des caractéristiques intéressantes comme l'exécution basée sur des graphes ce qui permet de définir facilement des stratégies en cas d'échec lors de l'exécution d'un service ou de blocage dû à l'absence d'un service. Une limitation certaine de SDL est l'utilisation d'interfaces classiques pour décrire les services et l'utilisation des règles standards de sous-typage pour vérifier la concordance de services. Comme évoqué dans nos travaux sur SCL, cela va à l'encontre du principe de non-anticipation des connexions. Le prototype de SaS a été testé sur des cas simples en terme de nombre d'équipements environnants, de services publiés et d'utilisateurs ce qui est représentatif dans le cas de maisons intelligentes par exemple mais pas dans le cas d'une utilisation plus mobile (maison, gare, bureau, lieux publics, ...).

2.3 TRAVAUX EN COURS ET PERSPECTIVES

Les travaux présentés dans ce chapitre se poursuivent actuellement selon deux axes.

LANGAGES À COMPOSANTS. Nous continuons à collaborer avec Christophe Dony et Chouki Tibermacine de l'équipe MaREL du LIRMM autour des langages à composants. Nous souhaitons par exemple faciliter l'expression des évolutions d'architectures à l'exécution. En effet, les applications sont de plus en plus amenées à changer d'architecture pour répondre à des contraintes de ressources ou à des changements de contexte d'exécution. Par ailleurs, nous souhaitons également étudier la possibilité d'implémenter efficacement un langage à composants réflexif. Ces perspectives de travail nous ont conduites à déposer conjointement un projet scientifique auprès de l'Agence nationale de la recherche (ANR) fin 2014. Bien que ce projet n'ait pas été retenu, nous envisageons d'autres pistes pour poursuivre ce travail.

COMPOSANTS ET MOBILITÉ. Répondre aux contraintes induites par les systèmes pervasifs est crucial lorsque l'on développe des applications de robo-

tique mobile. En effet, une mission robotique peut être décomposée comme une succession de missions plus élémentaires (se déplacer à un point donné, ouvrir une porte, rechercher un objet, ...). Chaque sous-mission nécessite d'exploiter au mieux les ressources du robot et donc de réorganiser l'architecture de contrôle du robot en fonction de la tâche à accomplir. Comme en SaS, l'utilisation de réseaux de Petri ou de machines à états hiérarchiques permet de modéliser efficacement les besoins en terme de changement d'architecture. Les réseaux de Petri ont déjà été utilisés dans de nombreux travaux pour exprimer une architecture de contrôle robotique. Par exemple dans COSARC [PADLR08], les composants sont exprimés sous la forme de réseaux de Petri pour mieux en valider l'assemblage. Le modèle MADCAR [Gro08b] quant à lui, s'intéresse plus à l'adaptation dynamique de l'architecture de contrôle d'une robot tout en conservant (i.e. migrant) l'état qui doit l'être. Tous ces travaux (dont SaS) ouvrent des perspectives intéressantes pour gérer les défaillances qui surviennent sur un robot en cours de mission. En effet, si la défaillance est logicielle ou contextuelle, on peut imaginer des solutions où l'architecture de contrôle du robot pourrait s'adapter afin de reprendre la mission éventuellement dans un mode dégradé.

2.4 SYNTHÈSE DES CONTRIBUTIONS

Le tableau ci-dessous synthétise mon activité dans ce thème à travers la co-signature d'articles scientifiques, le co-encadrement de thèses ou encore la participation à des projets scientifiques. L'annexe A contient plus de détails concernant ces éléments.

1	Revue internationale	[FBDH12]
5	Conférences internationales	[TDSF10, TSDF11, FFH+11d, FFH+11b, SDTF12]
5	Ateliers internationaux	[BF09, FBDH10, FFH+10, SDTF11, SDTF13]
3	Rapports d'activité de projet	[FFH+11a, FFH+11c, FHF+12]
1	Thèse co-encadrée soutenue	[Fau12]
1	Projet de recherche	Projet « Applications distribuées »
2	Collaborations scientifiques	EMA ISOE, LIRMM MaREL

PRIX ET RÉCOMPENSES. L'article [TSDF11] a reçu le prix « *ACM SIGSOFT Distinguished Paper Award* » lors de la conférence CBSE'11.

3 RÉFLEXION ET MODULARITÉ

> La perfection est atteinte, non pas lorsqu'il n'y a plus rien à ajouter, mais lorsqu'il n'y a plus rien à retirer.
>
> Antoine de Saint-Exupéry

> Les miroirs feraient bien de réfléchir avant de renvoyer les images.
>
> Jean Cocteau

La réflexion est un outil puissant sur lequel reposent la plupart de nos travaux. C'est pourquoi le support de la réflexion dans les langages à objets est également l'un de nos thèmes de recherche privilégiés. Ce chapitre présente selon deux axes, les travaux que j'ai menés ou auxquels j'ai participé sur la réflexion. Le premier axe correspond à l'un des verrous scientifiques identifié dans le projet « CAIRE » dont j'ai été coordinateur (cf. Annexe A, page 147). Ce verrou visait à modulariser la couche réflexive d'un langage afin de découpler le niveau de base et le niveau méta jusqu'à rendre ce dernier optionnel et injectable (*pluggable*) dynamiquement. Ce projet a permis le financement de la thèse de Nick Papoulias [Pap13] que j'ai co-encadrée et qui a été dirigée par Stéphane Ducasse de l'équipe RMoD de l'INRIA Lille, également partenaire du projet. Ces dernières années, un partenariat scientifique durable s'est installé entre les Mines de Douai et l'équipe RMoD de l'INRIA Lille. Le deuxième axe présenté dans ce chapitre en témoigne à travers le co-encadrement de la thèse de Guillermo Polito [Pol15] sur l'implémentation des langages à objets réflexifs qui reste un exercice complexe et souvent *ad hoc*. Nous avons donc proposé des mécanismes de haut niveau pour implémenter l'amorçage (*bootstrap*) d'un langage à objets réflexif et le faire évoluer sans avoir à modifier la machine virtuelle (VM) de ce langage.

3.1 MODULARISATION DU MÉTA-NIVEAU D'UN LANGAGE À OBJETS RÉFLEXIF

3.1.1 Motivation

Ce travail est né de la volonté de construire une infrastructure permettant le débogage à distance (cf. section 5.1) d'applications s'exécutant sur des CPS mobiles. La construction d'une telle infrastructure nécessite que l'application en cours d'exécution exporte tout ou partie d'une représentation d'elle-même au débogueur distant et lui offre la capacité d'agir sur elle-même. C'est exactement ce qu'offre un système réflexif qui a une représentation de lui-même qui lui permet de raisonner et d'agir sur lui-même également. Toutefois, le contexte contraint des CPS mobiles et la nécessité de faire de la réflexion à distance pose deux problèmes.

NÉCESSITÉ D'UNE COUCHE RÉFLEXIVE INJECTABLE. L'utilisation systématique de la réflexion introduit un surcoût lors de l'exécution d'une application ce qui n'est pas toujours souhaitable ou possible dans le contexte des CPS mobiles. La notion de réflexion injectable (*pluggable reflection*) [LV03, BU04] devient alors intéressante. Il s'agit de découpler le niveau de base d'un langage de son méta-niveau. Cela permet par exemple de fournir différentes implémentations des opérations réflexives pour des raisons d'efficacité ou de sécurité. Il est même possible de complètement décharger les opérations réflexives si l'application n'en a pas besoin.

PROBLÈME DE COHÉSION DES DONNÉES ET DES MÉTA-DONNÉES. Pattie Maes illustre ce problème ainsi : « *For example if we represent the concept of a book by means of an object, it may no longer be clear whether the slot with name "Author" represents the author of the book (i.e domain data) or the author of the object (i.e reflective data) [...].* » [Mae87]. Cet exemple illustre que dans les langages réflexifs, les données du niveau de base (données métier) et les méta-données sont mélangées dans les programmes. Or, dans l'optique du débogage à distance, les données métier font partie de l'application et doivent se situer sur le terminal mobile tandis que les méta-données ne sont nécessaires qu'en cas d'utilisation de la réflexion comme le débogage (e. g. noms des variables d'instances, ...) et peuvent donc être déchargées (au moins en partie) et situées à distance le cas échéant afin de minimiser l'impact sur les ressources du terminal mobile. L'impact des méta-informations ne doit pas être sous-estimé, surtout dans les langages à objets réflexifs qui les utilisent abondamment (e. g. noms des variables globales, noms des variables d'instance, noms des méthodes, ...). D'ailleurs, les compilateurs comme `gcc` et `javac` n'incorporent pas les méta-informations dans le binaire résultat par défaut (sauf si l'option de débogage est activée) afin de réduire l'empreinte mémoire des

programmes. Au contraire, les langages *cohésifs* comme Smalltalk disposent de beaucoup de méta-informations qu'il n'est pas toujours possible de décharger lors du déploiement.

En résumé, le travail présenté dans la suite de cette section est motivé par la nécessité de proposer architecture pour le cœur d'un langage à objets en vue de construire une solution de débogage à distance d'applications s'exécutant sur des équipements contraints. Cette architecture doit offrir un découplage aussi bien structurel (données vs méta-données) que fonctionnel (opérations de base et méta-opérations). Elle doit aussi permettre l'écriture de programmes qui supportent l'injection à chaud des méta-données et des méta-opérations notamment pour le débogage.

3.1.2 État de l'art

Cette section propose une synthèse de notre analyse de l'existant pour répondre à la problématique présentée. Tout d'abord, nous allons montrer que les langages à objets existants ont un problème de cohésion. Pour cela, nous avons choisi d'illustrer nos propos en Smalltalk, car il propose une couche réflexive parmi les plus complètes surtout du point de vue structurel.

SMALLTALK : UN LANGAGE COHÉSIF. Notre analyse de l'existant [Pap13] rappelle que de nombreuses classes de base en Smalltalk contiennent des méthodes qui normalement relèvent du méta-niveau. De même, de nombreuses méta-données sont stockées dans les objets de base. Les métaclasses jouent également un rôle ambigu car elles permettent de définir à la fois des fonctionnalités de base (e. g. `Point class»#fromUser`) et des fonctionnalités méta (e. g. `Class»#addInstVar:`). Le code source 3 montre un extrait de code qui illustre ce possible mélange des niveaux à travers l'utilisation indifférenciée de fonctionnalités de base comme `numberOfDoors` et de fonctionnalités réflexives comme `class`, `instVarNames` ou encore `superclass`.

```
1 myCar := Car new.
2 numberOfDoors := myCar numberOfDoors.
3 carClass := myCar class.
4 carClassIvNames := carClass instVarNames.
5 anotherCar := carClass new.
6 carSuperclass := carClass superclass.
```

Code source 3 : Code Smalltalk mêlant fonctionnalités de base et méta [BU04].

Nous avons identifié trois patrons d'architecture permettant de découpler les niveaux d'un langage à objets pour faire du débogage à distance : *remote proxy*, *remote façade* et miroirs. Les miroirs [BU04] proposent la solution la plus proche des caractéristiques souhaitées (cf. section 3.1.1) mais ils posent deux questions relatives aux notions d'état (les données de base et les méta-

données sont mélangées) et d'intercession. Nous allons donc présenter les miroirs et leurs limitations pour notre problème.

LES *miroirs*. La décomposition fonctionnelle [BU04] consiste à extraire l'API réflexive d'un langage de l'interface des objets de base. Les opérations réflexives sont regroupées et modularisées au sein de *miroirs* qui correspondent à des méta-objets sans état puisqu'ils continuent à utiliser les méta-données stockées dans les objets de base. La figure 7 illustre le principe des miroirs et le code source 4 montre l'impact de l'utilisation de miroirs pour implémenter un traitement équivalent au code source 3.

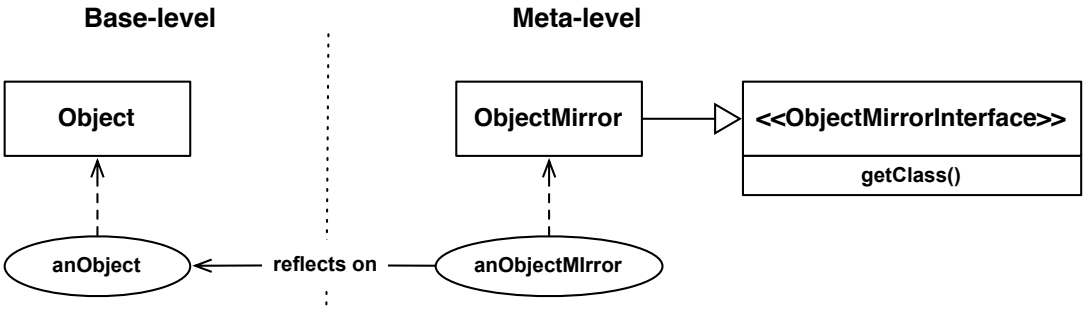


FIGURE 7 : Les miroirs sont des méta-objets explicites.

```
1 myCar := Car new.
2 numberOfDoors := myCar numberOfDoors.
3 carMirror := Mirror on: myCar.
4 carClassMirror := carMirror class.
5 carClassIvNames := carClassMirror instVarNames.
6 anotherCar := carClassMirror newInstance.
7 carClassSuperMirror := carClassMirror superclass.
```

Code source 4 : Code Smalltalk utilisant des miroirs [BU04].

Dans cet exemple, l'objet `myCar` ne répond plus qu'à des fonctionnalités de base. Toutes les fonctionnalités réflexives sont maintenant fournies par un objet miroir qui « reflète » un objet de base qui lui a été fourni au moment de sa création. L'utilisation des miroirs permet également de rendre explicite le saut entre le niveau de base et le méta-niveau (cf. ligne 3). Pour cela, la classe `Mirror` est une fabrique abstraite [GHJV95a] qui sert de point d'entrée. En fonction du contexte, cette fabrique peut retourner un objet miroir différent et ainsi proposer différentes implémentations des opérations réflexives. Le concepteur du langage peut ainsi proposer une hiérarchie extensible de miroirs réutilisables. À l'exécution, la fabrique abstraite permet d'instancier le miroir concret qui propose une implémentation (voire une API) adaptée au contexte, par exemple si le miroir est local ou distant par rapport à l'objet qu'il reflète. Un miroir distant par rapport à l'objet de base qu'il reflète peut offrir une API réflexive prenant en compte les transferts réseaux. Grâce

à toutes ces propriétés, la réflexion basée sur les miroirs est injectable car les opérations réflexives peuvent ne pas être chargées et injectées dynamiquement lorsqu'elles deviennent nécessaires.

MIROIRS ET DÉCOMPOSITION DE L'ÉTAT. Les miroirs permettent de séparer les opérations de base et les opérations réflexives mais pas les données de base (données métiers) et les méta-données (noms des variables d'instances, des méthodes, ...). Pourtant, dans le contexte distant, si les miroirs représentaient aussi l'état (éventuellement un cache) de l'objet qu'ils reflètent, il serait possible de minimiser les transferts réseau notamment dans le cas d'opérations d'introspection (accès aux méta-données de l'objet). Bien évidemment, le lien causal devrait être préservé. Les méta-objets de 3-KRS [Mae87] proposent ce découpage discipliné entre les données et les méta-données mais contrairement aux miroirs ils ne peuvent pas être déchargés.

MIROIRS ET INTERCESSION. Les miroirs constituent un modèle architectural de la couche réflexive d'un langage. Par contre, ce modèle ne spécifie pas comment représenter et implémenter les opérations d'intercession. L'extension des miroirs nommée Mirages [DVCM+05] propose une solution basée sur des méta-objets implicites. Toutefois, les opérations d'instrumentations supportées (envoi de message et *marshalling* d'objets) ne sont pas suffisantes pour implémenter une solution de débogage complète et haut-niveau (cf. section 5.1).

3.1.3 MetaTalk : un langage supportant l'injection dynamique de la réflexion

Cette section présente MetaTalk qui propose un modèle architectural de la couche réflexive d'un langage à objets en vue de construire une solution de débogage à distance. L'idée est que les objets de base (l'application) s'exécutent sur un équipement contraint tandis que les méta-objets à distance permettent à un débogueur d'introspecter et de modifier les objets de base. Pour cela, nous visons une solution mariant les miroirs pour pouvoir décharger et injecter la réflexion si nécessaire et les méta-objets de 3-KRS pour décomposer pleinement l'état des objets.

MÉTA-DONNÉES DE BAS NIVEAU ET DE HAUT NIVEAU. Par analogie avec la définition de la décomposition fonctionnelle [BU04], nous définissons la *décomposition structurelle* comme l'extraction des méta-données de l'état des objets de base. Toutefois, toutes les méta-données ne peuvent pas être retirées des objets de base car certaines d'entre elles sont utilisées par la machine virtuelle et donc indispensables à l'exécution. C'est ainsi que l'on distingue les méta-données de bas niveau i. e. celles qui ne peuvent être déchargées, et les méta-données de haut niveau. Notre analyse de Smalltalk nous a permis d'identifier : les noms de classe, les noms des variables d'instance, les

sélecteurs, le code source ou encore les informations relatives aux packages comme des méta-données de haut niveau non-utilisées par la machine virtuelle et donc déchargeables. Inversement, le format d'un objet, la référence vers sa classe ou le dictionnaire de méthodes d'une classe sont des méta-données de bas niveau directement accédées par la machine virtuelle et donc indispensables. Cet accès direct par la VM à des champs des objets de base limite les possibilités d'extension du langage. Par exemple en Pharo, il est impossible d'ajouter une variable d'instance dans la classe `Object` car cela décalerait les indices des variables d'instances déclarés dans les sous-classes (comme `superclass` défini dans `Behavior`) alors qu'ils sont supposés fixes par la VM. L'idée est donc de rendre les méta-données de bas niveau uniquement accessibles via des miroirs de bas niveau non déchargeables. La VM ne doit plus accéder directement aux méta-données de bas niveau stockées dans les objets de base mais doit utiliser les miroirs de bas niveau. Enfin, toutes les méta-données de haut niveau doivent être stockées dans des miroirs de haut niveau qui sont et déchargeables et injectables.

METATALK : LE MODÈLE DE RÉFÉRENCE. Notre modèle de noyau de langage à objets réflexif s'inspire de celui d'ObjVLisp [Coi87] et de Resilient [ABG+04] en ce qui concerne sa syntaxe déclarative. La figure 8 illustre notre proposition [NBD+11, Pap13]. Les niveaux de base et méta sont découplés. Tout objet du niveau de base (même les métaclasses) est associé à un miroir. Les miroirs – et uniquement les miroirs – fournissent les opérations réflexives, dont celles permettant l'accès aux méta-données. Les méta-données de haut niveau d'un objet de base sont stockées sous la forme de variable d'instances dans un unique miroir de haut niveau. Les méta-données de bas niveau d'un objet de base sont stockées dans l'entête de cet objet (*object header*) et uniquement accessibles via un miroir bas niveau sur cet objet. La VM doit impérativement passer par les miroirs bas niveau pour accéder aux méta-données d'un objet afin de garantir le découplage entre le modèle objet et son implémentation. Ce découplage assure que le modèle objet peut être étendu en toute circonstance puisque la VM ne fait aucune supposition sur le format interne des objets de base. Au niveau structure, la propriété de correspondance ontologique [BU04] impose que `ClassMirror` soit une sous-classe de `ObjetMirror` car `Class` hérite de `Object` dans le niveau de base.

En MetaTalk, le compilateur relève du méta-niveau puisque la compilation nécessite la présence des méta-informations. La définition dynamique de classe ne peut être faite que via des miroirs. Le niveau de base ne peut instancier que des classes de base pour pouvoir fonctionner en l'absence de miroirs. Le méta-niveau doit fournir une interface « standardisée » des miroirs et de leur acquisition pour autoriser différentes implémentations. La fabrique abstraite implantée dans la classe `Mirror` peut tout à fait retourner directement le miroir contenant les méta-données de l'objet reflété ou un autre miroir qui délèguerait certaines de ses opérations au miroir qui les contient. Enfin de

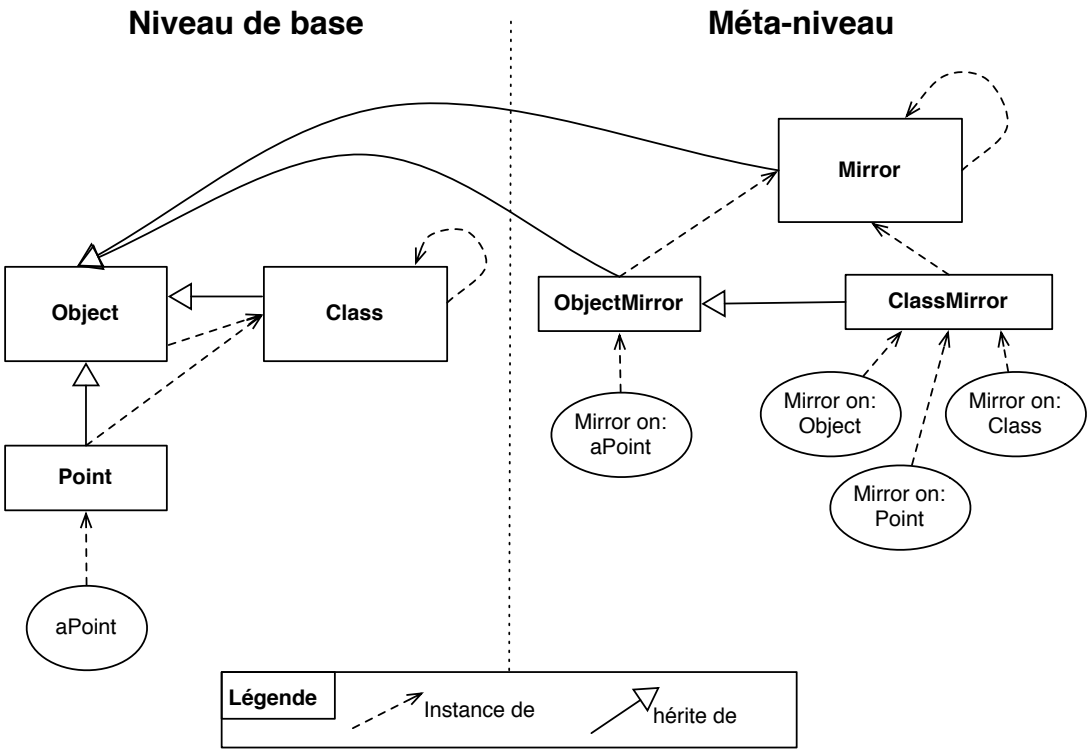


FIGURE 8 : Le modèle objet de MetaTalk.

compte, ce modèle permet de décharger tous les miroirs de haut niveau tout en assurant que le niveau de base est autosuffisant et peut être exécuté par la VM.

METATALK : EXEMPLE D'IMPLÉMENTATION. La figure 9 décrit comment sont stockées les méta-données dans cette implémentation. Contrairement à Pharo où les méta-données sont mélangées avec les données de base, MetaTalk propose une séparation stricte entre les niveaux. Les classes de base comme `Point` sont représentées par des objets possédant une entête contenant toutes les méta-données de bas niveau (`superclass`, ...). Aucune variable d'instance dans le niveau de base ne contient de méta-données. Un objet miroir sur une classe de base contient des références vers les méta-données de haut niveau et des références vers des miroirs de bas niveau permettant l'accès aux méta-informations stockées dans l'entête des objets de base. Cette implémentation permet au système d'être auto-décrit lorsque le méta-niveau est présent et assure également que le niveau de base est indépendant du niveau méta et peut fonctionner sans lui.

RÉIFICATION DU *réflectogramme*. Implémenter une solution de débogage à distance avancée implique l'utilisation de la réflexion comportementale pour instrumenter le code en cours d'exécution. La réflexion comportementale introduit un surcoût à l'exécution (acceptable dans les phases de débogage), mais surtout des problèmes spécifiques tels que la possibilité de méta-

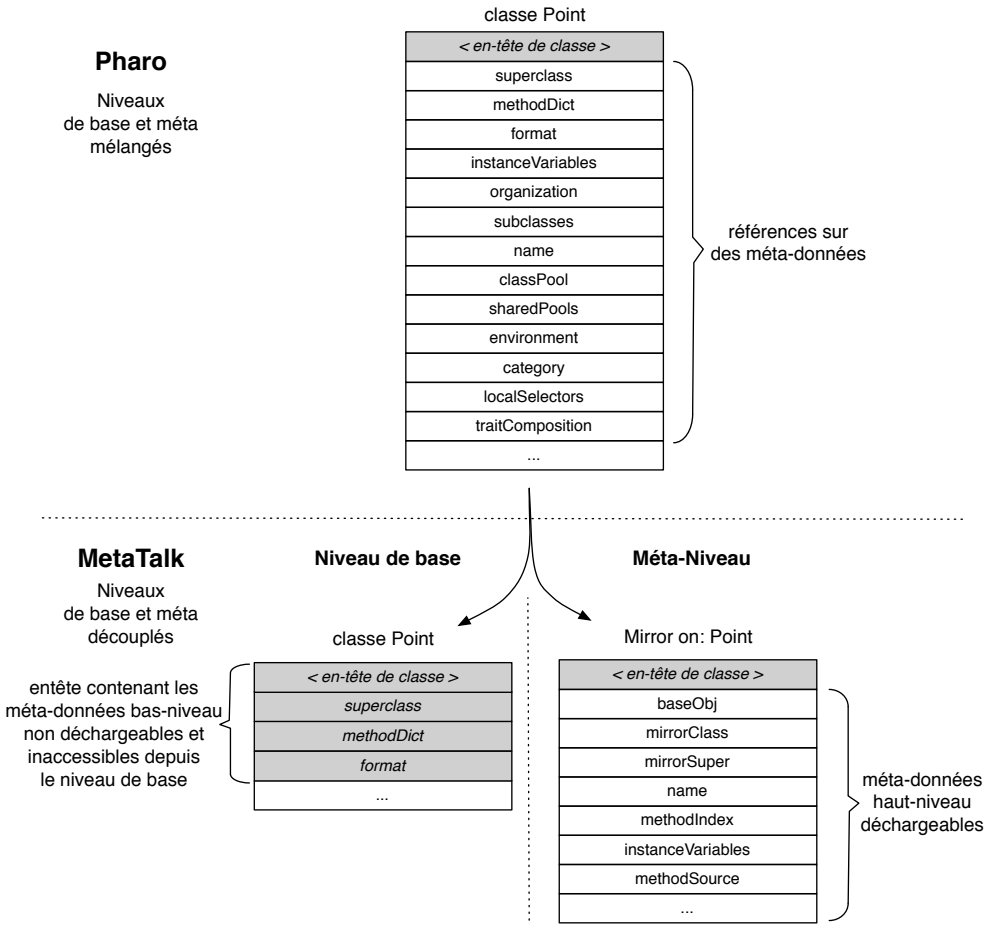


FIGURE 9 : Comparaison du stockage des méta-données en MetaTalk et en Pharo.

récursion [CKL96, DSD08] ou encore un mélange des niveaux qui rend difficile le déchargement du méta-niveau. Ces problèmes sont principalement liés au fait que la réflexion est implicite i. e. la réflexion est activée implicitement par la machine virtuelle lorsque certains événements surviennent lors de l'exécution. Les traitements réflexifs réalisés peuvent alors conduire à des inconsistances. Le code source 5 montre un exemple de réflexion implicite souvent similaire au modèle *Event-Condition-Action* [DGG95]. Dans cet exemple, lorsque l'événement « envoi de message » est rencontré par la VM, le bloc condition permet de filtrer les événements (e. g. `countingFlag` est vrai) pour éventuellement exécuter implicitement le bloc action qui contient des opérations réflexives (e. g. incrémentation de `messageCounter`). Ce code pose plusieurs problèmes. Tout d'abord, tous les envois de message subissent un ralentissement à l'exécution pour tester la condition. Ensuite, ce code pose le problème de méta-récursion. En effet, lorsque le message `increment` est envoyé à `messageCounter`, la VM teste à nouveau la condition et déclenche à nouveau l'exécution implicite du block action.

```
1 MsgSend
2     when: [ countingFlag = true ]
3     do: [ messageCounter increment ]
```

Code source 5 : Exemple de réflexion implicite.

Afin de proposer un contrôle explicite de la réflexion et une meilleure séparation des niveaux, nous avons proposé de réifier la notion de *réflectogramme* [TNCC03] qui illustre le flot de contrôle entre le niveau de base et le méta-niveau durant l'exécution. Le code source 6 illustre comment implémenter un traitement équivalent au code source 5 en utilisant le réflectogramme. Cet exemple n'introduit un surcoût que pour les envois de messages à un objet en particulier et il permet de prévenir la méta-récursion en désactivant la réflexion (`#enable`, `#disable`). Notre modèle de réflectogramme et son API [PDDF15] permettent de prendre en compte les cinq dimensions importantes pour contrôler la réflexion. Le contrôle temporel consiste à pouvoir activer et désactiver la réflexion (`#enable`, `#disable`). Le contrôle spatial permet de spécifier la portée de la réflexion (`#on:when:do:`, `#on:for:when:do:`). Le contrôle par placement permet d'exécuter des actions avant, après ou en remplacement (`#defaultAction`, `#returnValue:`) d'autres actions. Le contrôle de niveau d'exécution permettant d'éviter la méta-récursion (`#processMetaLevel`, `#objectMetaLevel`). Le contrôle d'identité qui permet de distinguer le méta-objet receveur d'une opération réflexive de l'objet de base reflété (`#at:`, `#at:put:` et `#perform:withArgs:`).

```
1 Reflectogram
2    on: MessageReceived
3    for: anObject
4    when: [:reflectogram | countingFlag = true ] do: [:reflectogram |
5       reflectogram disable.
6       messageCounter increment.
7       reflectogram returnValue: reflectogram defaultAction.
8       reflectogram enable
9    ]
```

Code source 6 : `Reflectogram` pour le contrôle explicite de la réflexion.

ÉVALUATION. Le modèle de Metatalk et la réification de la notion de réflectogramme ont permis de construire un modèle de noyau de langage à objets réflexifs basé sur les miroirs qui séparent strictement les opérations et les données du niveau de base et du méta-niveau. Cela permet de décharger complètement le méta-niveau tout en assurant que le niveau de base puisse être exécuté par la VM s'il n'utilise pas d'opération réflexive. La réification du reflectogramme permet de mieux contrôler la réflexion comportementale et surtout de la découpler du niveau de base. Enfin, les miroirs peuvent être dynamiquement injectés si besoin ce qui offre un support adéquat pour la

construction d'une solution de débogage à distance d'applications s'exécutant sur des CPS mobiles. Ce travail a été validé par plusieurs prototypes opérationnels.

3.2 AMORÇAGE DE LANGAGES À OBJETS RÉFLEXIFS

3.2.1 Motivation

L'amorçage (*Bootstrapping*) est plus connu dans le contexte des compilateurs. Un compilateur est dit amorcé lorsqu'il est capable de compiler son propre code source. Cette technique permet d'augmenter le niveau d'abstraction et de faciliter la compréhension (approche de plus haut niveau), l'optimisation et l'évolution des programmes. Amorcer un langage à objets réflexifs consiste à initialiser son environnement d'exécution (*language runtime*) avant de pouvoir exécuter un programme. L'amorçage d'un nouveau langage consiste aussi à initialiser son environnement d'exécution mais également à écrire son compilateur en même temps. L'environnement d'exécution d'un langage contient les classes du noyau (e. g. `Object`, `Class`, `Boolean`) ainsi que les objets initiaux (e. g. `true`, `false`).

L'amorçage d'un langage fixe généralement la sémantique de ce langage i. e. fixe son modèle objet et la structure des classes du noyau. Si ces éléments sont directement implémentés dans la machine virtuelle du langage (comme en Ruby), il devient très difficile de les modifier ou de les faire évoluer sans une connaissance avancée des détails internes de la VM. Or, il est important de pouvoir facilement modifier (sans changer la VM) ou de faire évoluer le cœur d'un langage afin d'introduire de nouveaux concepts ou de nouvelles fonctionnalités dont bénéficieront les applications écrites dans ce langage. Par exemple, Pharo [BDN+09] a introduit ces dernières années les notions de traits [SDNB03] ou de variables d'instances de première classe [VBLN11] ce qui a eu un impact conséquent sur son noyau. Dans le contexte des CPS mobiles, il peut au contraire être nécessaire de retirer des éléments du noyau du langage et des applications pour satisfaire les contraintes en ressources (capacité de calcul, consommation mémoire). Comme vu dans la section 3.1, on peut même envisager d'enlever complètement la réflexion et les métadonnées d'une application qui n'en a pas l'utilité. Or, les langages actuels (dont Pharo) n'ont pas été prévus pour permettre cela (ils sont cohésifs). Tout cela montre la nécessité de supporter les changements et notamment ceux qui n'ont pas été prévus [NDG+08].

La motivation derrière ce travail est donc de proposer un support pour l'amorçage de langage à objets réflexifs pour faciliter l'évolution des environnements d'exécution et des applications. Ces évolutions pouvant être l'ajout ou le retrait de fonctionnalités du noyau du langage.

3.2.2 État de l'art

CONCEPTS ET DÉFINITIONS. La figure 10 décrit les principaux éléments présents lorsqu'une application à objets est exécutée par une machine virtuelle. On distingue tout d'abord l'environnement d'exécution de la machine virtuelle (*VM runtime*) qui contient des fonctionnalités bas niveau comme le ramasse-miettes (GC), l'interpréteur de code-octet (*bytecode*) ou encore le compilateur à la volée (*JIT*). Cet environnement sert de support à l'exécution de l'environnement applicatif (*Application runtime*) qui englobe les éléments du noyau du langage (*language runtime*) et des éléments spécifiques à l'application en cours d'exécution.

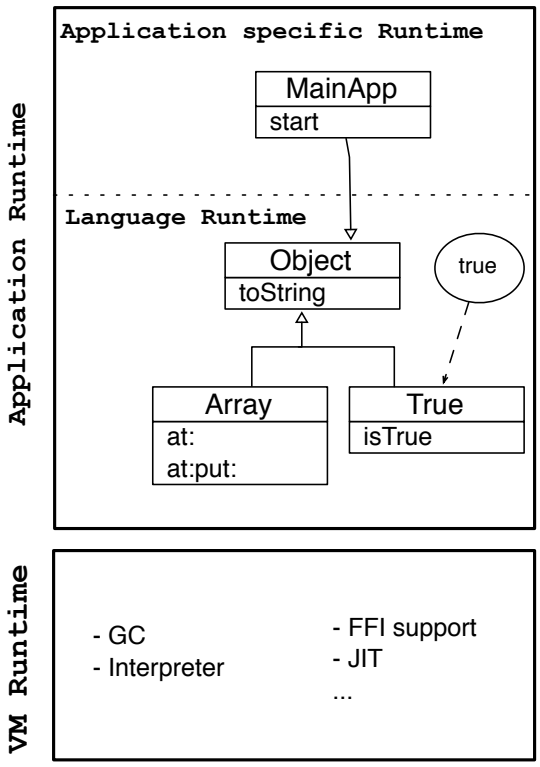


FIGURE 10 : Environnement d'exécution d'une application à objets s'exécutant au-dessus d'une machine virtuelle.

CRITÈRES DE COMPARAISON. L'amorçage consiste à générer des environnements d'exécution de langage. Cela nécessite de pouvoir manipuler un environnement d'exécution, c'est-à-dire pouvoir créer ou modifier des classes et des méthodes, instancier des objets ou encore démarrer et stopper des processus dans cet environnement. Toutes ces possibilités permettront également de faire évoluer un environnement d'exécution existant. Nous avons donc défini des critères pour évaluer les solutions permettant ces manipulations. Les critères importants selon nous sont les suivants :

MANIPULATION D'OBJETS. Il doit être possible d'accéder aux objets et à leurs champs ainsi que d'en instancier de nouveaux.

MANIPULATION DES ÉLÉMENTS DU LANGAGE. Il doit être possible de créer ou modifier les éléments qui font partie de l'environnement d'exécution du langage comme les classes et les méthodes.

MANIPULATION DE L'ENVIRONNEMENT. Il doit être possible de manipuler les éléments relatifs à l'exécution du programme i. e. créer, mettre pause ou redémarrer les processus ou encore inspecter les processus pour comprendre l'exécution d'un programme.

SÛRETÉ. La solution doit garantir que des modifications incorrectes ne peuvent pas être appliquées et que celles qui sont correctes peuvent être appliquées de façon sûre c'est-à-dire qu'elle ne laisse pas le système dans un état inconsistant.

NIVEAU D'ABSTRACTION UTILISATEUR. La solution doit offrir à son utilisateur la possibilité d'exprimer les manipulations qu'il souhaite réaliser à l'aide d'un langage de haut niveau afin de profiter des abstractions et de l'expressivité de ce langage.

NIVEAU D'ABSTRACTION DE L'API. L'API fournie par une solution de manipulation d'environnement d'exécution doit être de haut niveau. Le mieux pour cela est que les abstractions fournies pour la manipulation soient définies à partir des constructions constitutives de cet environnement.

SÉPARATION DES PRÉOCCUPATIONS. La solution doit clairement séparer les préoccupations de la VM de celles de l'application. Cela afin de ne pas polluer le développeur du langage avec les complexités inhérentes à la VM comme le ramasse-miettes ou le compilateur à la volée.

SOLUTIONS EXISTANTES. Nous avons classé les différentes solutions permettant de manipuler des environnements d'exécution en 3 catégories :

APPROCHES RÉFLEXIVES. Les modèles de réflexion et la méta-programmation permettent de manipuler l'environnement d'exécution applicatif. Ils permettent de modifier les classes et les objets, mais aussi la sémantique d'interprétation tout en assurant que ces modifications seront appliquées de manière sûre pour l'environnement dans lequel ils s'exécutent eux-mêmes. Dans cette famille, nous avons étudié trois approches. La première est le modèle général et flexible basé sur une tour d'interprètes [Smi82b]. Il permet de manipuler n'importe quel environnement d'exécution en modifiant son interprète situé dans le niveau supérieur (méta-niveau) mais il pose le problème de l'infinité de la hauteur de la tour. La deuxième approche est BLACK [Asa11], une

implémentation du modèle précédent en Scheme se limitant à un seul niveau d'interprétation extensible en certains points afin de modifier le comportement du niveau de base. Les niveaux supérieurs sont figés et directement exécutés par du code machine. La troisième approche est Reflectivity [DSD08], une architecture réflexive où des méta-objets représentent les éléments du langage et y sont causalement connectés. Reflectivity résout partiellement le risque de méta-récursion (quand le méta-niveau instrumente du code qu'il utilise lui-même) grâce à un niveau de profondeur méta associé à chaque contexte d'exécution. Les méta-objets ne peuvent agir que sur des objets associés à un niveau inférieur. Cette technique permet de manipuler l'environnement d'exécution avec un unique interpréteur tout en simulant plusieurs niveaux.

APPROCHES MÉTA-CIRCULAIRES. Cette approche correspond au principe « *high-level low-level programming* » [FBC+09] consistant à exprimer des éléments bas niveau en utilisant des langages de haut niveau. Par exemple, la VM Squeak [IKM+97] est une machine virtuelle (VM) méta-circulaire pour Smalltalk écrite en Slang (un sous-ensemble de Smalltalk projetable en C). La VM Cog[1] a étendu la VM Squeak pour y introduire un JIT notamment. Les VMs Jikes [AAB+00] et Maxine [WHVDV+13] sont des VMs méta-circulaires Java écrites en Java. L'avantage de ces approches est que le développeur de la VM peut utiliser des abstractions de haut niveau pour manipuler l'environnement d'exécution applicatif. Il peut ainsi accéder, contrôler et modifier les éléments internes à la VM mais aussi les objets et les classes dans l'environnement d'exécution applicatif. Toutefois, cela nécessite l'utilisation d'une API au niveau mémoire i. e. modifier un objet nécessite d'aller écrire la nouvelle valeur à la bonne adresse. De plus, dans la majorité des cas, cette interface disparait lors de la génération de la VM. Il n'est donc plus possible d'utiliser ces abstractions pour manipuler l'environnement d'exécution depuis ce dernier. On peut tout de même citer la VM méta-circulaire Klein [USA05] pour le langage Self. Contrairement aux approches précédentes, les abstractions de haut niveau fournies au développeur de la VM sont également accessibles depuis le langage même si l'API offerte consiste également à manipuler des adresses mémoire plutôt que des objets dans certains cas.

APPROCHES PAR VIRTUALISATION. Les systèmes de virtualisation tels que Xen [Chi07] permettent d'observer et contrôler les éléments internes d'une ou plusieurs VMs (chacune intégrant un système d'exploitation complet) grâce à un *hyperviseur*. Nous avons donc étudié les techniques liées à la virtualisation permettant de manipuler l'environnement d'exé-

1 Il s'agit de la VM actuellement utilisée par Pharo `http://www.mirandabanda.org/cogblog/`.

cution des applications virtualisées. Nous nous sommes également intéressés à la possibilité de faire coexister plusieurs environnements d'exécution d'applications isolément au-dessus de la même VM. Nous avons identifié et analysé de nombreux travaux relevant de cette famille d'approches : les *Class Loaders* en Java, le modèle de Changeboxes [DGL+07], Caja [MSL+08b] et Cajita, KaffeOS [BHL00], JVM TI [Ora13], MVM (Multi-User VM) [CDT03] ou encore Safe-Tcl [LOW97].

ÉVALUATION. La table 3 synthétise notre comparaison des techniques existantes [Pol15] pour manipuler l'environnement d'exécution d'une application (*application runtime*). Nous comparons donc les 3 familles d'approches décrites ci-dessus à travers les 7 critères précédemment définis. Les approches réflexives posent des problèmes de méta-stabilité comme la méta-récursion ce qui ne permet pas toujours d'appliquer des changements sans rendre en même temps le système inutilisable. Les approches méta-circulaires sont focalisées sur le développement de VMs et donc rarement utilisables depuis le langage pour manipuler son environnement d'exécution. De plus, l'API fournie par ces approches est généralement bas niveau (API orientée mémoire). Enfin, les approches par virtualisation permettent de faire coexister des environnements d'exécution d'applications complets au-dessus d'une même VM. Le contrôle de ces environnements est généralement limité à leur cycle de vie (démarrer, arrêter, mettre en pause). JVM TI propose tout même une interface de bas niveau pour manipuler les objets et certains éléments de l'environnement d'exécution.

	Approches réflexives	Approches méta-circulaires	Approches par virtualisation
Manip. d'objets	+	+	∼
Manip. des éléments du langage	+	+	∼
Manip. de l'environnement	∼	+	+
Sûreté	∼	+	+
Niv. d'abstraction utilisateur	+	+	∼
Niv. d'abstraction de l'API	+	∼	+
Séparation des préoccupations	+	–	+

TABLE 3 : Comparaison des familles d'approches existantes permettant de manipuler l'environnement d'exécution d'une application. (+ indique que le critère est satisfait, – qu'il ne l'est pas et ∼ qu'il l'est partiellement).

3.2.3 Espell : virtualisation d'environnements d'exécution de langages

PRÉSENTATION GÉNÉRALE DE L'ARCHITECTURE D'ESPELL. Espell [PDF+14] est une infrastructure de virtualisation d'environnements d'exécution de lan-

gages. Cette infrastructure permet de faire coexister plusieurs environnements d'exécution applicatifs au-dessus de la même VM. La figure 11 illustre cela avec deux environnements. Espell distingue l'environnement correspondant à l'hyperviseur et les environnements virtualisés. L'hyperviseur contient des objets de première classe spécifiques : des *Object spaces* [PDFB13] et un *hyperviseur*.

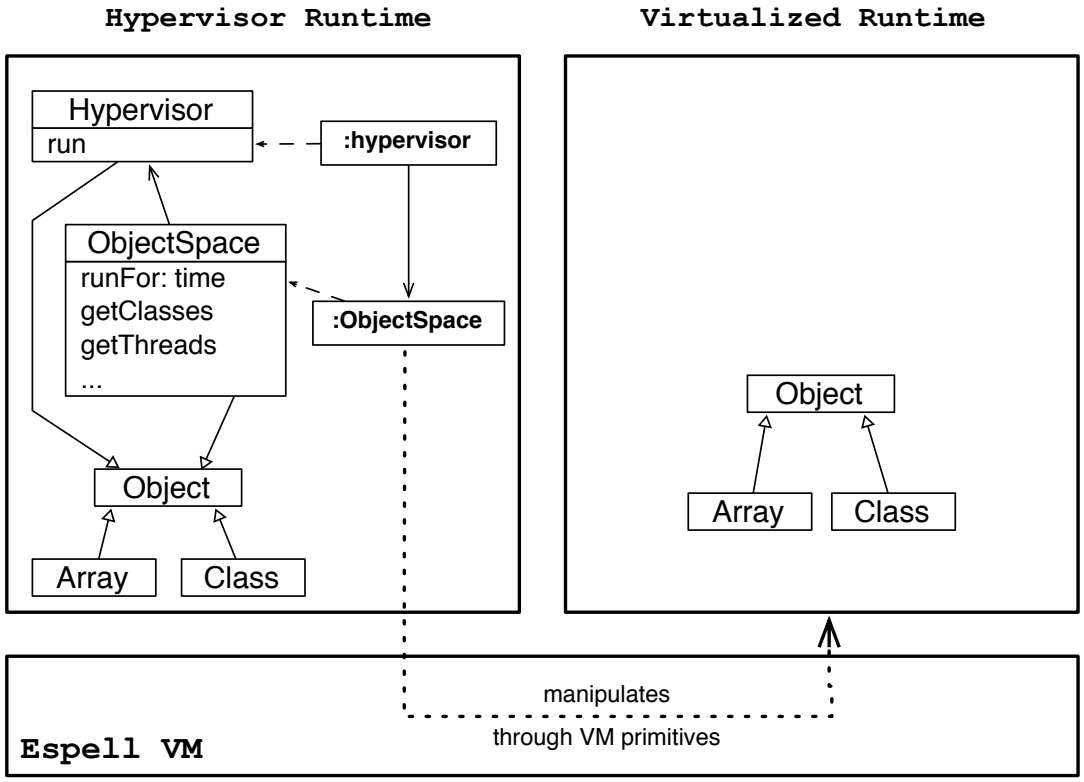


FIGURE 11 : Architecture d'Espell.

LE MODÈLE D'*object space*. Un *Object space* (OS) représente un environnement d'exécution complet et est l'unique point d'interaction avec ce dernier. Pour cela, un OS fournit une API de haut niveau basée sur des miroirs pour manipuler cet environnement ainsi que les objets qu'il contient. La classe `ObjectSpace` fournit de nombreuses méthodes pour manipuler l'environnement virtualisé ou les éléments qu'il contient (objets, classes, méthodes, processus, ...). Chacune de ces méthodes retourne une instance d'un objet miroir spécifique à l'élément manipulé i. e. une instance de `ClassMirror`, `ObjectMiror`, `ThreadMirror`, *etc.* Par exemple, `getClassNamed:` (défini sur `ObjetSpace`) retourne un objet miroir (instance de `ClassMirror`) pour manipuler une classe contenue dans l'OS virtualisé. Cette classe définit notamment les méthodes : `instantiate`, `intantiateWithSize:`, `compileMethod:`, `removeMethodNamed:`, *etc.*

INTERFACE VM/LANGAGE. En interne, les objets miroirs utilisent une interface spécifique avec la VM (VMSI) pour effectivement réaliser les opérations sur l'environnement virtualisé. Cette interface VMSI assure une séparation stricte entre la VM et l'environnement applicatif, ce qui permet de changer dynamiquement l'environnement en cours d'exécution sans redémarrer la VM. Pourtant, la VM a besoin de connaître l'identité (l'adresse) de certains objets spécifiques comme `nil`, `true` ou `false`. L'objet `nil` est en effet directement utilisé par le GC pour mettre à jour les références faibles (`WeakReference`). La VM doit également connaître certaines classes comme `Character` ou `BlockClosure` pour pouvoir en créer des instances. Enfin, la VM utilise aussi certains messages comme `doesNotUnderstand:`. Tous ces éléments dont la VM a besoin ne doivent pas être fixés en dur dans le code de la VM (contrairement à la VM de Ruby par exemple). La VM doit utiliser l'interface VMSI pour récupérer ces informations stockées dans l'environnement d'exécution. Depuis l'environnement d'exécution, le développeur doit aussi pouvoir utiliser l'interface VMSI pour modifier dynamiquement ces éléments.

LE MODÈLE D'HYPERVISEUR. L'hyperviseur est un objet de première classe qui permet de gérer à haut niveau la politique de supervision d'un *Object space*. Espell découpe l'exécution d'un *Object space* virtualisé en cycles. L'hyperviseur peut déclencher l'exécution d'un OS pour une fenêtre de temps donnée. Pendant cette fenêtre temporelle, l'OS virtualisé s'exécute sans contrôle et peut profiter pleinement de toutes les capacités de la VM. Lorsque cette fenêtre temporelle expire, l'OS virtualisé continue son exécution jusqu'à rencontrer un point de suspension sûr afin de rendre le contrôle à l'environnement d'hypervision. Dans notre implémentation, les points de suspension sûrs (i. e. point où l'interruption de l'exécution d'un OS ne va pas le laisser dans un état inconsistant) correspondent à l'activation d'un envoi de message ou aux codes-octet de saut (*bytecode backjumps*). Lorsque l'hyperviseur est réactivé, il peut introspecter l'état de l'environnement virtualisé, le modifier si nécessaire et relancer son exécution pour une nouvelle fenêtre temporelle.

SYNTHÈSE. Espell est une infrastructure pour la co-exécution d'environnements applicatifs. Cette infrastructure permet de contrôler un environnement virtualisé depuis un environnement d'hypervision. Espell fournit des interfaces de haut niveau pour manipuler un environnement virtualisé et le superviser grâce aux concepts de première classe : hyperviseur et *Object space*. Cette infrastructure ne fait aucune hypothèse sur l'application en cours d'exécution dans l'environnement virtualisé. L'implémentation d'Espell en Pharo permet de faire coexister deux images Pharo au-dessus de la même VM Pharo. Comme décrit ci-dessus, l'hyperviseur et l'*Object space* permettent de contrôler pleinement l'image virtualisée. Il est ainsi possible de démarrer l'exécution cette image, de la stopper, de modifier les objets, les classes, les méthodes

ou encore les processus qu'elle contient. Cette infrastructure est générale et ouvre de nombreuses perspectives [PDFB13].

3.2.4 Seed : amorçage de langages réflexifs avec Espell

Cette section décrit Seed [PDBF15] qui est une infrastructure pour l'amorçage de langages réflexifs tels que Pharo.

PRÉSENTATION GÉNÉRALE DU PROCESSUS D'AMORÇAGE. Comme illustré sur la figure 12, Seed repose sur un *interprète d'amorçage* (cf. `BootstrappingInterpreter`) qui exécute la *définition circulaire d'un langage* dans un environnement virtualisé par Espell. Le résultat final de ce processus d'amorçage est un environnement d'exécution d'application (cf. *Bootstrapped runtime* sur la figure 12) dont les entités du langage sont celles décrites dans la définition circulaire initiale. Le langage ainsi amorcé peut être directement exécuté par une VM standard et son initialisation est conforme à ce qui a été spécifié dans sa définition.

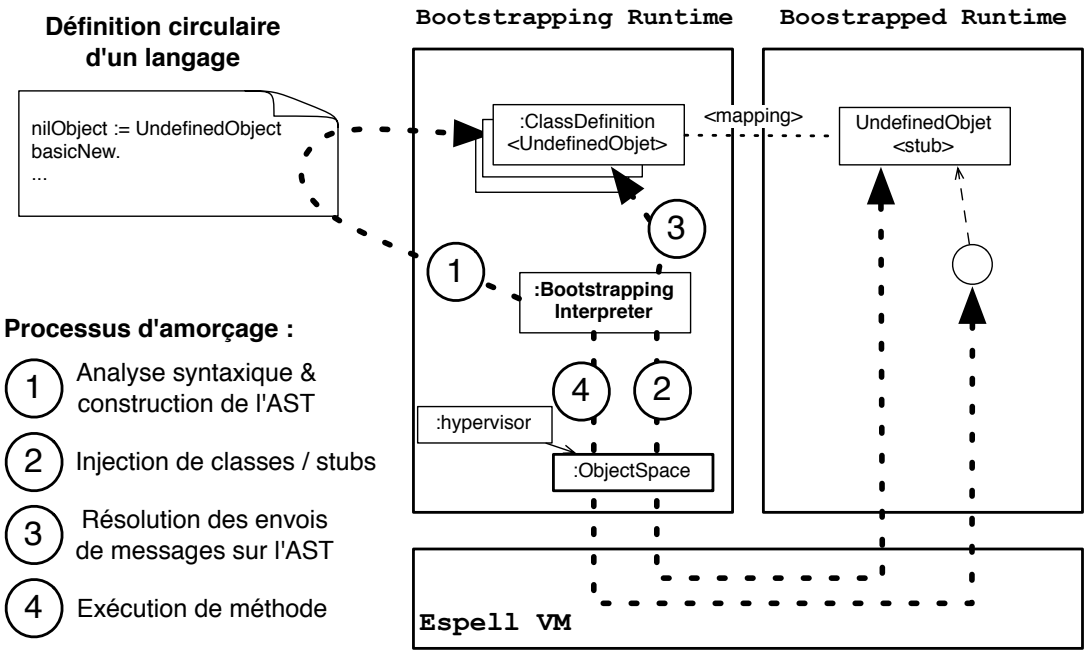


FIGURE 12 : Processus d'amorçage d'un langage avec Seed.

DÉFINITION CIRCULAIRE D'UN LANGAGE. La définition d'un langage consiste en la déclaration des classes (`Object`, `Class`, `Boolean`, ...), des méthodes (`new`, ...) ainsi que des initialisations nécessaires à l'exécution d'un programme (création des objets `nil`, `true`, `false`, ...). Cette définition est dite circulaire lorsqu'elle est écrite avec le langage qu'elle définit. Le code source 7 montre un extrait de la définition du langage Candle qui un langage à objets réflexif volontairement simplifié.

```
1  nilObject := UndefinedObject basicNew.
2  trueObject := True basicNew.
3  falseObject := False basicNew.
4
5  nil subclass: #ProtoObject
6      instanceVariableNames: ".
7
8  ProtoObject subclass: #Object
9      instanceVariableNames: ".
10
11 Object subclass: #UndefinedObject
12     instanceVariableNames: ".
13
14 ProtoObject >> isNil
15    ^ false
16
17 UndefinedObject >> isNil
18    ^ true
19
20 String initialize.
```

Code source 7 : Extrait de la définition circulaire de Candle.

INTERPRÈTE D'AMORÇAGE. L'interprète d'amorçage commence par lire la définition du langage écrit en lui-même pour créer un arbre de syntaxe abstraite de cette définition (étape 1 dans la figure 12). L'interprète d'amorçage va ensuite exécuter cette définition en parcourant cet arbre et en résolvant en même temps les problèmes inhérents à l'amorçage d'un langage. Typiquement, comment créer le premier objet sans classe et comment créer une classe sans classe (donc pas de super-classe possible). L'exécution directe de cette définition est donc impossible, car la classe `UndefinedObject` utilisé à la ligne 1 dans le code source 7) n'est définie qu'à la ligne 11. L'interprète d'amorçage résout ces inévitables paradoxes. Tout d'abord, lorsqu'il rencontre une classe encore inexistante dans l'environnement amorcé, il injecte une classe *stub* (étape 2 dans la figure 12) qui contient les éléments minimaux pour exécuter certaines opérations comme la primitive d'instanciation `basicNew`. Plus tard, lorsque la définition complète de la classe sera traitée, cette classe *stub* sera remplacée par la classe finale. Durant ce processus d'amorçage, l'environnement amorcé est d'abord incomplet (hiérarchie de classes non complète, ...). L'envoi de message est donc résolu sur l'arbre de syntaxe abstraite correspondant à la définition du langage pour identifier la méthode à exécuter (étape 3 dans la figure 12). Grâce à l'*Object Space*, l'interprète d'amorçage peut ensuite injecter cette méthode dans l'environnement amorcé si nécessaire et déclencher son exécution par l'objet normalement receveur de cet envoi de message (étape 4 dans la figure 12).

EXPÉRIMENTATIONS. Seed a permis d'amorcer trois langages [PDBF15], tous issus de la famille de Pharo mais présentant des modèles différents : Candle qui est une version minimale de Pharo supportant les classes et les métaclasses implicites, MetaTalk (cf. section 3.1) qui est un langage où le méta-niveau est optionnel et injectable dynamiquement grâce à une extension du modèle des miroirs et enfin le noyau de Pharo qui intègre une définition circulaire des Traits et la notion de variable d'instance de première classe. L'amorçage de ces trois langages aurait normalement demandé un effort conséquent de développement au niveau de la VM. Grâce à Seed et Espell, l'effort s'est concentré sur la définition circulaire de ces langages et quelques rares extensions de l'interpréteur d'amorçage tout cela en utilisant directement ces langages ou Pharo i. e. des langages de haut niveau et les outils de développement associés.

EXTENSIBILITÉ ET GÉNÉRALITÉ DE LA SOLUTION. L'interprète d'amorçage est écrit en Pharo. Cet élément est donc de haut niveau et facile à étendre. D'ailleurs, nous avons dû l'étendre dans certains cas pour prendre en compte des circularités spécifiques à certaines définitions de langage. L'architecture Espell est également extensible, car la majeure partie est également écrite en Pharo (*hyperviseur* et *Objet space*). La partie correspondante à l'interface entre la VM et le langage est réduite au maximum. De plus, cette interface révèle toutes les adhérences entre la VM et le langage donc il n'y pas lieu d'étendre cette interface sauf si la VM évolue. Enfin, après étude du code d'amorçage (généralement intégré dans leur VM) d'autres langages comme Ruby, Javascript ou encore Java, nous pensons que ce processus et son architecture leur sont transposables.

3.3 TRAVAUX EN COURS ET PERSPECTIVES

Les travaux présentés dans ce chapitre se poursuivent actuellement selon deux axes.

AUTO-AMORÇAGE DE PHARO. Nos travaux sur l'amorçage des langages réflexifs ont initialement été motivés par la difficulté de faire évoluer le noyau d'un langage réflexif tel que Pharo lorsque l'on souhaite y introduire de nouvelles fonctionnalités comme les Traits ou les variables d'instances de première classe ou encore un modèle de réflexion basé sur celui de MetaTalk. Comme nous l'avons montré, Seed est une infrastructure prometteuse dans ce domaine. Toutefois, amorcer le langage Pharo dans sa globalité (pas uniquement son noyau) pose encore de nombreuses difficultés [CDF+11]. L'une d'elles est liée au concept d'image (une sauvegarde de l'environnement d'exécution). Ce concept est particulièrement utile pour le déploiement à distance (auto-contenu) mais peu propice à l'amorçage, car il devient difficile d'assu-

rer la coévolution de la définition du langage et de son image. De plus, l'état global présent dans l'image Pharo peut être difficile à recréer à partir d'une définition statique car le code d'initialisation n'est plus disponible [PBDF14].

MISE À JOUR DYNAMIQUE DE CODE. La mise à jour dynamique du code d'un programme est particulièrement utile dans le contexte des CPS mobiles. En effet, pour s'adapter à son contexte, une application mobile doit pouvoir être mise à jour, et ce dynamiquement dans certains cas pour ne pas perdre l'état courant du programme. La mise à jour de code inclut le code de l'application, mais aussi celui des entités du langage sous-jacent comme les collections, les sockets ou encore le système de *threading*. Nous travaillons donc sur un système de mise à jour dynamique de code. Notre premier travail dans ce domaine a été d'utiliser Espell (cf. section 3.2.3) pour effectuer des mises à jour dynamiques des éléments de base. L'approche par virtualisation avec Espell a permis d'éviter les dépendances circulaires entre le code appliquant les mises à jours et celui des bibliothèques de base modifiées [PDB+15].

3.4 SYNTHÈSE DES CONTRIBUTIONS

Le tableau ci-dessous synthétise mon activité dans ce thème à travers la co-signature d'articles scientifiques, le co-encadrement de thèses ou encore la participation à des projets scientifiques. L'annexe A contient plus de détails concernant ces éléments.

1	Revue internationale	[PDF+14]
4	Conférences internationales	[CDF+11, PDDF15, PDBF15, PDB+15]
3	Ateliers internationaux	[NBD+11, PDFB13, PBDF14]
2	Thèses co-encadrées soutenues	[Pap13, Pol15]
1	Thèse co-encadrée en cours	Max Mattone
1	Projet de recherche	Projet « CAIRE » (coordinateur)
2	Collaborations scientifiques	INRIA RMoD, IEMN Telice

PRIX ET RÉCOMPENSES.

- En 2012, Guillermo Polito a remporté le 2e prix aux *ESUG innovation technology awards* avec son logiciel *Hazelnut* qui est le premier prototype pour auto-amorcer une image Pharo. `http://www.esug.org/wiki/pier/Conferences/2012/Innovation-Technology-Awards`
- En 2013, Guillermo Polito a remporté le 1er prix aux *ESUG innovation technology awards* avec son logiciel *Oz Recovery Tools* qui utilise Espell et son mécanisme de virtualisation pour « réparer » des images Pharo. `http://www.esug.org/wiki/pier/Conferences/2013/Innovation-Technology-Awards`

4 GESTION DES RESSOURCES

> [...] the real question is not when precisely Moore's Law will die; one can say it is already a walking dead. [...] since Moore's Law will not be handing us improved performance on a silver platter, we will have to deliver performance the hard way, by improved algorithms and systems. This is a great opportunity for computing research.
>
> Moshe Y. Vardi (Editor-in-chief)
> CACM, May 2014, Vol. 57(5)

> I have noted with dismay the bloatware phenomenon fueled by the expectation that Moore's Law would mask inefficient software. Developing resilient, secure, efficient software requires more skills, time, and money, along with a different mind-set, from what we see in the commercial software industry today.
>
> David K. Hemsath
> CACM, June 2014, Vol. 57(6)

Les ressources sont actuellement un frein important lorsqu'il s'agit de développer des applications pour CPS mobiles. C'est à cause des ressources disponibles sur ces équipements qu'il est nécessaire d'utiliser des bibliothèques et des *frameworks* logiciels dédiés pour développer pour ces plateformes. Ils permettent généralement une meilleure optimisation des performances (capacité de calcul) ou une consommation mémoire moindre (aussi bien pour la mémoire de stockage que la mémoire vive), car spécifiquement conçus pour l'équipement cible. Cette hétérogénéité logicielle (à cause de l'utilisation de différents *frameworks* dédiés) rend plus difficile la réutilisation du code d'une même application pour différents équipements. Cette volonté de réutilisation et de portabilité n'est pourtant pas nouvelle et est l'un des succès des machines virtuelles. Partant de cette idée, ce chapitre présente différents travaux que nous avons menés pour mieux gérer les ressources d'une

application s'exécutant au-dessus d'une machine virtuelle afin d'assurer sa portabilité. Dans le cadre de la collaboration entre les Mines de Douai et l'équipe RMoD de l'INRIA Lille, nous avons co-encadré deux thèses dont les travaux s'inscrivent dans ce thème. La première est celle de Guillermo Polito qui a proposé une approche générale pour automatiquement construire un environnement d'exécution sur mesure pour une application donnée afin de réduire son empreinte mémoire [Pol15]. La seconde est celle de Mariano Martinez-Peck, dont j'ai également été co-encadrant, qui a proposé un mécanisme similaire à une mémoire virtuelle au niveau applicatif permettant de décharger de façon transparente les objets non-utilisés dans la mémoire secondaire [MP12]. Plus récemment, je participe à un projet de recherche intitulé « JIT pour FPGA » avec l'ENSTA Bretagne Lab-STICC (cf. Annexe A, page 148). Ce projet nous a permis de co-financé la thèse de Xuan Sang Le dont le sujet porte sur la compilation à demande (JIT) vers des architectures matérielles spécialisées comme des *Field-programmable gate array* (FPGA). Ce travail vise à simplifier l'écriture d'applications haut niveau où certains calculs pourraient être ensuite déportés vers des architectures spécialisées afin de décharger le processeur central d'un robot mobile.

4.1 CONSTRUCTION D'ENVIRONNEMENTS D'EXÉCUTION DE TAILLE MINIMALE

4.1.1 Motivation

Les applications à objets sont constituées d'unités de code (e. g. *packages*, classes, méthodes). Lors du déploiement, toutes les unités de code d'une application sont transférées sur un équipement. Toutefois, certaines de ces unités ne sont jamais utilisées par l'application lorsqu'elle est exécutée et ce quel que soit les chemins d'exécutions empruntés par l'utilisateur. Ce problème est d'autant plus prégnant pour les bibliothèques tierces utilisées par une application (e. g. frameworks Web, bibliothèque de journalisation, ...). Ces bibliothèques sont souvent génériques et l'application n'en utilise généralement qu'une petite partie.

Pour mieux optimiser la consommation mémoire, les unités de code non-utilisées par une application ne doivent pas être déployées. Dans une certaine mesure, cela peut aussi permettre de meilleures performances, car il ne faut plus charger ces unités de code. Cette volonté de réduction des applications avant leur déploiement est également motivée par les limites matérielles et logicielles qu'imposent certains équipements. Par exemple, la VM Android

limite le nombre maximum de méthodes d'une application à 65535[1]. Toutes ces limitations peuvent conduire à l'impossibilité de déployer une application ou à limiter le nombre d'applications qu'il est possible de déployer sur un équipement. Pourtant de nombreuses briques logicielles sont nécessaires et doivent être déployées sur un robot mobile autonome pour qu'il puisse réaliser une mission.

Le travail présenté dans la suite de cette section est donc motivé par la nécessité d'identifier et de ne pas déployer les unités de code inutiles pour l'exécution d'une application donnée.

4.1.2 État de l'art

Cette section synthétise notre étude [Pol15] de l'état de l'art des approches de *tailoring* i. e. permettant la réduction de la taille des applications avant déploiement.

CRITÈRES D'ÉVALUATION. Pour évaluer l'existant, nous avons défini les critères importants que devrait satisfaire une solution idéale de réduction de la taille d'une application :

BIBLIOTHÈQUES DE BASE. Un langage de programmation contient de nombreuses bibliothèques de base souvent génériques et couvrant de larges besoins. Par exemple, Java 8 SE contient 4240 classes[2] et la version de développement de Pharo 3.0 contient 4115 classes et traits. Lors du déploiement d'une application, toutes ces unités de code standards ne sont pas forcément utilisées. Par exemple, une application « *hello world* » écrite en Java 1.6 SE occupe 25Mio de mémoire primaire lorsqu'elle est exécutée. Cette taille importante est la conséquence du chargement des bibliothèques de base. Une solution idéale doit donc pouvoir supprimer les éléments non-utilisés dans les bibliothèques de base.

BIBLIOTHÈQUES EXTERNES. De même, une application peut utiliser de nombreuses bibliothèques ou *frameworks* externes dont une partie seulement est utilisée pendant l'exécution. Une solution idéale doit pouvoir supprimer les éléments non-utilisés dans les bibliothèques externes.

CODE EXISTANT. Une solution idéale de *tailoring* doit être utilisable sur des applications existantes et ne pas nécessiter des modifications, quel que soit leur taille.

SUPPORT DE LA RÉFLEXION. L'utilisation de la réflexion dans le code source d'une application ne doit pas être un frein pour en réduire la taille. De

1 La documentation des codes-octet de la VM Dalvik (`http://source.android.com/devices/tech/dalvik/dalvik-bytecode.html`) indique que le `source register` accepte une valeur comprise entre 0 et 65535.

2 D'après la documentation Java `http://docs.oracle.com/javase/8/docs/api/`.

plus, l'application réduite doit fonctionner correctement et de la même manière que l'application originelle.

INFRASTRUCTURE STANDARD. Dans le cas idéal, l'application réduite doit pouvoir être exécutée avec l'infrastructure officielle (e. g. VM) et sans perte de performance.

CONFIGURABILITÉ. L'utilisateur de la solution doit pouvoir choisir la portée du processus de réduction. Par exemple, pour une application donnée, l'utilisateur peut choisir de réduire uniquement les unités de code de l'application elle-même et celles des bibliothèques externes et laisser intactes celles des bibliothèques de base. Ce besoin est justifié par le partage des bibliothèques de base dans certains équipements cibles.

SUPPORT DU TYPAGE DYNAMIQUE. Idéalement, la solution de réduction doit être utilisable et efficace sur du code écrit avec des langages dynamiquement typés i. e. sans annotations statiques de type.

RÉDUCTION MINIMALE. La solution doit permettre de sélectionner l'ensemble minimal des unités de code dont qu'une application déployée aura besoin pour fonctionner correctement i. e. elle ne doit pas contenir d'unité non-utilisée.

COMPLÉTUDE. La solution doit garantir qu'une application déployée contient toutes les unités de code dont elle a besoin pour fonctionner sans erreur selon tous ses chemins d'exécution possibles.

ÉVALUATION DES APPROCHES EXISTANTES. Nous avons étudié 4 familles d'approches permettant de réduire la taille d'une application avant son déploiement :

PLATEFORMES DÉDIÉES. Il s'agit de bibliothèques ou de *frameworks* spécifiquement conçus pour être exécutés sur une plateforme contrainte en mémoire et parfois mobile. Par exemple, Java Micro Edition (J2ME) est une version dédiée aux environnements contraints de la plateforme Java et Cocoa Touch[3] est une version dédiée aux smartphones et aux tablettes du framework Cocoa.

Les plateformes dédiées proposent effectivement une version réduite, mais fixe des bibliothèques de base qui ne peuvent donc pas être adaptées ou configurées pour une application donnée. Les applications doivent être spécifiquement écrites en utilisant ces plateformes dédiées, ce qui exclut la possibilité de réduire des applications existantes. Ces approches assurent la complétude mais pas la minimalité, car l'application déployée contient généralement des unités de code inutiles. Enfin,

3 https://developer.apple.com/technologies/ios/cocoa-touch.html

si des opérations réflexives sont fournies, elles fonctionneront correctement une fois l'application déployée, car il ne s'agit pas d'un processus automatique de construction d'une application réduite mais d'un développement manuel.

APPROCHES STATIQUES. Il s'agit d'utiliser les informations statiques (types, annotations, noms, ...) dans le code source (ou le code-octet) d'une application pour déterminer le sous-ensemble minimal des éléments utilisés. Pour cela, ces approches construisent généralement le graphe d'appels [GDDC97] à partir du point d'entrée de l'application. Différentes variations de cette technique ont permis de réduire la taille d'applications Java [RK02, TSL03, Tit06], d'*assemblies* .NET [SD10], d'applications Objective-C [BO14] et même du code écrit en Self [Age96]. Self étant un langage dynamiquement typé, une première étape a consisté à inférer les types pour ensuite utiliser ces techniques statiques.

Les approches statiques permettent de réduire les bibliothèques de base et externes ainsi que du code existant. L'application réduite produite peut être exécutée sur un environnement standard. Toutefois, ces approches statiques ne permettent pas de sélectionner l'ensemble minimal des unités de code utilisées lorsqu'elles sont appliquées sur du code écrit dans un langage dynamiquement typé. Elles souffrent également d'un support limité de la réflexion qu'elles tentent de combler grâce à des analyses spécifiques des chaines de caractères du programme par exemple. Enfin, elles ne sont pas configurables et s'appliquent à une application dans sa globalité.

APPROCHES DYNAMIQUES. Ces approches utilisent uniquement les informations disponibles durant l'exécution d'une application (e. g. flow d'exécution, objets vivants, ...) pour éliminer le code non-utilisé. Une première technique consiste à faire du chargement à la demande i. e. installer localement les classes et les méthodes nécessaires à l'exécution en les récupérant depuis un environnement externe (e. g. serveur). Une seconde technique consiste à déployer l'application complète et laisser le ramasse-miettes (GC) éliminer le code non-utilisé en se basant sur des statistiques d'utilisation. Des variations ou même des combinaisons de ces deux techniques ont notamment été utilisées dans JUCE [PRT+04], OLIE [GNM+03], SlimVM [WGF11] et Marea [PBD+13].

Les approches dynamiques ne permettent pas d'assurer la complétude. Différentes stratégies sont donc mises en place pour traiter les erreurs durant l'exécution à cause d'une unité de code manquante. Par exemple, charger l'unité de code manquante depuis un serveur distant si l'application est connectée. La plupart de ces approches (sauf Marea) nécessitent de déployer l'application réduite sur une infrastructure spécialisée comme une VM modifiée permettant le chargement paresseux et

distant des unités de code manquantes ou calculant des statistiques d'usage. Cela a un impact sur les performances de l'application mais elles permettent généralement de traiter correctement la réflexion et la méta-programmation. SlimVM propose même une version modifiée de certaines opérations réflexives car leur VM utilise une représentation mémoire des objets et des classes différente du format standard utilisé par la VM Java. Marea est la seule approche permettant de configurer le processus de réduction grâce à la possibilité d'indiquer des graphes d'objets potentiellement déchargeables. La minimalité de ces approches dépend avant tout de la granularité de leur mécanisme de chargement et de l'efficacité de leur mécanisme de détection de code non-utilisé pendant l'exécution. Une granularité importante chargera toujours plus de code que nécessaire mais une granularité trop faible induira une perte de performance due aux nombreux chargements.

APPROCHES HYBRIDES. Cette famille regroupe les travaux utilisant des informations statiques et dynamiques. L'idée commune consiste à démarrer l'application et la mettre en pause dès que suffisamment d'informations dynamiques sont disponibles i. e. pile d'appels créée, classes chargées et initialisées, quelques objets instanciés. Des techniques statiques sont ensuite utilisées sur les informations dynamiques disponibles à cet instant avant de continuer l'exécution.

Java in The Small (JITS) [CGV10] est une approche hybride permettant de sélectionner les unités de code utilisées par une application et de les sauvegarder dans une image binaire (forme réduite de l'application). Une VM spécifique peut ensuite charger et exécuter cette image. JITS permet de réduire les bibliothèques de base et externes ainsi que le code applicatif ou du code existant. JITS ne permet pas de configurer le processus de réduction. Concernant la réflexion, JITS présente les mêmes problèmes que les approches statiques, car une partie seulement des informations dynamiques sont utilisées.

SYNTHÈSE. La table 4 synthétise notre comparaison des techniques existantes [Pol15] pour fabriquer automatiquement un environnement d'exécution minimal pour une application donnée. Les approches dédiées sont limitées car elles sont manuelles et ne reposent pas sur un processus de réduction. Les approches automatiques permettent de réduire des applications existantes sans en modifier le code source. Parmi elles, les approches dynamiques et hybrides prennent en compte correctement la réflexion. La configurabilité n'est généralement pas supportée. Concernant la minimalité, les techniques dynamiques sont les plus efficaces car elles disposent des informations les plus précises. Enfin, aucune approche ne permet de garantir la complétude. Comme évoqué précédemment, différentes techniques comme le chargement paresseux sont employées dans l'existant pour pallier cet inconvénient.

	Plateformes dédiées	Approches statiques	Approches hybrides	Approches dynamiques
Bibliothèques de base	+	+	+	+
Bibliothèques externes	–	+	+	+
Code existant	–	+	+	∼
Réflexion	+	–	–	+
Infra. standard	+	–	–	∼
Configurabilité	–	–	–	∼
Typage dynamique	+	–	–	+
Minimalité	–	+	+	∼
Complétude	+	–	–	∼

TABLE 4 : Comparaison des familles d'approches existantes permettant la construction d'un environnement d'exécution minimal pour une application donnée (+ indique que le critère est satisfait, – qu'il ne l'est pas et ∼ qu'il l'est partiellement).

4.1.3 L'approche « *Run-fail-grow* »

PRÉSENTATION GÉNÉRALE. L'approche *Run-fail-grow* (RFG) est une solution générale qui à partir d'une application permet de produire une version réduite de cette application dépourvue des unités de code non-utilisées. Pour cela, RFG utilise les informations dynamiques pendant l'exécution d'une application de référence pour détecter les éléments utilisés et ainsi construire une application réduite. La figure 13 et l'algorithme 1 décrivent ce processus. En premier lieu, l'application est chargée et mise en pause (arrêt des *threads*) dans un environnement de référence (partie gauche de la figure 13 et ligne 1 de l'algorithme 1). Ensuite, une *graine* est installée dans un environnement de culture (*nurtured environnement* situé à droite sur la figure 13 et ligne 2 de l'algorithme 1). Cette graine doit permettre de démarrer l'exécution de l'application (étape 1 dans la figure 13 et lignes 3-5 de l'algorithme 1). Lorsqu'une erreur survient, car une unité de code (classe, méthode, ...) manque pour continuer l'exécution (étape 2 dans la figure 13 et ligne 6 de l'algorithme 1), cette unité est copiée depuis l'environnement de référence (étape 3 dans la figure 13 et ligne 7 de l'algorithme 1) et l'exécution peut reprendre dans l'environnement de culture (étape 4 dans la figure 13 et ligne 8 de l'algorithme 1). À la fin de ce processus, l'environnement de culture contient une application de taille réduite intégrant toutes les unités de code qui ont été nécessaires pendant l'exécution. Cet environnement peut être sauvegardé dans une image binaire et déployée pour s'exécuter sur une VM standard. Comme nous le verrons dans la suite, ce processus est configurable via le choix de la *graine*.

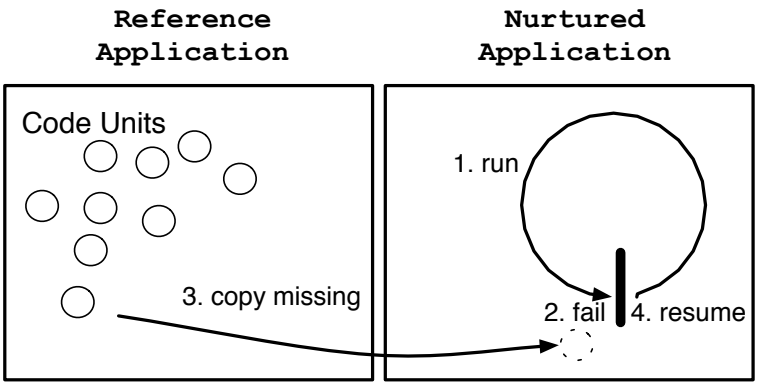


FIGURE 13 : Principe général de l'approche « *Run-fail-grow* » pour construire automatiquement des applications de taille réduite en éliminant le code non-utilisé.

```
1  Initialize reference application;
2  Initialize nurturing application with the seed;
3  Install entry point(s);
4  while not finished do
5  |   run the nurtured application;
6  |   if a code unit is missing then
7  |   |   install missing code units;
8  |   |   restart message send ;
9  |   end
10 end
```

Algorithme 1 : Description du processus « *Run-fail-grow* ».

EXEMPLE. Le code source 8 montre un extrait du code d'une application de journalisation. Certaines parties du code source de cette application ne sont pas utilisées à l'exécution (parties grises) et elles pourraient tout à fait être enlevées. Le point d'entrée de cette application est la méthode `MainApp»start` qui instancie un `StdoutLogger` (cf. ligne 2) qui utilise en interne la variable globale `stdout` et la classe `Time` de la bibliothèque de base de Pharo. Les parties grises de ce code correspondent aux unités de code non-utilisées : la classe `RemoteLogger` (lignes 15-24), la méthode `newLine` de la classe `StdoutLogger` (lignes 7-8). De plus, comme la classe `RemoteLogger` n'est pas utilisée et que les *sockets* ne sont pas utilisées ailleurs dans le code cette application, la classe `Socket` de la bibliothèque de base de Pharo peut être enlevée aussi. Il en est de même pour de nombreuses classes de la bibliothèque de base telles que : `Date`, ... Enfin, seule la méthode `now` de la classe `Time` est utilisée, on peut donc conserver uniquement les méthodes nécessaires à l'exécution de `Time»now`. Cet exemple illustre notre objectif de produire une application réduite qui contient uniquement les unités de code nécessaires à l'exécution d'une application donnée.

```
1  MainApp>>start
2      logger := StdoutLogger new.
3      logger log: 'Application has started'.
4      "do something"
5      logger log: 'Application has finished'.
6
7  StdoutLogger»newLine
8     stdout newLine.
9
10 StdoutLogger>>log: aMessage
11     stdout nextPutAll: Time now printString.
12     stdout nextPutAll: aMessage.
13     stdout newLine.
14
15 RemoteLogger»log: aMessage
16     | socket |
17     socket := self newSocket.
18     socket nextPutAll: Time now printString.
19     socket nextPutAll: aMessage.
20     socket newLine.
21
22 RemoteLogger»newSocket
23     "...."
24     "creates an instance of socket given some configuration"
```

Code source 8 : Exemple de code source d'une application de journalisation en Pharo (les parties grises dénotent du code non-utilisé).

LE PROCESSUS RFG SUR UN EXEMPLE. La figure 14 représente le contenu de l'environnement de culture à différentes étapes du processus :

ÉTAPE 0 Initialement, l'environnement de culture ne contient que les bibliothèques de base du langage Pharo (*graine*).

ÉTAPE 1 Le point d'entrée de l'application est ensuite injecté dans l'environnement de culture. Il s'agit d'un objet instance de la classe `MainApp`. Un processus (*thread*) est également injecté afin d'exécuter le code `aMainApp start` qui doit lancer l'application. Il est important de noter que la classe `MainApp` n'est pas encore injectée dans l'environnement. L'exécution démarre et l'objet `aMainApp` reçoit le message `start`.

ÉTAPE 2 Cet envoi de message déclenche une erreur, car classe de l'objet `aMainApp` n'est pas présente et il n'est donc pas possible d'accéder à son dictionnaire de méthode pour résoudre cet envoi de message. L'exécution est stoppée. Le processus RFG injecte donc la classe `MainApp` (dont son dictionnaire de méthode) et la méthode `start` en la copiant depuis l'environnement de référence. L'exécution de cette méthode est ensuite activée dans l'environnement de culture où l'exécution peut continuer.

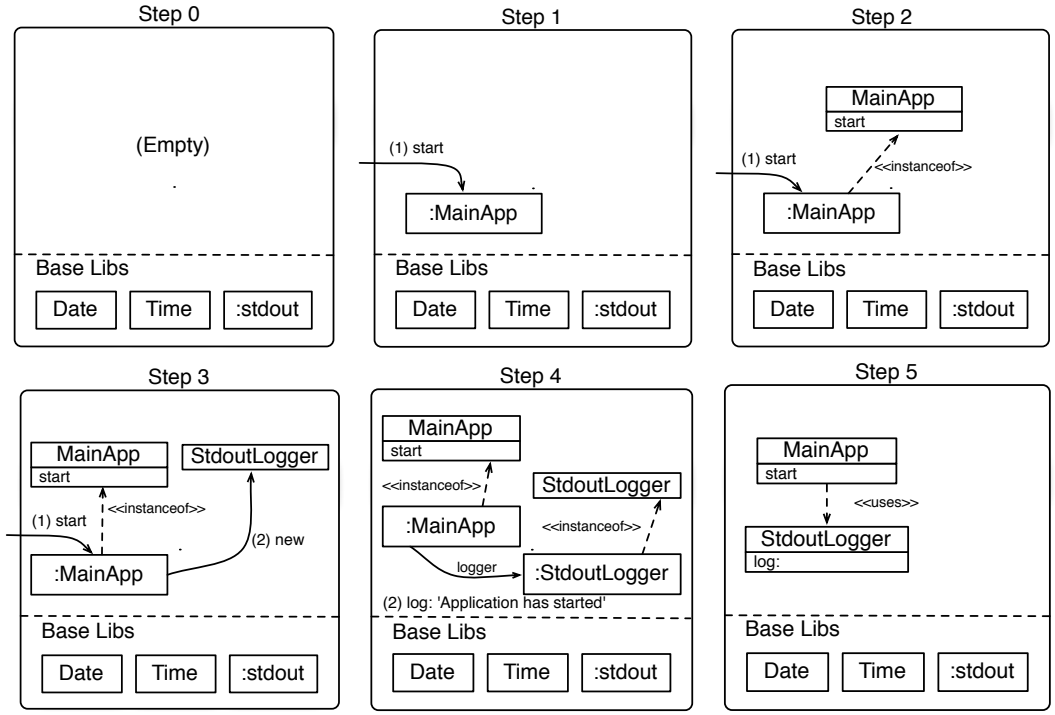


FIGURE 14 : Contenu de l'environnement de culture pendant l'exécution du processus RFG sur l'application décrite par le code source 8.

ÉTAPE 3 La première instruction de la méthode `start` (cf. ligne 2 du code source 8) déclenche une nouvelle erreur, car la classe `StdoutLogger` n'est pas présente. De même, l'exécution est stoppée et le processus RFG injecte cette classe, mais sans aucune méthodes ni méta-données (e. g. superclasse, package, sous-classes). Seule la structure de la classe a été copiée depuis l'environnement de référence. L'exécution peut à nouveau continuer dans l'environnement de culture. La classe (« vide ») `StdoutLogger` reçoit donc le message `new`. Cet envoi de message est parfaitement résolu et exécuté dans l'environnement de culture car la méthode `new` fait partie de la bibliothèque de base de Pharo et est donc présente dans la *graine* initialement injectée dans l'environnement de culture. Une instance de `StdoutLogger` est donc créée dans l'environnement de culture.

ÉTAPE 4 L'exécution se poursuit avec la ligne 2 du code source 8 mais stoppée car il manque la méthode `log:`. Le processus RFG copie donc cette méthode depuis l'environnement de référence dans l'environnement de culture et il insère une référence vers cette nouvelle méthode copiée dans le dictionnaire de méthode de la classe `StdoutLogger` qui était vide jusqu'à présent. L'exécution dans l'environnement de culture est re-démarrée et cette fois, l'envoi de message de la ligne 2 du code source 8 est parfaitement résolu et conduit à l'exécution de la méthode

`log:` qui vient juste d'être installée. Lorsque l'exécution de la méthode `log:` se termine, l'exécution se poursuit avec la ligne 5 du code source 8. Cette instruction ne provoque pas d'erreur car la méthode `log:` est déjà présente dans l'environnement de culture.

ÉTAPE 5 Cette étape montre le contenu de l'environnement de culture lorsque l'exécution de l'application se termine. Les objets feuilles (non référencés) ont été libérés par le ramasse-miettes. Il ne reste que les méthodes et les classes qui ont été exécutées par l'application excepté celles injectées initialement sous forme de *graine*.

DÉTECTION DES UNITÉS DE CODE MANQUANTES. Ce processus RFG repose sur la détection des unités de code manquantes pendant l'exécution de l'application dans l'environnement de culture. Cette détection est possible grâce à la mise en place de *détecteurs* qui sont installés dans l'environnement de culture à la place des éléments réels. A chaque fois que l'exécution accède à l'un de ces détecteurs, l'exécution est stoppée et le processus RFG peut copier les unités de code manquantes à la place des détecteurs qui ont été accédés avant de relancer l'exécution. Ces détecteurs sont installés dynamiquement dans l'environnement de culture en même temps qu'une nouvelle unité de code y est copiée. Par exemple, un détecteur d'objet manquant permet de capturer les envois de messages à un objet qui n'existe pas encore dans l'environnement de culture comme les classes. Lorsqu'un tel détecteur est accédé, le processus RFG installe une copie partielle de l'objet original (*stub*) i. e. la plupart de ses champs sont remplacés par des détecteurs. Lorsqu'une classe est copiée pour la première fois dans l'environnement de culture, son dictionnaire de méthode est rempli avec des détecteurs de méthode manquante permettant d'indiquer que cette méthode doit être installée. Ces détecteurs de méthode permettent de résoudre correctement l'envoi de message dans les cas de redéfinition.

CONFIGURABILITÉ À TRAVERS LA *graine*. Le processus RFG présenté est automatique mais configurable à travers le choix de la *graine*. Une graine est une collection d'unités de code que l'on injecte initialement dans l'environnement de culture. Il peut s'agir de classes, de méthodes ou encore d'objets pré-initialisés. Ces unités de code étant présentes, elles ne seront pas manquantes lors de l'exécution du processus RFG et ne sont donc pas soumises à une réduction de taille. Dans l'exemple décrit ci-dessus, la graine incluait toutes les bibliothèques de base de Pharo. Or, pour réduire au minimum la taille d'une application, il aurait mieux valu prendre une graine vide par exemple.

ÉVALUATION. Ce processus RFG a été implémenté grâce à l'infrastructure Espell (cf. section 3.2.3). Cette implémentation a permis de valider ce proces-

sus sur six applications différentes couvrant un large spectre : depuis l'addition de deux entiers (objets immédiats) jusqu'à une application Web complète et en passant par une application utilisant la réflexion. Les résultats montrent des réductions de taille comprises entre 95% et 99% pour ces applications (avec une graine vide). En plus de ces excellents résultats, l'approche RFG permet également de satisfaire 8 critères (cf. section 4.1.2) sur les 9 que nous avions définis. Comme toutes les autres approches (excepté les approches dédiées), RFG ne permet pas d'assurer le critère de complétude. L'application produite par RFG ne peut être complète que si tous les chemins d'exécution possibles ont été empruntés pendant le processus de réduction. L'exécution de tests fonctionnels couvrants peut aider à atteindre cet objectif. Toutefois, il est difficile à assurer la couverture en toute généralité. Comme pour les autres approches, RFG peut intégrer à une application déployée un mécanisme de chargement paresseux à distance si du code venait à manquer durant son exécution.

4.2 GESTION DE LA MÉMOIRE À L'EXÉCUTION

4.2.1 Motivation

Lorsqu'une application à objets est exécutée, les objets sont d'abord chargés en mémoire primaire i. e. en *Random Access Memory* (RAM). Ensuite, la quantité de mémoire utilisée par l'application au cours de son exécution dépend de la création et de la destruction des objets. De nombreux mécanismes sont à l'œuvre pour optimiser la consommation mémoire d'une application pendant sont exécution i. e. éviter de laisser des objets non-utilisés occuper de la mémoire primaire. Un objet non-utilisé désigne un objet qui n'a pas été utilisé par le programme auquel il appartient pendant un certain laps de temps. En effet, certains objets ne sont utiles que dans certaines situations ou conditions comme lors de l'initialisation de l'application par exemple.

Deux mécanismes importants existent pour optimiser la consommation mémoire d'une application. Le premier est le mécanisme de mémoire virtuelle fourni par les systèmes d'exploitation qui « découpent » la mémoire primaire en *pages* [Den70]. Une page peut être déchargée en mémoire secondaire si son contenu n'est pas utilisé et elle est rechargée en mémoire primaire de façon (presque) transparente dès que son contenu redevient nécessaire à l'exécution d'une application. Toutefois, ce mécanisme étant au niveau du système d'exploitation, il ne permet pas de profiter pleinement de la connaissance de la structure des applications. Par exemple, la granularité de ce mécanisme est une *page* et les objets utilisés et non-utilisés sont mélangés au sein des pages. Le second mécanisme est le ramasse-miettes (GC) souvent intégré à la machine virtuelle d'un langage. Le GC permet de libérer la mémoire allouée aux objets qui ne sont plus transitivement atteignables en suivant toutes les

références entre les objets [McC60]. Bien que le GC soit extrêmement utile, il ne permet pas de gérer les objets inutilisés, mais encore atteignables depuis d'autres objets qui encombrent pourtant la mémoire primaire [Kae86].

Le gaspillage de mémoire induit notamment par les objets inutiles conduit à pouvoir exécuter moins d'applications dans une quantité de mémoire donnée et a aussi un impact sur les performances à cause du mécanisme de mémoire virtuelle. Le cas extrême de ce gaspillage étant celui des fuites mémoire (*memory leaks*) [BM08] qui peuvent conduire au « *crash* » de l'application.

Pour mieux quantifier l'espace mémoire gaspillé par les objets inutiles, nous avons modifié une VM Pharo pour tracer l'utilisation des objets. Cette VM modifiée permet de marquer tous les objets comme inutiles et de démarrer le mode trace. Dans ce mode, la VM marque comme utilisé tous les objets qui reçoivent un message ou qui sont accédés directement par la VM. Enfin, il est possible de désactiver le mode trace et de récupérer l'ensemble des objets inutilisés. Grâce à cette VM, nous avons analysé 4 applications différentes [MPBD+10b] : un site Web écrit en Pier, la suite Moose, Dr. Geo et l'environnement Pharo. Pour chacune de ces applications, nous effectué de nombreuses actions permettant de couvrir plusieurs chemins d'exécution. Ces expériences ont permis de déterminer que les objets utilisés par ces applications n'occupent qu'entre 27% et 37% de la mémoire totale consommée.

Ce travail est donc motivé par ces résultats qui montrent le potentiel d'une *mémoire virtuelle au niveau applicatif* qui déchargerait les objets référencés mais inutilisés par l'application en cours d'exécution.

4.2.2 État de l'art

Cette section synthétise les solutions existantes de mémoires virtuelles au niveau applicatif au moment de la thèse de Mariano Martinez-Peck [MP12].

CRITÈRES DE COMPARAISON. Pour évaluer et comparer ces solutions, nous avons défini les critères suivants :

GRANULARITÉ. La granularité de la solution doit être en lien avec les objets plutôt que des pages de bits comme pour les mécanismes de mémoire virtuelle des systèmes d'exploitation. De plus, elle doit générer le moins de défauts d'objet (objet nécessaire à l'exécution, mais non présent en mémoire primaire) [HMB90] possible tout en assurant que lors des rechargements, le moins d'objets inutiles seront chargés. Une grande granularité permet de décharger beaucoup d'objets d'un coup (i. e. minimise le surcoût lié au déchargement) mais génère beaucoup de défauts d'objet ce qui provoque plus rapidement le rechargement de tous les objets. À l'inverse, une petite granularité nécessite de nombreux déchargements (surcoût) mais génère moins de défauts d'objet.

UNIFORMITÉ. Dans le cas de langages réflexifs, les méta-objets réifient la structure et même le comportement du programme en cours d'exécution (e. g. classes, méthodes, fermetures lexicales, processus, contextes, ...). Une solution uniforme doit pouvoir également s'appliquer à ces objets.

RÉVERSIBILITÉ. Il s'agit de pouvoir recharger les objets qui ont été déchargés. Contrairement à un ramasse-miettes qui libère complètement les objets non référencés, la solution doit placer les objets inutiles en mémoire secondaire par exemple pour pouvoir les recharger en cas de besoin.

CHARGEMENT AUTOMATIQUE. Si un objet déchargé devient utile pour continuer l'exécution de l'application, il doit être automatiquement rechargé.

DÉCHARGEMENT AUTOMATIQUE. La solution idéale doit décharger automatiquement les objets non-utilisés par une application en cours d'exécution.

TRANSPARENCE. Le comportement (i. e. les résultats obtenus) d'une application doit être le même qu'elle s'exécute avec ou sans mécanisme de mémoire virtuelle. De plus, le code de l'application ne doit pas être pollué par du code relatif au chargement ou déchargement des objets inutiles.

CONFIGURABILITÉ. Le programmeur doit pouvoir influencer quand décharger et recharger ainsi que comment le faire. Fournir cette possibilité au programmeur permet de profiter de sa connaissance du domaine pour définir des stratégies plus sophistiquées et basées sur les objets de son application plutôt qu'un simple *Least Recent Used* fourni par la mémoire virtuelle du système d'exploitation.

PORTABILITÉ. Autant que possible, le mécanisme de mémoire virtuelle ne doit pas nécessiter de nouvelles fonctionnalités dans la VM ou reposer sur des spécifités de certains systèmes d'exploitation. Dans l'idéal, le mécanisme de mémoire virtuelle doit être un outil portable s'exécutant au-dessus d'une machine virtuelle standard (VM non modifiée).

SOLUTIONS EXISTANTES. Nous avons analysé différentes solutions qui s'apparentent à des mémoires virtuelles ou en permettent la construction au sein d'un langage à objets s'exécutant au-dessus d'une VM :

MÉMOIRE VIRTUELLE AU NIVEAU SYSTÈME (OS). Une mémoire virtuelle au niveau système repose sur le (dé)chargement de pages. Ce mécanisme ne permet pas de décharger les objets inutilisés par une application. En effet, le développeur ne peut pas contrôler la répartition des objets dans les pages (i. e. regrouper les objets inutilisés). De plus, le

ramasse-miettes parcourt périodiquement tous les objets (même ceux en mémoire secondaire) ce qui peut conduire à une dégradation des performances [HFB05] due au chargement des pages. Enfin, le système d'exploitation ne peut pas prévoir quels sont les objets inutilisés et les stratégies de (dé)chargement utilisées (LRU, ...) ne bénéficient pas de ces informations.

MÉMOIRE VIRTUELLE AU NIVEAU APPLICATIF (APP). L'approche [KLVA93] permet au développeur d'une application d'enregistrer un gestionnaire de défaut de page pour les traiter directement depuis le code applicatif lorsqu'ils sont signalés par le système d'exploitation. Le développeur peut ainsi décider quelle page doit être déchargée et même implémenter ses propres stratégies de résolution adaptées à son application. Ce travail conclut que les mécanismes de mémoire virtuelle spécifiques à une application permettent d'obtenir de meilleures performances. Dans cette même famille, Exokernel [EKO95] est un système d'exploitation permettant au développeur d'une application d'adapter les stratégies de (dé)chargement mais aussi certaines abstractions du mécanisme de mémoire virtuelle comme l'utilisation d'une structure différente pour les pages directement depuis le code applicatif.

LOOM. LOOM [Kae86] (*Large Object-Oriented Memory*) propose un mécanisme de (dé)chargement d'objets entre une mémoire primaire et secondaire pour Smalltalk-80. La vision proposée est que les objets sont permanents en mémoire secondaire mais copiés en mémoire primaire temporairement s'ils sont nécessaires à l'exécution. Un objet en mémoire primaire (16 bits) ne référence pas directement un objet en mémoire secondaire (32 bits) mais il référence un *stub* qui contient l'adresse (32 bits) de ce dernier. Lorsque la VM accède à un *stub* (e. g. envoi de message), il est remplacé par l'objet original chargé depuis la mémoire secondaire avant de poursuivre l'exécution. Le modèle proposé par LOOM est intéressant mais complexifié par de nombreuses contraintes qui n'en sont plus de nos jours. La mémoire primaire de 16 bits et la mémoire secondaire de 32 bits. La mémoire secondaire est considérée comme extrêmement plus lente que la mémoire primaire or c'est moins le cas de nos jours grâce aux *Solid-State Drives* par exemple. LOOM intègre des mécanismes complexes pour gérer la création trop fréquente de certains objets comme les `MethodContext` or un ramasse-miettes générationnel [Ung84] permet de résoudre efficacement cela de nos jours.

PLATEFORMES DÉDIÉES. Java Micro Edition (J2ME) [Jav] est un exemple de plateforme (Java dans ce cas) dédiée aux environnements contraints. Ces plateformes permettent effectivement de réduire la consommation mémoire à l'exécution mais elles ne sont pas adaptées à chaque application et divergent totalement des plateformes standards.

APPROCHES HYBRIDES. *Java in The Small* (JITS) [CGV10] est outil permettant de sélectionner les unités de code utilisées par une application afin de produire une application réduite qui pourra être exécutée par la VM JITS. Contrairement aux plateformes dédiées, le développeur d'une application peut utiliser une bibliothèque standard (J2SE) et utiliser JITS pour réduire la taille de son application et des bibliothèques qu'elle utilise avant déploiement. De fait, la consommation mémoire à l'exécution sera réduite.

PERSISTANCE ET ODBMS. Les gestionnaires d'objets distribués [Dec86], les systèmes de persistance d'objets [MBMZ00] ou encore les bases de données objets (ODBMS) [BOS91] reposent tous sur la notion de défaut d'objet pour détecter qu'un objet distant, en mémoire secondaire ou stocké en base est nécessaire. Les ODBMS sont similaires à un mécanisme de mémoire virtuelle, car ils permettent de sauvegarder automatiquement un graphe d'objets en base. Toutefois, ils ne prennent pas en compte les objets à l'extérieur de ce graphe qui référencent des objets dans ce graphe. Les objets persistés ne peuvent donc pas être déchargés de la mémoire primaire sans introduire d'inconsistances.

MELT. Melt [BM08] décharge en mémoire secondaire les objets non-utilisés (*leaking objects*) par un programme mais référencés (non éliminés par le GC). Ces objets peuvent être chargés à nouveau en mémoire primaire s'ils redeviennent nécessaires à l'exécution. Comme pour LOOM, Melt utilise des objets *stub* en mémoire primaire pour stocker l'adresse des objets en mémoire secondaire (tailles des mémoires différentes). Les objets en mémoire primaire référencent donc les objets *stub*. Par contre, les adresses contenues dans les objets sauvegardées en mémoire secondaire peuvent ne plus être valides au moment du chargement, car le GC a déplacé les objets référencés. Pour cela, Melt propose des objets *scion* (indéplaçables par le GC) en mémoire primaire qui détiennent une référence version l'objet original en mémoire primaire également. Les objets en mémoire secondaire contiennent donc des références vers ces objets *scion*.

IMAGESEGMENT. ImageSegment [IKM+97] permet de (dé)charger un graphe d'objets en mémoire secondaire. Pour cela, le développeur commence par fournir une liste d'objets *racines* afin de déterminer le graphe d'objets à décharger par transitivité. Une instance de la classe `ImageSegment` est créée et elle contient 3 ensembles : les objets racines fournis, les objets internes au graphe (référencés uniquement par des objets appartenant au graphe) et enfin les objets partagés i. e. référencés par des objets externes au graphe. Détecter les objets partagés est nécessaire mais coûteux car n'importe quel objet du système peut référencer un objet dans le graphe. Lorsqu'un `ImageSegment` est déchargé, les objets

racines et internes sont sauvegardés en mémoire secondaire et les objets racines sont remplacés par des *proxys* ce qui provoque l'élimination des objets racines originaux ainsi que des objets internes par le GC car ils deviennent non atteignables. Notons que les objets partagés ne sont pas déchargés. Lorsqu'un développeur fournit un ensemble d'objets racines, il est difficile d'estimer si le graphe résultant contient beaucoup d'objets partagés ou non. Pourtant, plus il y aura d'objets partagés, moins d'espace mémoire sera libéré.

ÉVALUATION. La table 5 synthétise notre analyse de l'état de l'art [MP12]. La configurabilité des mécanismes de mémoires virtuelles au niveau applicatif est meilleure que celle des mémoires virtuelles au niveau système. Toutefois, ces deux familles restent dépendantes d'un système d'exploitation donné. La granularité de LOOM est un objet plutôt qu'une *page*. Or, des travaux [Sta82, Sta84] ont montré que cette granularité est trop fine et que les objets doivent être groupés judicieusement pour obtenir de bonnes performances. Les plateformes dédiées et les approches hybrides ne proposent pas de mécanisme pour (dé)charger dynamiquement les objets non-utilisés. Les systèmes de base de données objet (ODBMS) nécessitent pour la plupart que le développeur déclenche la persistance et écrive des requêtes. Melt suppose que les objets inutilisés forment un graphe transitif. Si cette hypothèse n'est pas vérifiée, Melt peut conduire à consommer plus de mémoire. Notre analyse détaillée [MPBD+10a, MPBDF12] d'ImageSegment nous a permis de mieux cerner ses limites. Par exemple, ne pas décharger les objets partagés, les classes ou encore les méthodes sont des freins pour vraiment réduire la consommation mémoire. La détection des objets partagés est optimisée en ImageSegment car elle repose sur une traversée complète de la mémoire par le GC (et donc au niveau VM) mais elle reste coûteuse. Pour finir, les graphes sont complexes et il est inévitable qu'un programmeur essaie de décharger un graphe contenant des *proxies* correspondants à un autre graphe déjà déchargé. Ce problème d'intersection de graphe n'est pas supporté par ImageSegment ce qui conduit à inconsistances.

4.2.3 Marea : une mémoire virtuelle au niveau applicatif

Cette section décrit Marea [PBD+13], un mécanisme de mémoire virtuelle au niveau applicatif pour gérer les objets non-utilisés.

CHOIX DE CONCEPTION. Notre analyse de l'existant (cf. section 4.2.2) ainsi que les défis liés à la construction d'un tel système [PBD+11b] nous ont permis d'étayer nos choix de conception pour Marea. Par exemple, la granularité utilisée par Marea est un graphe d'objets défini par la fermeture transitive d'objets racines. Cette granularité permet de réduire le nombre de *proxies*

	Mémoire virtuelle OS	*Mémoire virtuelle App*	*LOOM*	*Plateformes dédiées (J2ME)*	*Approches hybrides (JITS)*	*Persistance et ODBMS*	*Melt*	*ImageSegment*
Granularité	–	–	∼	/	/	∼	∼	+
Uniformité	+	+	∼	–	–	–	∼	–
Réversibilité	+	+	+	–	–	+	+	+
Chargement auto.	+	+	+	/	/	∼	+	+
Déchargement auto.	+	+	+	/	/	–	+	–
Transparence	+	+	+	–	–	∼	+	∼
Configurabilité	–	+	–	–	+	+	/	+
Portabilité	–	–	–	+	+	+	–	–

TABLE 5 : Synthèse comparative des solutions existantes pour (dé)charger les objets inutiles pendant l'exécution d'une application (+ indique que le critère est satisfait, – qu'il ne l'est pas, ∼ qu'il l'est partiellement et / qu'il ne s'applique pas).

nécessaires à quelques-uns lorsque le graphe est déchargé. Lors d'un défaut d'objet, les objets qu'il référence sont également chargés (tout le graphe auquel il appartient) ce qui permet de limiter les chargements en cascade. Toutefois, un graphe d'objets trop grand chargera beaucoup d'objets inutiles. Contrairement à ImageSegment, nous avons décidé de décharger aussi les objets partagés contenus dans un graphe. Ce choix est motivé par le fait que les objets partagés sont inévitables dans un graphe. De plus, nos expériences ont montré qu'un graphe utilisant une classe comme racine avait environ 17% d'objets partagés en Pharo. Enfin, un nombre important d'objets partagés diminue d'autant le gain de mémoire s'ils n'étaient pas déchargés.

PRÉSENTATION GÉNÉRALE. Marea repose sur 4 briques essentielles : un mécanisme de (dé)chargement de graphes d'objets (*Object Graph Swapper* – OGS), des *proxies*, un sérialiseur de graphe d'objets (*Object Serializer* – ObS) et un mécanisme de stockage. La première brique (OGS) est celle au cœur de Marea responsable du (dé)chargement de graphes d'objets entre les mémoires primaire et secondaire ce qui inclut la détection efficace et la gestion correcte des objets partagés. Un OGS utilise des *proxies* en mémoire primaire pour représenter un graphe déchargé et détecter qu'il doit être chargé à nouveau. Notre modèle de *proxy* [PBD+11a, PBF+14] est général et extensible ; il per-

met de capturer tous les accès à un objet avec une empreinte mémoire la plus compacte possible. Un OGS utilise également un sérialiseur de graphes responsable de la conversion d'un graphe d'objets en binaire (sérialisation) et inversement (matérialisation). La matérialisation doit être particulièrement efficace, car l'exécution de l'application a été stoppée par un défaut d'objet. Le sérialiseur doit fonctionner pour tous les types de données possibles, dont les classes, les méthodes, les fermetures lexicales ou encore les processus. Enfin, la brique de stockage consiste à transférer le binaire représentant un graphe d'objets vers une mémoire secondaire comme le système de fichiers local ou une base de données. La suite de cette section décrit plus particulièrement les algorithmes de déchargement et chargement d'un graphe et la gestion des objets partagés.

ALGORITHME DE DÉCHARGEMENT D'UN GRAPHE. La figure 15 illustre l'algorithme de déchargement d'un graphe d'objets à travers 5 étapes :

1. Le développeur commence par fournir le ou les objets racines qui permettent de définir le graphe (e. g. objet A).

2. Un unique identifiant est associé à ce nouveau graphe (e. g. 42). Pour chaque objet de ce graphe, un *proxy* est créé ainsi qu'une table de correspondance (cf. `proxiesDict`) entre les objets du graphe et leurs *proxies*. Tous les objets du graphe sont sérialisés dans un flot binaire. Chaque proxy stocke l'identifiant du graphe et la position de l'objet qu'il représente dans le flot binaire. Ce flot binaire peut être transféré en mémoire secondaire comme dans un fichier (e. g. 42.swap) par exemple.

3. Tous les objets du graphe sont remplacés par leur *proxy* en parcourant la table `proxiesDict`. Cette étape nécessite que le langage d'implémentation supporte cette opération de remplacement. C'est notamment le cas de Smalltalk grâce à sa primitive `become:`.

4. Cette étape consiste à remplir une table appelée `GraphTable` qui pour chaque identifiant de graphe maintient la liste (avec des références faibles) des *proxies* associés à ce graphe. Lors du chargement du graphe, cette table permet de retrouver tous les *proxies* et donc de les remplacer correctement par les objets matérialisés qu'ils représentent.

5. Enfin, la dernière étape de ce processus de déchargement d'un graphe consiste à supprimer les structures temporaires comme `proxiesDict`. Cette suppression entraine la libération des objets du graphe (e. g. A, B et C) par le ramasse-miettes, car ils ne sont plus référencés. De même, les *proxies* représentant les objets internes ont été libérés, car uniquement faiblement référencés depuis la liste des *proxies* contenues dans `GraphTable`. Cette liste contient maintenant des emplacements vides (contenant la valeur `nil`). Le résultat final montre que seuls les *proxies*

pour les objets racines (e. g. A) et les objets façades (objets référencés directement par un objet externe au graphe) ont survécu au passage du ramasse-miettes.

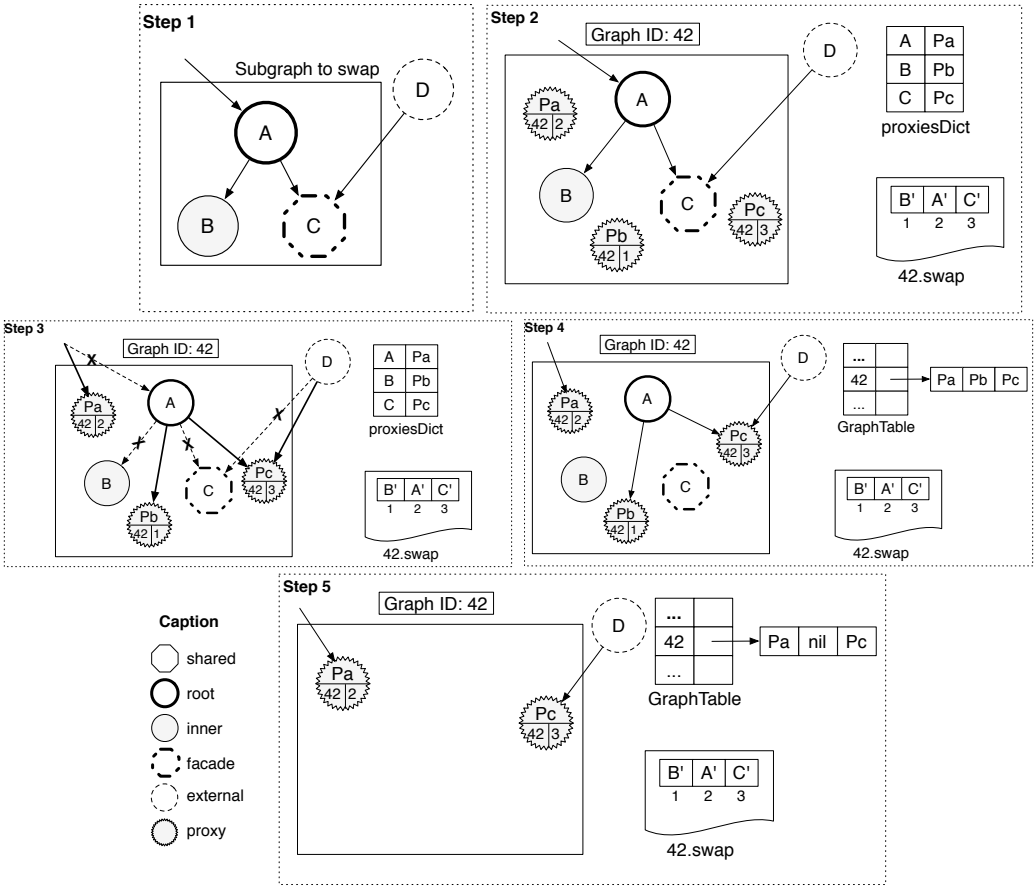


FIGURE 15 : Étapes du déchargement d'un graphe d'objets avec Marea.

ALGORITHME DE CHARGEMENT D'UN GRAPHE. Le chargement d'un graphe est automatiquement déclenché lorsqu'un *proxy* est accédé (e. g. envoi de message, accès à une variable d'instance), il s'agit d'un défaut d'objet. À partir du contexte qui a déclenché le chargement et du *proxy*, Marea peut charger le graphe correspondant et relancer l'exécution de façon transparente pour l'application en cours d'exécution. Les principales étapes du chargement d'un graphe sont illustrées sur la figure 16 :

1. La première étape consiste à matérialiser les objets appartenant au graphe en mémoire primaire (e. g. A', B' et C'). Attention, certains objets ne doivent pas être créés comme `true`, `false`, `nil` ou encore `Transcript`. Pour traiter ces cas spéciaux, Marea utilise des `hooks` avant et après la sérialisation et la matérialisation des objets. Ensuite, la table `proxiesDict` peut être reconstruite à partir de la liste des *proxies vivants* contenue

dans `GraphTable` et de l'index des objets référencés dans chaque proxy (e. g. Pa est associé au 2e objet dans le flot binaire qui est le nouvel objet matérialisé A'). Certains objets comme B' ne sont associés à aucun *proxy*.

2. Chaque *proxy* est remplacé par l'objet nouvellement matérialisé qui lui correspond. Lorsque la table `proxiesDict` est libérée, tous les *proxies* sont collectés par le ramasse-miettes. L'entrée associée au graphe chargé est supprimée de `GraphTable` de même que sa sauvegarde en mémoire secondaire si nécessaire.

3. La dernière étape montre que le graphe chargé (reconstruit) correspond bien au graphe original.

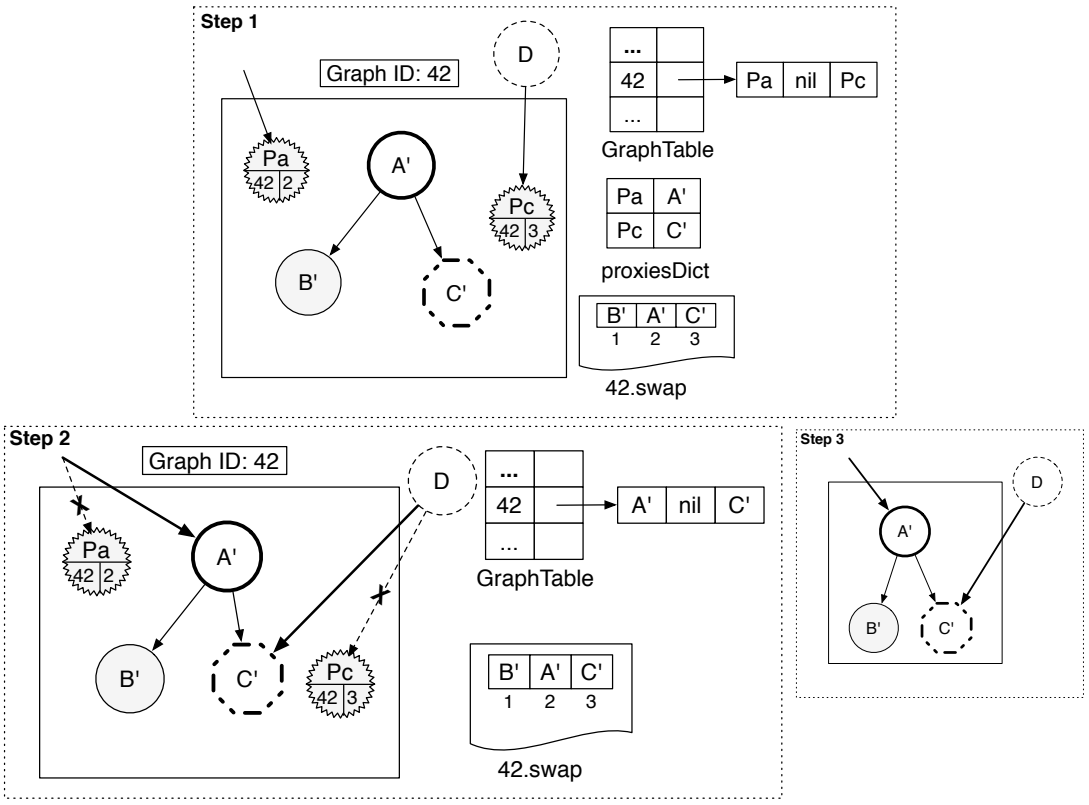


FIGURE 16 : Étapes du chargement d'un graphe d'objets avec Marea.

INTERSECTION DE GRAPHES. Décharger un graphe contenant des *proxies* suite au déchargement d'un autre graphe est une situation inévitable. Par exemple, vouloir décharger le graphe de racine D (e. g. id 43) à partir de la situation représentée sur l'étape 5 de la figure 15. Cela montre que `Pc` devient un proxy partagé entre les graphes 42 et 43. Les deux algorithmes présentés ci-dessus ont été étendus pour permettre à Marea de gérer ces cas.

DÉCHARGEMENT AVEC *proxies* PARTAGÉS. Seuls les objets standards (i. e. pas les *proxies*) sont remplacés par des *proxies*. Il ne peut pas y avoir un *proxy* pour un *proxy*. Les *proxies* appartenant à un graphe sont insérés dans une table `SharedProxiesTable` dont les clés sont des `proxyID` et les valeurs des références (fortes) vers un *proxy*. Un `proxyID` est un identifiant unique construit à partir d'un identifiant de graphe et la position de l'objet *proxy* dans le flot sérialisé de ce graphe. La figure 17 montre le résultat du déchargement du graphe de racine A (`graphID` 42) suivi du déchargement du graphe de racine D (`graphID` 43). On remarque que `Pc` est présent à la fois dans `GraphTable` et dans `SharedProxiesTable`.

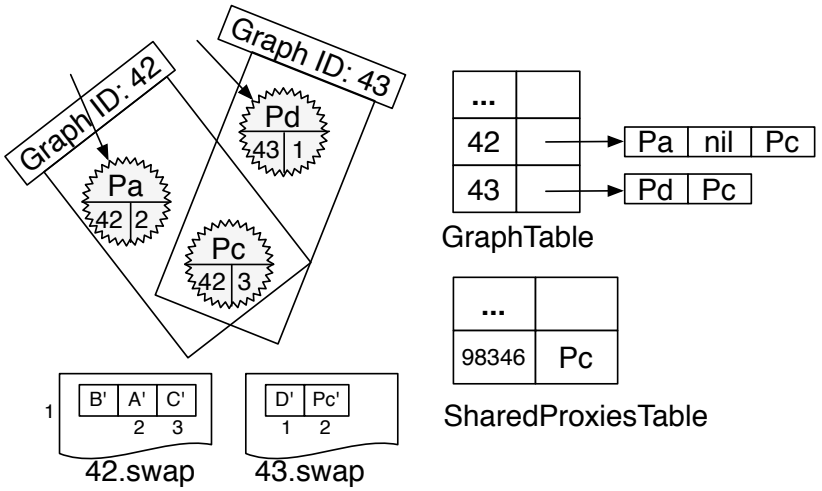


FIGURE 17 : Déchargement de graphes d'objets avec proxies partagés.

CHARGEMENT AVEC *proxies* PARTAGÉS. L'ordre de chargement des graphes n'est pas forcément le même que celui qui a été utilisé pour leur déchargement. L'algorithme de chargement d'un graphe précédemment décrit ne diffère que pour les *proxies* partagés présents parmi les objets matérialisés. Ces *proxies* ne sont pas créés mais directement remplacés par l'objet existant qui leur est associé dans la table `SharedProxiesTable` qui est retrouvé grâce de l'identifiant `proxyID`. Suivant l'ordre de chargement des graphes, `SharedProxiesTable` contient un *proxy* (Pc) ou un objet matérialisé (C′) qui a remplacé ce *proxy*. Dans tous les cas, les graphes sont correctement reconstruits tout en préservant les objets partagés.

ÉVALUATION. À travers Marea et son implémentation, nous avons proposé un mécanisme de déchargement des objets inutilisés pendant l'exécution d'un programme qui sont ensuite automatiquement rechargés si besoin. Nous avons validé cette approche à travers différentes expérimentations sur des applications réelles [PBD+13]. Nos résultats montrent une réduction de la mémoire consommée comprise entre 25% et 40% suivant les applications.

Il reste tout de même une marge d'amélioration conséquente puisque notre étude montrait un potentiel compris entre 63% et 73%. L'impact sur les performances de l'application est principalement lié à l'utilisation de la primitive de remplacement des objets. Par exemple, décharger ou recharger un graphe contenant entre 1 et 1000 objets prend en moyenne 40ms. Bien que cette baisse de performance reste tout à fait acceptable pour de nombreux domaines d'applications, cela peut être un frein pour certaines applications de robotique mobile. L'évaluation de Marea montre que tous les critères initiaux (cf. section 4.2.2) ont été satisfaits hormis le déchargement automatique qui nécessite de pouvoir détecter automatiquement (au bout de combien de temps un objet non accédé doit-il être considéré comme inutile ?) et efficacement (comment tracer l'historique de l'activité des objets ?) les objets non-utilisés pendant l'exécution d'une application.

4.3 TRAVAUX EN COURS ET PERSPECTIVES

INTERFACE DE FONCTIONS ÉTRANGÈRES (FFI) HAUT NIVEAU. Nous travaillons également sur l'optimisation de la consommation CPU des applications s'exécutant sur un robot mobile. Notre idée est d'utiliser des langages réflexifs pour fournir au développeur un niveau d'abstraction et des outils de haut niveau. Toutefois, ces langages ne permettent pas dans certains cas d'atteindre des performances acceptables comme traiter les images ou les vidéos provenant des caméras embarquées sur le robot avec un débit suffisant par rapport à la vitesse de déplacement du robot. Pour cela, nous nous sommes intéressés à l'interaction entre les langages haut niveau et les bibliothèques natives i. e. les interfaces de fonctions étrangères ou *Foreign Function Interface* (FFI) en Smalltalk [BFDS13]. L'idée consiste à appeler depuis le langage haut niveau des fonctions situées dans des bibliothèques compilées nativement. Les FFI se doivent d'être efficaces en ce qui concerne le *marshalling* i. e. la conversion des types de données entre le langage haut niveau (e. g. des objets Smalltalk) et le code natif. Mais il est également important de pouvoir facilement ajouter de nouvelles bibliothèques.

VERS DES VMS HAUT NIVEAU. Une FFI efficace permet d'imaginer que certains mécanismes normalement enfouis dans le code de la VM soient « remontés » au niveau applicatif sans dégrader trop fortement les performances. Cette perspective permet d'envisager des VMs plus réduites et donc mieux adaptées aux contraintes en mémoire [BF10] sans pour autant perdre en portabilité. Ensuite, les mécanismes remontés au niveau applicatif deviennent des objets manipulables sur lesquels nos techniques de réduction de la consommation mémoire sont tout à fait applicables.

DÉPORT DE CALCULS VERS DES ARCHITECTURES SPÉCIALISÉES. Je co-encadre la thèse de Xuan Sang Le qui vise à simplifier l'écriture d'applications où certains calculs peuvent ensuite être déportés vers des architectures spécialisées comme des FPGAs afin de décharger le processeur central d'un robot mobile. L'idée est d'implémenter une architecture de contrôle d'un robot mobile en Pharo et de pouvoir « automatiquement » projeter certaines parties de cette architecture sur des FPGAs afin de décharger le processeur central du robot de certaines tâches coûteuses comme le traitement d'image par exemple [LLB+14]. Actuellement, nous disposons d'une chaîne permettant de décrire des circuits FPGA en Pharo [LLF+15]. Ces circuits peuvent être simulés au-dessus de la VM Pharo ou déployés automatiquement sur une carte FPGA et appelés depuis Pharo comme dans le cas d'une FFI. Nous travaillons maintenant à la généralisation de cette technique pour du code Smalltalk.

4.4 SYNTHÈSE DES CONTRIBUTIONS

Le tableau ci-dessous synthétise mon activité dans ce thème à travers la co-signature d'articles scientifiques, le co-encadrement de thèses ou encore la participation à des projets scientifiques. L'annexe A contient plus de détails concernant ces éléments.

3	Revues internationales	[MPBDF12, PBD+13, PBF+14]
4	Conférences internationales	[MPBD+10b, MPBD+10a, BF10, PBD+11b]
4	Ateliers internationaux	[PBD+11a, BFDS13, LLB+14, LLF+15]
1	Rapports internes	[PDBF14]
2	Thèses co-encadrées soutenues	[MP12, Pol15]
1	Thèse co-encadrée en cours	Xuan Sang Le
1	Projet de recherche	Projet CARNOT « JIT pour FPGA »
2	Collaborations scientifiques	INRIA RMoD, ENSTA-Bretagne Lab-STICC

PRIX ET RÉCOMPENSES

- L'article [LLF+15] a reçu le 1er prix des « *Best Papers Award* » de l'atelier international IWST'15 lors de la conférence ESUG'15.

5 INFRASTRUCTURES ET OUTILS POUR LE DÉVELOPPEMENT D'APPLICATIONS ROBOTIQUES

> The most exciting phrase to hear in science, the one that heralds new discoveries, is not 'Eureka!' (I found it!) but rather, 'hmm.... that's funny...'
>
> Isaac Asimov

Ce chapitre présente les travaux que j'ai menés ou auxquels j'ai participé sur l'amélioration des outils et des infrastructures pour le développement d'applications de robotique de services. Ce domaine constitue le cadre expérimental privilégié de mes travaux. Par exemple, dans le cadre du projet « CAIRE » (cf. Annexe A, page 147) dont j'ai été coordinateur, nous avons proposé de construire automatiquement de cartes de rayonnements (WiFi, électromagnétiques, . . .) par un robot mobile équipé des capteurs adéquats. Dans le cadre du projet « RoboShop » (cf. Annexe A, page 148), nous avons proposé différents scénarios d'usage d'un ou plusieurs robots dans le cadre du commerce. Ce projet s'est terminé par la démonstration d'un prototype opérationnel de robot guide en magasin sur le stand New Shopping Experience 3 du Pôle de Compétitivité du Commerce (PICOM) lors du salon VAD conext à Lille Grand Palais[1]. Ce chapitre décrit les travaux menés dans le cadre de ces projets à travers le co-encadrement de doctorants et de post-docs. Dans la thèse de Nick Papoulias [Pap13], nous avons proposé une infrastructure pour le débogage à distance d'applications s'exécutant sur des CPS mobiles. Durant le post-doc de Zhi Yan, nous avons travaillé sur une infrastructure de type banc d'essai pour évaluer et comparer des applications multi-robots. Cette infrastructure repose sur la simulation réaliste de ces applications par une grappe de machines en exécutant le même code que celui déployé sur les robots réels.

1 http://car.mines-douai.fr/2013/11/roboshop-demo-16oct2013/

5.1 DÉBOGAGE À DISTANCE

5.1.1 Motivation

Le test et le débogage sont des étapes du cycle de développement d'une application estimées jusqu'à 50% du temps total de développement [Bei90b]. Mais ces étapes couvrent aussi 50% à 75% du coût total de développement [BH02].

À travers la figure 1 (cf. chapitre 1), nous avons présenté le cycle de développement classique d'une application pour un CPS mobile. Il s'agit d'un développement généralement distant, car l'équipement cible ne permet pas l'installation d'un environnement de développement ou est dépourvu de certains périphériques d'entrées/sortie (écran, clavier). Ce mode de développement à distance complexifie encore les étapes de débogage et de tests. En effet, les phases de re-compilation et de re-déploiement systématiques de l'application pour chaque changement allongent le temps nécessaire au débogage. Des études empiriques ont permis d'estimer à 10,5 minutes par heure de développement comme étant le temps moyen consacré au re-déploiement [Zer11]. Raccourcir ces phases de re-compilation et de re-déploiement lors du débogage d'une application distante est donc prometteur pour réduire également le temps et les coûts de développement.

L'une des techniques les plus utilisées pour le débogage distant repose sur l'analyse *post-mortem* de journaux (traces d'exécution) pour identifier les causes d'une erreur survenue durant l'exécution [Zel05]. Le succès de cette analyse dépend de la verbosité des traces. Mais, plus les traces sont verbeuses et plus l'analyse est difficile à cause du volume de données. De plus, pour déterminer précisément les causes d'une erreur, il est souvent nécessaire d'affiner les traces produites à l'exécution ce qui nécessite une modification du code source, une re-compilation et le re-déploiement de l'application. Il existe toutefois des outils de débogage à distance permettant de s'affranchir de cette analyse des traces. Mais les solutions existantes ne permettent pas de profiter des mêmes mécanismes de débogage avancés que dans le cas d'un développement en local. Le fait que l'application et le débogueur soient distants rend difficile la mise en place de solutions avancées permettant par exemple d'analyser les interactions entre les objets durant l'exécution (e. g. *objet-centric debugging* [RBN12]). Des études empiriques [SMDV06] ont pourtant montré que l'analyse dynamique des programmes est plus utile au programmeur que la possibilité d'examiner l'exécution en spécifiant des points d'arrêts sur des lignes de code.

En résumé, le travail présenté dans la suite de cette section est motivé par la nécessité de proposer une architecture permettant le débogage interactif d'applications s'exécutant sur des équipements contraints (cf. figure 18). Cette architecture se doit d'être légère et minimale (en l'absence d'erreur) du côté de l'application tout en permettant à un débogueur distant d'inspecter et modifier cette même application en cours d'exécution si une erreur survient.

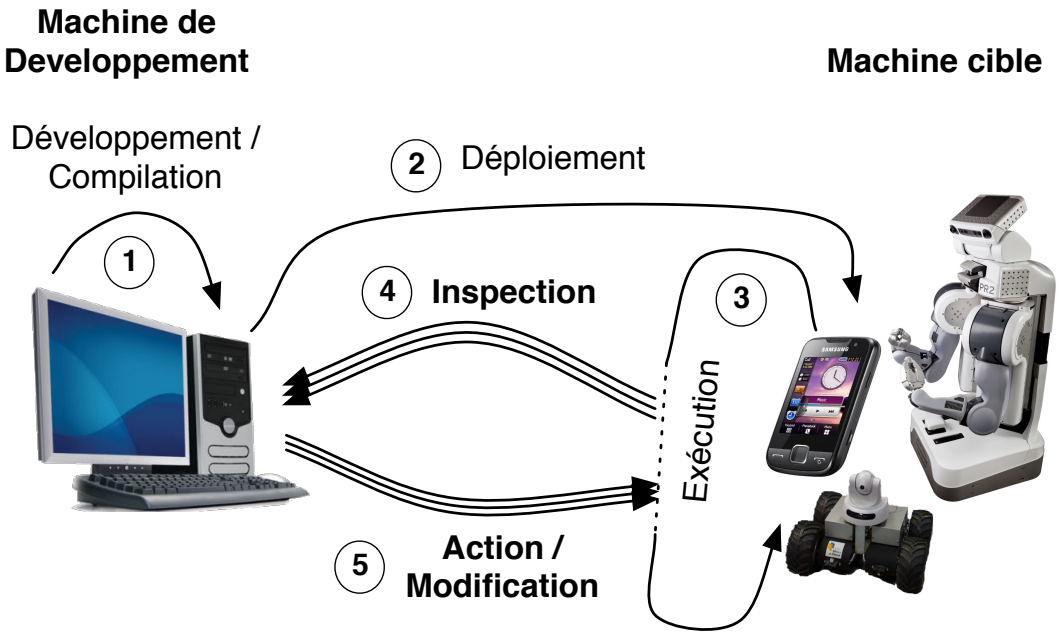


FIGURE 18 : Idée générale du débogage interactif à distance.

Pour cela, l'exécution de l'application déboguée doit être *interruptible* lorsque certains événements surviennent durant l'exécution comme la levée d'une exception par exemple. Lors d'une interruption, l'exécution de l'application est suspendue et les étapes 4 et 5 sur la figure 1 permettent au développeur d'inspecter et de modifier (e. g. *incremental updates*) l'application à distance directement dans le contexte d'exécution où l'erreur est survenue. Ce mode de débogage à distance sans perdre le contexte d'exécution de l'erreur est réel un avantage par rapport à l'analyse de traces.

5.1.2 État de l'art

DÉBOGAGE À DISTANCE. Le *débogage* est un processus en deux étapes. Dans un premier temps, le programmeur doit associer un échec qu'il observe lors de l'exécution d'un programme à un défaut dans son code source [Bei90a, Hum99]. Dans un second temps, le programmeur corrige ce défaut en appliquant une modification dans le code source et valide cette modification à travers le succès qu'il observe lors de l'exécution du programme ainsi corrigé. Un *débogueur* est un processus externe à celui de l'application déboguée et est utilisé pour analyser l'exécution de cette dernière. Le *débogage à distance* est un processus de débogage où le processus du débogueur s'exécute sur une machine différente de celle où s'exécute le processus de l'application déboguée.

La figure 19 présente une vue abstraite de l'architecture logicielle nécessaire au débogage à distance. La machine cible (un robot mobile par exemple) exécute l'application à déboguer. Elle doit intégrer un intergiciel pour la ges-

tion des communications distantes ainsi qu'un support permettant d'examiner les processus, leurs piles d'exécution, les variables d'instances des objets ou encore les variables locales. À distance, la machine de développement doit également intégrer un intergiciel, mais aussi les outils de débogage (inspecteurs, débogueur) et surtout un modèle qui décrit l'application distante déboguée (e. g. code source, points d'arrêts, ...).

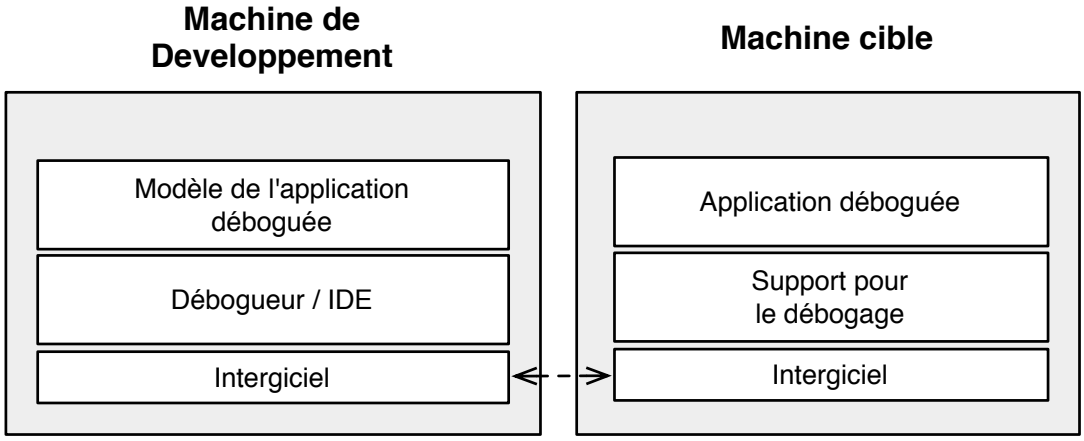


FIGURE 19 : Architecture logicielle macroscopique pour le débogage à distance.

CRITÈRES DE COMPARAISON. Afin d'évaluer et de comparer différentes solutions pour le débogage à distance, nous avons identifié trois propriétés importantes :

INTERACTIVITÉ. Ce besoin consiste à permettre au développeur de pouvoir inspecter et modifier dynamiquement le code ou l'état de l'application déboguée pendant son exécution sur l'équipement cible. Pendant le débogage d'une application à objets, le développeur doit pouvoir ajouter, modifier ou enlever des packages, des classes, des méthodes et des variables d'instance ou encore modifier la hiérarchie des classes. Pour remplir pleinement ce besoin, l'application cible doit pouvoir être déboguée sans être re-démarrée afin de conserver le contexte d'exécution dans lequel l'application a été interrompue. De plus, le modèle de l'application déboguée sur la machine de développement doit toujours refléter l'état courant de l'application en cours d'exécution sur la machine cible (cf. figure 19).

INSTRUMENTATION. Il s'agit de la capacité de la solution de débogage à altérer la sémantique de l'exécution d'une application pour supporter son débogage comme la mise en place de points d'arrêts par exemple. La solution de débogage instrumente le processus d'exécution de l'application déboguée pour qu'il s'interrompe lorsque certains événements surviennent (comme lors de l'accès à une variable d'instance) afin de passer le contrôle au processus de débogage. Plusieurs travaux ont recensé et

catégorisé tous les événements nécessaires pour déboguer une application à objets [McA95, RRGN10] : exécution d'une instruction spécifique ou d'une méthode, création d'un objet, accès en lecture ou en écriture à l'un des champs d'un objet, tous les envois de message depuis un objet ou reçus par un objet, à chaque fois qu'un objet est reçu en paramètre, *etc*.

DISTRIBUTION. Le débogage à distance requiert une infrastructure de communication. Toutefois, proposer une solution générique est difficile, car il faut prendre en compte les contraintes liées aux nombreux équipements cibles (smartphone, robot mobile, serveur dans le nuage, ...) en terme de ressources (capacité de calcul, mémoire, bande passante) ou de besoins spécifiques comme la sécurité ou un format de sérialisation donné. Les intergiciels adaptables [KCBC02] sont une réponse possible à ces besoins, car ils offrent une flexibilité de configuration durant l'exécution [RKC01] dont pourrait bénéficier une solution de débogage à distance. C'est ainsi que nous avons défini 4 différents niveaux d'adaptabilité de la distribution : pas de distribution, intergiciel figé (i. e. protocole de communication dédié et figé), intergiciel extensible (comme CORBA ou DCOM qui proposent une solution générale pour l'informatique répartie) et enfin intergiciel adaptable (i. e. un intergiciel extensible qui permet de s'adapter dynamiquement [DL02]).

SOLUTIONS EXISTANTES. Nous avons étudié différentes solutions existantes permettant le débogage d'applications à objets au sens large mais également des solutions spécifiques aux CPS mobiles comme l'utilisation d'émulateurs ou de simulateurs :

ÉMULATEURS ET SIMULATEURS. L'utilisation d'un simulateur de smartphone ou de robot comme Morse [ELD+12a] ou Gazebo [KH04a] permet d'exécuter l'application cible en local sur la machine de développement. Cette solution offre l'avantage de s'affranchir des contraintes liées à la distance. Toutefois, il n'existe pas forcément d'émulateur spécifique pour le système cible visé qui simule efficacement les capteurs et actionneurs désirés. Cette solution pose également le problème de la prise en compte des contraintes en ressources des équipements finaux. De plus, même si le simulateur s'exécute sur la machine développement, il ne propose pas forcément d'interface pour le débogage de l'application en cours de simulation.

GDB. Gnu-debugger [RS03] permet le débogage à distance. Pour cela, il utilise un processus séparé sur la machine cible appelé *gdb-server* auquel il attache les processus en cours d'exécution. Pour permettre le débogage, l'application doit être compilée, déployée et exécutée spécifiquement.

Gdb offre un support limité aux méthodes existantes (dans le cas de code écrit en ObjC) en ce qui concerne la modification de l'application en cours d'exécution.

JPDA/JVM. JPDA (*Java Platform Debugger Architecture*) [Ora13] est l'architecture Java permettant le débogage à distance de programmes s'exécutant au-dessus de la JVM (*Java Virtual Machine*). L'application à déboguer doit obligatoirement s'exécuter sur une JVM fournissant l'interface native JVMTI. Cette interface de bas niveau (au niveau de la VM) offre des opérations permettant d'accéder à l'état de la machine virtuelle en cours d'exécution, de suspendre, d'activer ou d'examiner la pile des *threads*, de placer des points d'arrêts, de modifier une variable temporaire ou un paramètre, d'envoyer des messages aux objets ou encore d'accéder en lecture ou en écriture à leurs champs. Par contre, il n'est pas possible de modifier une classe déjà chargée (ajout d'une variable d'instance par exemple). Finalement, un débogueur peut communiquer avec la JVM distante via son interface JVM TI grâce au protocole JDWP. La plupart des langages s'exécutant au-dessus de la JVM (comme Groovy ou Scala) reposent sur JDI (*Java Debugging Interface*) qui l'implémentation Java du protocole JDWP.

.NET. L'infrastructure de débogage de .NET [Mic12b] repose également sur une machine virtuelle modifiée. Cela permet au débogueur de s'attacher à un processus distant en cours d'exécution sans perdre le contexte d'exécution. De même qu'en Java, il est possible de remplacer des méthodes existantes. Par contre, cette infrastructure ne permet pas la mise à jour incrémentale de code avec de nouveaux packages ou de nouvelles méthodes.

JREBEL ET DCE. DCE [WWS10] et JRebel [Zer12] sont des modifications de la JVM permettant de modifier les classes Java déjà chargées. Couplées à JPDA, ces extensions de la JVM constituent une solution pour le débogage à distance supportant la mise à jour incrémentale de code.

MAXINE. L'inspecteur de Maxine [WHV+12] permet le débogage de la machine virtuelle Java écrite en Java nommée Maxine. Cet outil est spécialisé dans le débogage de machines virtuelles méta-circulaires. En plus des points d'arrêts classiques sur le code source et les objets, le développeur peut aussi mettre des points d'arrêts sur des éléments internes à la VM comme des événements liés au ramasse-miettes par exemple.

TOD. TOD [PTP07] est un débogueur omniscient pour le langage Java. Il permet de ne pas interrompre l'exécution du programme tout en permettant ensuite au développeur de « remonter dans le temps » et de voir l'état des objets à différents points de l'exécution. Le débogage omniscient n'est pas lié à notre travail. Par contre, TOD repose sur une archi-

tecture originale pour l'instrumentation qui se base sur des événements comme l'interception de messages ou l'accès à des champs.

SMALLTALK. Smalltalk propose un support pour le débogage qui permet de ne pas perdre le contexte d'exécution [LP90] mais surtout de bénéficier de nombreuses réifications comme les contextes ou les processus. Smalltalk permet également la mise à jour incrémentale de code en ajoutant des classes, des méthodes ou en modifiant des éléments existants. Les capacités de débogage offertes par Smalltalk reposent en grande partie ses capacités réflexives. Dans le cas du débogage à distance, cela pose le problème de la réflexion à distance (cf. section 3.1).

BIFROST. Le débogage centré sur les objets [RBN12] propose des techniques avancées de débogage grâce aux possibilités d'instrumentation offertes par le framework Bifrost [RRGN10] qui est une extension du MOP Smalltalk. Par exemple, Bifrost permet d'intercepter explicitement l'accès à une variable d'instance d'un objet en particulier.

JS/RIVET [Mic12a] Les outils de débogage pour les applications Javascript s'exécutant dans les navigateurs comme IE, Firefox ou encore Chrome fournissent la possibilité de contrôler l'exécution et de spécifier des points d'arrêts. Toutefois, ils sont également limités en ce qui concerne la mise à jour incrémentale de code qui se limite aux slots et aux méthodes existantes. La solution la plus proche de notre objectif est Rivet [Mic12a] qui permet le débogage d'applications Javascript côté serveur. Rivet supporte la mise à jour incrémentale de code et propose une infrastructure de débogage à distance basée sur HTTP.

AMBIENTTALK. AmbientTalk est un langage dédié au développement d'applications pour CPS mobiles. Le modèle de réflexion utilisé par AmbientTalk supporte les mirages [MCTT07] qui sont une extension des miroirs pour supporter la réflexion implicite. Cette extension permet de supporter l'instrumentation d'événements tels que l'instanciation d'objets par exemple. AmbientTalk propose aussi un support à la distribution permettant de s'adapter dynamiquement car il a été spécifiquement conçu pour gérer les réseaux mobiles *ad hoc*.

ÉVALUATION. Le tableau 6 synthétise notre évaluation [Pap13] des approches présentées ci-dessus avec les besoins que nous avons identifiés pour la construction d'une solution de débogage à distance. Cette comparaison met en exergue que les solutions classiques comme JPDA, .NET ou Gdb et leurs extensions comme JRebel et DCE sont incomplètes en ce qui concerne l'interactivité et les possibilités d'instrumentation. Bifrost (mais dans un contexte de débogage local) offre un support complet de ces deux propriétés mais il lui manque un support pour la distribution comme c'est le cas en AmbientTalk par exemple.

	Interactivité	Instrumentation	Distribution
GDB	∼	∼	∼
JPDA/JVM	∼	∼	∼
.NET	∼	∼	∼
JRebel/DCE	+	∼	∼
Maxine	∼	∼	∼
TOD	∼	∼	∼
Smalltalk	+	∼	–
Bifrost	+	+	–
JS/Rivet	+	∼	∼
AmbientTalk	∼	∼	+

TABLE 6 : Évaluation des solutions de débogage à distance [Pap13] (+ indique que le critère est satisfait, – qu'il ne l'est pas et ∼ qu'il l'est partiellement).

5.1.3 Mercury : une infrastructure de débogage à distance

Mercury est notre proposition d'architecture pour la construction d'une solution de débogage à distance [PBF+15]. Cette architecture repose sur trois caractéristiques principales (cf. figure 20) : la réflexion distante (point 1), l'instrumentation par l'intercession (point 2) et un support adaptatif de la distribution (point 3).

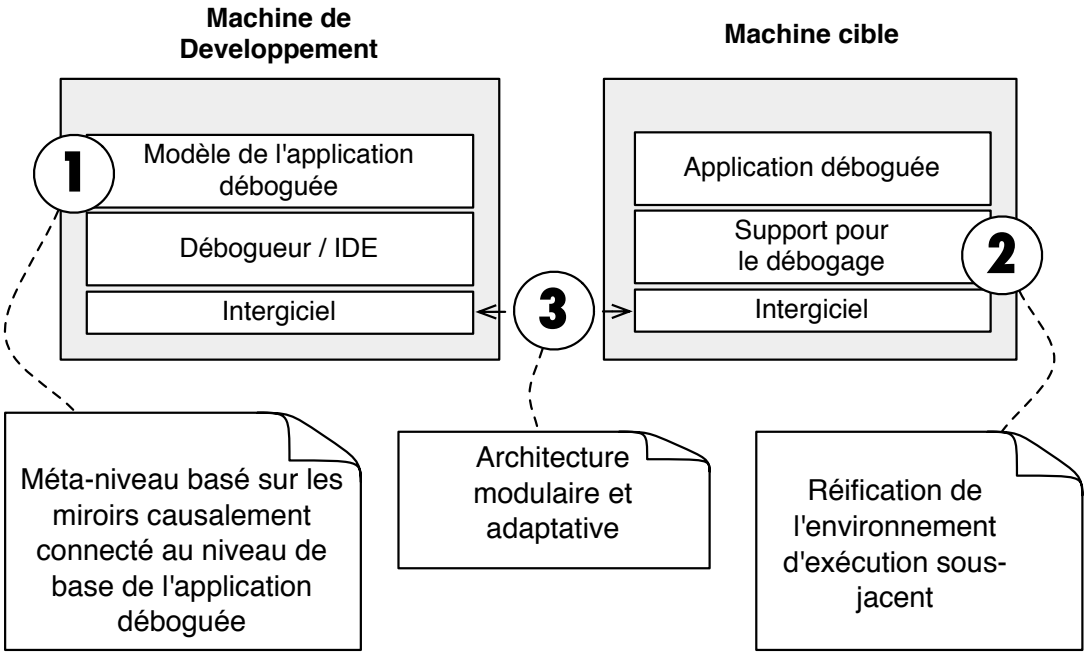


FIGURE 20 : Architecture générale de Mercury.

RÉFLEXION DISTANTE. L'évaluation de l'état a montré que la réflexion permet de supporter pleinement l'interactivité et l'instrumentation. Notre solu-

tion propose donc que le méta-niveau de l'application déboguée s'exécute sur la machine de développement localement aux outils de débogage. Bien évidemment, ce méta-niveau représente et agit sur le niveau de base de l'application qui s'exécute sur la machine cible. Ce découplage entre le méta-niveau et le niveau de base a été spécifiquement étudié dans la section 3.1. En résumé, les miroirs (point 1) fournissent une API réflexive et contiennent les méta-données de haut niveau nécessaires au débogage l'application à distance. Lorsque le débogage est en cours, les miroirs communiquent avec un objet façade sur la machine cible qui est dédié au support pour le débogage (point 2). La réflexion impose de maintenir le lien causal entre le méta-niveau (les miroirs) et le niveau de base (l'application déboguée). Cela implique que tout changement dans l'un des deux niveaux doit être répercuté dans l'autre niveau. Par exemple, si le développeur ajoute un package, une classe ou une méthode à travers un miroir sur la machine de développement, cela doit se traduire automatiquement par une mise à jour dynamique et incrémentale de l'application en cours d'exécution sur la machine cible. Pour cela, nous avons défini une hiérarchie de miroirs (e. g. `ObjectMiror`, `EnvironmentMiror`, `ClassMiror`, `ProcessMirror`) qui fournissent les méta-opérations nécessaires (e. g. `setClass:`, `newPackageNamed:`, `addInstVarName:`).

INSTRUMENTATION PAR INTERCESSION. Assurer la connexion causale nécessite de détecter les changements dans le niveau de base i. e. l'application déboguée du côté de la machine cible. Par exemple, lorsqu'un nouveau processus est créé, il faut qu'un miroir instance de `ProcessMiror` soit automatiquement créé du côté de la machine de développement. Pour détecter les changements, le support pour le débogage (point 2) doit permettre d'instrumenter l'exécution de l'application dynamiquement. Plutôt qu'une approche bas niveau basée sur une extension de la VM, nous utilisons l'intercession et notamment la réflexion implicite grâce la réification du réflectogramme (cf. section 3.1).

SUPPORT ADAPTATIF POUR LA DISTRIBUTION. Mercury repose sur une solution de distribution spécifiquement conçue pour le débogage. Cette solution s'inspire fortement des intergiciels adaptables et repose sur le schéma de conception fabrique abstraite [GHJV95b] qui permet d'instancier, de paramétrer et d'assembler les objets dynamiquement en fonction du contexte.

ÉVALUATION. Mercury supporte pleinement les trois propriétés : interactivité, instrumentation et distribution. Pour cela, des idées issues de différentes approches comme AmbientTalk, Bifrost et MetaTalk ont été combinées. Un prototype de Mercury a été implémenté et a permis d'en valider l'approche. Ce prototype n'a aucun impact sur les performances de l'application tant qu'une erreur ou un point d'arrêt n'est pas atteint. Pendant le débogage, l'instrumentation induit inévitablement un surcoût que nous avons quantifié

à 8x par rapport à l'exécution sans instrumentation. Mais les performances peuvent également être limitées par le réseau de communication. Une instrumentation bas niveau (VM) aurait permis de meilleures performances ainsi que la possibilité de déboguer la VM elle-même comme c'est le cas en Maxine mais au prix de l'extensibilité de la solution.

5.2 BANC D'ESSAI POUR APPLICATIONS MULTI-ROBOTS

5.2.1 Motivation

De nombreuses applications robotiques peuvent bénéficier de l'utilisation d'une flotte de plusieurs robots au lieu de compter sur un seul robot [Par08]. En effet, avoir plusieurs robots signifie une augmentation de robustesse grâce à la redondance. En outre, plusieurs robots peuvent effectuer des tâches en parallèle et donc accélérer l'exécution et l'efficacité de certaines applications comme la recherche de survivants après une catastrophe (e. g. tremblement de terre), la recherche de foyer incendiaire dans des bâtiments, l'exploration minière ou encore le déminage.

Cependant, l'utilisation de systèmes multi-robots soulève le défi de la coordination [YJC13]. Pour vraiment profiter du parallélisme potentiel d'une flotte robotique, il faut des stratégies pour organiser l'activité des robots qui garantissent des performances accrues. À titre d'exemple, considérons l'exploration d'un environnement inconnu [Sta09] qui est une tâche commune à de nombreuses applications. Une stratégie de coordination doit attribuer à chaque robot un ensemble de zones à explorer de façon à réduire à la fois le temps nécessaire pour construire une carte complète de l'environnement ainsi que l'énergie totale consommée par la flotte robotique. Malheureusement, l'élaboration de stratégies optimales ou quasi-optimales pour la coordination est complexe. C'est pourquoi la communauté de recherche fait des efforts considérables pour traiter ce problème [BMF+00, DBD+09, How06, Sta09, Yam98a, ZSDT02].

L'abondance des algorithmes d'exploration multi-robots est en soit un problème quand on doit choisir le plus approprié à utiliser pour une situation donnée. Dans la littérature, les auteurs évaluent leurs solutions avec différents robots, différents simulateurs, dans différents environnements et sous différentes conditions. Par conséquent, les résultats présentés ne sont pas toujours comparables. Une évaluation mathématique des algorithmes - comme l'analyse de la complexité - est difficile à cause de la complexité des systèmes multi-robots et de leurs environnements. Il y a trop de paramètres qui peuvent influer sur les performances. Des exemples de ces paramètres sont :

le nombre de robots, la puissance de calcul disponible, la mémoire sur chaque robot, le fait que la flotte soit homogène ou hétérogène.

Une approche alternative consiste à faire une évaluation empirique basée sur le *benchmarking* (évaluation par essais) des algorithmes[2] d'exploration multi-robots [BPM14, Pob06]. Un *benchmark* doit être *reproductible* afin que les chercheurs puissent comparer leur approche avec celles existantes. La reproductibilité implique l'absence d'hypothèses cachées et assure la possibilité de reproduire l'expérimentation. Le réseau européen de la recherche en robotique (EURON) a attiré l'attention de la communauté robotique sur ce besoin en 2006[3]. Toutefois, aujourd'hui encore, il existe peu de bancs d'essais réutilisables pour évaluer une application multi-robots. Les compétitions robotiques comme le DARPA Challenge ou la Robocup permettent dans une certaine mesure de comparer différentes approches [AJO11]. Toutefois, participer à ce genre de compétition reste un effort coûteux qui nécessite une intégration complète d'un système robotique.

Simuler des applications multi-robots permet de facilement répéter une expérimentation avec différentes stratégies de coordination par exemple. La simulation doit être le plus réaliste possible, car l'objectif est de comparer une application multi-robots réelle et non pas une version simplifiée. Par exemple, simuler de nombreux robots sur le même ordinateur peut cacher le fait que les communications entre les robots consomment plus de bande passante qu'il ne serait possible en pratique. Dans tous les cas, les simulations sont importantes avant d'effectuer des tests sur des robots réels. De nombreux travaux proposent donc de simuler le plus précisément possible les robots (capteurs, actionneurs) et leur environnement comme Webots [GVH03], V-REP [FSOM10], Gazebo [KH04b] ou encore MORSE [ELD+12b]. L'utilisation d'un simulateur n'est toutefois pas suffisante pour faire un *benchmark* et vraiment comparer quantitativement des applications robotiques. Pour cela, il est nécessaire de définir des *métriques* représentatives et objectives permettant de mesurer la performance de chacune des expérimentations. Il faut également que l'infrastructure de simulation permette de collecter toutes les données nécessaires au calcul des métriques définies.

En résumé, ce travail est motivé par le besoin de comparer objectivement différentes approches pour la coordination multi-robots notamment dans le cadre de missions d'exploration.

5.2.2 État de l'art

Cette section décrit l'existant en ce qui concerne les infrastructures et les métriques permettant le *benchmarking* d'applications multi-robots. Cet état de l'art se focalise sur l'exploration multi-robots.

2 En fait, ce sont des implémentations de ces algorithmes qui sont effectivement comparées.
3 http://www.robot.uji.es/EURON/en/index.htm

INFRASTRUCTURES POUR LE *benchmarking* D'APPLICATIONS MULTI-ROBOTS. Peu de travaux décrivent vraiment l'infrastructure de tests qu'ils ont mise en place pour évaluer leur solution. Nous pouvons néanmoins citer :

- Un banc d'essai dédié à la validation expérimentale d'applications distribuées sur un système multi-robots [MFK08]. Ce système est construit autour : des modèles de robot Scarab et Khepri, de l'intergiciel robotique Player [GVH03] et du simulateur 3D Gazebo [KH04b]. L'une des limitations de cette approche est qu'elle repose sur une architecture centralisée qui ne reflète pas la réalité d'une flotte de robots. De plus, il est difficile de reproduire les expérimentations présentées dans ces travaux par manque d'informations sur le système proposé ainsi que sur le processus qu'ils ont utilisé pour les effectuer.

- Un environnement de simulation pour des flottes de robots reposant sur un réseau à large échelle [OaRBTS08]. Cet environnement contient jusqu'à 800 ordinateurs connectés par des commutateurs intelligents qui permettent d'expérimenter différentes topologies de réseau pour la communication inter-robot. Ce travail est très centré sur la partie réseau de communication et délaisse la partie simulation des robots et de leur environnement. De plus, cette infrastructure matérielle pointue ne peut pas être facilement reproduite.

- *Advanced ROS Network Introspection* (ARNI) [BW14] qui propose d'étendre les capacités de l'intergiciel ROS pour mieux collecter des données sur les communications réseau via le topic `/statistics`. ARNI introduit un nœud de supervision qui compare l'état courant du système avec un état de référence. L'objectif est donc d'assurer un mode de fonctionnement sûr et fiable plutôt que de faire un environnement pour le *benchmarking*. Néanmoins, ce système peut tout à fait être utilisé pour analyser les communications et collecter des mesures relatives au trafic réseau entre les nœuds ROS.

- RoboNetSim [KGC13] est un *framework* pour la simulation réaliste des communications inter-robots qui peut être intégré avec différents simulateurs. RoboNetSim supporte jusqu'à des centaines de robots. Ce découplage entre la simulation des communications et le simulateur des robots est particulièrement intéressant.

La plupart des infrastructures existantes permettant la simulation d'applications multi-robots reposent sur l'utilisation d'un seul simulateur s'exécutant sur une seule machine. Notre analyse a montré que d'autres solutions existent, mais elles sont peu décrites ou alors très spécialisées sur la partie réseau de communication.

MÉTRIQUES POUR L'ÉVALUATION DE STRATÉGIES D'EXPLORATION. Dans la littérature, chaque nouvelle stratégie d'exploration s'accompagne généralement d'une évaluation qui lui est propre. Nous présentons ci-après un aperçu des méthodes utilisées :

- Bautin *et al.* [BSC12] ont comparé leur algorithme d'affectation des frontières pour l'exploration multi-robots nommé MinPos avec deux algorithmes bien connus dans l'état de l'art (Yamauchi [Yam98a] et Burgard [BMF+00]), dans trois environnements différents. Ils ont mesuré le temps d'exploration donné par les étapes de simulation, tout en faisant varier le nombre de robots dans différentes méthodes. La comparaison a été effectuée sur deux simulateurs différents développés par eux-mêmes. L'un est discret, l'autre est plus réaliste et il permet d'utiliser exactement le même code en simulation et sur les vrais robots.

- Faigl *et al.* [FSC14] ont proposé un framework d'évaluation pour évaluer des stratégies d'exploration de façon indépendante. Ils ont comparé cinq algorithmes de répartition des tâches dans des scénarios d'exploration multi-robots. Pour cela, ils ont défini quatre environnements différents qui représentent des espaces ouverts (e. g. bureaux) et effectué des simulations discrètes. La métrique de performance utilisée dans ce travail est le nombre d'étapes de simulation nécessaires pour explorer tout l'environnement.

- Couceiro *et al.* [CVRF14b] ont présenté plusieurs expériences de simulation pour évaluer cinq algorithmes d'exploration et de cartographie multi-robots. Ils ont utilisé deux métriques de performance : 1) le « taux d'exploration » de l'environnement au fil du temps, calculé par la division de la surface de la carte construite par les robots à un instant donné par la surface totale du terrain ; 2) l'aire sous la courbe (*area under the curve – AUC*) qui est obtenue en calculant la moyenne du « taux d'exploration » sur 500 itérations. Dans leurs expériences, le temps est représenté par le nombre d'itérations d'exploration. Les expériences ont été effectuées en utilisant le simulateur MRSim[4], qui est un simulateur discret basé sur Matlab.

- Frank *et al.* [FLHW10a] ont comparé quatre stratégies différentes de sélection de frontières pour l'exploration multi-robots et analysé la performance en termes : (1) du temps (compté en nombre de pas de simulation) nécessaire pour explorer tout l'environnement, ainsi que de (2) la zone explorée par pas de temps. Les expériences réalisées en Matlab en sont discrètes et idéales : elles négligent le bruit, les problèmes de localisation et ne considèrent que les environnements convexes et sans obstacles.

4 http://www.mathworks.com/matlabcentral/fileexchange/38409-mrsim-multi-robot-simulator--v1-0-

- Amigoni [Ami08] a expérimentalement comparé quatre stratégies d'exploration afin d'évaluer leurs forces et leurs faiblesses. Les expériences ont été effectuées sur un simulateur discret avec un seul robot. Seulement deux métriques ont été prises en compte pour comparer les performances des stratégies d'exploration : (1) le nombre d'opérations de détection nécessaires pour réaliser l'exploration et (2) la distance totale parcourue par le robot pendant l'exploration.

- Scrapper *et al.* [SML+09] se sont concentrés sur le développement de méthodes et techniques de test standard pour évaluer quantitativement la performance des systèmes de cartographie robotique par rapport aux exigences définies par l'utilisateur. Ils ont défini la qualité de la carte comme le rapport entre le nombre de points caractéristiques trouvés dans la carte construite par le robot et le nombre de points caractéristiques présents dans la carte réelle (*ground truth*). Toutefois, cette métrique n'évalue pas si les cartes construites et réelles sont de la « même forme ».

- Lass *et al.* [LSR09] ont examiné plusieurs métriques d'évaluation pour les systèmes multi-agents. Ils ont classé les métriques le long de deux axes : la performance et les types de données. Les métriques de performance quantifient la consommation des ressources du système, telles que l'utilisation de bande passante, la consommation d'énergie, et le temps d'exécution. Les types de données incluent le nominal, l'ordinal, l'intervalle, et le rapport. Ce travail peut être utilisé comme une référence, car les systèmes multi-robots peuvent être considérés comme un cas particulier de systèmes multi-agents.

Ce panorama montre la diversité des métriques utilisées pour un même problème. Dans la suite, nous définissons notre propre jeu de métriques basé sur une synthèse de cet état de l'art. Notre proposition se fonde sur l'idée que les métriques doivent être définies précisément et indépendamment du matériel et du logiciel pour permettre la reproductibilité des expérimentations.

5.2.3 Nos travaux

Dans cette section, nous présentons une infrastructure permettant de faire du *benchmarking* d'applications multi-robots [YFLB14a]. Nous détaillons ensuite deux utilisations de cette infrastructure : une première pour déterminer la taille optimum d'une flotte de robot lors d'une mission d'exploration [YFLB14b, BFD12] et une seconde pour comparer deux stratégies de coordination multi-robots lors d'une mission d'exploration [YFLB15a, YFLB15b].

NOTRE BANC D'ESSAI POUR APPLICATIONS MULTI-ROBOTS. La figure 21 présente l'architecture générale de notre solution [YFLB14a]. Elle repose sur l'in-

tergiciel ROS qui est un standard de fait pour le développement d'applications robotiques. Utiliser un intergiciel comme ROS permet de découpler une application robotique du robot sur lequel elle s'exécute. Cela nous permet d'exécuter le *même* code lors des *benchmarks* que lors du déploiement sur des robots réels. Pour renforcer le réalisme, chaque contrôleur de robot, constitué d'un ensemble de nœuds ROS, est déployé sur une machine différente. Pour s'affranchir des contraintes logicielles liées aux machines utilisées, nous avons virtualisé l'environnement logiciel de nos robots réels dans des VM (grâce au logiciel *VirtualBox*[5]). Ce choix est un compromis entre la réutilisabilité et les performances. Toutefois, comme tous nos robots utilisent la même infrastructure, la baisse de performances est uniformément répartie. Actuellement, nous utilisons MORSE [ELD+12b] pour simuler tous les robots et leur environnement en 3D. Ensuite, pour répondre à la reproductibilité des expérimentations, nous avons choisi d'exprimer des *plans d'expériences* précis qui décrivent les paramètres, les conditions d'arrêt et les métriques à collecter pour chaque expérience. Enfin, le moniteur (cf. figure 21) est en charge de superviser l'exécution d'une expérimentation conformément au plan d'expérience.

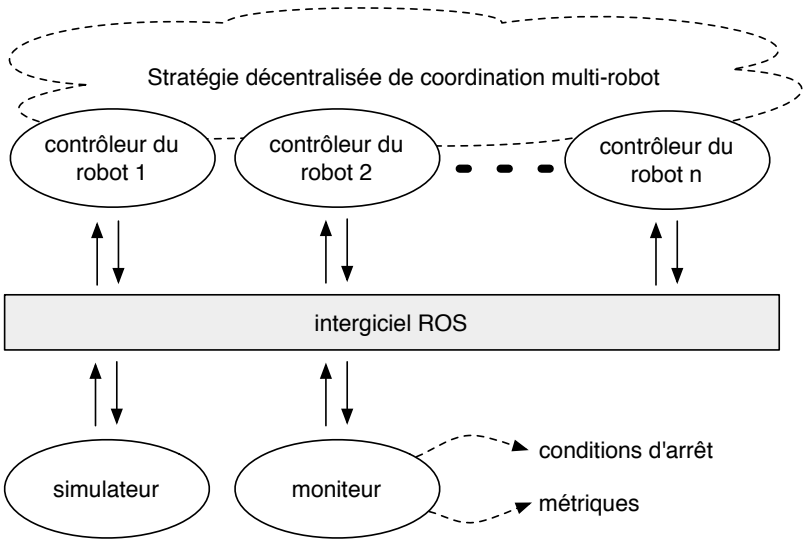


FIGURE 21 : Architecture de notre banc d'essai pour applications multi-robots.

Au niveau réseau, il y a trois voies de communication :

- Celle entre le simulateur et les robots. Le simulateur envoie les données de leurs capteurs (laser, odométrie, ...) à chaque robot et il reçoit des informations de commande.
- Celle entre le moniteur et les robots. Le moniteur reçoit les mesures liées à l'expérimentation en cours. En fonction des métriques souhaitées, le trafic peut être important. Par exemple, dans le cas d'applications

5 http://virtualbox.org

d'exploration, si la condition d'arrêt est un pourcentage de couverture de la carte, il est nécessaire d'échanger les cartes fréquemment.

- Celle entre les robots. Ces communications sont propres à l'application en cours d'exécution comme une stratégie de coordination pour l'exploration.

Nous avons effectivement mis en œuvre cette architecture sur un cluster de calcul pour simuler des flottes robots [YFLB14a]. Pour cela, nous avons répondu à de nombreuses questions telles que : Combien faut-il allouer de processeurs par contrôleur de robot ? Quelle architecture physique de réseau pour assurer des expérimentations réalistes ? Comment automatiser les expérimentations ?

PROCESSUS DE *benchmarking*. Notre processus de *benchmarking* repose sur la notion de plan d'expériences (cf. figure 22). Il s'agit de la description de toutes les expérimentations (simulations) à effectuer. Un plan d'expériences (cf. exemple figure 22) décrit les :

- *Paramètres.* Nous avons proposé une classification des différents paramètres de la littérature [YFLB14a]. Pour chaque paramètre, l'utilisateur doit fournir l'ensemble des valeurs qui vont être utilisées par les simulations.

- *Vecteurs de paramètres.* Un vecteur de paramètres contient la valeur à utiliser pour chacun des paramètres pour une simulation donnée. Ce vecteur de vecteurs peut être fourni sous la forme d'une fonction pour le calculer (e. g. toutes les combinaisons possibles).

- *Itérations.* C'est le nombre de fois qu'il faut répéter une simulation. En robotique, de nombreux composants ne sont pas déterministes à cause du matériel, mais aussi du logiciel comme les algorithmes de SLAM (*Simultaneous Localization And Mapping*).

- *Conditions d'arrêt.* Ces conditions expriment quand il faut stopper une simulation (i. e. terminée avec succès, temps écoulé atteint, . . .)

- *Mesures.* Il s'agit des données à collecter pendant chaque simulation. Il faut aussi préciser quand les données doivent être collectées via une fréquence ou l'occurrence d'un événement particulier.

- *Metriques.* Il s'agit de fonctions précises qu'il faut évaluer avec les données collectées.

Parameters	**Robot**	Robot Model: Pioneer 3-DX
		Computing Capabilities: 2 CPUs, 2GB RAM
		Maximum speed: 1.2m/s, 5.24rad/s
		Laser rangefinder: SICK LMS500
	Fleet	Number of robots: [1,30]
		Homogeneity: homogeneous (Pioneer 3-DX)
		Robot initial positions: top left to bottom left corner, every 2m
		Communication network: gigabit wired Ethernet
		Communication range: 200m
	Environment	Terrains: *loop*, *cross*, *zigzag*, *maze*
		Terrain size: 80m × 80m
		Obstacle height: 2m
		Corridor width: 8m
Parameter vectors	All possible combinations	
Iterations	5	
Stop conditions	99% of explored terrain	
	2000 seconds elapsed	
Measurements	Explored area per simulation step	
	% of used CPU each robot per 5 seconds	
	% of used RAM each robot per 5 seconds	
	Kbytes received each robot per 5 seconds	
	Kbytes sent each robot per 5 seconds	
Metrics	Exploration Time	
	Exploration Cost	
	Exploration Efficiency	
	Map Quality	

FIGURE 22 : Plan d'expériences pour l'exploration multi-robots basée sur les frontières [Yam98a].

MÉTRIQUES POUR QUANTIFIER LA PERFORMANCE DE SOLUTIONS D'EXPLORATION MULTI-ROBOTS. Le jeu de métriques que nous avons défini [YFLB15b] est dédié à l'évaluation de la performance des algorithmes d'exploration multi-robots. Il comprend les éléments suivants :

- **Robot.** Il s'agit de quantifier la consommation en ressources localement sur chaque robot :
  - **Temps de calcul.** Somme des temps d'utilisation utilisateur et système ;
  - **Quantité mémoire.** Pourcentage de mémoire utilisé ;
  - **Bande passante.** Nombre total de Kio reçus ou émis par seconde.
- **Flotte.** Il s'agit de quantifier la performance d'une flotte :
  - **Temps d'exploration.** Cette métrique est la plus utilisée dans la littérature. Il s'agit du temps total nécessaire à la flotte pour terminer une mission d'exploration. Notre définition est que l'exploration commence lorsqu'un des robots commence à explorer et se termine lorsqu'au moins un des robots a exploré un pourcentage donné de la carte (e. g. 99%) ;
  - **Coût d'exploration.** Cette métrique peut faire intervenir de nombreuses données comme la consommation totale en ressources, le

prix d'achat des robots ou encore leur coût de maintenance. Toutefois, elle est souvent définie ainsi :

$$explorationCost(n) = \sum_{i=1}^{n} d_i$$

où n est le nombre de robot dans la flotte et d_i la distance parcourue par chaque robot i.

– **Efficacité de l'exploration.** Cette métrique est directement proportionnelle à la quantité d'information récupérée à partir de l'environnement et inversement proportionnelle au coût que cela a engendré [ZSDT02] :

$$explorationEfficiency(n) = \frac{M}{explorationCost(n)}$$

où n est le nombre de robots dans la flotte et M est l'aire de la zone explorée en mètre carré. Par exemple, si cette métrique vaut 1.6, cela signifie qu'à chaque fois que les robots parcourent une distance de $1m$, ils découvrent en moyenne une zone de $1.6m^2$.

– **Complétude de la carte.** Le résultat de l'exploration est une carte de l'environnement découvert. La connaissance préalable du terrain (*ground truth map*) permet d'estimer si la carte produite est complète ou non [CVRF14a, FLHW10b, SML+09]. Cette métrique quantifie la complétude comme le ratio entre l'aire explorée M et l'aire totale de la « vraie » carte (*ground truth map*) P :

$$mapCompleteness = \frac{M}{P}$$

– **Qualité de la carte.** À cause des imprécisions, du bruit et des algorithmes utilisés (SLAM), les cartes produites lors d'une exploration contiennent des erreurs par rapport à la carte *ground truth*. Une carte est généralement représentée par une grille d'occupation des zones inconnues, occupées et libres. Nous définissons donc l'erreur d'une carte ($mapError(M)$) comme le nombre total de cellules dans cette carte qui ont une valeur différente de leur cellule correspondante dans la carte *ground truth*. L'erreur d'une carte sera d'autant plus importante que sa résolution sera élevée.
La performance de l'exploration est donc un compromis entre la résolution de la carte produite et les erreurs qu'elle contient. Nous définissons ainsi la qualité d'une carte comme le ratio entre l'aire explorée sans erreur et l'aire totale de la carte *ground truth* :

$$mapQuality = \frac{M - A(mapError(M))}{P}$$

où M est l'aire explorée, $A(mapError(M)$ est l'aire occupée par l'erreur de la carte et P est l'aire totale de la carte *ground truth map*. Notre définition de qualité de carte tient compte de la précision topologique au lieu de se concentrer uniquement sur la complétude comme dans [CVRF14a, SML+09].

OPTIMISATION DE LA TAILLE D'UNE FLOTTE DE ROBOTS POUR L'EXPLORATION. L'idée de ce travail [YFLB14b, BFD12] est d'optimiser la taille d'une flotte de robots (nombre de robots dans la flotte) effectuant une mission d'exploration de sorte que le temps d'exploration et le coût d'exploration soient minimisés. Il s'agit d'explorer le plus rapidement possible une zone avec un nombre minimum de robots consommant le moins d'énergie possible. Pour résoudre ce problème, nous avons effectué une série d'expérimentations grâce à l'infrastructure décrite ci-dessus. Le code simulé est exactement le même que celui qui s'exécute sur les robots. Toutefois, les simulations permettent d'effectuer plus rapidement de nombreuses expérimentations avec des conditions différentes comme la position initiale des robots. Nous avons effectué un premier plan d'expériences [YFLB14b] pour déterminer si la position initiale des robots a un impact sur l'exploration. Pour cela, nous avons fixé tous les paramètres : flotte de robots homogène, un terrain de $80m^2$ de type labyrinthe, condition d'arrêt à 99% d'exploration, 5 simulations par expérimentations, *etc*. Les robots ne se coordonnent qu'à travers l'échange de leur carte locale. Chacun robot décide seul de la direction à prendre en utilisant un algorithme de plus proche frontière [Yam98a]. Nous avons fait varier le nombre de robots ainsi que les positions initiales des robots selon trois configurations : *blind* (A), un robot par point d'entrée (B), deux robots par point d'entrée (C). La figure 23 montre les résultats de nos expérimentations. Les meilleurs résultats sont atteints avec 12 robots dans ce cas. Les expérimentations avec les positions initiales B et C ont été ajoutés ensuite pour confirmer l'hypothèse que les positions initiales des robots ont un impact significatif sur l'exploration. Les dégradations de coût et de temps lorsqu'on augmente encore la taille de la flotte (plus de 12 robots) s'expliquent par le fait que les robots se gênent mutuellement et aussi parce que la carte contient des culs-de-sac.

BENCHMARKING DE SOLUTIONS MULTI-ROBOTS POUR L'EXPLORATION. Nous avons réalisé un *benchmark* de solutions d'exploration multi-robots [YFLB15a] en fixant de nombreux paramètres et faisant varier l'algorithme de fusion de cartes ainsi que le terrain utilisé. L'idée est de déterminer l'impact de la fusion de cartes sur les performances de l'exploration. Nous avons choisi : un algorithme glouton [Yam98b] et un algorithme probabiliste [BMF+00]. Ces deux algorithmes nécessitent de connaître la position

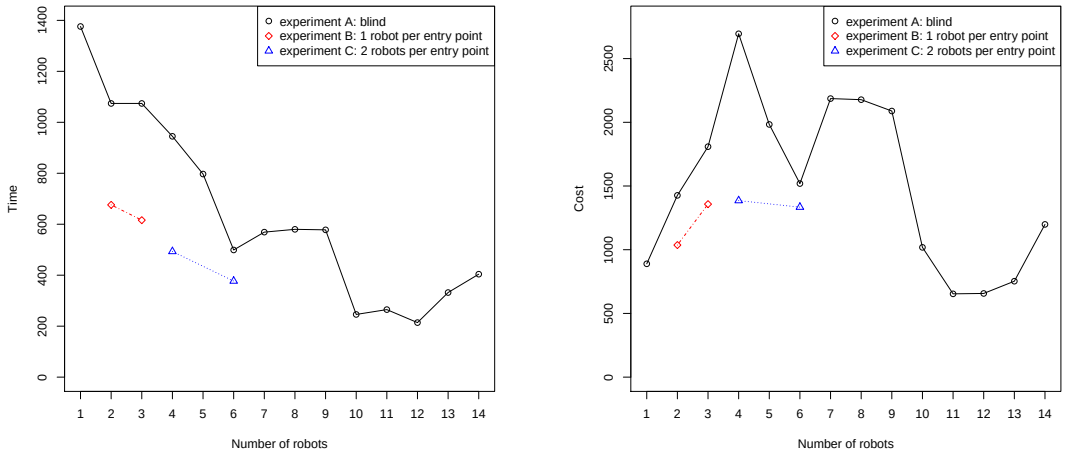
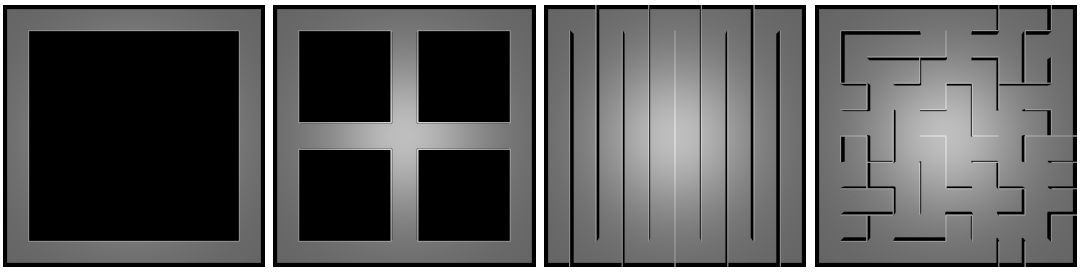


FIGURE 23 : Moyenne du temps et du coût d'exploration en fonction du nombre de robots et de leurs positions initiales.

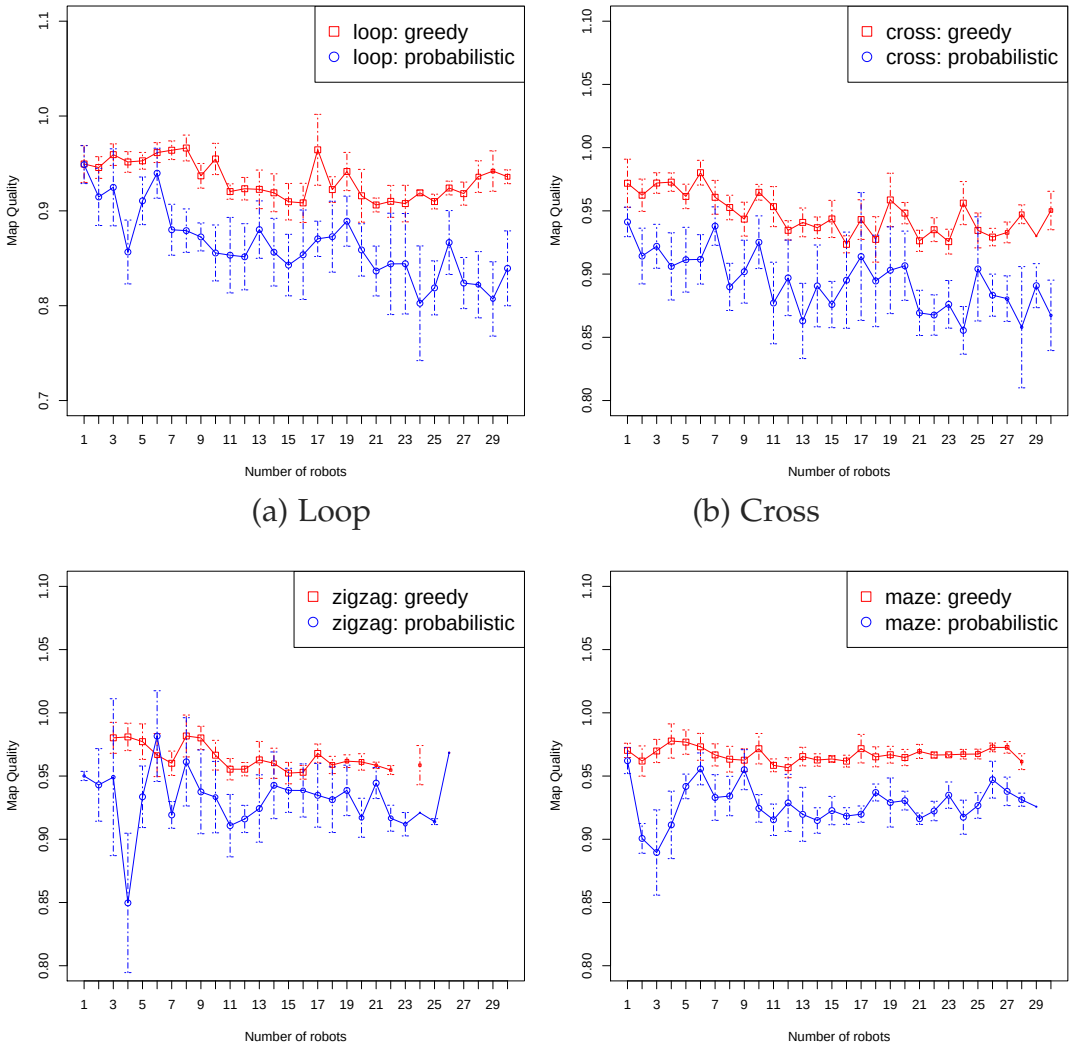
initiale des robots les uns par rapport aux autres. Nous avons utilisé 4 terrains de types différents (cf. figure 24) inspirés de la Robocup. La figure 25 montre uniquement les résultats concernant la qualité de la carte (cf. [YFLB15a] pour tous les résultats). Ces résultats expérimentaux montrent que la taille de la flotte et le type de terrain influent principalement sur le temps, le coût et l'efficacité de l'exploration alors que l'algorithme de fusion de cartes influe plutôt sur la qualité de la carte produite.



(a) Loop (b) Cross (c) Zigzag (d) Maze

FIGURE 24 : Terrains utilisés pour le *benchmarking* de solutions d'exploration (obstacles en noir).

ÉVALUATION. Notre processus de *benchmarking* permet d'évaluer et comparer expérimentalement des solutions multi-robots grâce à l'explicitation des plans d'expériences et à la définition de métriques claires. Notre infrastructure de simulation basée sur l'intergiciel ROS permet effectivement de simuler le même code que celui qui s'exécute sur les robots réels. Nous avons d'ailleurs utilisé le même code d'exploration que celui décrit ci-dessus sur



(a) Loop (b) Cross

(c) Zigzag (d) Maze

FIGURE 25 : Résultats d'un *benchmark* de solutions d'exploration pour la métrique qualité de carte.

deux robots réels[6]. Notre infrastructure de simulation peut encore être améliorée. Par exemple, en ce qui concerne la simulation de différentes configurations réseau comme cela est possible avec RoboNetSim ou encore en utilisant un simulateur robotique distribué.

5.3 TRAVAUX EN COURS ET PERSPECTIVES

DÉBOGAGE À DISTANCE. Nos travaux sur le débogage à distance se poursuivent naturellement à travers le projet « *RDebug* » (cf. 146) porté par Marcus Denker de l'INRIA Lille qui vise à transférer nos travaux communs au

6 http://car.mines-douai.fr/2015/03/collaborative-exploration

sein de l'environnement Pharo. Cela permettra de faciliter le développement d'applications pour CPS mobiles avec Pharo et surtout ouvrira de nouvelles directions de recherche. En effet, le débogage à distance n'est qu'une facette du développement. C'est pourquoi nous nous intéressons aussi au développement à distance et surtout à la possibilité de développer de façon interactive à distance i. e. sans arrêter le programme distant. C'est notamment ce que nous avons proposé dans le projet « CAIRE » (cf. Annexe A, page 146) et mis en pratique à travers notre plateforme PhaROS (une passerelle entre *Robotic Operating System* (ROS) et Pharo). Dans le cadre de son stage de master, Pablo Estefó a expérimenté cette idée avec le langage dédié LRP (*Live Robot Programming*) [ECF+14]. Ce langage est conçu par Johan Fabry de l'université du Chili à Santiago. Ensembles, nous sommes d'ailleurs en train de démarrer une collaboration scientifique qui s'est concrétisée à ce jour par des séjours invités (cf. Annexe A, page 154 pour plus de détails).

TESTS D'APPLICATIONS ROBOTIQUES. Lors du développement d'applications robotiques, l'intégration entre le logiciel et le matériel peut conduire à des problèmes difficiles à identifier. De même, la maintenance ou l'assurance qualité des systèmes robotiques nécessitent d'effectuer des tests pour vérifier le bon fonctionnement du système dans son ensemble (matériel+logiciel). Ces tests doivent être répétables et les plus automatisés possible. Nous avons donc proposé une méthodologie de développement d'applications robotiques [FLB13b, FLB13a, LFB13] basée sur des tests répétables, réutilisables et semi-automatisés. Cette méthodologie propose de maximiser la sûreté pour les opérateurs humains et le robot, même en cas de défaillance de ce dernier. À long terme, nous souhaitons proposer une infrastructure permettant de tester fonctionnellement des systèmes robotiques, de façon analogue aux processus d'intégration continue en génie logiciel. L'idée est de réaliser progressivement des tests (simulations) d'abord sur le logiciel de contrôle robotique, puis d'intégrer progressivement les capteurs et les actionneurs, pour arriver enfin au système complet (matériel et logiciel).

INFRASTRUCTURE POUR LE BENCHMARKING. Nous continuons notre effort pour la mise en place d'une infrastructure permettant le *benchmarking* d'applications pour flottes de robots mobiles. À travers le post-doc de Khelifa Baizid, nous travaillons actuellement sur des évolutions de notre infrastructure et sur le benchmarking de différents algorithmes de cartographie (GMapping[7], HectorSLAM [KMvSK11], KartoSLAM[8], ...) ainsi que sur des métriques plus fines pour quantifier la qualité de la carte obtenue. Nous souhaitons ensuite nous concentrer sur l'amélioration des techniques de cartographie multi-robots principalement limitées par la quantité d'information à échanger entre les robots d'après nos expérimentations.

7 http://wiki.ros.org/gmapping
8 http://www.kartorobotics.com

5.4 SYNTHÈSE DES CONTRIBUTIONS

Le tableau ci-dessous synthétise mon activité dans ce thème à travers la co-signature d'articles scientifiques, le co-encadrement de thèses ou encore la participation à des projets scientifiques. L'annexe A contient plus de détails concernant ces éléments.

2	Revues internationales	[PBF+15, BFB+13]
3	Conférences internationales	[YFLB15a, YFLB14b, LFB13]
1	Atelier international	[FLB13b]
4	Ateliers nationaux	[YFLB15b, YFLB14a, FLB13a, BFD12]
2	Rapports d'activité de projet	[BFLB13, BFLB14]
1	Thèse co-encadrée soutenue	[Pap13]
2	Post-doc co-encadrés	Zhi Yan, Khelifa Baizid
4	Projets de recherche	Projet « CAIRE » (coordinateur), Projet « Roboshop », Projet « SUCRé » Projet « RDebug »

6 SYNTHÈSE ET PERSPECTIVES

> On ne va jamais aussi loin que lorsqu'on ne sait pas où l'on va.
>
> CHRISTOPHE COLOMB

> Much to learn you still have... my old padawan. [...] This is just the beginning!
>
> YODA

Ce chapitre commence par une synthèse des travaux présentés dans ce mémoire. J'ai menés la plupart de ces travaux dans le cadre de projets scientifiques en partenariat avec d'autres équipes de recherche ainsi qu'à travers le co-encadrements d'étudiants de master, d'ingénieurs, de doctorants ou encore de post-doctorants (cf. Annexe A, page 146). Cette synthèse permet de mettre en perspective les résultats obtenus par rapport à l'objectif général qui guide mes recherches présenté en introduction (cf. section 1.3). Ce chapitre conclue ensuite ce mémoire en présentant trois axes de recherches possibles suite aux travaux déjà menés.

6.1 SYNTHÈSE

Le cycle de développement « classique » (cf. figure 1, page 3) pour développer des applications pour des systèmes cyber-physiques (CPS) mobiles est un frein à l'efficacité des développeurs. Pour remédier à cela, je me suis attaché à proposer des langages, des modèles et des outils de plus haut niveau offrant ainsi au développeur la possibilité de développer interactivement une application pour CPS mobiles. Cet objectif global s'est concrétisé au fil des ans à travers des recherches menées conjointement avec d'autres équipes scientifiques à travers le co-encadrements de doctorants notamment.

La figure 26 illustre le nouveau cycle de développement que je propose ainsi que les différentes contributions présentées dans ce manuscrit pour le

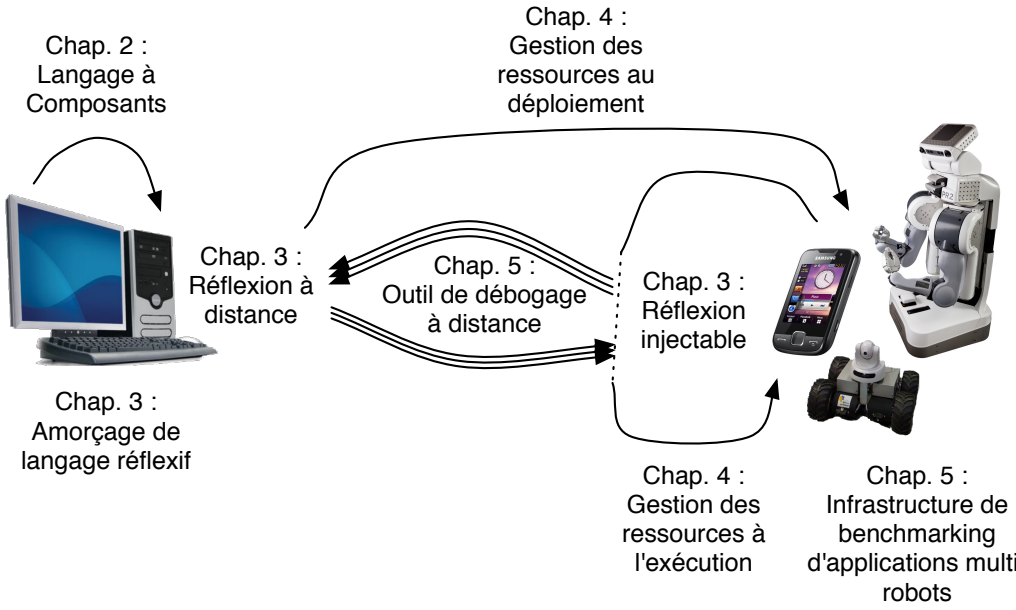


FIGURE 26 : Synthèse de travaux présentés dans ce mémoire pour supporter le développement interactif et haut niveau d'applications pour CPS mobiles.

supporter. Dans le chapitre 2, nous avons proposé des langages à composants. Ces travaux ont notamment été menés dans le cadre d'un projet de recherche avec les équipes MaREL du LIRMM et ISOE des Mines d'Alès (cf. Annexe A, page 148) et le co-encadrement de la thèse de Matthieu Faure [Fau12]. Les langages à composants permettent au développeur d'exprimer l'architecture d'une application directement dans son code source. Cette explicitation de l'architecture d'une application permet de la manipuler et de la réviser si besoin pour l'adapter. Pour encore mieux exprimer des adaptations logicielles, nous avons même proposé un langage à composants réflexif où tous les concepts structurels du langage sont représentés par des composants. Implémenter ou simplement étendre un langage réflexif est toutefois une tâche complexe (cf. chapitre 3, section 3.2). En ce sens, nous avons proposé avec l'équipe RMoD de l'INRIA dans le cadre du co-encadrement de la thèse de Guillermo Polito [Pol15], une approche par virtualisation consistant à faire coexécuter plusieurs environnements au-dessus de la même machine virtuelle. Un environnement pouvant contrôler l'exécution des autres. Cette même infrastructure a été adaptée pour réduire efficacement la taille des applications avant leur déploiement sur un CPS mobile (cf. chapitre 4, section 4.1). Pendant l'exécution d'une application sur un CPS mobile, nous avons proposé d'utiliser un mécanisme de mémoire virtuelle au niveau applicatif (cf. chapitre 4, section 4.2). Ce travail est le résultat de la thèse Mariano Martinez-Peck [MP12] également co-encadrée avec l'équipe RMoD de l'INRIA Lille avec laquelle une collaboration durable s'est installée. Ce mécanisme permet de décharger en mémoire secondaire les objets non-utilisés (mais référencés) par l'application et de les recharger si besoin de façon transparente pour

l'application en cours d'exécution. Lorsqu'une erreur survient pendant l'exécution, nous avons proposé une infrastructure (cf. chapitre 5, section 5.1) permettant le débogage à chaud et à distance de cette erreur depuis la machine de développement. Cette infrastructure repose notamment sur la possibilité d'injecter dynamiquement des opérations réflexives (cf. chapitre 3, section 3.1) sur le terminal mobile et sur la possibilité d'avoir des méta-objets sur la machine de développement, causalement connectés aux objets de base sur le terminal mobile. Ces travaux sur le débogage à distance sont le résultat de la thèse de Nick Papoulias [Pap13] également co-encadrée avec l'équipe RMoD de l'INRIA Lille.

Le domaine d'application privilégié qui motive ces travaux est le développement d'applications de cartographie multi-robots. Pour développer ce type d'applications, il faut non seulement développer la partie contrôle de chaque robot, mais également s'intéresser à la coordination des robots de la flotte pour accomplir la mission d'exploration. Pour faciliter la comparaison objective des solutions d'exploration multi-robots existantes, nous avons proposé une infrastructure permettant de simuler le même code que celui qui serait déployé sur des robots réels ainsi que des métriques quantifiables (cf. chapitre 5, section 5.2).

6.2 PERSPECTIVES

À la fin de chaque chapitre de ce mémoire, les perspectives à court et moyen terme pour poursuivre les travaux ont déjà été présentées. La plupart de ces perspectives s'inscrivent dans le cadre de projets scientifiques, d'encadrements de thèses, de post-doc ou encore de collaborations scientifiques auxquels je participe déjà. Dans cette section, je vais donc me concentrer uniquement sur trois perspectives à plus long terme.

6.2.1 Développement « agile » d'applications de robotique de services

Les approches de développement dites « agiles » (XP, TDD, intégration continue, ...) sont désormais une alternative crédible et de plus en plus utilisée par rapport aux méthodes classiques (cycle en V, ...) pour faire rapidement face aux besoins des usagers. Pour cela, ces méthodes pragmatiques mettent l'accent sur les tests et l'intégration en conditions réelles dès le début du cycle de développement. Cela est rendu possible par un ensemble de pratiques et d'outils (débogueur, outils de refactoring, serveur d'intégration, ...).

Dans le domaine du développement d'applications robotiques, les méthodes agiles sont encore marginales. Une explication est que peu de chercheurs issus du génie logiciel s'intéressent aux problématiques liées au développement de logiciels pour les systèmes robotiques (contraintes matérielles

et temporelles). Ainsi, les méthodes classiques, les langages et les outils bas niveau (assembleur, C, C++) sont encore majoritairement utilisés dans ce domaine ce qui ne permet pas d'utiliser pleinement les méthodes agiles.

À long terme, mon objectif est de fédérer plusieurs chercheurs autour de l'utilisation des méthodes agiles pour le développement de logiciels modulaires pour la robotique. Nous nous concentrerons sur les couches de contrôle hautes des robots mobiles, car les contraintes matérielles et temporelles y sont plus faibles. De même que pour les travaux présentés dans ce mémoire, l'originalité de tels travaux réside dans l'utilisation de langages dynamiques et réflexifs, car ils permettent une meilleure rapidité pour la mise au point de programmes et surtout ils facilitent la réalisation d'extensions du langage et des outils de développement pour les adapter aux besoins de la robotique.

Globalement, il faut raccourcir le cycle développement traditionnel pour le rendre plus adapté au développement interactif. Les différentes tâches liées au développement (compilation, debug, tests, analyse de performances, ...) pourront s'effectuer soit sur la machine de développement soit sur le robot en fonction de ses capacités (CPU, mémoire, ...) et des besoins du développeur. Plus précisément, il s'agit de permettre :

- le développement et le déploiement incrémental du code sur le robot de manière transparente afin que le développeur puisse travailler directement sur le robot tout en profitant de la puissance de sa machine pour l'environnement et les outils de développement ;
- l'analyse *in vivo* des programmes en cours d'exécution sur un robot ;
- la modification du code et des données du programme à chaud c'est-à-dire pendant son exécution afin de faciliter la mise au point des programmes ;
- l'optimisation du programme en fonction des ressources réelles du robot ;
- la possibilité de tester automatiquement un système robotique (matériel et logiciel) ;
- l'intégration continue i. e. regrouper fréquemment et automatiquement toutes les briques logicielles et les déployer sur le robot afin d'en vérifier leur cohérence et leur compatibilité avec les ressources et les capacités matérielles du robot.

Notre solution de débogage à distance permet déjà de répondre à certaines de ces problématiques. Toutefois, cette solution nécessite d'être adaptée et pleinement intégrée à un environnement complet comme Pharo pour devenir un standard de développement. Cet objectif est d'ailleurs celui du projet « RDebug » (cf. page 146) dans lequel nous collaborons avec l'équipe RMoD

de l'INRIA Lille Nord Europe. D'autres modèles et outils sont nécessaires pour mieux supporter le développement interactif, les tests ou encore l'intégration continue. Cet axe de recherche est également le cadre de ma collaboration avec l'équipe Pleiad de l'Université du Chili qui m'a invité deux semaines en novembre 2014 pour utiliser nos outils de développement autour de PhaROS sur leur robot PR2 (cf. Annexe A, page 154 pour plus de détails).

6.2.2 Adaptation et mise à jour dynamique d'applications

De nombreux domaines d'applications (robotique, cloud, ...) nécessitent des systèmes continuellement en fonctionnement. Et même si le fonctionnement perpétuel n'est pas impératif, il peut être nécessaire d'adapter le comportement d'une application pendant son exécution dans certains cas. Dans le cadre d'une mission robotique, il peut être nécessaire de changer dynamiquement de mode de fonctionnement i. e. d'état dans le cas d'une modélisation par réseau de Petri. Ce changement d'état peut s'accompagner d'une reconfiguration de l'architecture logicielle de contrôle du robot mais ce dernier ne peut pas s'arrêter de fonctionner dans certains cas. Par exemple, un drone volant au dessus de l'eau doit continuer à voler pendant la reconfiguration. Même les systèmes déployés dans le nuage sont reconfigurés (i. e. stoppés, redémarrés, déplacés) en permanence pour optimiser la consommation énergétique des serveurs sur lesquels ils s'exécutent par rapport à leur taux d'utilisation. Ces besoins d'adaptation et/ou de mise à jour dynamique d'une application en cours d'exécution soulèvent de nombreux défis. Dans le cas d'applications de robotique mobile, nous pouvons citer :

- Comment détecter qu'une adaptation est nécessaire ? Cette question majeure soulève le problème de la modélisation du contexte d'exécution qui inclut l'environnement physique du robot, mais aussi celui de l'application (batterie déchargée, surchauffe processeur, ...).
- Comment prévoir tous les scénarios d'adaptation possibles ? Il n'est en effet pas concevable que le développeur d'une application ait à imaginer toutes les adaptations possibles.
- Comment exprimer les adaptations i. e. quelles abstractions sont nécessaires ?
- Quels mécanismes pour adapter effectivement une application en cours d'exécution ? Modifier dynamiquement le code d'une application peut conduire à des inconsistances. Les travaux existants sur la quiescence ou les mécanismes transactionnels apportent déjà des éléments de réponse à cette question.
- Pendant l'adaptation d'une application, comment assurer un mode de fonctionnement « minimal » qui garanti la sécurité des personnes et

du matériel dans le cas d'une application robotique ? Par exemple, un drone ne doit pas tomber pendant l'adaptation de son application de contrôle.

- Comment transférer l'état d'une application vers sa forme adaptée ?
- Peut-on borner le temps d'une adaptation et est-ce déterministe ?

Je travaille déjà sur cet axe de recherche de l'adaptation et de la mise à jour dynamique d'applications à travers le co-encadrement d'une thèse avec l'équipe RMoD de l'INRIA Lille Nord Europe.

6.2.3 *Benchmarking* de systèmes multi-robots

Dans de nombreux domaines, utiliser une flotte de robots permet de réaliser la mission plus rapidement ou plus efficacement. L'objectif du projet « SUCRé » (cf. page 146) est de fournir un support aux équipes de secours (e. g. pompiers) dans le cas de scénarios à risque dans des bâtiments (e. g. incendie, fuite gaz, . . .). L'idée est qu'une flotte de robots explore le bâtiment avec différents objectifs : cartographie, localisation de personnes, localisation des foyers d'incendie ou encore détection des voies de passage pour les pompiers. Ce projet prospectif et ambitieux n'est pourtant qu'un exemple des possibles applications qu'offrent les systèmes multi-robots autonomes. C'est pourquoi il est important de proposer dès aujourd'hui des outils et des infrastructures pour développer et comparer objectivement des applications multi-robots. En ce sens, nous avons déjà proposé une première infrastructure permettant de simuler le code réel d'applications multi-robots [YFLB15a]. À partir de données récoltées durant ces simulations, nous pouvons calculer des métriques et comparer objectivement différentes approches de coordination multi-robots aussi bien en terme d'efficacité qu'en terme de réalisme par rapport aux capacités réelles des robots. Je souhaite continuer à enrichir cette plateforme jusqu'à proposer une collection d'expérimentations de références pour certaines classes de problèmes de façon analogue à ce qui est proposé dans d'autres communautés scientifiques. Bien évidemment, cela nécessite également de comparer les résultats obtenus par différentes approches d'où la nécessité d'un consensus sur les métriques à utiliser. Ce travail ouvre également la voie pour proposer de nouvelles stratégies de coordinations multi-robots décentralisées et efficaces.

GLOSSAIRE

CPS *Cyber-Physical System*

MOP *Meta-Object Protocol*

CBSD *Component-based Software Development*

GPS *Global Positioning System*

DSL *Domain-Specific Languages*

FPGA *Field-programmable gate array*

FFI *Foreign Function Interface*

PPC Programmation par composants

ADL *Architecture Description Language*

OCL *Object Constraint Language*

ANR Agence nationale de la recherche

EJB *Enterprise JavaBeans*

CCM *Corba Component Model*

RAM *Random Access Memory*

ROS *Robotic Operating System*

BIBLIOGRAPHIE

[AAB+00] B. Alpern, C. R. Attanasio, J. J. Barton, M. G. Burke, P. Cheng, J. D. Choi, A. Cocchi, S. J. Fink, D. Grove, M. Hind, S. F. Hummel, D. Lieber, V. Litvinov, M. F. Mergen, T. Ngo, J. R. Russell, V. Sarkar, M. J. Serrano, J. C. Shepherd, S. E. Smith, V. C. Sreedhar, H. Srinivasan, and J. Whaley. The Jalapeño virtual machine. *IBM Systems Journal*, 39(1) :211–238, 2000. (Cité page 49.)

[ABG+04] Jakob R. Andersen, Lars Bak, Steffen Grarup, Kasper V. Lund, Toke Eskildsen, Klaus Marius Hansen, and Mads Torgersen. Design, implementation, and evaluation of the resilient smalltalk embedded platform. In *Proceedings of ESUG International Smalltalk Conference 2004*, September 2004. (Cité page 42.)

[ACN02] Jonathan Aldrich, Craig Chambers, and David Notkin. ArchJava : Connecting Software Architecture to Implementation. In *Proceedings of the 22rd International Conference on Software Engineering, ICSE 2002, 19-25 May 2002, Orlando, Florida, USA*, pages 187–197. ACM, 2002. (Cité page 20.)

[Age96] Ole Agesen. *Concrete Type Inference : Delivering Object-Oriented Applications*. Ph.D. thesis, Stanford University, December 1996. (Cité page 61.)

[AJO11] Monica Anderson, Odest Chadwicke Jenkins, and Sarah Osentoski. Recasting robotics challenges as experiments. *IEEE Robotics & Automation Magazine*, 18(2) :10–11, June 2011. (Cité page 91.)

[Ami08] Francesco Amigoni. Experimental evaluation of some exploration strategies for mobile robots. In *Proceedings of ICRA'08*, pages 2818–2823, 2008. (Cité page 94.)

[Asa11] Kenichi Asai. Reflection in direct style. *SIGPLAN Not.*, 47(3) :97–106, October 2011. (Cité page 48.)

[BC86] J.-P. Briot and P. Cointe. The objvlisp model : Definition of a uniform, reflexive and extensible object oriented language. In *Proc. of the 7th ECAI*, pages 270–277, 1986. (Cité page 28.)

[BCL+04] Eric Bruneton, Thierry Coupaye, Matthieu Leclercq, Vivien Quéma, and Jean-Bernard Stefani. An Open Component Model and Its Support in Java. In Ivica Crnkovic, Judith A. Stafford, Heinz W. Schmidt, and Kurt C. Wallnau, editors, *Component-Based Software Engineering, 7th International Symposium, CBSE 2004, Edinburgh, UK, May 24-25, 2004, Proceedings*, volume 3054 of *Lecture Notes in Computer Science*, pages 7–22. Springer, 2004. (Cité pages 18 et 22.)

[BDN+09] Andrew P. Black, Stéphane Ducasse, Oscar Nierstrasz, Damien Pollet, Damien Cassou, and Marcus Denker. *Pharo by Example*. Square Bracket Associates, Kehrsatz, Switzerland, 2009. (Cité page 46.)

[Bei90a] Boris Beizer. *Software Testing Techniques*. Thomson Computer Press, 1990. (Cité page 83.)

[Bei90b] Boris Beizer. *Software testing techniques (2nd ed.)*. Van Nostrand Reinhold Co., New York, NY, USA, 1990. (Cité page 82.)

[BF09] Noury Bouraqadi and Luc Fabresse. CLIC : A component model symbiotic with smalltalk. In *Proceedings of the International Workshop on Smalltalk Technologies*, Brest, France, August 2009. ACM. (Cité pages 21 et 36.)

[BF10] Noury Bouraqadi and Luc Fabresse. Towards Small Portable Virtual Machines. In Marcus Denker and Gabriela Arévalo, editors, *Proceedings of International Conference on Smalltalk Technologies (Smalltalks'10)*, Concepcion del Uruguay, Argentina, 2010. (Cité pages 79 et 80.)

[BFB+13] Noury Bouraqadi, Luc Fabresse, Alexandre Bergel, Damien Cassou, Stéphane Ducasse, and Jannik Laval. Sockets. In *Deep Into Pharo*, page 21. Square Bracket Associates, September 2013. (Cité page 103.)

[BFD12] Noury Bouraqadi, Luc Fabresse, and Arnaud Doniec. On fleet size optimization for multi-robot frontier-based exploration. In *Proceedings of 7th National Conference on "Control Architecture of Robots" Control Architectures of Robots (CAR'12)*, Nancy, France, May 2012. (Cité pages 94, 99 et 103.)

[BFDS13] Camillo Bruni, Luc Fabresse, Stéphane Ducasse, and Igor Stasenko. Language-side Foreign Function Interfaces with NativeBoost. In *Proceedings of International Workshop on Smalltalk Technologies (IWST'13)*, Annecy, France, September 2013. (Cité pages 79 et 80.)

[BFLB13] Noury Bouraqadi, Luc Fabresse, Jannik Laval, and Santiago Bragagnolo. PhaROS : Concepts and Architecture. Technical report, Ecole des Mines de Douai, Avr 2013. (Cité page 103.)

[BFLB14] Noury Bouraqadi, Luc Fabresse, Jannik Laval, and Santiago Bragagnolo. A Robotic Infrastructure for Map Construction and Navigation. Technical report, Ecole des Mines de Douai, Jan 2014. (Cité page 103.)

[BH02] Padmanabhan Santhanam Brent Hailpern. Software debugging, testing, and verification. *IBM Systems Journal*, 2002. (Cité page 82.)

[BHL00] G. Back, W. Hsieh, and J. Lepreau. Processes in kaffeos : Isolation, resource management and sharing in java. In *4th USENIX International Symposium on Operating System Design and Implementation (OSDI)*, 2000. (Cité page 50.)

[BM08] Michael D. Bond and Kathryn S. McKinley. Tolerating memory leaks. In Gail E. Harris, editor, *OOPSLA : Proceedings of the 23rd Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, October 19-23, 2008, Nashville, TN, USA*, pages 109–126. ACM, 2008. (Cité pages 69 et 72.)

[BMF+00] W. Burgard, M. Moors, D. Fox, R. G. Simmons, and S. Thrun. Collaborative multi-robot exploration. In *Proc. ICRA'00*, pages 476–481, April 2000. (Cité pages 90, 93 et 99.)

[BO14] Garo Bournoutian and Alex Orailoglu. On-device objective-c application optimization framework for high-performance mobile processors. In *Proceedings of the Conference on Design, Automation & Test in Europe*, DATE '14, pages 85 :1–85 :6, 3001 Leuven, Belgium, Belgium, 2014. European Design and Automation Association. (Cité page 61.)

[BOS91] Paul Butterworth, Allen Otis, and Jacob Stein. The GemStone object database management system. *Commun. ACM*, 34(10) :64–77, 1991. (Cité page 72.)

[BPM14] Fabio Bonsignorio, Angel P. Del Pobil, and Elena Messina. Fostering progress in performance evaluation and benchmarking of robotic and automation systems. *IEEE Robotics & Automation Magazine*, 21(1) :22–25, March 2014. (Cité page 91.)

[Bru97] David Bruce. What makes a good domain-specific language ? apostle, and its approach to parallel discrete event simulation, 1997. Technical Report. (Cité page 7.)

[BSC12] Antoine Bautin, Olivier Simonin, and François Charpillet. MinPos : A novel frontier allocation algorithm for multi-robot exploration. In *Proceedings of ICIRA'12*, pages 496–508, 2012. (Cité page 93.)

[BSW96] Jan Bosch, Clemens A. Szyperski, and Wolfgang Weck. Workshop on component-oriented programming (wcop'96), 1996. (Cité page 8.)

[BU04] Gilad Bracha and David Ungar. Mirrors : design principles for meta-level facilities of object-oriented programming languages. In *Proceedings of the International Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA'04), ACM SIGPLAN Notices*, pages 331–344, New York, NY, USA, 2004. ACM Press. (Cité pages 11, 38, 39, 40, 41 et 42.)

[BW14] Andreas Bihlmaier and Heinz Wörn. Increasing ROS reliability and safety through advanced introspection capabilities. In *Proceedings of the INFORMATIK 2014*, pages 1319–1326, 2014. (Cité page 92.)

[CDF+11] Gwenael Casaccio, Stéphane Ducasse, Luc Fabresse, Jean-Baptiste Arnaud, and Benjamin van Ryseghem. Bootstrapping a smalltalk. In *Proceedings of International Conference on Smalltalk Technologies (Smalltalks'11)*, Bernal, Buenos Aires, Argentina, 2011. (Cité pages 55 et 56.)

[CDT03] Grzegorz Czajkowski, Laurent Daynès, and Ben Titzer. A multi-user virtual machine. In *USENIX Annual Technical Conference, General Track*, pages 85–98, 2003. (Cité page 50.)

[CGV10] Alexandre Courbot, Gilles Grimaud, and Jean-Jacques Vandewalle. Efficient off-board deployment and customization of virtual machine-based embedded systems. *ACM Transaction on Embedded Computer Systems*, 9 :21 :1–21 :53, mar 2010. (Cité pages 62 et 72.)

[Chi07] David Chisnall. *The Definitive Guide to the Xen Hypervisor*. Open Source Software Development Series. Prentice Hall, 2007. (Cité page 49.)

[CKL96] Shigeru Chiba, Gregor Kiczales, and John Lamping. Avoiding confusion in metacircularity : The meta-helix. In Kokichi Futatsugi and Satoshi Matsuoka, editors, *Proceedings of ISOTAS '96*, volume 1049 of *Lecture Notes in Computer Science*, pages 157–172. Springer, 1996. (Cité pages 11 et 44.)

[Coi87] Pierre Cointe. Metaclasses are first class : the ObjVlisp model. In *Proceedings OOPSLA '87, ACM SIGPLAN Notices*, volume 22, pages 156–167, December 1987. (Cité page 42.)

[CPJ+02] Dipanjan Chakraborty, Filip Perich, Anupam Joshi, Timothy W. Finin, and Yelena Yesha. A reactive service composition architecture for pervasive computing environments. In *Proceedings of the IFIP TC6/WG6.8 Working Conference on Personal Wireless Communications*, PWC '02, pages 53–62, Deventer, The Netherlands, The Netherlands, 2002. Kluwer, B.V. (Cité page 31.)

[CSVC10] Ivica Crnković ;, Severine Sentilles, Aneta Vulgarakis, and Michel R.V. Chaudron. A classification framework for software component models. *IEEE Transactions on Software Engineering*, 99(PrePrints), 2010. (Cité page 16.)

[CVRF14a] Micael S. Couceiro, Patricia A. Vargas, Rui P. Rocha, and Nuno M. F. Ferreira. Benchmark of swarm robotics distributed techniques in a search task. *Robotics and Autonomous Systems*, 62(2) :200–213, January 2014. (Cité pages 98 et 99.)

[CVRF14b] Micael S. Couceiro, Patricia A. Vargas, Rui P. Rocha, and Nuno M.F. Ferreira. Benchmark of swarm robotics distributed techniques in a search task. *Robotics and Autonomous Systems*, 62(2) :200–213, February 2014. (Cité page 93.)

[DBD+09] Arnaud Doniec, Noury Bouraqadi, Michael Defoort, Van Tuan Le, and Serge Stinckwich. Distributed constraint reasoning applied to multi-robot exploration. In *Proceedings of ICTAI'09*, pages 159–166, 2009. :ictai2009 :ictai2009 :ictai2009 :ictai2009

[Dec86] Dominique Decouchant. Design of a distributed object manager for the Smalltalk-80 system. In *Conference proceedings on Object-oriented programming systems, languages and applications*, OOPLSA '86, pages 444–452, New York, NY, USA, 1986. ACM. (Cité page 72.)

[Den70] Peter J. Denning. Virtual memory. *ACM Comput. Surv.*, 2(3) :153–189, September 1970. (Cité page 68.)

[Den08] Marcus Denker. *Sub-method Structural and Behavioral Reflection*. PhD thesis, University of Bern, May 2008. (Cité page 10.)

[DGG95] KlausR. Dittrich, Stella Gatziu, and Andreas Geppert. The active database management system manifesto : A rulebase of adbms features. In Timos Sellis, editor, *Rules in Database*

Systems, volume 985 of *Lecture Notes in Computer Science*, pages 1–17. Springer Berlin Heidelberg, 1995. (Cité page 44.)

[DGL+07] Marcus Denker, Tudor Gîrba, Adrian Lienhard, Oscar Nierstrasz, Lukas Renggli, and Pascal Zumkehr. Encapsulating and exploiting change with Changeboxes. In *Proceedings of the 2007 International Conference on Dynamic Languages (ICDL 2007)*, pages 25–49. ACM Digital Library, 2007. (Cité page 50.)

[DHU+07] Nicolas Desnos, Marianne Huchard, Christelle Urtado, Sylvain Vauttier, and Guy Tremblay. Automated and unanticipated flexible component substitution. In H. W. Schmidt et al., editors, *Proceedings of the 10th ACM SIGSOFT Symposium on Component-Based Software Engineering (CBSE2007)*, volume 4608 of *LNCS*, pages 33–48, Medford, MA, USA, July 2007. Springer. (Cité page 17.)

[DL02] Pierre-Charles David and Thomas Ledoux. An infrastructure for adaptable middleware. In *On the Move to Meaningful Internet Systems 2002 : CoopIS, DOA, and ODBASE*, volume 2519 of *Lecture Notes in Computer Science*, pages 773–790. Springer Berlin Heidelberg, 2002. (Cité page 85.)

[DSD08] Marcus Denker, Mathieu Suen, and Stéphane Ducasse. The meta in meta-object architectures. In *Proceedings of TOOLS EUROPE 2008*, volume 11 of *LNBIP*, pages 218–237. Springer-Verlag, 2008. (Cité pages 44 et 49.)

[DVCM+05] Jessie Dedecker, Tom Van Cutsem, Stijn Mostinckx, Theo D'Hondt, and Wolfgang De Meuter. Ambient-oriented programming. In *Companion to the 20th Annual ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications*, OOPSLA '05, pages 31–40, New York, NY, USA, 2005. ACM. (Cité pages 9 et 41.)

[ECF+14] Pablo Estefo, Miguel Campusano, Luc Fabresse, Johan Fabry, Jannik Laval, and Noury Bouraqadi. Towards live programming in ROS with pharos and LRP. In *5th International Workshop on Domain-Specific Languages and models for ROBotic systems (DSLRob-14)*, volume abs/1412.4629, 2014. (Cité page 102.)

[EKO95] Dawson R. Engler, M. Frans Kaashoek, and James O'Tool. Exokernel : An operating system architecture for application-level resource management. In *SOSP*, pages 251–266, 1995. (Cité page 71.)

[ELD+12a] Gilberto Echeverria, Séverin Lemaignan, Arnaud Degroote, Simon Lacroix, Michael Karg, Pierrick Koch, Charles Lesire, and Serge Stinckwich. Simulating complex robotic scenarios with morse. In *SIMPAR*, pages 197–208, 2012. (Cité page 85.)

[ELD+12b] Gilberto Echeverria, Severin Lemaignan, Arnaud Degroote, Simon Lacroix, Michael Karg, Pierrick Koch, Charles Lesire, and Serge Stinckwich. Simulating complex robotic scenarios with MORSE. In *In Proceedings of the 3rd International Conference on Simulation, Modeling, and Programming for Autonomous Robots (SIMPAR 2012)*, pages 197–208, Tsukuba, Japan, November 2012. (Cité pages 91 et 95.)

[Fab07] Luc Fabresse. *Du découpage à l'assemblage non anticipé de composants : Conception et mise en oeuvre du langage à composants Scl*. PhD thesis, Université de Montpellier 2, dec 2007. (Cité pages 8, 15, 24 et 26.)

[Fau12] Matthieu Faure. *Management of Scenarized User-centric Service Compositions for Collaborative Pervasive Environments*. PhD thesis, LIRMM & Ecole des Mines d'Alès & Ecole des Mines de Douai & Université Montpellier 2 - France, December 2012. (Cité pages 15, 29, 31, 35, 36 et 106.)

[FBC+09] Daniel Frampton, Stephen M. Blackburn, Perry Cheng, Robin J. Garner, David Grove, Eliot, and Sergey I. Salishev. Demystifying magic : high-level low-level programming. In *Proceedings of the 2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, VEE '09, pages 81–90, New York, NY, USA, 2009. ACM. (Cité pages 7 et 49.)

[FBDH10] Luc Fabresse, Noury Bouraqadi, Christophe Dony, and Marianne Huchard. Component-Oriented Programming : From Requirements to Language Support. In Marcus Denker and Gabriela Arévalo, editors, *Proceedings of International Conference on Smalltalk Technologies (Smalltalks'10)*, Concepcion del Uruguay, Argentina, 2010. (Cité pages 8, 18 et 36.)

[FBDH12] Luc Fabresse, Noury Bouraqadi, Christophe Dony, and Marianne Huchard. A language to bridge the gap between component-based design and implementation. *Journal of Computer Languages, Systems and Structures*, 38(1) :29–43, jan 2012. (Cité pages 18, 24, 26 et 36.)

[FDH08] Luc Fabresse, Christophe Dony, and Marianne Huchard. Foundations of a simple and unified component-oriented lan-

guage. *International Journal of Computer Languages, Systems and Structures*, 34(2-3) :130–149, 2008. (Cité page 21.)

[FFH+10] Matthieu Faure, Luc Fabresse, Marianne Huchard, Christelle Urtado, and Sylvain Vauttier. Towards Scenario Creation by Service Composition in Ubiquitous Environments. In L. Duchien S. Ducasse and L. Seinturier, editors, *Proceedings of the 9th BElgian-NEtherlands software eVOLution seminar (BENEVOL 2010)*, pages 145–155, Lille, France, December 2010. (Cité pages 32 et 36.)

[FFH+11a] Matthieu Faure, Luc Fabresse, Marianne Huchard, Christelle Urtado, and Sylvain Vauttier. Construction d'applications distribuees a base de composants par composition de services. Technical report, Ecole des Mines de Douai, Jan 2011. (Cité page 36.)

[FFH+11b] Matthieu Faure, Luc Fabresse, Marianne Huchard, Christelle Urtado, and Sylvain Vauttier. A service component framework for multi-user scenario management in ubiquitous environments. In *Proceedings of the 6th International Conference on Software Engineering Advances (ICSEA 2011)*, Barcelona, Spain, October 2011. XPS (Xpert Publishing Services). Acceptance rate : 30%. (Cité pages 32 et 36.)

[FFH+11c] Matthieu Faure, Luc Fabresse, Marianne Huchard, Christelle Urtado, and Sylvain Vauttier. The SaS Platform for Ubiquitous Environments. Technical report, Ecole des Mines de Douai, April 2011. (Cité page 36.)

[FFH+11d] Matthieu Faure, Luc Fabresse, Marianne Huchard, Christelle Urtado, and Sylvain Vauttier. User-defined scenarios in ubiquitous environments : Creation, execution control and sharing. In *Proceedings of the 23rd International Conference on Software Engineering and Knowledge Engineering (SEKE 2011)*, pages 302–307, Miami, USA, July 2011. Acceptance rate : 31%. (Cité pages 32 et 36.)

[FHF+12] Matthieu Faure, Marianne Huchard, Luc Fabresse, Christelle Urtado, and Sylvain Vauttier. Composition de services scénarisée et centrée utilisateur pour les environnements pervasifs collaboratifs. Technical report, Ecole des Mines de Douai, Dec 2012. (Cité page 36.)

[FLB13a] Luc Fabresse, Jannik Laval, and Noury Bouraqadi. Towards quality assurance tests for mobile robots. In *Proceedings of 8th National Conference on "Control Architecture of Robots" Control*

Architectures of Robots (CAR'13), Angers, France, June 2013. (Cité pages 102 et 103.)

[FLB13b] Luc Fabresse, Jannik Laval, and Noury Bouraqadi. Towards Test-Driven Development for Mobile Robots. In Davide Brugali, editor, *Proceedings of the eighth full-day Workshop on Software Development and Integration in Robotics (SDIR VIII), in conjunction with the IEEE International Conference on Robotics and Automation (ICRA)*, pages 12–13, Karlsruhe, Germany, May 2013. (Cité pages 102 et 103.)

[FLHW10a] Sebastian Frank, Kim Listmann, Dominik Haumann, and Volker Willert. Performance analysis for multi-robot exploration strategies. In *proceedings of SIMPAR'10*, pages 399–410. Springer, 2010. (Cité page 93.)

[FLHW10b] Sebastian Frank, Kim D. Listmann, A. Dominik Haumann, and Volker Willert. Performance analysis for multi-robot exploration strategies. In *In Proceedings of the 2nd International Conference on Simulation, Modeling, and Programming for Autonomous Robots (SIMPAR 2010)*, pages 399–410, Darmstadt, Germany, November 2010. (Cité page 98.)

[FSC14] Jan Faigl, Olivier Simonin, and François Charpillet. Comparison of task-allocation algorithms in frontier-based multi-robot exploration. In *Proceedings of EUMAS'14*. Springer, 2014. (Cité page 93.)

[FSOM10] Marc Freese, Surya P. N. Singh, Fumio Ozaki, and Nobuto Matsuhira. Virtual robot experimentation platform V-REP : A versatile 3d robot simulator. In *In Proceedings of the 2nd International Conference on Simulation, Modeling, and Programming for Autonomous Robots (SIMPAR 2010)*, pages 51–62, Darmstadt, Germany, November 2010. (Cité page 91.)

[GBV06] Guillaume Grondin, Noury Bouraqadi, and Laurent Vercouter. Madcar : an abstract model for dynamic and automatic (re-)assembling of component-based applications. In *Proceedings of the 9th International Symposium on CBSE (Component-Based Software Engineering)*, LNCS, pages 360–367, Sweden, June 2006. Springer. (Cité page 17.)

[GDDC97] David Grove, Greg DeFouw, Jeffrey Dean, and Craig Chambers. Call graph construction in object-oriented languages. In *Proceedings of the 12th ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications*,

OOPSLA '97, pages 108–124, New York, NY, USA, 1997. ACM. (Cité pages 12 et 61.)

[GHJV95a] Erich Gamma, Richard Helem, Ralph Johnson, and John Vlissides. *Design Patterns – Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995. (Cité page 40.)

[GHJV95b] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns : Elements of Reusable Object-Oriented Software*. Addison Wesley, March 1995. (Cité pages 20 et 89.)

[GNM+03] Xiaohui Gu, Klara Nahrstedt, Alan Messer, Ira Greenberg, and Dejan Milojicic. Adaptive offloading inference for delivering applications in pervasive computing environments. In *Proceedings of the First IEEE International Conference on Pervasive Computing and Communications*, PERCOM '03, pages 107–, Washington, DC, USA, 2003. IEEE Computer Society. (Cité page 61.)

[Gro08a] Guillaume Grondin. *MaDcAr-Agent : un modèle d'agents auto-adaptables à base de composants*. PhD thesis, Ecole des Mines de Saint Etienne, November 2008. (Cité page 17.)

[Gro08b] Guillaume Grondin. *Un modèle d'agents auto-adaptables à base de composants*. PhD thesis, Ecole des Mines de Saint Etienne, November 2008. Thèse co-encadrée par Noury Bouraqadi (Mines Douai) et Laurent Vercouter (Ecole des Mines de St Etienne) et sous la direction d'Olivier Boissier (Ecole des Mines de St Etienne). (Cité page 36.)

[GVH03] Brian P. Gerkey, Richard T. Vaughan, and Andrew Howard. The Player/Stage project : Tools for multi-robot and distributed sensor systems. In *Proceedings of the 11th International Conference on Advanced Robotics (ICAR'03)*, pages 317–323, Coimbra, Portugal, June 2003. (Cité pages 91 et 92.)

[Ham97] Graham Hamilton. JavaBeans. API Specification, Sun Microsystems, July 1997. Version 1.01. (Cité page 19.)

[HC01] George T. Heineman and William T. Councill, editors. *Component-based software engineering : putting the pieces together*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001. (Cité page 8.)

[HFB05] Matthew Hertz, Yi Feng, and Emery D. Berger. Garbage collection without paging. In *Proceedings of the 2005 ACM SIGPLAN conference on Programming language design and implementation*,

PLDI '05, pages 143–153, New York, NY, USA, 2005. ACM. (Cité page 71.)

[HHUV09] Fady Hamoui, Marianne Huchard, Christelle Urtado, and Sylvain Vauttier. Specification of a Component-based Domotic System to Support User-Defined Scenarios. In Du Zhang Guido Wirtz, Jerry Gao, editor, *SEKE 2009 : 21st International Conference on Software Engineering and Knowledge Engineering*, pages 597–602, Boston, USA, July 2009. Knowledge Systems Institute Graduate School. (Cité page 31.)

[HK05] Thomas Heider and Thomas Kirste. Multimodal appliance cooperation based on explicit goals : concepts & potentials. In *Proceedings of the 2005 joint conference on Smart objects and ambient intelligence : innovative context-aware services : usages and technologies*, pages 271 – 275, 2005. (Cité page 31.)

[HL07] Galen C. Hunt and James R. Larus. Singularity : Rethinking the software stack. *SIGOPS Oper. Syst. Rev.*, 41(2) :37–49, April 2007. (Cité page 11.)

[HMB90] Antony L. Hosking, J. E Moss, and Cynthia Bliss. Design of an object faulting persistent Smalltalk. Technical report, University of Massachusetts, Amherst, MA, USA, 1990. (Cité page 69.)

[How06] A. Howard. Multi-robot simultaneous localization and mapping using particle filters. *The International Journal of Robotics Research*, 25 :1243–1256, December 2006. (Cité page 90.)

[Hum99] Watts S. Humphrey. Bugs or defects ? *Technical Report Vol. 2, Issue 1*, 1999. (Cité page 83.)

[IKM+97] Dan Ingalls, Ted Kaehler, John Maloney, Scott Wallace, and Alan Kay. Back to the future : The story of Squeak, a practical Smalltalk written in itself. In *Proceedings of the 12th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications (OOPSLA'97)*, pages 318–326. ACM Press, November 1997. (Cité pages 49 et 72.)

[Jav] Java micro edition. http ://java.sun.com/javame/index.jsp. (Cité page 71.)

[JE07] Diane Jordan and John Evdemon. Web services business process execution language version 2.0, 2007. (Cité page 33.)

[Kae86] Ted Kaehler. Virtual memory on a narrow machine for an object-oriented language. *Proceedings OOPSLA '86, ACM SIGPLAN Notices*, 21(11) :87–106, November 1986. (Cité pages 12, 69 et 71.)

[KCBC02] Fabio Kon, Fabio Costa, Gordon Blair, and Roy H. Campbell. The case for reflective middleware. *Commun. ACM*, 45(6) :33–38, June 2002. (Cité page 85.)

[KdRB91] Gregor Kiczales, Jim des Rivières, and Daniel G. Bobrow. *The Art of the Metaobject Protocol*. MIT Press, 1991. (Cité pages 7, 8, 10 et 11.)

[KGC13] Michal Kudelski, Luca Maria Gambardella, and Gianni A. Di Caro. RoboNetSim : An integrated framework for multi-robot and network simulation. *Robotics and Autonomous Systems*, 61(5) :483–496, January 2013. (Cité page 92.)

[KH04a] Nathan Koenig and Andrew Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *In IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2149–2154, 2004. (Cité page 85.)

[KH04b] Nathan P. Koenig and Andrew Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *Proceedings of the 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'04)*, pages 2149–2154, Sendai, Japan, September 2004. (Cité pages 91 et 92.)

[KLVA93] Keith Krueger, David Loftesness, Amin Vahdat, and Thomas Anderson. Tools for the development of application-specific virtual memory management. In *Proceedings OOPSLA '93, ACM SIGPLAN Notices*, volume 28, pages 48–64, October 1993. (Cité page 71.)

[KMvSK11] S. Kohlbrecher, J. Meyer, O. von Stryk, and U. Klingauf. A flexible and scalable slam system with full 3d motion estimation. In *Proc. IEEE International Symposium on Safety, Security and Rescue Robotics (SSRR)*. IEEE, November 2011. (Cité page 102.)

[LB85] M. M. Lehman and L. A. Belady, editors. *Program evolution : processes of software change*. Academic Press Professional, Inc., San Diego, CA, USA, 1985. (Cité page 19.)

[Lee08] Edward A. Lee. Cyber physical systems : Design challenges. Technical Report UCB/EECS-2008-8, EECS Department, University of California, Berkeley, Jan 2008. (Cité page 1.)

[LFB13] Jannik Laval, Luc Fabresse, and Noury Bouraqadi. A methodology for testing mobile autonomous robots. In *Proceedings of the 2013 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2013)*, Tokyo, Japan, November 2013. IEEE. Acceptance rate : 43%. (Cité pages 102 et 103.)

[LLB+14] Xuan Sang Le, Loïc Lagadec, Noury Bouraqadi, Luc Fabresse, and Jannik Laval. From smalltalk to silicon : Towards a methodology to turn smalltalk code into fpga. In *Proceedings of International Workshop on Smalltalk Technologies (IWST'14)*, Cambridge, England, September 2014. (Cité page 80.)

[LLF+15] Xuan Sang Le, Loïc Lagadec, Luc Fabresse, Jannik Laval, and Noury Bouraqadi. A meta-model supporting both hardware and smalltalk-based execution of fpga circuits. In *Proceedings of International Workshop on Smalltalk Technologies (IWST'15)*, Cambridge, England, July 2015. 1st price at the Best Paper Award of IWST'15. (Cité page 80.)

[LOW97] Jacob Y. Levy, John K. Ousterhout, and Brent B. Welch. The safe-tcl security model. Technical report, Sun Microsystems, Inc., Mountain View, CA, USA, 1997. (Cité page 50.)

[LP90] Wilf R. LaLonde and John R. Pugh. *Inside Smalltalk : vol. 1*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1990. (Cité page 87.)

[LSR09] Robert N Lass, Evan A Sultanik, and William C Regli. Metrics for multiagent systems. In *Performance Evaluation and Benchmarking of Intelligent Systems*, pages 1–19. Springer, 2009. (Cité page 94.)

[LV03] David H. Lorenz and John Vlissides. Pluggable reflection : Decoupling meta-interface and implementation. In *Proceedings of International Conference on Software Engineering (ICSE)*, pages 3–13, Portland, Oregon, May 2003. IEEE Computer Society. (Cité pages 11 et 38.)

[LW05] Kung-Kiu Lau and Zheng Wang. A taxonomy of software component models. In *31st EUROMICRO Conference on Software Engineering and Advanced Applications (EUROMICRO-SEAA 2005), 30 August - 3 September 2005, Porto, Portugal*, pages 88–95. IEEE Computer Society, 2005. (Cité page 16.)

[LWL05] Benjamin Livshits, John Whaley, and Monica S. Lam. Reflection analysis for java. In *Proceedings of the Third Asian Conference on Programming Languages and Systems*, APLAS'05, pages

139–160, Berlin, Heidelberg, 2005. Springer-Verlag. (Cité page 11.)

[Mae87] P. Maes. Concepts and experiments in computational reflection. In *Proceedings of OOPSLA'87*, pages 147–155, Orlando, Florida, October 1987. ACM. (Cité pages 7, 11, 38 et 41.)

[MBMZ00] Alonso Marquez, Stephen M Blackburn, Gavin Mercer, and John Zigman. Implementing orthogonally persistent Java. In *In Proceedings of the Workshop on Persistent Object Systems (POS*, pages 218–232, 2000. (Cité page 72.)

[McA95] Jeff McAffer. Meta-level programming with coda. In W. Olthoff, editor, *Proceedings ECOOP '95*, volume 952 of *LNCS*, pages 190–214, Aarhus, Denmark, August 1995. Springer-Verlag. (Cité page 85.)

[McC60] John McCarthy. Recursive functions of symbolic expressions and their computation by machine, part I. *CACM*, 3(4) :184–195, April 1960. (Cité pages 6 et 69.)

[McI68] M. D. McIlroy. Mass produced software components. In P. Naur and B. Randell, editors, *Proceedings, NATO Conference on Software Engineering*, Garmisch, Germany, October 1968. (Cité pages 8 et 16.)

[MCTT07] Stijn Mostinckx, Tom Van Cutsem, Stijn Timbermont, and Eric Tanter. Mirages : Behavioral intercession in a mirror-based architecture. In *Proceedings the ACM Dynamic Languages Symposium (DLS 2007)*, October 2007. (Cité page 87.)

[MFK08] Nathan Michael, Jonathan Fink, and Vijay Kumar. Experimental testbed for large multirobot teams. *IEEE Robotics & Automation Magazine*, 15(1) :53–61, March 2008. (Cité page 92.)

[MH99] Richard Monson-Haefel. *Enterprise JavaBeans*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 1999. (Cité page 19.)

[MHS05] Marjan Mernik, Jan Heering, and Anthony M. Sloane. When and how to develop domain-specific languages. *ACM Computer Surveys*, 37(4) :316–344, 2005. (Cité page 7.)

[Mic12a] James Mickens. Rivet : Browser-agnostic remote debugging for web applications. In *USENIX Annual Technical Conference*, pages 333–345, 2012. (Cité page 87.)

[Mic12b] Microsoft. How to : Set up remote debugging, visual studio 2012. `http://msdn.microsoft.com/en-us/library/bt727f1t.aspx`, 2012. (Cité page 86.)

[MJD96] J. Malenfant, M. Jacques, and F.-N. Demers. A tutorial on behavioral reflection and its implementation. In *In proceedings of Reflection*, pages 1–20, 1996. (Cité page 10.)

[Moo65] Gordon E. Moore. Cramming more components onto integrated circuits. *Electronics*, 38(8) :114–117, april 1965. (Cité page 1.)

[MP12] Mariano Martinez Peck. *Application-Level Virtual Memory for Object-Oriented Systems*. PhD thesis, Ecole des Mines de Douai - France & Université Lille 1 - France, October 2012. (Cité pages 58, 69, 73, 80 et 106.)

[MPBD+10a] Mariano Martinez Peck, Noury Bouraqadi, Marcus Denker, Stéphane Ducasse, and Luc Fabresse. Experiments with a fast object swapper. In *Proceedings of International Conference on Smalltalk Technologies (Smalltalks'10)*, 2010. (Cité pages 73 et 80.)

[MPBD+10b] Mariano Martinez Peck, Noury Bouraqadi, Marcus Denker, Stéphane Ducasse, and Luc Fabresse. Visualizing objects and memory usage. In *Proceedings of International Conference on Smalltalk Technologies (Smalltalks'10)*, 2010. (Cité pages 69 et 80.)

[MPBDF12] Mariano Martinez Peck, Noury Bouraqadi, Stéphane Ducasse, and Luc Fabresse. Object swapping challenges : an evaluation of imagesegment. *Journal of Computer Languages, Systems and Structures*, 38(1) :1–15, January 2012. (Cité pages 73 et 80.)

[MSL+08a] Mark S. Miller, Mike Samuel, Ben Laurie, Ihab Awad, and Mike Stay. Caja : Safe active content in sanitized javascript, 2008. Technical Report, Google Inc. (Cité page 11.)

[MSL+08b] Mark S. Miller, Mike Samuel, Ben Laurie, Ihab Awad, and Mike Stay. Caja safe active content in sanitized javascript. Technical report, Google Inc., 2008. (Cité page 50.)

[MT00] Nenad Medvidovic and Richard N. Taylor. A Classification and Comparison Framework for Software Architecture Description Languages. *Software Engineering*, 26(1) :70–93, 2000. (Cité page 18.)

[NBD+11] Papoulias Nikolaos, Noury Bouraqadi, Marcus Denker, Stéphane Ducasse, and Luc Fabresse. Towards structural decomposition of reflection with mirrors. In *Proceedings of International Workshop on Smalltalk Technologies (IWST'11)*, Edingburgh, United Kingdom, 2011. (Cité pages 42 et 56.)

[NDG+08] Oscar Nierstrasz, Marcus Denker, Tudor Gîrba, Adrian Lienhard, and David Röthlisberger. Change-enabled software systems. In Martin Wirsing, Jean-Pierre Banâtre, and Matthias Hölzl, editors, *Challenges for Software-Intensive Systems and New Computing Paradigms*, volume 5380 of *LNCS*, pages 64–79. Springer-Verlag, 2008. (Cité page 46.)

[NDP10] Oscar Nierstrasz, Stephane Ducasse, and Damien Pollet. *Pharo by Example*. Square Bracket Associates, July 2010. (Cité page 21.)

[NS04] T. Nakajima and I. Satoh. Personal home server : enabling personalized and seamless ubiquitous computing environments. In *Pervasive Computing and Communications, 2004. PerCom 2004. Proceedings of the Second IEEE Annual Conference on*, pages 341–345, March 2004. (Cité page 31.)

[OaRBTS08] Takashi Okada, Junya Nakata abd Razvan Beuran, Yasuo Tan, and Yoichi Shinoda. Large-scale simulation method of mobile robots. In *Proceeding of the IEEE 2nd International Symposium on Universal Communication (ISUC'08)*, Osaka, Japan, December 2008. (Cité page 92.)

[OMG02a] Object Management Group. *Manual of CORBA Component Model V3.0*, 2002. *http ://www.omg.org/technology/documents/formal/components.htm*. (Cité page 20.)

[Omg02b] Omg. OMG IDL Syntax and Semantics. Technical report, OMG, 2002. (Cité page 19.)

[OMG12] OMG. OMG Object Constraint Language (OCL), Version 2.3.1, January 2012. (Cité page 28.)

[Ora13] Oracle. Java platform debugger architecture (jpda). `http://docs.oracle.com/javase/7/docs/technotes/guides/jpda/`, 2013. (Cité pages 50 et 86.)

[Ous05] M. Oussalah. *Ingénierie des composants : concepts, techniques et outils*. Editions Vuibert, 2005. (Cité page 17.)

[PADLR08] Robin Passama, David Andreu, Christophe Dony, and Thérèse Libourel Rouge. COSARC : Un langage de composants pour l'ingénierie des architectures robotiques. Pourquoi utiliser des composants, et quels composants ? *Journal Européen des Systèmes Automatisés*, 42 :439–458, 2008. (Cité page 36.)

[Pap13] Nikolaos Papoulias. *Remote Debugging and Reflection in Resource Constrained Devices*. Theses, Ecole des Mines de Douai

& Université des Sciences et Technologie de Lille - Lille I - France, December 2013. (Cité pages 37, 39, 42, 56, 81, 87, 88, 103 et 107.)

[Par08] Lynne E Parker. Multiple mobile robot systems. *Springer Handbook of Robotics*, pages 921–941, 2008. (Cité page 90.)

[PBD+11a] Mariano Martinez Peck, Noury Bouraqadi, Marcus Denker, Stéphane Ducasse, and Luc Fabresse. Efficient proxies in smalltalk. In *Proceedings of ESUG International Workshop on Smalltalk Technologies (IWST 2011)*, Edinburgh, Scotland, 2011. (Cité pages 74 et 80.)

[PBD+11b] Mariano Martinez Peck, Noury Bouraqadi, Marcus Denker, Stéphane Ducasse, and Luc Fabresse. Problems and challenges when building a manager for unused objects. In *Proceedings of International Conference on Smalltalk Technologies (Smalltalks'11)*, Bernal, Buenos Aires, Argentina, 2011. (Cité pages 73 et 80.)

[PBD+13] Mariano Martinez Peck, Noury Bouraqadi, Marcus Denker, Stéphane Ducasse, and Luc Fabresse. Marea : An efficient application-level object graph swapper. *Journal of Object Technology*, 12(1) :2 :1–30, jan 2013. (Cité pages 61, 73, 78 et 80.)

[PBDF14] Guillermo Polito, Noury Bouraqadi, Stéphane Ducasse, and Luc Fabresse. Understanding pharo's global state to move programs through time and space. In *Proceedings of International Workshop on Smalltalk Technologies (IWST'14)*, Cambridge, England, September 2014. (Cité page 56.)

[PBF+14] Mariano Martinez Peck, Noury Bouraqadi, Luc Fabresse, Marcus Denker, and Camille Teruel. Ghost : A uniform and general-purpose proxy implementation. *Science of Computer Programming*, 98(3) :339–359, 2014. (Cité pages 74 et 80.)

[PBF+15] Nick Papoulias, Noury Bouraqadi, Luc Fabresse, Stéphane Ducasse, and Marcus Denker. Mercury : Properties and Design of a Remote Debugging Solution using Reflection. *Journal of Object Technology*, 14(2) :1 :1–36, May 2015. (Cité pages 88 et 103.)

[PDB+15] Guillermo Polito, Stéphane Ducasse, Noury Bouraqadi, Luc Fabresse, and Max Mattone. Virtualization support for dynamic core library update. In *Proceedings of the ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software (SPLASH/OnWard !15)*. ACM, 2015. (Cité page 56.)

[PDBF14] Guillermo Polito, Stéphane Ducasse, Noury Bouraqadi, and Luc Fabresse. Extended results of Tornado : A Run-Fail-Grow approach for Dynamic Application Tayloring. Research report, Inria, 2014. (Cité page 80.)

[PDBF15] Guillermo Polito, Stéphane Ducasse, Noury Bouraqadi, and Luc Fabresse. A bootstrapping infrastructure to build and extend pharo-like languages. In *Proceedings of the ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software (SPLASH/OnWard !15)*. ACM, 2015. (Cité pages 53, 55 et 56.)

[PDDF15] N. Papoulias, M. Denker, S. Ducasse, and L. Fabresse. Reifying the reflectogram : Towards explicit control for implicit reflection. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, SAC'15, pages 1978–1985, New York, NY, USA, 2015. ACM. Acceptance rate : 24%. (Cité pages 45 et 56.)

[PDF+14] Guillermo Polito, Stéphane Ducasse, Luc Fabresse, Noury Bouraqadi, and Benjamin Van Ryseghem. Bootstrapping Reflective Systems : The Case of Pharo. *Science of Computer Programming*, 96(1) :141–155, dec 2014. (Cité pages 50 et 56.)

[PDFB13] Guillermo Polito, Stéphane Ducasse, Luc Fabresse, and Noury Bouraqadi. Virtual Smalltalk Images : Model and Applications. In *Proceedings of International Workshop on Smalltalk Technologies (IWST'13)*, Annecy, France, September 2013. (Cité pages 51, 53 et 56.)

[Pob06] Angel P. Del Pobil. Why do we need benchmarks in robotics research. In *Proceedings of the IROS'06 Workshop on Benchmarks in Robotics Research*, Beijing, China, October 2006. (Cité page 91.)

[Pol15] Guillermo Polito. *Virtualization Support for Application Runtime Specialization and Extension*. Theses, Ecole des Mines de Douai & Université des Sciences et Technologie de Lille - Lille I - France, April 2015. (Cité pages 37, 50, 56, 58, 59, 62, 80 et 106.)

[PRT+04] Lucian Popa, Costin Raiciu, Radu Teodorescu, Irina Athanasiu, and Raju Pandey. Using code collection to support large applications on mobile devices. In *Proceedings of the 10th Annual International Conference on Mobile Computing and Networking*, MobiCom '04, pages 16–29, New York, NY, USA, 2004. ACM. (Cité page 61.)

[PTP07] Guillaume Pothier, Éric Tanter, and José Piquer. Scalable omniscient debugging. *Proceedings of the 22nd Annual SCM SIGPLAN Conference on Object-Oriented Programming Systems, Languages and Applications (OOPSLA'07)*, 42(10) :535–552, 2007. (Cité page 86.)

[RBN12] Jorge Ressia, Alexandre Bergel, and Oscar Nierstrasz. Object-centric debugging. In *Proceeding of the 34rd international conference on Software engineering*, ICSE '12, 2012. (Cité pages 82 et 87.)

[RDT08] David Röthlisberger, Marcus Denker, and Éric Tanter. Unanticipated partial behavioral reflection : Adapting applications at runtime. *Journal of Computer Languages, Systems and Structures*, 34(2-3) :46–65, July-October 2008. (Cité page 10.)

[RHT+13] Daniel Romero, Gabriel Hermosillo, Amirhosein Taherkordi, Russel Nzekwa, Romain Rouvoy, and Frank Eliassen. The DigiHome Service-Oriented Platform. *Software : Practice and Experience*, 43(10) :1143–1239, October 2013. (Cité page 31.)

[Riv96] Fred Rivard. Smalltalk : a reflective language. In *Proceedings of REFLECTION '96*, pages 21–38, April 1996. (Cité page 7.)

[RK02] Derek Rayside and Kostas Kontogiannis. Extracting java library subsets for deployment on embedded systems. *Sci. Comput. Program.*, 45(2-3) :245–270, November 2002. (Cité page 61.)

[RKC01] Manuel Rom, Fabio Kon, and Roy H. Campbell. Reflective middleware : From your desk to your hand. *IEEE Distributed Systems Online*, 2(5), 2001. (Cité page 85.)

[RRGN10] Jorge Ressia, Lukas Renggli, Tudor Gîrba, and Oscar Nierstrasz. Run-time evolution through explicit meta-objects. In *Proceedings of the 5th Workshop on Models@run.time at the ACM/IEEE 13th International Conference on Model Driven Engineering Languages and Systems (MODELS 2010)*, pages 37–48, October 2010. http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-641/. (Cité pages 85 et 87.)

[RS03] Stan Shebs Richard Stallman, Roland Pesch. *Debugging with GDB*. Gnu Press, 2003. (Cité page 85.)

[SC00] João Costa Seco and Luís Caires. A basic model of typed components. *Lecture Notes in Computer Science*, 1850 :108–129, 2000. (Cité page 20.)

[SD10] Olivier Sallenave and Roland Ducournau. Efficient compilation of .net programs for embedded systems. In *Proceedings of the Workshop on the Implementation, Compilation, Optimization of Object-Oriented Languages, Programs and Systems*, ICOOOLPS '10, pages 3 :1–3 :8, New York, NY, USA, 2010. ACM. (Cité page 61.)

[SDK+95] Mary Shaw, Robert DeLine, Daniel V. Klein, Theodore L. Ross, David M. Young, and Gregory Zelesnik. Abstractions for Software Architecture and Tools to Support Them. *Software Engineering*, 21(4) :314–335, 1995. (Cité pages 16 et 18.)

[SDNB03] Nathanael Schärli, Stéphane Ducasse, Oscar Nierstrasz, and Andrew P. Black. Traits : Composable units of behavior. In *Proceedings of European Conference on Object-Oriented Programming*, volume 2743 of *ECOOP'03*, pages 248–274. Springer Verlag, July 2003. (Cité pages 16 et 46.)

[SDTF11] Petr Spacek, Christophe Dony, Chouki Tibermacine, and Luc Fabresse. A smalltalk implementation of exil, a component-based programming language. In *Proceedings of ESUG International Workshop on Smalltalk Technologies (IWST 2011)*, Edinburgh, Scotland, August 2011. ESUG. (Cité page 36.)

[SDTF12] Petr Spacek, Christophe Dony, Chouki Tibermacine, and Luc Fabresse. An inheritance system for structural & behavioral reuse in component-based software programming. In *In proceedings of the 11th International Conference on Generative Programming and Component Engineering (GPCE'12)*, Dresden, Germany, September 2012. ACM Press. Acceptance rate : 40%. (Cité pages 27 et 36.)

[SDTF13] Petr Spacek, Christophe Dony, Chouki Tibermacine, and Luc Fabresse. Wringing out Objects for Programming and Modeling Component-based Systems. In *Proceedings of the Second International Workshop on Combined Object-Oriented Modeling and Programming Languages (COOMPL at ECOOP 2013)*, Montpellier, France, July 2013. (Cité pages 28 et 36.)

[SK08] Bruno Siciliano and Oussama Khatib, editors. *Handbook of Robotics*. Springer, 2008. (Cité page 1.)

[SMDV06] J. Sillito, G.C. Murphy, and K. De Volder. Questions programmers ask during software evolution tasks. In *Proceedings of the 14th International Symposium on Foundations on Software Engineering*, SIGSOFT '06/FSE-14, pages 23–34. ACM, 2006. (Cité page 82.)

[Smi82a] Brian Cantwell Smith. *Procedural Reflection in Programming Languages*. PhD thesis, Massachusetts Institute of Technology, Laboratory for Computer Science, 1982. (Cité page 7.)

[Smi82b] Brian Cantwell Smith. *Reflection and Semantics in a Procedural Language*. Ph.D. thesis, MIT, Cambridge, MA, 1982. (Cité page 48.)

[Smi84] B. C. Smith. Reflection and semantics in lisp. In *Proceedings of the 14th Annual ACM Symposium on Principles of Programming Languages, POPL'84*, pages 23–35, Salt Lake City, Utah, USA, January 1984. (Cité page 6.)

[SML+09] Chris Scrapper, Raj Madhavan, Rolf Lakaemper, A Censi, A Godil, A Wagan, and A Jacoff. Quantitative assessment of robot-generated maps. In *Performance Evaluation and Benchmarking of Intelligent Systems*, pages 221–248. Springer, 2009. (Cité pages 94, 98 et 99.)

[Spa13] Petr Spacek. *Conception et mise en oeuvre d'un langage réflexif de modélisation et programmation par composants*. PhD thesis, LIRMM & Ecole des Mines d'Alès & Ecole des Mines de Douai & Université Montpellier 2 - France, 2013. Thèse de doctorat dirigée par Dony, Christophe Informatique Montpellier 2 2013. (Cité pages 15 et 27.)

[Sta82] James William Stamos. A large object-oriented virtual memory : Grouping strategies, measurements, and performance. Technical Report SCG-82-2, Xerox Palo Alto Research Center, Palo Alto, California, may 1982. (Cité page 73.)

[Sta84] James W. Stamos. Static grouping of small objects to enhance performance of a paged virtual memory. *ACM Trans. Comput. Syst.*, 2(2) :155–180, May 1984. (Cité page 73.)

[Sta09] Cyrill Stachniss. *Robotic Mapping and Exploration*. Springer, 2009. (Cité page 90.)

[Str14] Robert Strandh. Resolving metastability issues during bootstrapping. In *International Lisp Conference*, 2014. (Cité page 11.)

[Szy98] C. Szyperski. *Component software : beyond object-oriented programming*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 1998. (Cité page 21.)

[Szy02] C. Szyperski. *Component Software : Beyond Object-Oriented Programming (2nd Edition)*. Addison-Wesley, 2002. (Cité pages 8 et 16.)

[TDSF10] Chouki Tibermacine, Christophe Dony, Salah Sadou, and Luc Fabresse. Software architecture constraints as customizable, reusable and composable entities. In *Proceedings of the 4th European Conference on Software Architecture (ECSA'10)*, Copenhagen, Denmark, August 2010. Springer-Verlag. Short paper. (Cité pages 28 et 36.)

[Tib06] Chouki Tibermacine. *Contractualisation de l'évolution architecturale de logiciels à base de composants : Une approche pour la préservation de la qualité*. Theses, Université de Bretagne Sud, October 2006. (Cité page 28.)

[Tit06] Ben L. Titzer. Virgil : objects on the head of a pin. In *OOPSLA '06 : Proceedings of the 21st annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications*, pages 191–208, New York, NY, USA, 2006. ACM. (Cité page 61.)

[TLR+09] Jean-Yves Tigli, Stéphane Lavirotte, Gaëtan Rey, Vincent Hourdin, Daniel Cheung-Foo-Wo, Eric Callegari, and Michel Riveill. Wcomp middleware for ubiquitous computing : Aspects and composite event-based web services. *annals of telecommunications - annales des télécommunications*, 64(3-4) :197–214, 2009. (Cité page 31.)

[TNCC03] Éric Tanter, Jacques Noyé, Denis Caromel, and Pierre Cointe. Partial behavioral reflection : Spatial and temporal selection of reification. In *Proceedings of OOPSLA '03, ACM SIGPLAN Notices*, pages 27–46, nov 2003. (Cité page 45.)

[TSDF11] Chouki Tibermacine, Salah Sadou, Christophe Dony, and Luc Fabresse. Component-based specification of software architecture constraints. In *In proceedings of the 14th International ACM SIGSOFT Symposium on Component Based Software Engineering (CBSE'11)*, Boulder, Colorado, USA, June 2011. ACM Press. Acceptance rate : 29%, ACM SIGSOFT *Distinguished Paper Award*. (Cité pages 28 et 36.)

[TSL03] Frank Tip, Peter F. Sweeney, and Chris Laffra. Extracting library-based java applications. *Commun. ACM*, 46(8) :35–40, August 2003. (Cité page 61.)

[Ung84] Dave Ungar. Generation scavenging : A non-disruptive high performance storage reclamation algorithm. *ACM SIGPLAN Notices*, 19(5) :157–167, 1984. (Cité page 71.)

[USA05] David Ungar, Adam Spitz, and Alex Ausch. Constructing a metacircular virtual machine in an exploratory programming environment. In *OOPSLA '05 : Companion to the 20th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, pages 11–20, New York, NY, USA, 2005. ACM. (Cité page 49.)

[VBG+10] Toon Verwaest, Camillo Bruni, David Gurtner, Adrian Lienhard, and Oscar Niestrasz. Pinocchio : Bringing reflection to life with first-class interpreters. In *Proceedings of the ACM International Conference on Object Oriented Programming Systems Languages and Applications*, OOPSLA '10, pages 774–789, New York, NY, USA, 2010. ACM. (Cité page 11.)

[VBLN11] Toon Verwaest, Camillo Bruni, Mircea Lungu, and Oscar Nierstrasz. Flexible object layouts : enabling lightweight language extensions by intercepting slot access. In *Proceedings of 26th International Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA '11)*, pages 959–972, New York, NY, USA, 2011. ACM. (Cité page 46.)

[WGF11] Gregor Wagner, Andreas Gal, and Michael Franz. “slimming” a java virtual machine by way of cold code removal and optimistic partial program loading. *Sci. Comput. Program.*, 76(11) :1037–1053, November 2011. (Cité page 61.)

[WHV+12] Christian Wimmer, Michael Haupt, Michael L. Van De Vanter, Mick Jordan, Laurent Daynes, and Douglas Simon. Maxine : An approachable virtual machine for, and in, java. Technical Report 2012-0098, Oracle Labs, 2012. (Cité page 86.)

[WHVDV+13] Christian Wimmer, Michael Haupt, Michael L. Van De Vanter, Mick Jordan, Laurent Daynès, and Douglas Simon. Maxine : An approachable virtual machine for, and in, java. *ACM Trans. Archit. Code Optim.*, 9(4), January 2013. (Cité page 49.)

[WLF07] Chao-Lin Wu, Chun-Feng Liao, and Li-Chen Fu. Service-Oriented Smart-Home Architecture Based on OSGi and Mobile-Agent Technology. *IEEE Transactions on Systems, Man and Cybernetics, Part C (Applications and Reviews)*, 37(2) :193–205, March 2007. (Cité page 31.)

[WWS10] Thomas Würthinger, Christian Wimmer, and Lukas Stadler. Dynamic code evolution for java. In *Proceedings of the 8th International Conference on the Principles and Practice of Programming in Java*, PPPJ '10. ACM, 2010. (Cité page 86.)

[Yam98a] B. Yamauchi. Frontier-based exploration using multiple robots. In *Proceedings of Agent'98*, 1998. (Cité pages 90, 93, 97 et 99.)

[Yam98b] Brian Yamauchi. Frontier-based exploration using multiple robots. In *Proceedings of the 2nd International Conference on Autonomous Agents (Agents'98)*, pages 47–53, Minneapolis, MN, USA, May 1998. (Cité page 99.)

[YFLB14a] Zhi Yan, Luc Fabresse, Jannik Laval, and Noury Bouraqadi. Simulating multi-robot exploration using ros and morse. In *Proceedings of 9th National Conference on "Control Architecture of Robots" Control Architectures of Robots (CAR'14)*, Paris, France, June 2014. (Cité pages 94, 96 et 103.)

[YFLB14b] Zhi Yan, Luc Fabresse, Jannik Laval, and Noury Bouraqadi. Team size optimization for multi-robot exploration. In *4th International Conference on Simulation, Modeling and Programming for Autonomous Robots (SIMPAR 2014)*, Bergamo, Italy, October 2014. Springer. (Cité pages 94, 99 et 103.)

[YFLB15a] Zhi Yan, Luc Fabresse, Jannik Laval, and Noury Bouraqadi. Metrics for performance benchmarking of multi-robot exploration. In *Proceedings of the 2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2015)*, Hamburg, Germany, November 2015. IEEE. Acceptance rate : 46%. (Cité pages 94, 99, 100, 103 et 110.)

[YFLB15b] Zhi Yan, Luc Fabresse, Jannik Laval, and Noury Bouraqadi. Métriques pour le benchmarking de performance d'exploration multi-robots. In Laurent Vercouter and Gauthier Picard, editors, *23es Journées Francophones sur les Systèmes Multi-Agents (JFSMA'15)*, pages 9–18, Rennes, France, June 2015. Cépaduès. (Cité pages 94, 97 et 103.)

[YJC13] Zhi Yan, Nicolas Jouandeau, and Arab Ali Cherif. A survey and analysis of multi-robot coordination. *International Journal of Advanced Robotic Systems*, 10, December 2013. (Cité page 90.)

[Zel05] Andreas Zeller. *Why Programs Fail : A Guide to Systematic Debugging*. Morgan Kaufmann, October 2005. (Cité page 82.)

[Zer11] ZeroTurnAround. Java ee productivity report 2011. `http://zeroturnaround.com/wp-content/uploads/2010/11/Java_EE_Productivity_Report_2011_finalv2.pdf`, 2011. (Cité page 82.)

[Zer12] ZeroTurnAround. What developers want : The end of application re-deploys. http://files.zeroturnaround.com/pdf/JRebelWhitePaper2012-1.pdf, 2012. (Cité page 86.)

[ZSDT02] R. Zlot, A. Stentz, M.B. Dias, and S. Thayer. Multi-robot exploration controlled by a market economy. In *Proceedings of ICRA'02*, volume 3, pages 3016–3023, 2002. (Cité pages 90 et 98.)

ANNEXES

A | CURRICULUM VITÆ DÉTAILLÉ

It's nice to be important but it's more important to be nice.

John Templeton
Scooter

L'expérience, ce n'est pas ce qui arrive à quelqu'un, c'est ce que quelqu'un fait avec ce qui lui arrive.

Aldous Huxley

On ne peut rien enseigner à autrui. On ne peut que l'aider à le découvrir lui-même.

Galilée

Un professeur qui essaie d'enseigner sans inspirer à ses élèves le désir d'apprendre frappe sur des têtes dures.

Horace Mann

Fabresse Luc

Maître-Assistant en Informatique

☎ 03 27 71 23 58
06 78 59 67 91
✉ luc.fabresse@mines-douai.fr
http://car.mines-douai.fr/luc/

État civil

Nom	: Fabresse
Prénom	: Luc
Date de naissance	: 21 avril 1980 (35 ans)
Lieu de naissance	: Alfortville (Val-de-Marne), France
Situation familiale	: Marié, 2 enfants

Fonctions et Expériences

Depuis 2008 **Maître-Assistant en informatique**, *Ecole des Mines de Douai, Institut Mines-Télécom*.

2006–2008 **ATER**, *IUT informatique de Montpellier puis Institut des Sciences de l'Entreprise et du Management de Montpellier*.

2003–2006 **Doctorant**, *LIRMM, Equipe MaREL*, bourse MENRT et moniteur CIES à l'université Montpellier 2 et à Polytech'Montpellier.

2003 **Stagiaire**, *Société Gatonero S.A.*, Stage DEA/Ingénieur en collaboration avec le LIRMM.

Formations et Diplômes

2008 **Qualification aux fonctions de maître de conférences**, *section CNU 27*.
Renouvelée en 2012.

2007 **Doctorat**, *Université Montpellier 2*, LIRMM, Equipe MaREL.
Titre : « Du découplage à l'assemblage non-anticipé de composants: conception et mise en œuvre du langage à composants Scl »,
Co-directeurs: C. Dony, M. Huchard.

2003 **Diplôme d'Études Approfondies (DEA) en informatique**, *Université Montpellier 2*.
Titre: « Proposition d'un langage de programmation à base de composants logiciels interfaçables et application à la génération semi-automatisée d'IHM »,
Encadrant: C. Dony,
Cadre de travail: collaboration Gatonero S.A. / LIRMM.

2003 **Diplôme d'Ingénieur en Informatique et Gestion**, *Polytech'Montpellier*.
Titre projet: « Implémentation Java d'un algorithme de manipulation de hiérarchies de classes »,
Encadrant: C. Dony.

2000 **DEUG Sciences et Techniques pour l'Ingénieur**, *Université Montpellier 2*.

Activités de Recherche (synthèse)

Mes activités de recherches visent à proposer des modèles, des mécanismes et des outils permettant le développement interactif et haut niveau d'applications de robotique mobile grâce aux *langages réflexifs et dynamiquement typés*. Pour cela, mes travaux s'articulent autour de la modularité et de la réutilisation (objets, composants, modules, services, ...), des infrastructures d'exécution (machines virtuelles, mécanismes de gestion de la mémoire, ...) et des outils pour le développement (débogueur à distance, simulations multi-robots distribuées, ...). Un de mes domaines d'application privilégiés à l'école des Mines de Douai est celui de l'exploration d'un territoire inconnu par une flotte de robots mobiles et autonomes.

Synthèse chiffrée depuis 2007 :

- 8 revues internationales et chapitres de livres
- 17 conférences internationales dont 1 primée dans une conférence de rang A
- 13 ateliers internationaux avec comité et acte dont 1 primé
- 5 ateliers nationaux
- 4 rapports internes
- 2 post-doc co-encadrés
- 4 thèses co-encadrées soutenues. Taux d'encadrements : 30%, 40%, 30%, 50%
- 2 thèses co-encadrées en cours. Taux d'encadrements : 40%, 50%
- 1 stagiaire de master encadré
- 1 ingénieur co-encadré pendant 18 mois
- 3 participations à des projets de recherche terminés dont 1 en tant que coordinateur
- 3 participations à des projets de recherche en cours
- 5 logiciels libres co-développés
- 5 collaborations scientifiques : INRIA RMoD, LIRMM MaREL, Mines Alès, ENSTA Bretagne, DCC PLEIAD (University of Chile)

Références

- DR. Stéphane Ducasse (stephane.ducasse@inria.fr) de l'INRIA Lille - Nord Europe, **garant** de mon HDR.
  Téléphone (bureau) : 03 20 43 42 56
- Prof. Noury Bouraqadi (noury.bouraqadi@mines-douai.fr) de l'école des Mines de Douai.
  Téléphone (bureau) : 03 27 71 23 60
- Prof. Christophe Dony (christophe.dony@lirmm.fr) de l'Université Montpellier 2.
  Téléphone (bureau) : 04 67 41 85 33
- Prof. Marianne Huchard (marianne.huchard@lirmm.fr) de l'Université Montpellier 2.
  Téléphone (bureau) : 04 67 41 86 58

Activités d'enseignement

Depuis mon arrivée à l'Ecole des Mines de Douai (EMD) en 2008, j'assure chaque année les Cours, TD/TP, examens et rattrapages de plusieurs matières principalement au niveau de la Majeure ISIC « Ingénierie des systèmes d'information et de communication (ISIC) » (BAC+4). Je m'investis également dans l'encadrement de différents types de projets étudiants chaque année comme les Projets Découverte Recherche (PDR) et les Projets Scientifique et Technique (PST). Je participe aussi à de nombreux jurys comme les soutenances stages en 1^{re}, 2^{e}et 3^{e}année ou encore aux oraux de recrutements à l'EMD (admissibilité). Je vais également intervenir prochainement dans le Master international « Robotique et Transport » (MRT) de l'école Centrale de Lille et Polytech'Lille pour des cours/TD/TP sur la robotique mobile de services.

Intitulé de l'enseignement	Niveau	Nature	Nb heures
Algorithmique & Langage C	BAC+4	Cours/TD/TP	28
Technologies Internet	BAC+4	Cours/TD/TP	32
UML	BAC+4	Cours/TD/TP	16
Programmation Web	BAC+3	Cours/TD/TP	24
Développement mobile avec HTML5	BAC+4	Cours/TD/TP	20
TIC	BAC+3	TD/TP	8
Programmation par Objets (Java)	FCDD*	TD	8
Projets Découverte Recherche (76h)	BAC+4	~3 groupes	3*20
Projet Scientifique et Technique	BAC+5	~1 groupe	20
Soutenances Stages	BAC+3,+4,+5	~15 soutenances	
		Volume horaire présentiel moyen par an :	136h (hors projets)
		Volume horaire moyen par an en équivalent TD :	235h

Le tableau ci-dessous présente mes activités d'enseignement et d'encadrement entre 2003 et 2008, d'abord en tant que Moniteur de l'Université Montpellier 2 et de Polytech'Montpellier puis en tant qu'ATER.

Intitulé de l'enseignement	Niveau	Nature	Heures TD
Bases de données: modélisation, conception et réalisation	IUP1*	TD/TP	20
Introduction à la programmation (ADA)	PM1*/IUT1	TD/TP	101
Introduction à OpenGL	M2 pro.	TP	10
Programmation par assemblage de composants	M2 pro.	TD/TP	36
Introduction aux technologies J2EE	M2 pro.	TD/TP	36
Découverte de l'informatique	L1	TD/TP	106
Programmation par Objets (C++)	L3	TD/TP	24
Programmation par Objets avancée	IUT3	Cours/TD/TP	12
Algorithmique avec PHP	IUT1 (DU*)	Cours/TD/TP	36
Programmation par assemblage de composants logiciels	M2 rech.	Cours	10
Encadrement de projets	Tous niveaux	TP	37
		Volume horaire total équivalent TD entre 2003-2008 :	428h

*Abbreviations:

FCDD : Formation Continue Diplômante à Distance
PM : élèves ingénieurs de Polytech'Montpellier
IUP : institut universitaire professionnalisé
L1,L3 : Licence 1^{re}et 3^{e}années (anciennement DEUG et Licence)
M2 pro. : Master 2^{e}année professionnel (anciennement DESS)
M2 rech. : Master 2^{e}année recherche (anciennement DEA)
EMD : élèves ingénieurs de l'école des Mines de Douai
DU : Diplôme Universitaire

Participation à des projets d'enseignement

Projet Web

Quelques années après mon arrivée à l'EMD, j'ai mis en place un projet d'élèves transverse à plusieurs matières en collaboration avec les enseignants de différentes matières: charte graphique, ateliers de génie logiciel, SGBD, gestion de code et mon propre cours technologies Internet. Pour ce projet, un cahier des charges (nouveau chaque année) est donné aux élèves regroupés en binôme ou trinôme. À travers les différents cours qu'ils suivent, les élèves doivent avancer sur leur projet qui consiste à réaliser un site Web complet, déployé et opérationnel en PHP et MySql. Lors d'une soutenance finale devant tous les enseignants, ils doivent démontrer qu'ils ont utilisé les connaissances acquises dans les différentes matières et que leur projet répond bien au cahier des charges. Ce projet permet aux élèves de mieux s'investir dans les différents cours, car motivés pour avancer sur leur projet. Il est même fréquent que des élèves apprennent par eux-mêmes certaines notions pour avancer plus vite. Ce premier projet informatique en équipe et de bout en bout permet de décloisonner les enseignements. Aujourd'hui, ce projet est un élément fort de la formation comme en témoignent les anciens élèves.

MOOC « Langage C »

En tant que responsable d'un cours de langage C, j'ai décidé de participer à un dépôt de projet pour concevoir un MOOC pour l'apprentissage de ce langage et des mécanismes associés. Ce projet a été financé et démarre officiellement en fin 2015 pour une mise en ligne d'un premier volet de 3 semaines en juin 2016. S'ensuivront deux autres volets de 3 semaines.

Coordinateur : Rémi Sharrock (Telecom ParisTech)
Participants :
- Guillaume Duc (Télécom ParisTech)
- Luc Fabresse et Anthony FLEURY (Mines Douai)
- Philippe Lalevée (Mines Saint-Etienne)
- Christian Bac (Telecom SudParis)
- Jean-François Colin (Telecom Lille)
- Mathias Hiron (président de France-IOI)

MOOC « Pharo »

Le développement par objets nécessite de vraiment comprendre les mécanismes tels que l'envoi de message, la redéfinition, le polymorphisme et même la réflexion. Pour illustrer ces notions importantes sans être confronté à des problématiques secondaires comme la compilation, l'installation et la configuration d'un environnement de développement complexe, nous choisi d'utiliser la plateforme Pharo qui est simple et puissante. En ce sens, nous avons déposé en 2015 plusieurs demandes de financement (MOOCLAB, UNIT, UNISCIEL) pour concevoir un MOOC complet d'approfondissement de la conception et de la programmation objet dans un environnement réflexif et immersif grâce à la plateforme Pharo.

Coordinateur : Stéphane Ducasse (INRIA RMoD)
Participants :
- Noury Bouraqadi et Luc Fabresse (Mines Douai)
- Damien Cassou (Université de Lille)
- Alain Plantec (Université de Bretagne Occidentale)
- Hervé Verjus (Université de Savoie Chamberry)
- Serge Stinckwich (Université Marie Currie)

Autres activités pédagogiques

Traduction de chapitres pour le livre « Squeak par l'exemple » `http://www.squeakbyexample.org/fr/`

Participation à des projets scientifiques

Projet « RDDebug » (en cours)

- Période : 2015-2017
- Titre : « *RDDebug: Remote Distributed Debugging* »
- Financement : ADT INRIA
- Partenaires :
  - Dr. Marcus Denker (Coordinateur) (INRIA - RMoD)
  - Pr. Noury Bouraqadi et Dr. Luc Fabresse (Mines Douai)
  - Mike Filonov, PharoCloud (http://pharocloud.com), Russia
- Résumé : The goal of the request is to put research results on remote debugging into practice. We have worked together with Ecole de Mine des Douai the last 4 years on remote reflection as the basis for remote debugging and development. This ADT will allow us to take the results of this research together with existing prototypes and integrate it into our Pharo system. Having remote debugging available will ease the development of distributed systems with Pharo and enable new research directions. The system will be used by: (1) researchers from RMoD and its partners (language design, robotics, software main- tenance) to design, deploy and maintain new distributed architecture based on Pharo; (2) SMEs to develop, deploy and maintain quickly distributed Pharo applications when they deploy in the cloud.

Projet « SUCRé » (en cours)

- Période : 2014 à 2017
- Titre : « Coopération Homme(s)-Robot(s) en milieu hostile »
- Financement : ARCir dynamique région Nord-Pas de Calais
- Partenaires :
  - LAMIH de l'Université de Valenciennes et du Hainaut-Cambrésis
  - LAGIS de l'Université de Lille 1
  - LGI2A de l'Université d'Artois
  - GEMTEX de l'ENSAIT (Ecole d'ingéniérie et d'innovation textile) de Roubaix
  - L'URIA de l'Ecole des Mines de Douai.
  - CRESTIC de l'Université de Reims
  - Tech-CICO de l'Université de Troyes
  - Ecole des Mines Paris-Tech
  - Carinna (Recherche et Innovation en Champagne-Ardenne) à l'Université de Troyes
- Résumé : Le projet SUCRé se focalise sur la coopération Homme(s)-Robot(s). Elle inclut la coopération entre opérateurs humains, la coopération entre un opérateur humain et un robot, la coopération entre un humain et plusieurs robots et la coopération entre les robots. La coopération entre les opérateurs humains concerne à la fois les agents de la cellule de crise et les acteurs déployés sur le terrain. L'objectif est de soutenir l'activité des opérateurs humains sur les décisions à prendre et les actions à engager. Des aides à la décision globale pour les agents de la cellule de crise (niveau stratégique/tactique), locales pour les acteurs du terrain (niveau tactique/opérationnel) sont ainsi conçues (cf. figure 1). Ces aides soutiennent l'activité individuelle et coopérative des agents. Elles sont alimentées par les informations issues du terrain par le biais de capteurs présents sur les robots et sur les acteurs équipés de vêtements intelligents. Les informations sont analysées de façon à les rendre pertinentes et intelligibles et à faire des propositions de décision sur la planification des missions des agents.
  La coopération entre un opérateur humain et un robot concerne principalement les acteurs du terrain. C'est une activité coopérative programmée par la cellule de crise au travers de la mission ou initiée par l'homme ou le robot en fonction de l'évolution de la situation. La situation évolue de par sa propre dynamique (diminution ou augmentation de l'incendie, de l'inondation, ...), de par la présence de victime ou de nouvel objectif fixé par la cellule de crise, ou encore de par l'évolution des compétences et capacités des agents déployés sur le terrain (secouriste blessé, robot en panne, ...). Les agents humains ou robotisés communiquent aux partenaires de

nouvelles informations pour réactualiser le but commun ou font appel aux partenaires pour leur venir en aide dans la décision ou dans l'action.
La coopération robot-robot se base sur une auto organisation des agents robotisés pour répondre à une mission confiée par la cellule de crise (exploration d'une zone, recherche d'une cible, ...). Plusieurs approches de l'automatique et de l'informatique se complémentent pour assurer la résilience et la robustesse de la flotte. La flotte de robots est cependant supervisée par un opérateur humain appartenant à la cellule de crise ou sur le terrain, mais de toute façon déporté. C'est une problématique particulière aussi en plein développement dans le domaine de l'interaction Homme-Robots.

Projet « JIT pour FPGA » (en cours)

- Période : 2014-2017
- Titre : « Compilation logicielle à la demande vers matériel »
- Financement : Carnot Mines
- Partenaires :
  - Pr. Loïc Lagadec (ENSTA Bretagne)
  - Pr. Noury Bouraqadi, Luc Fabresse et Jannik Laval (Mines Douai)
- Résumé : Le développement de logiciel de contrôle de robot doit répondre à une double contrainte. L'autonomie des robots exige d'embarquer le logiciel qui régit le comportement du robot. Ce logiciel effectue généralement des traitements lourds et complexes (typiquement du traitement d'image). La complexité spatiale et temporelle de ces calculs doit cependant être limitée afin de minimiser leur consommation énergétique et ainsi maximiser la durée des missions ou le rayon d'action des robots.
  Une solution possible à ce problème est de remplacer les logiciels par du matériel, plus efficace. La contrepartie est qu'un matériel spécifique à une application est plus coûteux à réaliser. Le présent projet a pour objectif de s'attaquer à la réduction de ce coût par la synthèse automatique de FPGAs. Un mécanisme de compilation à la demande remplacerait à chaud, les fragments de code les plus utilisés par une contrepartie matérielle. Le code du logiciel sera écrit dans le langage de programmation par objets Smalltalk. De par sa dynamicité Smalltalk présente l'intérêt d'offrir un support élaboré à la réflexion et la méta-programmation. Ces capacités simplifient les opérations d'analyse de programme et sa modification à l'exécution. La réalisation des objectifs du projet requiert la levée des verrous suivants : (i) définir des règles de programmation qui permettent d'écrire en Smalltalk du code projetable vers des FPGA et (ii) identifier des mécanismes qui permettent de remplacer à l'exécution des fragments de code par du matériel, sans compromettre la cohérence de l'état du robot.

Projet « CAIRE »

- Période : 2013 à 2015
- Titre : « Cartographie Automatique en Intérieur de Rayonnements Electromagnétiques »
- Financement « Jeunes chercheurs » de la région Nord-Pas de Calais
- Partenaires :
  - Luc Fabresse (Coordinateur), Arnaud Doniec, Noury Bouraqadi (Mines Douai)
  - Stéphane Ducasse (INRIA - RMoD)
  - Philippe Mariage (IEMN - Telice)
- Résumé : L'objectif global est de faciliter la construction d'applications de robotique mobile. L'application visée dans ce projet est celle de la construction automatique de cartes de rayonnements électromagnétiques par un robot mobile équipé des capteurs adéquats. Toutefois, ce projet n'est qu'une première étape et il s'intéresse plus spécifique au processus de développement d'applications pour robots mobiles. De manière analogue au marché actuel des logiciels pour smartphones et tablettes, nous serons confrontés dans un avenir proche au besoin de développer rapidement et massivement des logiciels pour répondre à ces nouveaux besoins et usages liés à l'essor de la robotique.
  Les approches de développement dites « agiles » (XP, TDD, intégration continue, ...) sont désormais une alternative crédible et de plus en plus utilisée par rapport aux méthodes classiques (cycle en V, ..) pour faire rapidement face aux besoins des usagers. Pour cela, ces méthodes pragmatiques mettent l'accent sur les tests

et l'intégration en conditions réelles dès le début du cycle de développement. Cela est rendu possible par un ensemble de pratiques et d'outils (débogueur, outils de *refactoring*, serveur d'intégration, ...).
Dans le domaine du développement d'applications robotiques, les méthodes agiles sont encore marginales. Une explication est que peu de chercheurs issus du génie logiciel s'intéressent aux problématiques liées au développement de logiciels pour systèmes robotiques (contraintes matérielles et temporelles). Ainsi, les méthodes classiques, les langages et les outils bas niveau (assembleur, C, C++) sont encore majoritairement utilisés dans ce domaine ce qui ne permet pas d'utiliser pleinement les méthodes agiles.
À travers ce projet, notre objectif est de fédérer plusieurs chercheurs de notre laboratoire autour de l'utilisation des méthodes agiles pour le développement de logiciels modulaires pour la robotique. Nous nous concentrerons sur les couches de contrôle hautes du robot (contraintes matérielles et temporelles faibles). L'originalité de nos travaux réside dans l'utilisation de langages dynamiques. D'une part, car ils permettent une meilleure flexibilité et rapidité pour le prototypage et la mise au point. D'autre part, car ils facilitent la réalisation d'extensions du langage et des outils de développement pour les adapter aux besoins de la robotique.
Une première étape de ce projet consiste à proposer un nouveau modèle de stratification de la réflexivité (basé sur des miroirs). Notre idée est de profiter de la réflexivité lors du développement et la mise au point d'outils : débogueur, outil d'analyse et de génération de code compact avant déploiement, mais de ne pouvoir ensuite garder que les miroirs nécessaires à l'exécution du code.

Projet « RoboShop »

- Période : 2012-2013
- Titre : « Robots en galerie marchande »
- Financement : Pôle de compétitivité du commerce (PICOM)
- Partenaire : Mines Douai
- Salon technologique : En fin de projet, nous avons effectué des démonstrations d'usage d'un robot mobile sur le stand New Shopping Experience 3 du PICOM lors du salon VAD conext à Lille Grand Palais les 20,21 et 22 octobre 2013.
- Résumé : Ce projet s'intéresse à l'usage de robots *mobiles* et *autonomes* dans le cadre d'applications liées au commerce.
  Scientifiquement, ce projet vise à définir une architecture de contrôle d'un robot mobile spécifiquement conçue pour la construction d'applications répondant à des scénarios d'usage dans le cadre du commerce. Cette architecture de contrôle devra être indépendante de toute application ou scénario d'usage et extensible afin de permettre la construction d'applications répondant aux usages futurs. Technologiquement, nous travaillons à travers ce projet sur la définition et la mise en place de l'infrastructure matérielle et logicielle d'un robot. Il s'agit de fournir un robot complet et prêt à l'emploi disposant des fonctions basiques telles que: localisation, navigation dans un environnement contrôlé, interaction avec l'utilisateur (synthèse vocale, tablette tactile). Il s'agit d'une part d'adapter matériellement l'un des robots disponibles à Mines Douai (ajout de capteurs, ...) et d'autre part de rendre accessible à l'architecture de contrôle du robot toutes ces fonctionnalités de base via l'intergiciel ROS (Robot Operating System).

Projet « Applications distribuées »

- Période : 2009-2012
- Titre : « Construction d'applications distribuées à base de composants par composition de services »
- Financement : Carnot Mines
- Partenaires :
  - Dr. Christelle Urtado et Dr. Sylvain Vauttier (Mines Alès - ISOE)
  - Dr. Luc Fabresse (Mines Douai - DIA)
  - Pr. Marianne Huchard (Université Montpellier 2, LIRMM - MaREL)
- Résumé : On peut en effet constater que les plateformes à composants proposent des mécanismes de construction et de manipulation d'architectures d'applications qui sont locales à un serveur. C'est le cas pour OSGi, qui définit un standard de gestion de composants logiciels Java. OSGi dispose de nombreuses implémentations, commerciales

ou open-source, utilisées dans des domaines d'applications très variés (systèmes embarqués, serveurs applicatifs, grilles de calcul, outils de développement, ...). OSGi définit des mécanismes dynamiques de gestion du cycle de vie des composants permettant leur déploiement, leur lancement, leur arrêt, leur mise à jour et leur désinstallation. Il propose même un modèle de programmation orienté-services utilisant un référentiel dans lequel les composants publient les types de services qu'ils fournissent ou recherchent des composants fournissant les services dont ils ont besoin. Mais tous ces mécanismes ne sont destinés qu'à la gestion des composants déployés localement sur le dispositif géré par OSGi.

L'objet de ce projet est donc d'étudier la combinaison des paradigmes composants et services afin de proposer un ensemble de modèles, de mécanismes et d'outils permettant : (i) d'une part l'interopérabilité des composants issus de différents modèles et langages (Fractal, Scl, Maleva, ...). Par exemple, des approches comme Frogi permettent l'empaquetage de composants Fractal dans des "bundles" OSGI. Dans quelle mesure d'autres modèles de composants écrits dans différents langages de programmation (Java, Smalltalk, C#, ...) peuvent bénéficier d'un tel empaquetage et quels seraient les impacts sur la composition des services ; et (ii) d'autre part la gestion d'applications construites à partir de composants hétérogènes déployés (« installés ») sur un ensemble de dispositifs (ordinateur, serveur multimédia, appareil ménager intelligent, capteurs sans fil, ...). Le travail se concentrera notamment sur la constitution de catalogues de services et de composants en se basant sur différents points de vue comme leur description intrinsèque, leurs propriétés, leurs usages, leur généalogie. Il prendra aussi en compte, de façon autonomique, diverses opérations de gestion du cycle de vie des composants (évolution, adaptation, reconfiguration).

Co-encadrements de post-doctorants

Khelifa BAIZID (en cours)

- **Post-doc** de Juin 2015 à Juin 2016 (12 mois)
- Financement : 100% Projet « Sucré »
- Sujet: « *Benchmarking* d'algorithmes de cartographie multi-robots »
- Co-encadrants: Dr. Noury Bouraqadi, Dr. Luc Fabresse, Dr. Jannik Laval
- Résumé des travaux : Dans ce post-doc, Khelifa Baizid utilise notre infrastructure de simulation multi-robots pour proposer de nouveaux algorithmes d'exploration et de coordination multi-robot. L'un des points importants de ce travail est d'ouvrir la possibilité de télé-opérer tout ou partie de la flotte pendant la mission d'exploration. Cette capacité est nécessaire pour le projet SUCRé pour les équipes de secours et implique de doter les robots d'une autonomie ajustable.

Zhi YAN

- **Post-doc** de septembre 2013 à mars 2015 (18 mois),
- Financement : 100% Projet CAIRE (Region Nord-Pas-de-Calais)
- Sujet: « Infrastructure pour la simulation et le *benchmarking* de systèmes multi-robots »
- Co-encadrants: Dr. Noury Bouraqadi, Dr. Luc Fabresse, Dr. Jannik Laval
- Résumé des travaux : Dans ce post-doc, Zhi Yan a travaillé sur une infrastructure de simulation réaliste pour des flottes de robots mobiles. En utilisant la puissance du cluster de l'Ecole des Mines de Douai, nous avons pu réaliser des *benchmarks* de solutions de coordination multi-robots lors de l'exploration d'environnements intérieurs inconnus. Cette infrastructure repose sur l'intergiciel *Robot Operating System* et sur MORSE (simulateur robotique 3D développé au LAAS) et permet d'exécuter le même code en simulation et sur des robots réels. Ensuite, grâce à des métriques, nous évaluons la qualité des résultats obtenus (des cartes essentiellement) par différentes approches ainsi que leur consommation en ressources (processeur, mémoire, bande passante).

Co-encadrements de doctorants

Max MATTONE (en cours)

- Période : thèse débutée en novembre 2014
- Sujet : « *Dynamic Updating for New Generation Robots* »
- Financement: 50% Mines Douai et 50% INRIA
- Localisation du doctorant : 50% à Mines Douai et 50% à l'INRIA Lille
- Co-directeurs de thèse :
  - DR. Stéphane Ducasse (INRIA Lille Nord Europe)
  - Pr. Noury Bouraqadi (Mines Douai)
- Co-encadrants :
  - Dr. Luc Fabresse (Mines Douai) (taux d'encadrement : 50%)
  - Dr. Jannik Laval (Université Lyon 2)
- Résumé du sujet : Being able to build the next generation of systems that cannot be stopped is an interesting challenge for our industry. Indeed right now we often shutdown one element, replace it by another, fix it, improve it and put it back working. However, there are some situations (production chain, robot on mars. . .) where we cannot do such basic process. We believe that software built with the property of running forever in mind will be by construction more robust to changes, more evolvable and agile by their intrinsic nature.
  Current dynamically typed languages such CLOS, Ruby, python, Pharo support software updating when they load code. Even if they are ahead from other languages from a certain perspective, it is important to rethink such behavior because it is ad-hoc. We believe that it is important to offer safer and more customizable software update mechanisms and infrastructure. By safer we mean for example that code should not break when loaded due to wrong definition order, instances should be adequately migrated. Safety can also mean that code could be loaded, tested on the spot and swapped on success.
  The questions that we want to cover in this PhD are the following ones: what is a good infrastructure to support safe dynamic code update? Do we need change and temporal analysis? is an infrastructure based on separate environments enough to support safe update? What notion of atomicity is necessary? what is the new generation of metaobject protocol for object migration? would it make sense to have multiple versions of the same classes/packages in the same VM?

Xuan Sang LE (en cours)

- Période : thèse débutée en février 2014
- Sujet : « Intégration d'un langage réflexif avec des FPGAs pour le développement d'applications robotiques »
- Financement: 50% projet « JIT Smalltalk pour FPGA » et 50% ENSTA Bretagne
- Localisation du doctorant : 50% à Mines Douai et 50% à l'ENSTA Bretagne
- Co-directeurs de thèse :
  - Pr. Loïc Lagadec (ENSTA Bretagne)
  - Pr. Noury Bouraqadi (Mines Douai)
- Co-encadrants :
  - Dr. Luc Fabresse (Mines Douai) (taux d'encadrement : 40%)
  - Dr. Jannik Laval (Université Lyon 2)
- Résumé du sujet : Les robots mobiles autonomes se caractérisent par une double contrainte. Ils doivent d'une part embarquer les calculs nécessaires à la réalisation de leurs missions. D'autre part, ils doivent minimiser leur consommation énergétique afin de maximiser la durée des missions ou leur rayon d'action. Ces deux contraintes sont contradictoires, car les algorithmes embarqués sur les robots pour naviguer ou reconnaître des objets de leur environnement sont généralement complexes et donc coûteux en terme d'énergie. Partant du précédent constat, nous proposons de remplacer une partie des logiciels embarqués sur les robots par du matériel, énergétiquement plus efficace. Cependant, pour ne pas perdre la flexibilité apportée par les logiciels, nous proposons de recourir

aux FPGA. Ce projet s'inscrit dans la lignée de nos travaux de recherche axés sur l'emploi de méthodes agiles et de langages dynamiques et réflexifs comme Smalltalk. Ces langages favorisent la capitalisation et la pérennité des résultats produits ainsi que l'élévation du niveau d'abstraction lors de la programmation. L'objectif du projet est de mettre en œuvre un mécanisme de compilation logicielle à la demande (*Just in Time Compilation* ou JIT) vers le matériel, permettant de produire des primitives matérielles. Ces primitives se substituent alors au code de départ, faisant à la fois office de spécification et de solution de repli logicielle. Idéalement, la compilation de ces primitives permettra par réflexivité de produire une VM complète dans le matériel. Nous pourrons alors bénéficier de la forte puissance d'expression du logiciel, couplée aux performances largement accrues par l'emploi de matériel. La programmation d'applications complexes sera très largement facilitée.

Guillermo POLITO

- Période : thèse débutée en avril 2012 et soutenue en avril 2015
- Titre : « *Virtualization Support for Application Runtime Specialization and Extension* »
- Financement: 50% Mines Douai et 50% INRIA
- Localisation du doctorant : 50% à Mines Douai et 50% à l'INRIA Lille
- Directeur de thèse : DR. Stéphane Ducasse (INRIA Lille Nord Europe)
- Co-encadrants :
  - Dr. Noury Bouraqadi (Mines Douai)
  - Dr. Luc Fabresse (Mines Douai) (taux d'encadrement : 50%)
- Résumé : Un environnement d'exécution est l'ensemble des éléments logiciels représentant une application pendant son exécution. Les environnements d'exécution doivent être adaptables à différents contextes. Les progrès des technologies de l'information, tant au niveau logiciel qu'au niveau matériel, rendent ces adaptations nécessaires. Par exemple, nous pouvons envisager d'étendre un langage de programmation pour améliorer la productivité des développeurs. Aussi, nous pouvons envisager de réduire la consommation mémoire des applications de manière transparente afin de les adapter à certaines contraintes d'exécution e.g. des réseaux lents ou de la mémoire limités. Nous proposons Espell, une infrastructure pour la virtualisation d'environnement d'exécution de langages orientée objets haut niveau. Espell fournit une infrastructure généraliste pour le contrôle et la manipulation d'environnements d'exécution pour différentes situations. Une représentation de 'premier-ordre' de l'environnement d'exécution orienté-objet fournit une interface haut niveau qui permet la manipulation de ces environnements. Un hyperviseur est client de cette représentation de 'premier-ordre' et le manipule soit directement, soit en y exécutant des expressions arbitraires.

  Nous montrons au travers de notre prototype que cette infrastructure supporte le *bootstrapping* (i.e. l'amorçage ou initialisation circulaire) des langages et le *tailoring* (i.e. la construction sur-mesure ou 'taille') d'environnement d'exécution. En utilisant l'amorçage nous initialisons un langage orienté-objet haut niveau qui est auto-décrit. Un langage amorcé profite de ses propres abstractions se montrant donc plus simple à étendre. La taille d'environnements d'exécution est une technique qui génère une application spécialisée en extrayant seulement le code utilisé pendant l'exécution d'un programme. Une application taillée inclut seulement les classes et méthodes qu'elle nécessite, et évite que des librairies et des *frameworks* externes surchargent inutilement la base de code.

Nick PAPOULIAS

- Période : thèse débutée en septembre 2010 et soutenue en décembre 2013
- Titre : « *Remote Debugging and Reflection in Resource Constrained Devices* »
- Financement: 50% Mines Douai et 50% Région Nord–Pas de Calais
- Localisation du doctorant : 80% à Mines Douai et 20% à l'INRIA Lille
- Directeur de thèse : DR. Stéphane Ducasse (INRIA Lille Nord Europe)
- Co-encadrants :
  - Dr. Noury Bouraqadi (Mines Douai)
  - Dr. Marcus Denker (INRIA Lille Nord Europe)
  - Dr. Luc Fabresse (Mines Douai) (taux d'encadrement : 30%)
- Résumé : La construction de logiciels pour des appareils qui ne peuvent pas accueillir localement des outils de

développement peut être difficile. Les solutions développement à distance et notamment les outils de débogage sont parfois délicats à utiliser en raison de leur nature distribuée. Dans cette thèse, afin de surmonter ces problèmes, nous identifions d'abord quatre propriétés désirables qu'une solution idéale pour le débogage à distance doit présenter : *l'interactivité, l'instrumentation, la distribution et la sécurité*. L'interactivité est la capacité d'une solution de débogage à distance de mise à jour incrémentale de toutes les parties d'une application sans perdre le contexte d'exécution (sans arrêter l'application). L'instrumentation est l'aptitude d'une solution de modifier la sémantique d'un processus en cours en vue d'aider le débogage. La distribution est la capacité d'une solution de débogage à adapter son cadre alors que le débogage d'une cible à distance. Enfin, la sécurité fait référence à la disponibilité de conditions préalables pour l'authentification et la restriction d'accès.
Compte tenu de ces propriétés, nous proposons Mercury, un modèle de débogage à distance et une architecture pour des langages à objets réflexifs. Mercury permet (1) l'interactivité grâce à un méta-niveau à distance basé sur les miroirs, (2) l'instrumentation à travers une intercession réflective basée sur la réification de l'environnement d'exécution sous-jacent, (3) la distribution grâce à un intergiciel adaptable et (4) la sécurité par la décomposition et l'authentification de l'accès aux aspects réflexifs. Nous validons notre proposition à travers un prototype dans le langage de programmation Pharo à l'aide d'un cadre expérimental diversifié de multiples dispositifs contraints. Nous illustrons des techniques de débogage à distance supportées par les propriétés de Mercury, tels que le *débogage agile distant* et *l'instrumentation objet à distance* et montrons comment elles peuvent résoudre dans la pratique, les problèmes que nous avons identifiés.

Matthieu FAURE

- Période : thèse débutée en décembre 2009 et soutenue en décembre 2012
- Titre : « *Management of scenarized user-centric service compositions for collaborative pervasive environments* »
- Financement: Projet « Applications distribuées »
- Localisation du doctorant : 50% à Mines Douai et 50% à Mines Alès
- Directeur de thèse : Pr. Marianne Huchard (Univ. Montpellier 2, LIRMM)
- Co-encadrants :
  - Dr. Luc Fabresse (Mines Douai) (taux d'encadrement : 40%)
  - Dr. Christelle Urtado (Mines Alès)
  - Dr. Sylvain Vauttier (Mines Alès)
- Résumé : L'informatique pervasive est un support pour des environnements contenant de nombreux équipements électroniques disséminés et interconnectés. Ces dispositifs fournissent un accès distant à une multitude de fonctionnalités qui nous aident dans notre vie quotidienne. Les Architectures Orientées Services sont adaptées à la conception de logiciels pervasifs. En effet, chaque dispositif fournit son propre ensemble de fonctionnalités sous la forme de services. Ainsi, en l'absence de mécanismes complémentaires, les utilisateurs se trouvent limités à utiliser les services isolément alors que leurs besoins correspondent à des scénarios qui impliquent une composition de multiples services offerts par plusieurs appareils. Dans cette thèse, nous défendons qu'un système pervasif doit : d'une part, permettre aux utilisateurs d'exprimer facilement leurs besoins en créant des scénarios et d'autre part, proposer à ses utilisateurs une représentation et des moyens de gestion de leur contexte afin qu'ils puissent tirer le meilleur parti de leur environnement et de ses changements. Par ailleurs, l'exécution de scénarios doit être résiliente aux changements environnementaux et aux actions des utilisateurs. Elle doit ainsi s'adapter dynamiquement et, si possible, tirer profit du contexte et des changements de l'environnement.
  Notre contribution, nommée SaS (Scénarios as Services), répond à ces objectifs. Elle propose une approche interopérable capable de s'adapter à l'environnement. Elle fournit une représentation persistante et personnalisable du contexte et inclut un langage de description de scénarios destiné aux utilisateurs. Ces scénarios sont facilement contrôlables, personnalisables et réutilisables. Elle planifie l'exécution pas-à-pas des scénarios, afin de s'adapter aux changements de l'environnement et de bénéficier des avantages de la mobilité des utilisateurs (exécution d'un scénario, dans la durée, sur plusieurs lieux). Enfin, elle inclut le partage de scénarios qui permet aux utilisateurs de collaborer. Un prototype de SaS, basé sur des normes industrielles (telle qu'OSGi), prouve la faisabilité de notre contribution et nous permet de l'évaluer sur un cas d'étude simple.

Mariano MARTINEZ-PECK

- Période : thèse débutée novembre 2009 et soutenue en décembre 2012
- Titre : « *Application-Level Virtual Memory for Object-Oriented Systems* »,
- Financement : Bourse Ecole des Mines de Douai,
- Localisation du doctorant : 80% à Mines Douai et 20% à l'INRIA Lille
- Directeur de thèse : DR. Stéphane Ducasse (INRIA Lille Nord Europe)
- Co-encadrants:
  - Dr. Noury Bouraqadi (Mines Douai)
  - Dr. Marcus Denker (INRIA Lille Nord Europe)
  - Dr. Luc Fabresse (Mines Douai) (taux d'encadrement : 30%)
- Résumé de la thèse : Lors de l'exécution des applications à base d'objets, plusieurs millions d'objets peuvent être créés, utilisés et enfin détruits s'ils ne sont plus référencés. Néanmoins, des dysfonctionnements peuvent apparaître, quand des objets qui ne sont plus utilisés ne peuvent être détruits, car ils sont référencés. De tels objets gaspillent la mémoire principale et les applications utilisent donc davantage de mémoire que ce qui est effectivement requis. Nous affirmons que l'utilisation du gestionnaire de mémoire virtuel du système d'exploitation ne convient pas toujours, car ce dernier est totalement isolé des applications. Le système d'exploitation ne peut pas prendre en compte ni le domaine ni la structure des applications. De plus, les applications n'ont aucun moyen de contrôler ou influencer la gestion de la mémoire virtuelle.

  Dans cette thèse, nous présentons Marea, un gestionnaire de mémoire virtuelle piloté par les applications à base d'objets. Il constitue une solution originale qui permet aux développeurs de gérer la mémoire virtuelle au niveau applicatif. Les développeurs d'une application peuvent ordonner à notre système de libérer la mémoire principale en transférant les *objets inutilisés, mais encore référencés* vers une mémoire secondaire (telle qu'un disque dur). En plus de la description du modèle et des algorithmes sous-jacents à Marea, nous présentons notre implémentation dans le langage Pharo. Notre approche a été validée à la fois qualitativement et quantitativement. Ainsi, nous avons réalisé des expérimentations et des mesures sur des applications grandeur nature pour montrer que Marea peut réduire l'empreinte mémoire de 25% et jusqu'à 40%.

Encadrement d'ingénieurs et de masters

Pablo ESTEFO

- Période : **Stagiaire Master** de mai 2014 à novembre 2014 (6 mois),
- Financement : 100% Mines Douai - DIA
- Sujet : « Développement interactif d'applications robotiques avec PhaROS »
- Encadrant : Dr. Luc Fabresse
- Résumé du travail : Pablo Estefo a travaillé sur notre prototype logiciel PhaROS construit au-dessus de ROS. Dans un premier temps, il s'est attaché à développer différentes missions robotiques avec PhaROS. Cette expérience a permis d'évaluer les forces et les faiblesses de PhaROS du point de vue d'un non-expert. Pablo Estefo a ensuite proposé des améliorations de la plateforme PhaROS pour faciliter le développement. En collaboration avec l'université du Chili à Santiago, ce stage de master a permis de co-publier un article dans un atelier international sur l'utilisation d'un DSL pour la construction d'applications robotiques (DSLRob 2014).

Santiago BRAGAGNOLO

- Période : **Ingénieur** de septembre 2012 à février 2014 (18 mois)
- Financement : 100% Projet RoboShop
- Sujet : « Infrastructure pour le développement d'applications de services avec des robots mobiles »
- Résumé du travail : Dans le cadre du projet RoboShop, Santiago Bragagnolo a posé les fondations technologiques pour l'usage de robots comme nouveau média dans la relation client. Il a réalisé une plate-forme logicielle (PhaROS) et matérielle qui est le point de départ à de futurs projets mettant en œuvre la robotique dans le contexte du

service au sens large. Un démonstrateur a été construit avec cette infrastructure dans le cadre du commerce. Il s'agit d'un prototype d'application robotique pour accompagner et guider des clients. Ce scénario d'usage général a permis de lever de nombreux verrous et sert de catalyseur pour de nouvelles idées d'usages.

Co-organisation de conférences, d'ateliers ou d'événements

- Depuis 2010, je participe chaque année à l'organisation de la conférence ESUG (European Smalltalk User Group) dont je suis également membre du *steering commitee* (`http://www.esug.org`).
- PharoDays'15 (29-30 janvier 2015), deux journées techniques centrées sur les nouvelles innovations de l'écosystème Pharo.
- En 2013, j'ai co-organisé des démonstrations d'usage d'un robot mobile sur le stand New Shopping Experience 3 du PICOM lors du salon VAD conext à Lille Grand Palais les 20,21 et 22 octobre 2013.
- Dans le cadre des Journées Européennes de la robotique de 2013, nous avons également organisé des démonstrations grand public de robotique mobile dans le département DIA de l'Ecole de Mines.
- PharoConf'12 (24-25 mai 2012), deux journées techniques centrées sur les nouvelles innovations de l'écosystème Pharo.
- Deep into Smalltalk'11 (7 au 11 mars 2011), école d'une semaine avec des présentations d'académiques et industriels sur des sujets pointus tels: machine virtuelle, compilation, réseau.
- Innovation Technology Awards en 2009, 2010 et 2011 lors de la conférence ESUG.
- *Control Architecture of Robots* CAR'10 (`http://car.mines-douai.fr/CAR2010/`), 5^{e}conférence nationale sur les architectures de contrôle pour robots. CAR est le rendez-vous annuel du groupe de travail GT4 du GDR Robotique.

Séjours invités

Séjour invité à l'Université du Chili à Santiago

Suite à l'accueil du stagiaire Pablo Estéfo, j'ai commencé à échanger régulièrement avec le Dr. Johan Fabry qui travaille au DCC (*departamento de ciencias de la computación*) de l'université du Chili à Santiago au sein de l'équipe Pleiad. En novembre 2014, Johan Fabry m'a invité 2 semaines au sein de son équipe pour travailler ensemble sur l'intégration de la plateforme PhaROS (Mines Douai) avec leur robot PR2. J'ai également profité de cette occasion pour faire un exposé invité dans son laboratoire :

- Title : « *Leveraging the development of mobile robots* »
- Résumé : At Ecole des Mines de Douai (Lille Area, Northern France), we study mobile and autonomous robotics from two complementary perspectives: Software Engineering (SE) and Artificial Intelligence (AI). From the SE perspective, we study software architectures, infrastructures and tools for controlling individual mobile robots. We focus on reflective and dynamic languages, as well as component models, for a modular, high-level and agile development of robotic software architectures. From the AI perspective, we study coordination and cooperation in robotic fleets. We mainly focus on communication models as well as emerging or predefined organizations for multi-agent robotic systems. In this presentation, I will give an overview of some of our recent PhaROS-based solutions we have been developing for both single robots and multi-robot fleets. I will illustrate each contribution with experiments we have been conducting that illustrate targeted applications.

Accueil du Dr. Johan Fabry (Université du Chili à Santiago, DCC - Pleiad)

Suite à ma demande, Johan Fabry a également été invité 2 semaines à l'Ecole des Mines de Douai en janvier 2015. Il a pu ainsi expérimenter l'utilisation de son langage LRP (Live Robot Programming) sur nos robots mobiles. Cela a été particulièrement aisé, car LRP dispose maintenant d'une interface avec PhaROS. Johan Fabry a également effectué l'exposé invité suivant au sein du département Informatique et Automatique :

- Title : « *Live Robot Programming* »
- Résumé : Typically, development of robot behavior entails writing the code, deploying it on a simulator or robot

and running it for testing. If this feedback reveals errors, the programmer mentally needs to map the error in behavior back to the source code that caused it before being able to fix it. This process suffers from a large cognitive distance between the code and the resulting behavior, which slows down development and can make experimentation with different behaviors prohibitively expensive. In contrast, Live Programming tightens the feedback loop, minimizing cognitive distance. As a result, programmers benefit from an immediate connection with the program that they are making thanks to an immediate, 'live' feedback on program behavior. This allows for extremely rapid creation, or variation, of robot behavior and for dramatically increased debugging speed. To enable such Live Robot Programming, we propose a language and program visualization that provides for live programming of nested state machines and integrates in the Robot Operating System (ROS). We outline the language, named LRP, give an illustration of its use and discuss the key points of the language that enables its liveness.

Participation à des jurys de thèse

2015 (en cours) **Rapporteur**, *de la thèse de Jean-Philippe Schneider*.
Sujet : « Mise en relation de modèles de systèmes hétérogènes grâce à un langage de rôles »,
Date de soutenance prévue en novembre 2015,
Directeur de thèse : Pr. Joel Champeau (ENSTA Bretagne).

Décembre 2013 **Examinateur**, *de la thèse de Petr Spacek*.
Titre : « *Design and Implementation of a Reflective Component-Oriented Programming and Modeling Language* »,
Jury :
(Président) Roland DUCOURNAU, Professeur, LIRMM, Université Montpellier II
(Rapporteur) Lionel SEINTURIER, Professeur, Inria, University Lille 1
(Rapporteur) Ivica CRNKOVIC, Professeur, IDT, Mälardalen University, Sweden
(Examinateur) Pierre COINTE, Professeur, LINA, Ecole des Mines de Nantes
(Examinateur) Luc FABRESSE, Maître-Assistant, Institut Mines-Telecom, Mines Douai
(Directeur de thèse) Christophe DONY, Professeur, LIRMM, Université Montpellier II
(Co-encadrant) Chouki TIBERMACINE, Maître de Conférences, LIRMM, Université Montpellier II

Relecture d'articles

2015
- Re-lecteur pour un numéro spécial du « *Journal on Software Engineering for Robotics* » (JOSER) sur la thématique des « *Domain-Specific Languages and Models for Robotic Systems* (parution prévue en 2016)
- Membre du comité editorial du livre « Pharo for the Enterprise : a Web Perspective » (parution prévue fin 2015)
- Membre du comité de programme de « *International Workshop on Domain-Specific Languages and Models for Robotic Systems* » (DSLRob-15)

2014 Re-lecteur pour :
- « *Journal of Computer Engineering and Informatics* » (JCEI)
- « *IEEE International Conference on Robotics and Automation* » (ICRA 2014)
- « *International Workshop on Domain-Specific Languages and Models for Robotic Systems* » (DSLRob-14)

2013 Re-lecteur pour :
- « *Elsevier Journal Science of Computer Programming* » (SCP)
- « *IEEE International Conference on Robotics and Automation* » (ICRA 2013)

2012
- Membre du comité de programme de « *International Workshop on Smalltalk Technologies* » (IWST'12)
- Re-lecteur pour la « Conférence francophone sur l'Architecture Logicielle » (CAL'12)

2010
- Membre du comité de programme de « *International Workshop on Dynamic languages for RObotic and Sensors systems* » (DYROS'10)
- Re-lecteur pour :
  - Conférence internationale sur « *Technology of Object-Oriented Languages and Systems* » (TOOLS'10)
  - « *International Conference on Software Maintenance* » (ICSM'10)
  - Conférence nationale « Langages et Modèles à Objets » (LMO'10)

2009 Re-lecteur pour :
- « Conférence Internationale MCETECH sur les technologies du e-Commerce » (MCETECH'09)
- « *International Symposium on Programming and Systems* » (ISPS'09)

2005 Re-lecteur pour:
- *European Conference on Object-Oriented Programming* » (ECOOP'05)
- Conférence nationale « Langages et Modèles à Objets » (LMO'05)
- Atelier du groupe « Objets Composants Modèles » (OCM'05)

Implication dans la communauté scientifique

Participation au développement de logiciels libres

- Pharo, un langage à objets pur couplé à un environnement de développement puissant
  `http://www.pharo.org`
- Ghost, un framework général de *proxies* légers
  `http://smalltalkhub.com/#!/~CAR/Ghost`
- Ocean (*Object-oriented, Cross-platform, and Effective Api for Networking*)
  `http://smalltalkhub.com/#!/~CAR/Ocean`
- PhaROS, un client ROS en Pharo
  `http://smalltalkhub.com/#!/~CAR/PhaROS`
- BOSS, un simulateur mulit-robots discret
  `http://smalltalkhub.com/#!/~CAR/BOSS`

Membre du bureau d'ESUG et du consortium Pharo

Aussi bien en recherche qu'en enseignement, mes activités reposent sur le langage Smalltalk et notamment la plateforme open-source Pharo. C'est pourquoi je contribue au développement et à la diffusion de ce langage et de Pharo au côté de l'équipe RMoD de l'INRIA Nord Europe. D'ailleurs, Mines Douai fait officiellement partie du « consortium Pharo » qui fédère une vingtaine d'entreprises et quinzaine de groupes de recherche autour de cette plateforme. Notre implication à pérenniser et documenter une plateforme que nous utilisons au quotidien pour nos enseignements et notre recherche me parait indispensable pour continuer demain à travailler avec une plateforme innovante et robuste.

Participation à des ateliers ou groupes de travail

Je participe à deux communautés scientifiques nationales : au groupe de travail sur « les architectures de contrôle pour la robotique » du GDR Robotique et le groupe de travail COSMAL du GDR GPL (Génie de

la Programmation et du Logiciel).
Au niveau international, je participe régulièrement à un atelier sur les langages dynamiques pour la robotique (DSLRob) ou encore des ateliers internationaux liés au langage Smalltalk (IWST, Smalltalks).
Ces ateliers sont des lieux d'échange scientifiques entre les acteurs d'un même domaine et j'encourage même les masters, doctorants et post-doctorants que j'encadre à y participer.

Diffusion de la culture scientifique

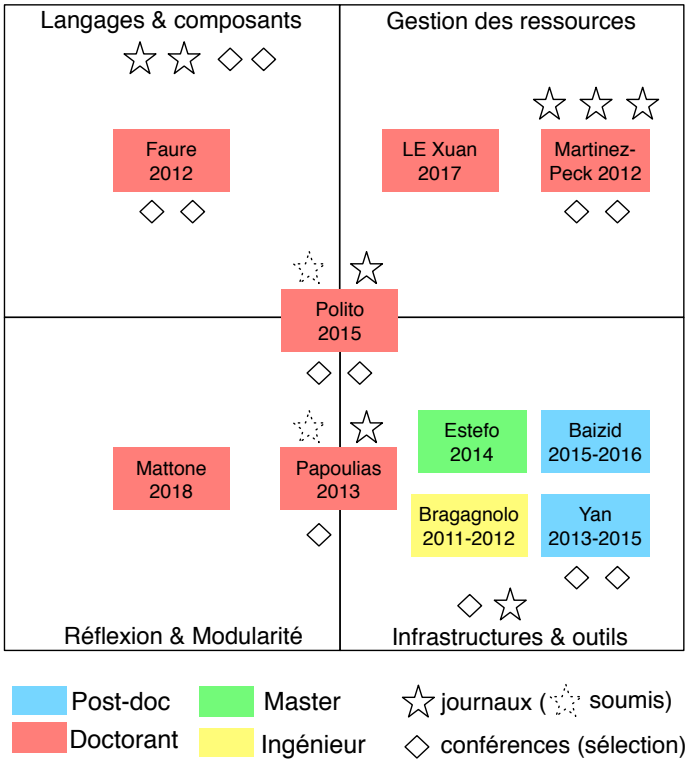
Je réalise régulièrement (plusieurs fois par an) des visites et des démonstrations de robotique mobile à Mines Douai pour des publics variés : enfants de maternelle, collégiens, lycéens, post-bac, futurs élèves de l'Ecole des Mines et industriels.

Activités administratives

Je suis un membre élu (collège des enseignants-chercheurs) au conseil de département Informatique et Automatique de l'Ecole des Mines de Douai. Ce conseil a pour objectif de faire un bilan des activités du département et de présenter le plan de développement.

Synthèse des travaux (après 2007)

De manière analogue au marché actuel des logiciels pour smartphones et tablettes, nous serons confrontés dans un avenir proche au besoin de développer rapidement et massivement des logiciels pour répondre aux nouveaux besoins et usages liés à l'essor de la robotique mobile. Dans cette perspective, mes travaux visent à *proposer des langages, des modèles et des outils permettant le développement interactif et haut niveau d'applications de robotique mobile*. Il s'agit d'améliorer les pratiques actuelles en terme de développement d'applications pour robots mobiles en se concentrant sur les langages et les outils offerts aux développeurs et les infrastructures pour les supporter. Mes travaux de recherche se fondent sur l'idée les langages réflexifs sont de meilleurs candidats pour répondre à cet objectif car il permet un haut niveau d'abstraction et intègrent déjà des abstractions et des mécanismes permettant l'adaptation et l'extension des programmes. Dans ce contexte, mes travaux s'articulent autour de quatre thèmes principaux représentés ci-dessous.



peuvent ensuite être déportés vers des architectures spécialisées comme des FPGAs afin de décharger le processeur central du robot.

Infrastructures & Outils

Lors du développement d'une application pour un robot mobile, le développeur programme généralement sur une machine de développement et compile (il peut s'agir d'une compilation croisée) son application pour la déployer sur le robot cible. Après exécution, le développeur peut récupérer des traces (journaux) qu'il pourra analyser sur sa machine de développement afin d'identifier d'éventuels dysfonctionnements de l'application. Ce cycle de développement pourtant classique est fastidieux. Pour l'améliorer, notre travail se focalise sur le développement à distance et la possibilité de simuler une application multi-robots.
Le débogage d'une application peut prendre jusqu'à 50% du temps total de développement d'après des études empiriques. Dans le cas d'une application robotique, le débogage est encore complexifié par la présence du matériel de ses possibles pannes ou défaillances. C'est ainsi que nous avons proposé une architecture permettant le débogage interactif d'applications s'exécutant sur des équipements contraints comme un robot mobile. Cette architecture est légère et minimale (en l'absence d'erreur) sur le robot tout en permettant à un débogueur distant d'inspecter et de modifier cette même application en cours d'exécution si une erreur survient.
Pour faciliter la mise au point d'applications multi-robots, nous avons proposé une infrastructure pour simuler le code des robots. Cette infrastructure repose sur l'intergiciel *Robot Operating System* et le simulateur 3D MORSE (développé au LAAS) et permet d'exécuter le même code en simulation et sur des robots réels. Nous utilisons principalement cette infrastructure pour évaluer la qualité de différents algorithmes de coordination multi-robots ainsi que leur consommation en ressources (processeur, mémoire, bande passante).

Résumé des travaux de doctorat (avant 2007)

La programmation par assemblage de composants logiciels (PPC) promet une réduction des coûts liés au développement, à la maintenance et à l'évolution d'un logiciel. Dans l'existant, les langages de programmation permettant la PPC, les « langages à composants », sont peu nombreux et disparates de par leurs origines, leurs objectifs, leurs concepts ou encore leurs mécanismes. Cette thèse propose donc Scl, un langage à composants minimal, simple et spécifiquement conçu pour faire de la PPC. La conception de Scl nous a permis de mieux identifier les notions clefs de la PPC comme le *découplage* et la *non-anticipation* ainsi que d'aborder un ensemble de questions souvent oubliées dans les autres propositions comme l'auto-référence, le passage d'arguments ou le statut des composants de base (collections, entiers, etc) dans un monde unifié. Scl permet aussi la séparation des préoccupations au niveau du code puisqu'un même composant peut être utilisé de façon standard ou de façon transversale. Cela est possible grâce à une unification des concepts d'aspect – issu de la programmation par aspects – et de composant et via le mécanisme général d'assemblage de composants de Scl. Ce mécanisme d'assemblage permet également d'établir des connexions entre composants basées sur les changements d'état de leurs propriétés sans que leurs programmeurs n'aient à écrire une ligne de code spécifique à cet effet. Il existe deux prototypes de Scl, le premier et le plus abouti est écrit en Smalltalk et le second en Ruby.

Publications depuis 2007

Ouvrages

[1] **(to be published)** Damien Cassou, Stéphane Ducasse, Luc Fabresse, Johan Fabry, and Sven Van Caekenberghe. *Enterprise Pharo: a Web Perspective*. Square Bracket LLC, 2015.

Revues internationales et chapitres de livres

[10] **(to be submitted)** Guillermo Polito, Noury Bouraqadi, Stéphane Ducasse, and Luc Fabresse. Run-fail-grow: a dynamic dead code elimination technique. *Journal of Object Technology*, 2015.

[9] **(to be submitted)** N. Papoulias, M. Denker, S. Ducasse, and L. Fabresse. End-user abstractions for meta-control: Reifying the reflectogram. *Science of Computer Programming*, 2015.

[8] Nick Papoulias, Noury Bouraqadi, Luc Fabresse, Stéphane Ducasse, and Marcus Denker. Mercury: Properties and Design of a Remote Debugging Solution using Reflection. *Journal of Object Technology*, 14(2):1:1–36, May 2015.

[7] Mariano Martinez Peck, Noury Bouraqadi, Luc Fabresse, Marcus Denker, and Camille Teruel. Ghost: A uniform and general-purpose proxy implementation. *Science of Computer Programming*, 98(3):339–359, 2014.

[6] Noury Bouraqadi, Luc Fabresse, Alexandre Bergel, Damien Cassou, Stéphane Ducasse, and Jannik Laval. Sockets. In *Deep Into Pharo*, page 21. Square Bracket Associates, September 2013.

[5] Guillermo Polito, Stéphane Ducasse, Luc Fabresse, Noury Bouraqadi, and Benjamin Van Ryseghem. Bootstrapping Reflective Systems: The Case of Pharo. *Science of Computer Programming*, 96(1):141–155, dec 2014.

[4] Mariano Martinez Peck, Noury Bouraqadi, Marcus Denker, Stéphane Ducasse, and Luc Fabresse. Marea: An efficient application-level object graph swapper. *Journal of Object Technology*, 12(1):2:1–30, jan 2013.

[3] Luc Fabresse, Noury Bouraqadi, Christophe Dony, and Marianne Huchard. A language to bridge the gap between component-based design and implementation. *Journal of Computer Languages, Systems and Structures*, 38(1):29–43, jan 2012.

[2] Mariano Martinez Peck, Noury Bouraqadi, Stéphane Ducasse, and Luc Fabresse. Object swapping challenges: an evaluation of imagesegment. *Journal of Computer Languages, Systems and Structures*, 38(1):1–15, January 2012.

[1] Luc Fabresse, Christophe Dony, and Marianne Huchard. Foundations of a simple and unified component-oriented language. *International Journal of Computer Languages, Systems and Structures*, 34(2-3):130–149, 2008.

Conférences internationales avec comité de lecture et actes

[17] Zhi Yan, Luc Fabresse, Jannik Laval, and Noury Bouraqadi. Metrics for performance benchmarking of multi-robot exploration. In *Proceedings of the 2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2015)*, Hamburg, Germany, November 2015. IEEE. Acceptance rate: 46%.

[16] Guillermo Polito, Stéphane Ducasse, Noury Bouraqadi, Luc Fabresse, and Max Mattone. Virtualization support for dynamic core library update. In *Proceedings of the ACM International Symposium on*

New Ideas, New Paradigms, and Reflections on Programming and Software (SPLASH/OnWard!15). ACM, 2015.

[15] Guillermo Polito, Stéphane Ducasse, Noury Bouraqadi, and Luc Fabresse. A bootstrapping infrastructure to build and extend pharo-like languages. In *Proceedings of the ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software (SPLASH/OnWard!15)*. ACM, 2015.

[14] N. Papoulias, M. Denker, S. Ducasse, and L. Fabresse. Reifying the reflectogram: Towards explicit control for implicit reflection. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, SAC'15, pages 1978–1985, New York, NY, USA, 2015. ACM. Acceptance rate: 24%.

[13] Zhi Yan, Luc Fabresse, Jannik Laval, and Noury Bouraqadi. Team size optimization for multi-robot exploration. In *4th International Conference on Simulation, Modeling and Programming for Autonomous Robots (SIMPAR 2014)*, Bergamo, Italy, October 2014. Springer.

[12] Jannik Laval, Luc Fabresse, and Noury Bouraqadi. A methodology for testing mobile autonomous robots. In *Proceedings of the 2013 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2013)*, Tokyo, Japan, November 2013. IEEE. Acceptance rate: 43%.

[11] Petr Spacek, Christophe Dony, Chouki Tibermacine, and Luc Fabresse. An inheritance system for structural & behavioral reuse in component-based software programming. In *In proceedings of the 11th International Conference on Generative Programming and Component Engineering (GPCE'12)*, Dresden, Germany, September 2012. ACM Press. Acceptance rate: 40%.

[10] Matthieu Faure, Luc Fabresse, Marianne Huchard, Christelle Urtado, and Sylvain Vauttier. User-defined scenarios in ubiquitous environments: Creation, execution control and sharing. In *Proceedings of the 23rd International Conference on Software Engineering and Knowledge Engineering (SEKE 2011)*, pages 302–307, Miami, USA, July 2011. Acceptance rate: 31%.

[9] Matthieu Faure, Luc Fabresse, Marianne Huchard, Christelle Urtado, and Sylvain Vauttier. A service component framework for multi-user scenario management in ubiquitous environments. In *Proceedings of the 6th International Conference on Software Engineering Advances (ICSEA 2011)*, Barcelona, Spain, October 2011. XPS (Xpert Publishing Services). Acceptance rate: 30%.

[8] Mariano Martinez Peck, Noury Bouraqadi, Marcus Denker, Stéphane Ducasse, and Luc Fabresse. Problems and challenges when building a manager for unused objects. In *Proceedings of International Conference on Smalltalk Technologies (Smalltalks'11)*, Bernal, Buenos Aires, Argentina, 2011.

[7] Chouki Tibermacine, Salah Sadou, Christophe Dony, and Luc Fabresse. Component-based specification of software architecture constraints. In *In proceedings of the 14th International ACM SIGSOFT Symposium on Component Based Software Engineering (CBSE'11)*, Boulder, Colorado, USA, June 2011. ACM Press. Acceptance rate: 29%, ACM SIGSOFT *Distinguished Paper Award*.

[6] Gwenael Casaccio, Stéphane Ducasse, Luc Fabresse, Jean-Baptiste Arnaud, and Benjamin van Ryseghem. Bootstrapping a smalltalk. In *Proceedings of International Conference on Smalltalk Technologies (Smalltalks'11)*, Bernal, Buenos Aires, Argentina, 2011.

[5] Luc Fabresse, Noury Bouraqadi, Christophe Dony, and Marianne Huchard. Component-Oriented Programming: From Requirements to Language Support. In Marcus Denker and Gabriela Arévalo, editors, *Proceedings of International Conference on Smalltalk Technologies (Smalltalks'10)*, Concepcion del Uruguay, Argentina, 2010.

[4] Mariano Martinez Peck, Noury Bouraqadi, Marcus Denker, Stéphane Ducasse, and Luc Fabresse. Visualizing objects and memory usage. In *Proceedings of International Conference on Smalltalk Technologies (Smalltalks'10)*, 2010.

[3] Chouki Tibermacine, Christophe Dony, Salah Sadou, and Luc Fabresse. Software architecture constraints as customizable, reusable and composable entities. In *Proceedings of the 4th European Conference on Software Architecture (ECSA'10)*, Copenhagen, Denmark, August 2010. Springer-Verlag. Short paper.

[2] Noury Bouraqadi and Luc Fabresse. Towards Small Portable Virtual Machines. In Marcus Denker and Gabriela Arévalo, editors, *Proceedings of International Conference on Smalltalk Technologies (Smalltalks'10)*, Concepcion del Uruguay, Argentina, 2010.

[1] Mariano Martinez Peck, Noury Bouraqadi, Marcus Denker, Stéphane Ducasse, and Luc Fabresse. Experiments with a fast object swapper. In *Proceedings of International Conference on Smalltalk Technologies (Smalltalks'10)*, 2010.

Ateliers internationaux avec comité de lecture et actes

[13] Xuan Sang Le, Loïc Lagadec, Luc Fabresse, Jannik Laval, and Noury Bouraqadi. A meta-model supporting both hardware and smalltalk-based execution of fpga circuits. In *Proceedings of International Workshop on Smalltalk Technologies (IWST'15)*, Cambridge, England, July 2015. 1st price at the Best Paper Award of IWST'15.

[12] Xuan Sang Le, Loïc Lagadec, Noury Bouraqadi, Luc Fabresse, and Jannik Laval. From smalltalk to silicon: Towards a methodology to turn smalltalk code into fpga. In *Proceedings of International Workshop on Smalltalk Technologies (IWST'14)*, Cambridge, England, September 2014.

[11] Pablo Estefo, Miguel Campusano, Luc Fabresse, Johan Fabry, Jannik Laval, and Noury Bouraqadi. Towards live programming in ROS with pharos and LRP. volume abs/1412.4629, 2014.

[10] Guillermo Polito, Noury Bouraqadi, Stéphane Ducasse, and Luc Fabresse. Understanding pharo's global state to move programs through time and space. In *Proceedings of International Workshop on Smalltalk Technologies (IWST'14)*, Cambridge, England, September 2014.

[9] Guillermo Polito, Stéphane Ducasse, Luc Fabresse, and Noury Bouraqadi. Virtual Smalltalk Images: Model and Applications. In *Proceedings of International Workshop on Smalltalk Technologies (IWST'13)*, Annecy, France, September 2013.

[8] Camillo Bruni, Luc Fabresse, Stéphane Ducasse, and Igor Stasenko. Language-side Foreign Function Interfaces with NativeBoost. In *Proceedings of International Workshop on Smalltalk Technologies (IWST'13)*, Annecy, France, September 2013.

[7] Petr Spacek, Christophe Dony, Chouki Tibermacine, and Luc Fabresse. Wringing out Objects for Programming and Modeling Component-based Systems. In *Proceedings of the Second International Workshop on Combined Object-Oriented Modeling and Programming Languages (COOMPL at ECOOP 2013)*, Montpellier, France, July 2013.

[6] Luc Fabresse, Jannik Laval, and Noury Bouraqadi. Towards Test-Driven Development for Mobile Robots. In Davide Brugali, editor, *Proceedings of the eighth full-day Workshop on Software Development and Integration in Robotics (SDIR VIII), in conjunction with the IEEE International Conference on Robotics and Automation (ICRA)*, pages 12–13, Karlsruhe, Germany, May 2013.

[5] Mariano Martinez Peck, Noury Bouraqadi, Marcus Denker, Stéphane Ducasse, and Luc Fabresse. Efficient proxies in smalltalk. In *Proceedings of ESUG International Workshop on Smalltalk Technologies (IWST 2011)*, Edinburgh, Scotland, 2011.

[4] Papoulias Nikolaos, Noury Bouraqadi, Marcus Denker, Stéphane Ducasse, and Luc Fabresse. Towards structural decomposition of reflection with mirrors. In *Proceedings of International Workshop on Smalltalk Technologies (IWST'11)*, Edingburgh, United Kingdom, 2011.

[3] Petr Spacek, Christophe Dony, Chouki Tibermacine, and Luc Fabresse. A smalltalk implementation of exil, a component-based programming language. In *Proceedings of ESUG International Workshop on Smalltalk Technologies (IWST 2011)*, Edinburgh, Scotland, August 2011. ESUG.

[2] Matthieu Faure, Luc Fabresse, Marianne Huchard, Christelle Urtado, and Sylvain Vauttier. Towards Scenario Creation by Service Composition in Ubiquitous Environments. In L. Duchien S. Ducasse and L. Seinturier, editors, *Proceedings of the 9th BElgian-NEtherlands software eVOLution seminar (BENEVOL 2010)*, pages 145–155, Lille, France, December 2010.

[1] Noury Bouraqadi and Luc Fabresse. CLIC: A component model symbiotic with smalltalk. In *Proceedings of the International Workshop on Smalltalk Technologies*, Brest, France, August 2009. ACM.

Ateliers nationaux

[5] Zhi Yan, Luc Fabresse, Jannik Laval, and Noury Bouraqadi. Métriques pour le benchmarking de performance d'exploration multi-robots. In Laurent Vercouter and Gauthier Picard, editors, *23es Journées Francophones sur les Systèmes Multi-Agents (JFSMA'15)*, pages 9–18, Rennes, France, June 2015. Cépaduès.

[4] Zhi Yan, Luc Fabresse, Jannik Laval, and Noury Bouraqadi. Simulating multi-robot exploration using ros and morse. In *Proceedings of 9th National Conference on "Control Architecture of Robots" Control Architectures of Robots (CAR'14)*, Paris, France, June 2014.

[3] Luc Fabresse, Jannik Laval, and Noury Bouraqadi. Towards quality assurance tests for mobile robots. In *Proceedings of 8th National Conference on "Control Architecture of Robots" Control Architectures of Robots (CAR'13)*, Angers, France, June 2013.

[2] Noury Bouraqadi, Luc Fabresse, and Arnaud Doniec. On fleet size optimization for multi-robot frontier-based exploration. In *Proceedings of 7th National Conference on "Control Architecture of Robots" Control Architectures of Robots (CAR'12)*, Nancy, France, May 2012.

[1] Luc Fabresse. Des modèles aux programmes à base de composants : Besoin de langages à composants. In *GT COSMAL du GDR GPL, Journée thématique Composition d'objets, de composants et de services*, Janvier 2009.

Rapports

[4] Guillermo Polito, Stéphane Ducasse, Noury Bouraqadi, and Luc Fabresse. Extended results of Tornado: A Run-Fail-Grow approach for Dynamic Application Tayloring. Research report, Inria, 2014.

[3] Noury Bouraqadi, Luc Fabresse, Jannik Laval, and Santiago Bragagnolo. A Robotic Infrastructure for Map Construction and Navigation. Technical report, Ecole des Mines de Douai, Jan 2014.

[2] Noury Bouraqadi, Luc Fabresse, Jannik Laval, and Santiago Bragagnolo. PhaROS: Concepts and Architecture. Technical report, Ecole des Mines de Douai, Avr 2013.

Ce document a été mis en page à l'aide du style visuel et typographique `classicthesis` développé par André Miede. Ce style fût inspiré par le livre de Robert Bringhurst sur la typographie intitulé *"The Elements of Typographic Style"*.

`http://code.google.com/p/classicthesis/`

Final Version as of 9 décembre 2015.

Langages réflexifs pour le développement d'applications de robotique mobile

Luc Fabresse

Dans un avenir proche, nous serons confrontés au besoin de développer rapidement et massivement des logiciels pour répondre aux nouveaux usages liés à l'essor de la robotique mobile et autonome. Or, le cycle de développement actuel de ces applications est généralement long : développement, compilation, déploiement et exécution. Dans ce contexte, mon objectif de raccourcir ce cycle et permettre un retour immédiat (dans une logique *live programming*) en proposant de nouveaux langages, outils et infrastructures aux développeurs. Mes travaux de recherche se concentrent sur les couches hautes des applications robotiques (avec des contraintes matérielles et temporelles plus faibles) et se fondent sur l'idée que les *langages réflexifs* sont de meilleurs candidats pour répondre à cet objectif, car ils offrent un haut niveau d'abstraction et intègrent déjà des mécanismes permettant l'adaptation et l'extension des programmes. Dans ce contexte, ce mémoire décrit les travaux que j'ai menés ou auxquels j'ai participé selon quatre thèmes principaux.

Dans le premier thème, nous avons proposé des *langages à composants* permettant d'exprimer l'architecture d'un logiciel directement dans son code source comme un assemblage de composants pouvant être distants ou pervasifs. Cette explicitation de l'architecture facilite ainsi l'adaptation des programmes.
Le deuxième thème est celui des langages à objets réflexifs qui offrent une représentation de leur propre infrastructure grâce à des méta-objets ainsi que des méta-opérations pour la modifier. Étendre le noyau d'un langage réflexif est toutefois complexe. Cela nécessite de modifier son amorçage qui est généralement en partie enfoui dans le code de sa machine virtuelle. Nous avons donc proposé une approche de plus haut niveau basée sur la co-exécution d'environnements par une même machine virtuelle pour amorcer un langage réflexif à partir d'une définition circulaire de lui-même.
Le troisième thème est lié aux ressources limitées disponibles sur un robot mobile (e. g. mémoire, capacité de calcul). Pour réduire la mémoire consommée par une application, nous avons proposé deux nouveaux mécanismes basés sur la réflexion : le premier élimine le code non-utilisé avant son déploiement et le second est une mémoire virtuelle au niveau applicatif permettant de gérer les objets non-utilisés mais référencés durant son exécution. Nous travaillons également sur la possibilité de décharger le processeur central du robot en déportant automatiquement certains calculs coûteux vers des architectures matérielles spécialisées comme des FPGAs.
Le quatrième thème présente des outils et des infrastructures pour améliorer le développement d'applications mono et multi-robots. Nous décrivons ainsi une infrastructure utilisant la réflexion pour déboguer à distance une application s'exécutant sur un robot mobile. Notre application cible est la cartographie d'un territoire inconnu par une flotte de robots. Pour faciliter la comparaison objective de solutions de cartographie multi-robots, nous avons proposé une infrastructure permettant de simuler le même code que celui qui serait déployé sur des robots réels ainsi que des métriques quantifiables (e. g. temps d'exploration, mémoire et CPU consommés).

À travers ces quatre thèmes, ces travaux proposent des solutions originales pour développer, tester, déployer, exécuter et évaluer des applications contraintes et mobiles comme celles pour robots autonomes. Ces travaux ouvrent de nouvelles pistes de recherche comme le développement agile d'applications robotiques ou encore l'adaptation dynamique de ces applications pour une meilleure prise en compte du contexte.

Mots-clés : langage réflexif, langage à composants, mémoire virtuelle applicative, débogage à distance, robotique mobile et autonome

Mines Douai
Département Informatique et Automatique,
941 rue Charles Bourseul, CS 10838, 59508 Douai Cedex