HAL
open science

Reconfiguration dynamique d'architectures logicielles : des métaclasses aux " nuages verts "

Thomas Ledoux

▶ **To cite this version:**

Thomas Ledoux. Reconfiguration dynamique d'architectures logicielles : des métaclasses aux " nuages verts ". Génie logiciel [cs.SE]. Université de Nantes, Faculté des sciences et des techniques, 2018. ⟨tel-01864344⟩

HAL Id: tel-01864344

https://theses.hal.science/tel-01864344v1

Submitted on 3 Sep 2018

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Autorisation HAL

HAL Authorization

École doctorale n° 601 : MathSTIC

Habilitation à Diriger les Recherches

HDR

Université de Nantes

Spécialité doctorale "Informatique"

présentée et soutenue publiquement par

Thomas Ledoux

le 17 Juillet 2018

Reconfiguration dynamique d'architectures logicielles : des métaclasses aux « nuages verts »

Jury

Mme Laurence DUCHIEN	Professeur, Université de Lille	Rapporteur
M. Michel RIVEILL	Professeur, Polytech'Nice Sophia	Rapporteur
M. Jean-Bernard STEFANI	Directeur de recherche, INRIA Grenoble Rhône-Alpes	Rapporteur
M. Pierre COINTE	Professeur, IMT Atlantique	Examinateur
M. Thierry COUPAYE	VP Recherche Internet des objets, Orange Labs	Examinateur
M. Rachid GUERRAOUI	Professeur, École polytechnique fédérale de Lausanne	Examinateur
M. Claude JARD	Professeur, Université de Nantes	Président

Laboratoire LS2N, IMT Atlantique Campus de Nantes

Remerciements

A mes grands-parents

Je remercie très chaleureusement toutes les personnes qui ont contribué de près ou de loin à l'aboutissement de cette HDR.

Je tiens tout d'abord à remercier les membres de mon jury pour leur participation et leur bienveillance. Merci à Laurence Duchien, Michel Riveill et Jean-Bernard Stéfani pour leurs commentaires constructifs en tant que rapporteurs. Merci à Thierry Coupaye, Rachid Guerraoui, Claude Jard pour l'intérêt qu'ils ont manifesté pour mes travaux. Un *special thanks* pour Pierre Cointe pour ses relectures attentives ainsi que pour son soutien tout au long de ces années.

Les travaux présentés ici sont le fruit de collaborations avec de nombreux collègues et d'excellents doctorants. Je tiens tout particulièrement à remercier certains d'entre eux. Tout d'abord, mes collègues de tous les jours, principalement Rémi Douence, Hervé Grall, Adrien Lèbre, Jean-Marc Menaud, Jacques Noyé, Jean-Claude Royer, Mario Südholt (et depuis peu, Hugo Brunelière et Charles Prud'homme) avec lesquels j'ai partagé débats passionnés et contributions. Par ailleurs, je veux aussi mentionner des coopérations fructueuses avec d'autres collègues parmi lesquels Olivier Barais, Françoise Baude, Sara Bouchenak, Fabienne Boyer, Noury M. Bouraqadi-Saâdani, Thierry Coupaye, Noël De Palma, Daniel Hagimont, Jean-Marc Jézéquel, Frédéric Jouault, Jonathan Lejeune, Michel Riveill, Jean-Louis Pazat, Eric Rutten, Pierre Sens, Rémi Sharrock. Et enfin, je ne veux pas oublier mes doctorants Frederico Alvares, Fabien Baligand, Pierre-Charles David, Simon Dupont, Sabbir Hasan, Yousri Kouki, Marc Léger avec lesquels les échanges scientifiques ont été fertiles et humainement riches.

Je tiens également à remercier la Direction et certains membres de IMT Atlantique pour leurs encouragements, mes collègues de la formation par apprentissage FIL et bien sûr Narendra Jussien qui a m'a chaleureusement soutenu quand il était à la direction du Département.

Enfin, je veux remercier du fond du cœur ma famille, mes parents et mes proches pour leur soutien dans ce voyage au long cours. J'ai du temps à rattraper auprès de mes enfants Baptiste, Ferdinand et Garance. Pour finir, il est toujours délicat de remercier cet Autre qui vous accompagne. Merci Laure-Anne pour ton abnégation et ton soutien inconditionnel même par gros temps.

Thomas

Table des matières

1 Introduction **1**
1.1 Fil conducteur de mes travaux . . . 1
1.1.1 Eco-système professionnel . . . 1
1.1.2 Problématique et approche . . . 2
1.1.3 Démarche et aperçu des résultats . . . 3
1.2 Contributions principales . . . 4
1.2.1 Préambule . . . 4
1.2.2 Adaptation dynamique comme objet de première classe . . . 4
1.2.3 Reconfiguration dynamique et fiable des architectures logicielles . . . 5
1.2.4 Eco-élasticité dans le Cloud « vert » et frugal . . . 5
1.3 Organisation du document . . . 6
1.3.1 Comment lire ce document ? . . . 6

I Reconfiguration dynamique : pourquoi ? **7**

2 Adaptation du logiciel à l'exécution **9**
2.1 Contexte et problématique . . . 9
2.2 Définitions et terminologie . . . 10
2.2.1 Du dictionnaire... . . . 10
2.2.2 ... aux types d'adaptation du logiciel . . . 11
2.2.3 Premier bilan . . . 11
2.3 Motivations de l'adaptation . . . 12
2.3.1 Les raisons de l'adaptation . . . 12
2.3.2 Atouts de l'adaptation dynamique . . . 13
2.3.3 Quelques illustrations d'adaptation dynamique . . . 14
2.4 Modélisation de l'adaptation . . . 16
2.4.1 Étapes du processus d'adaptation . . . 16
2.4.2 Sujets de l'adaptation (*What*) . . . 16
2.4.3 Mécanismes d'adaptation (*How*) . . . 17
2.4.4 Moments d'adaptation (*When*) . . . 18
2.4.5 Acteurs de l'adaptation (*Who*) . . . 19
2.4.6 Quelques mots sur l'observation (*Where*) . . . 20
2.4.7 Et la prise de décision ? . . . 21
2.5 Retour d'expériences : OpenCorba . . . 21
2.5.1 Problématique et motivations . . . 22
2.5.2 Description . . . 22
2.5.3 Relecture de mes travaux . . . 23
2.6 Synthèse . . . 23

3 Adaptation dynamique : pistes de travail 25
3.1 Propriétés d'adaptabilité du logiciel . . . 25
3.1.1 Réification . . . 25
3.1.2 Modularité et Composabilité . . . 27
3.1.3 Support des adaptations non anticipées . . . 27
3.2 Adaptation comme préoccupation . . . 27
3.2.1 Externalisation . . . 28
3.2.2 Expression . . . 28
3.3 Adaptation dynamique, garanties du processus et qualité logicielle . . . 29
3.3.1 Fiabilité . . . 29
3.3.2 Performance . . . 32
3.3.3 Évaluation qualitative de l'adaptation dynamique . . . 33
3.4 Bilan . . . 33

II Reconfiguration dynamique : comment ? 35

4 Auto-reconfiguration d'architectures logicielles et langages dédiés 37
4.1 Prérequis (a.k.a. interfaces requises pour comprendre ce chapitre) . . . 37
4.1.1 Modèle de composants Fractal . . . 37
4.1.2 Langages dédiés . . . 39
4.1.3 Autonomic computing . . . 40
4.2 Safran : une extension de Fractal pour créer des applications auto-adaptables . . . 41
4.2.1 Conception et architecture générale . . . 42
4.2.2 Mise en oeuvre d'un langage dédié à l'adaptation . . . 43
4.3 WildCAT : modélisation et observation du contexte d'exécution . . . 47
4.3.1 Motivations et objectifs . . . 47
4.3.2 Modélisation du contexte d'exécution . . . 47
4.3.3 Interface de programmation . . . 49
4.4 FPath/FScript : navigation et reconfiguration dans les architectures Fractal . . . 52
4.4.1 Problématique et motivations . . . 53
4.4.2 Navigation avec FPath . . . 53
4.4.3 Reconfiguration avec FScript . . . 56
4.5 Application adaptative avec Safran . . . 59
4.6 Conclusion et perspectives (a.k.a. interfaces fournies) . . . 61
4.6.1 Lien avec la programmation par aspects . . . 61
4.6.2 Généricité de l'approche FPath/FScript . . . 62
4.6.3 Bilan . . . 63

5 Fiabilité des reconfigurations dynamiques dans les architectures Fractal 65
5.1 Prérequis (a.k.a. interfaces requises pour comprendre ce chapitre) . . . 65
5.1.1 Sûreté de fonctionnement et transactions . . . 65
5.1.2 Logique du premier ordre et Alloy . . . 67
5.2 Approche multi-étapes pour des reconfigurations dynamiques fiables . . . 68
5.2.1 Motivation et problématique . . . 68
5.2.2 Chaîne de validation des reconfigurations . . . 69
5.3 Spécification des (re)configurations Fractal . . . 71
5.3.1 Spécification des configurations . . . 72
5.3.2 Vérification de la cohérence du modèle et des configurations . . . 75
5.3.3 Spécification des reconfigurations . . . 77
5.4 Reconfigurations transactionnelles . . . 80

5.4.1 Modèle transactionnel 80
5.4.2 Propriétés ACID 81
5.5 Evaluation et expérimentations de Fractal TXR 84
5.5.1 Evaluation des performances 84
5.5.2 Auto-réparation d'un serveur Java EE dans un cluster 86
5.6 Conclusion et perspectives (a.k.a. interfaces fournies) 88
5.6.1 Généralisation à d'autres modèles de composants 88
5.6.2 De la difficulté à valoriser un prototype de recherche 88
5.6.3 Bilan 89

III Reconfiguration dynamique : pour qui ? 91

6 Cloud computing et empreinte énergétique 93
6.1 Contexte 93
6.1.1 L'informatique en nuage 93
6.1.2 Transition écologique et développement durable 95
6.2 Problématique : enjeux énergétiques des TIC 95
6.2.1 Informatique durable 95
6.2.2 Empreinte énergétique liée à l'informatique en nuage et aux centres de données 96
6.2.3 Un début de réponse : l'élasticité 97
6.2.4 Etat des lieux – Bilan 98
6.3 Pistes de travail 98
6.3.1 Hypothèses de départ 98
6.3.2 Effacement de la consommation énergétique du logiciel à l'exécution 101
6.3.3 Synergie entre les modèles de service 102
6.3.4 Intégration des énergies renouvelables 102
6.4 Bilan : vers un Cloud frugal 103

7 Eco-élasticité logicielle pour un Cloud frugal 105
7.1 Vers une auto-administration fine de la gestion de ressources 105
7.1.1 Architecture en nuages 105
7.1.2 Auto-configuration et Auto-optimisation 107
7.2 Etablir un contrat entre couches XaaS : le langage CSLA 107
7.2.1 Motivations et survol du langage 107
7.2.2 Evaluation et expérimentations 110
7.2.3 Conclusion 115
7.3 Réaliser l'éco-élasticité multi-boucle dans le Cloud 117
7.3.1 Motivations 117
7.3.2 Coordination de boucles autonomiques 118
7.3.3 Evaluation et expérimentation 122
7.3.4 Conclusion 123
7.4 Programmer l'éco-élasticité 124
7.4.1 Motivations 124
7.4.2 Gestion de l'élasticité multi-couche de bout en bout 124
7.4.3 Evaluation et expérimentation 129
7.4.4 Conclusion 132
7.5 Proposer des services sans empreinte carbone dans le SaaS 133
7.5.1 Motivations 133
7.5.2 Intégrer l'énergie verte dans le Cloud 133
7.5.3 Vers des services verts 136

7.5.4 Evaluation et expérimentation 138
7.5.5 Conclusion 140
7.6 Bilan 141

8 Bilan des contributions et perspectives 143
8.1 Bilan 143
8.1.1 Résumé des travaux 143
8.1.2 Contributions dans le domaine du Cloud computing 144
8.2 Perspectives 144
8.2.1 Vers un gestionnaire autonomique générique pour les couches XaaS 145
8.2.2 Vers le développement de microservices verts pour le Cloud 147
8.2.3 Vers des patterns de coordination de boucles de contrôle hétérogènes pour l'auto-adaptation dans le Fog 148

IV Annexe : Curriculum Vitae détaillé 171

Table des figures

2.1 Les trois états d'un logiciel (auto-)adaptable *context-aware*. 12
2.2 Middleware adaptatif pour la mobilité [FHS+06] 15
2.3 Complex Event Processing [CM12] 21
2.4 Un titre intrigant, une couverture pleine de mystère qui font rêver ! [KR91] 22

3.1 MOP, introspection, intercession [MSKC04] 26
3.2 Exemple de configuration avec l'ADL Wright 30
3.3 Transitions d'états pour obtenir un état "quiescent" [KM98] 31

4.1 Une architecture Fractal 38
4.2 Boucle MAPE-K [KC03] 41
4.3 Architecture générale Safran [DL06] 42
4.4 Contrôleur d'adaptation Fractal [Dav05] 44
4.5 Méta-modèle de WildCAT [Dav05] 48
4.6 Domaine contextuel `sys` [Dav05] 48
4.7 Arborescence WildCAT 50
4.8 Méta modèle FPath [DLLC09] 54
4.9 Exemple modèle FPath [DLLC09] 55
4.10 Code FPath [DLLC09] 55
4.11 Reconfiguration d'une architecture Fractal 59

5.1 Taxonomie de la sûreté de fonctionnement [ALRL04] 66
5.2 Une chaîne de validation pour la reconfiguration 69
5.3 Prévention des fautes : (a) Analyse statique (b) Validation dynamique 70
5.4 Les trois niveaux de contraintes d'intégrité [Lég09]. 74
5.5 Processus de vérification des contraintes. 75
5.6 Deux types de mise à jour [Lég09]. 81
5.7 Vérification des contraintes d'intégrité [Lég09]. 83
5.8 Architecture Fractal du composant HelloWorld. 85
5.9 Scénario de reconfiguration transactionnelle dans un cluster Java EE [Lég09]. . . . 87

6.1 Une architecture multi-couche orientée services [Dup16]. 94
6.2 Consommation d'électricité des TIC en 2012. 96
6.3 Élasticité, sur-dimensionnement et sous-dimensionnement [Dup16]. 97
6.4 Un exemple d'architecture multi-couche orientée services avec SLA [KADL14]. . . 99
6.5 Consommation en RAM et CPU de Windows dans le temps (source blog GreenIT). . 100
6.6 Communication inter-boucle multi-couche [AdOLLM12]. 102

7.1 Architecture en nuages. 106
7.2 Exemple d'évaluation d'un SLO dans CSLA. 109
7.3 Modélisation des applications Cloud 111

7.4 Dépendances entre SLA 112
7.5 Une architecture n-tiers dans le Cloud [KADL14]. 113
7.6 Résultats des expériences [KADL14]. 116
7.7 Exemple d'une interférence multi-boucle. 117
7.8 Architecture multi-boucle pour le Cloud Computing [ASL12]. 118
7.9 Modèle architectural pour la coordination des boucles autonomiques [ASL13]. . . . 119
7.10 Hiérarchie des événements/actions [ASL13]. 120
7.11 Reconfiguration multi-couche suite à une pénurie d'énergie [SBK+16]. 123
7.12 Gestionnaire autonomique ElaStuff [DBAL17]. 126
7.13 Règles ElaScript dans Eclipse [Dup16]. 128
7.14 Architecture globale [Dup16]. 129
7.15 Résultats des expériences [Dup16]. 131
7.16 *Energy as a Service* et « green SLA ». 134
7.17 Virtualisation de l'énergie verte. 135
7.18 SaaS *green energy-aware* [Has17]. 136
7.19 Architecture SaaScaler [Has17]. 137
7.20 Contrôleurs hybrides [Has17]. 138
7.21 Consommation énergétique de RUBiS avec 2 recommandations. 139
7.22 Expérimentation SaaScaler [HALP17]. 140

8.1 Boucle autonomique générique XaaS. 145
8.2 Architecture CoMe4ACloud [BASA+18]. 146

Liste des tableaux

2.1 Synthèse terminologie d'adaptation 23

4.1 Actions primitives FScript 57
4.2 Comparaison de code [DLLC09] 59
4.3 Correspondance entre les concepts AOP et les mécanismes de Safran 62

5.1 Comparaison du temps d'exécution (*ms*) avec/sans transaction [LLC10]. 86
5.2 Comparaison du temps d'exé. (*ms*) d'une reconfiguration avec propriétés ACID [LLC10]. 86

7.1 SLA entre le IaaS et le SaaS 114
7.2 SLA entre le SaaS et ses clients 114
7.3 API élasticité multi-couche [DBAL17]. 125
7.4 Actions d'élasticité et critères d'adaptation : tendances [Dup16]. 127
7.5 Résultat des *stratégies d'élasticité* sur le choix des *tactiques*. 132

Listings

4.1 WildCAT : exemple mode PULL . . . 50
4.2 WildCAT : exemple n°1 mode PUSH . . . 52
4.3 WildCAT : exemple n°2 mode PUSH . . . 52
4.4 WildCAT : exemple n°3 mode PUSH . . . 52
4.5 Reconfiguration Fractal en Java. . . . 53
4.6 Fonction et action en FScript. . . . 57
4.7 Création d'un composant cache en FScript. . . . 58
4.8 Connexion d'un composant cache en FScript. . . . 58
4.9 Politique d'adaptation Safran. . . . 60
5.1 Action introduisant un cycle en FScript. . . . 68
5.2 Action potentiellement correcte en FScript. . . . 70
5.3 *noCycle* en Alloy. . . . 76
5.4 *noSharing* en Alloy. . . . 76
5.5 Contraintes exprimées en FScript. . . . 76
5.6 *add* en Alloy. . . . 79
5.7 *addSafe* en FScript. . . . 79
5.8 Exemple de reconfiguration en FScript. . . . 85
7.1 Exemple CSLA. . . . 109
7.2 Règles App tier . . . 114
7.3 Règles Opt tier . . . 114
7.4 Règle applicative . . . 114
7.5 Planification d'un *Energy Shortage* . . . 121
7.6 Exemple d'élasticité multi-couche dans ElaScript . . . 128
7.7 Tactiques IaaS dans ElaScript . . . 130
7.8 Tactiques SaaS dans ElaScript . . . 130
7.9 Tactique *cross-layer* dans ElaScript . . . 130

Chapitre 1

Introduction

1.1 Fil conducteur de mes travaux

Ce mémoire présente une synthèse des travaux de recherche que j'ai réalisés depuis 1998. Ils ont pour fil conducteur la problématique de la *reconfiguration dynamique d'architectures logicielles* que j'avais déjà initié lors de ma thèse avec la contribution OpenCorba [Led98] [Led99].

1.1.1 Eco-système professionnel

Ces recherches ont été menées au sein du département informatique de l'Ecole des Mines de Nantes [1], dans l'équipe « objet » (1994-2002), puis Obasco (2002-2008) et enfin Ascola (2008-2017). Obasco et Ascola sont des équipes communes INRIA Rennes-Bretagne Atlantique et UMR LINA [2]. Pour bien comprendre le contexte scientifique dans lequel j'ai évolué et son influence, je présente brièvement ces équipes dont le domaine de recherche est le génie logiciel, les langages de programmation et les systèmes distribués.

En rejoignant Pierre Cointe à l'Ecole des Mines de Nantes, j'ai intégré l'équipe en construction autour de la programmation par objets (équipe « objet »). Ma communauté scientifique était celle du futur GDR GPL (Génie de la Programmation et du Logiciel) et mes conférences cibles ECOOP, OOPSLA pour l'international et LMO pour le national. Avec l'équipe « objet », nous avons participé à la diffusion de la technologie Java (conférences OCM, compilateur et machine virtuelle de l'IDE Eclipse) et à l'émergence de la programmation par aspects (et sa conférence internationale AOSD, puis le réseau d'excellence européen AOSD).

L'équipe Obasco (2002-2008), dirigée par Pierre Cointe, s'intéressait à la « programmation post-objets ». Obasco s'attaquait au problème général d'adapter le logiciel à ses usages en partant des techniques de réflexion [Smi84] et de méta-programmation [KR91] et en développant des outils de construction d'architectures logicielles à base de patrons de conception [GHJV95], de composants [Szy02] et d'aspects [KLM+97]. Notre fil conducteur était la programmation modulaire, l'évolution d'une programmation à petite échelle (*in the small*) avec des langages à objets à la Smalltalk ou Java vers une programmation à plus grande échelle (*in the large*) [DK75] telle qu'elle a émergé avec les modèles de composants et la programmation par aspects.

Dans la continuité du projet Obasco, l'équipe Ascola (2008-2017), dirigée par Mario Südholt, s'est attaquée au problème de la structuration et de l'évolution du logiciel en développant de nouvelles abstractions pour la programmation des architectures logicielles, leur représentation en termes de langages de programmation expressifs et leur implémentation à la fois correcte et efficace. L'investigation des relations entre langages dédiés [vDKV00], langages d'aspects [EFB01] et langages de composition [TOHS99] comme l'étude des fondements de la programmation par aspects et de leurs

1. IMT Atlantique, campus de Nantes depuis le 1er janvier 2017 suite à la fusion avec Telecom Bretagne.
2. LS2N depuis le 1er janvier 2017 suite à la fusion avec l'UMR IRCCyN.

propriétés de composition au moyen de sémantiques formelles sont des exemples de recherche de l'équipe Ascola. Au fil des ans, une majorité des chercheurs de l'équipe s'est tournée vers les systèmes distribués comme domaine d'application de ses recherches.

Au cours des années 2000, j'ai élargi le champ de ma communauté scientifique en me rapprochant du futur GDR RSD (Réseaux et Systèmes Distribués) notamment du fait de mes travaux sur les composants/services répartis. A l'aube des années 2010, une activité « systèmes virtualisés [BDF+03] et *Green IT* » a émergé au sein de l'équipe Ascola sous l'impulsion de Jean-Marc Menaud et Adrien Lèbre. L'objectif était d'optimiser la consommation de ressources (applicative, virtuelle, matérielle) pour diminuer l'empreinte énergétique des centres de données. Cette problématique peut être abordée de différentes façons en adressant le niveau infrastructure, via des techniques de virtualisation [HLM+09], ou encore le niveau langage, via des langages dédiés [PLM10]. Les activités actuelles de ce sous-groupe [3] – dont je fais partie – concernent aujourd'hui aussi bien l'éco-élasticité multi-niveau dans les infrastructures virtualisées de type *Fog/Cloud computing* que l'intégration d'énergie renouvelable dans les centres de données.

1.1.2 Problématique et approche

Aujourd'hui, la complexité, la diffusion et l'ubiquité croissante des systèmes logiciels autant que l'environnement économique compétitif à l'extrême rendent nécessaire la conception et le développement de logiciels capables d'évoluer dans l'espace et dans le temps. Dans l'espace, car l'omni-présence du réseau Internet, la prolifération des objets numériques et des réseaux sans-fil (Internet des objets), ont donné une nouvelle vision de la production/consommation du logiciel. Dans le temps, car les cycles de vie du logiciel sont de plus en plus courts pour rester compétitif dans le contexte de la mondialisation.

Pour faire face à une hétérogénéité importante des environnements logiciels, à une variabilité et disponibilité dynamique des ressources, à une forte croissance des évolutions technologiques, à des systèmes d'information large échelle, les principaux acteurs du logiciel (l'architecte, le développeur) doivent reconsidérer le développement et l'exécution d'applications dans un contexte d'un genre nouveau. En effet, dans les environnements traditionnels avant l'essor de l'Internet, le développement et l'exécution d'application s'effectuaient en supposant que le support d'exécution était connu à l'avance. Lorsqu'une application allouait une ressource, une non-disponibilité de celle-ci en cours d'exécution conduisait à une terminaison non prévue de l'application. Dans les environnements que nous considérons – du fait de l'hétérogénéité, de la variabilité et de l'évolutivité – de nombreux changements peuvent intervenir et ceux-ci ne doivent pas être considérés comme des erreurs fatales, mais doivent au contraire être pris en compte pour ajuster, adapter dynamiquement l'application à son nouveau contexte.

Pour répondre à cet enjeu, l'objectif est tout d'abord de déterminer une unité de construction du logiciel qui permettra par assemblage de construire des logiciels plus conséquents mais qui sera aussi une unité de substitution pour ajuster le logiciel en cours de vie. Un *composant logiciel* tel que décrit par [BCL+04] peut être vu comme un module unitaire de granularité intéressante dans un système complexe. Plus gros qu'une classe, plus petit qu'un *framework* (cadriciel), le composant sera le grain privilégié dans des approches de programmation modulaire à large échelle. Les composants logiciels créés doivent savoir évoluer et s'adapter pour répondre à de nouveaux besoins, prendre différentes formes et passer de l'une à l'autre via une *reconfiguration*. Pour répondre au défi de la variabilité dynamique du contexte, cette transformation doit être possible pendant l'exécution du logiciel. Ainsi, la *reconfiguration dynamique d'architectures logicielles à base de composants* est une piste de recherche importante pour répondre aux défis posés par l'informatique d'aujourd'hui. Cet axe de recherche a été le fil conducteur de mes travaux.

3. Depuis le 1 novembre 2017, ce sous-groupe est devenu une équipe à part entière : il s'agit de l'équipe Stack (`http://stack.inria.fr/`) dirigée par Adrien Lèbre.

1.1.3 Démarche et aperçu des résultats

Depuis le début de mes travaux de recherche, je m'intéresse au paradigme de modularité, de « séparation des préoccupations » (*separation of concerns* [Dij76]) qui proposent de mieux structurer les systèmes logiciels (applications, architectures logicielles, mais aussi langages et intergiciels) pour favoriser leur réutilisation et leur évolution.

En effet, dans un premier temps, j'ai utilisé comme moyen d'expression la réflexion pour proposer une taxonomie de métaclasses pour rétro-concevoir des bibliothèques de classes Smalltalk justifiant ainsi l'intérêt de la réflexion dans la conception de langages à objets [LC96]. C'est également à cette époque que j'ai proposé avec deux autres doctorants (N. Bouraqadi, F. Rivard) un nouveau modèle pour organiser les architectures à méta-niveaux [BSLR98]. Finalement, l'application de ces techniques au domaine des intergiciels a été le sujet de ma thèse et m'a permis de concevoir et d'implémenter OpenCorba un bus logiciel ouvert basé sur un protocole à méta-objets en Smalltalk [Led98] [Led99]. Ensuite, intéressé par les liens entre la réflexion et la programmation par aspects, j'ai montré que la méta-programmation pouvait être un substrat possible pour la conception de langages d'aspects [BSL04], offrant ainsi de nouvelles propriétés comme le tissage dynamique dans AspectJ [DLBs01].

Ensuite, dans le cadre du projet RNTL ARCAD (2000-2004) – qui m'a permis de fréquenter des personnalités aussi diverses que M. Riveill (ESSI), D. Caromel (I3S), T. Coupaye (France Telecom R&D), J-B. Stefani (INRIA Rhône-Alpes) – je me suis concentré sur le concept d'adaptabilité dynamique dans les architectures logicielles à base de composants. ARCAD a participé à l'émergence du modèle de composants Fractal (fractal.ow2.org) [BCL$^+$04], à la définition d'un patron d'auto-administration pour logiciel à base de composants avec le framework Safran [Dav05], à la réalisation d'un intergiciel pour le monitoring (WildCAT [DL05]) mais également à la proposition d'un aspect d'adaptation pour la reconfiguration dynamique de composants Fractal [DL06].

Avec le projet RNTL Selfware (2005-2008) – et ses partenaires en particulier D. Hagimont (ENSEEIHT), N. De Palma (INRIA Rhône-Alpes) et encore T. Coupaye – j'ai abordé un nouveau domaine qui était déjà un domaine cible mais implicite de mes précédents travaux : celui de l'*autonomic computing* [KC03]. Mon idée était d'utiliser les approches langages dédiés (au sens *Domain Specific Language* [vDKV00]) pour développer des langages sur mesure pour l'administration autonomique de systèmes répartis. L'intérêt du langage dédié est d'être spécifiquement conçu pour un domaine d'application particulier, celui de la reconfiguration dynamique dans notre cas. J'ai notamment contribué à la définition du langage de reconfiguration FScript [DLLC09] pour le modèle de composants Fractal ; j'ai participé à la spécification des (re)configurations Fractal et à l'intégration d'un moniteur transactionnel dans l'interpréteur FScript pour rendre les reconfigurations fiables [LLC10] ; enfin, en parallèle avec mes partenaires de France Telecom R&D (N. Rivierre), intéressé par la gestion de la qualité de service (QoS) dans les orchestrations de Web services, j'ai contribué à la conception d'un langage dédié permettant de spécifier des règles d'adaptation dynamique pour garantir la QoS lors de fluctuations de l'environnement d'exécution [BRL08].

A partir de 2010, je me suis rapproché de mon collègue Jean-Marc Menaud, et je me suis orienté vers la thématique du *Green computing* dans le but d'optimiser l'empreinte énergétique des centres données qui accompagnent l'essor spectaculaire du *Cloud computing* [dOL12]. L'idée phare est que le logiciel hébergé par un centre de données doit lui-même participer à la réduction de la consommation énergétique. D'où des problématiques d'optimisation multi-niveau, multicritère, de coordination de boucles autonomiques entre l'application et l'infrastructure virtuelle l'hébergeant [ASL12]. La reconfiguration dynamique d'architectures Cloud multi-niveau sera dirigée par des compromis qualité service (QoS)–empreinte énergétique (un composant gourmand en énergie rendant en général un service de meilleure qualité) [KADL14]. Ces compromis seront possibles car les critères de QoS seront spécifiés en amont dans un contrat de service SLA (*Service Level Agreement*). C'est dans le cadre du projet ANR MyCloud (2010-2013) – et ses partenaires, en particulier Pierre Sens (LIP6) et Sara Bouchenak (INRIA Rhône-Alpes) – que nous avons abordé cette problématique et défini le

langage CSLA [Kou13][SBK+16], prenant finement en compte les caractéristiques du Cloud. Pour conclure, la gestion autonomique de l'élasticité multi-couche des applications dans le Cloud pour une utilisation efficiente des ressources [Dup16] devient ainsi un axe majeur de ma recherche ces dernières années [DLAL15, DBAL17].

Avec le projet EPOC (2013-2017) du Labex CominLabs – et ses partenaires en particulier A.-C. Orgerie (IRISA) et J.-L. Pazat (INSA Rennes) – l'idée est d'intégrer des sources d'énergie renouvelable comme objet de première classe dans la prise de décision pour optimiser l'empreinte carbone du Cloud (*green energy vs brown energy*) [HKLP14][HKLP17]. Mon objectif est alors de relier ces résultats à nos travaux précédents en proposant des contrats de type « Green SLA » à l'utilisateur final des architectures en nuage (Cloud) [HALP16, HALP17].

1.2 Contributions principales

L'objectif de cette section est d'introduire les contributions principales de ce mémoire. J'ai volontairement choisi d'exclure les contributions qui s'écartait du fil conducteur choisi à savoir la *reconfiguration dynamique d'architectures logicielles.*

1.2.1 Préambule

Les travaux présentés ici reposent sur des collaborations riches et variées. Tout d'abord, des collaborations scientifiquement et humainement fructueuses avec mes doctorants : Pierre-Charles David (2005), Fabien Baligand (2008), Marc Léger (2009), Frederico Alvares (2013), Yousri Kouki (2013), Simon Dupont (2016), Sabbir Hasan (2017). Mais également des travaux communs avec mes collègues de tous les jours, principalement Pierre Cointe, Rémi Douence, Hervé Grall, Adrien Lèbre, Jean-Marc Menaud, Jacques Noyé, Jean-Claude Royer, Mario Südholt (et depuis peu, Hugo Brunelière et Charles Prud'homme). Et enfin, des coopérations avec d'autres collègues au niveau national parmi lesquels Olivier Barais, Sara Bouchenak, Fabienne Boyer, Noury M. Bouraqadi-Saâdani, Thierry Coupaye, Noël De Palma, Daniel Hagimont, Jean-Marc Jézéquel, Frédéric Jouault, Jonathan Lejeune, Michel Riveill, Jean-Louis Pazat, Pierre Sens, Rémi Sharrock.

Les travaux rapportés dans le présent document ont fait l'objet de publications internationales dans les conférences OOPSLA, CCGrid, Coordination, CBSE, ICSOC, …, les journaux IEEE Transactions on Sustainable Computing, IEEE Transactions on Cloud Computing, Future Generation Computer Systems, SIGOPS Operating Systems Review, Annals of Telecommunications, …, mais aussi des chapitres de livre aux éditions Wiley-IEEE Press, Addison-Wesley, IGI Global, ….

La plupart de ces travaux ont été réalisés/utilisés dans le cadre des projets RNTL ARCAD (adaptation dynamique de composants répartis, 2000-2004), RNTL Selfware (plate-forme répartie sous administration autonomique, 2005-2008), ADT INRIA Galaxy (*an open agile SOA framework*, 2008-2010), ANR MyCloud (SLA et qualité de service pour Cloud Computing, 2010-2014), FSN OpenCloudware (*Think to PaaS for Multi-IaaS Cloud Computing*, 2012-2014), Labex CominLabs EPOC (*Energy proportional and opportunistic computing systems*, 2013-2017).

1.2.2 Adaptation dynamique comme objet de première classe

Pour favoriser la réutilisation et l'évolution des applications informatiques, nous proposons de considérer l'adaptation dynamique comme une préoccupation transverse, un *objet de première classe* [DL03]. La modularisation de l'adaptation (*découplage spatial*) évite de polluer le code métier d'une application avec la logique d'adaptation. La possibilité d'attacher et de détacher dynamiquement des politiques d'adaptation à l'application (*découplage temporel*) évite de prévoir à l'avance tous les cas de figure possibles et les adaptations appropriées à chacun.

Les applications à base de composants forment un bon substrat pour la construction d'applications adaptables. En nous appuyant sur les propriétés intrinsèques du modèle de composants

Fractal [BCL+04], nous avons proposé des langages pour faciliter l'adaptation dynamique d'architectures : (i) FPath, un langage dédié à la navigation dans les architectures Fractal (introspection) ; et (ii) FScript, un langage dédié à la reconfiguration dynamique d'architectures Fractal (intercession) [DLLC09]. En plus de leur syntaxe concise et dédiée à leur domaine, FPath et FScript garantissent par construction de bonnes propriétés comme la terminaison (requête s'exécutant en temps borné dans FPath, pas de boucle infinie dans FScript).

Le processus d'adaptation étant par essence réactif, nous avons étendu le modèle Fractal pour rendre les applications auto-adaptables avec le framework Safran [Dav05] [DL06]. Ce dernier permet d'attacher dynamiquement des politiques d'adaptation aux composants Fractal d'une application, faisant de ces derniers des composants adaptatifs. Safran utilise FScript pour l'écriture des règles de reconfiguration. Safran embarque également WildCAT, un framework facilitant la construction d'applications sensibles au contexte (*context-aware*) [DL05]. WildCAT est basé sur un moteur de *Complex Event Processing* (CEP), permettant de traiter/filtrer des événements dans le but d'identifier les événements significatifs dans un nuage d'événements [HSB09]. Finalement, Safran peut être vu comme un patron d'auto-administration pour les architectures à base de composants Fractal, l'adaptation dynamique étant externalisée comme objet de première classe.

1.2.3 Reconfiguration dynamique et fiable des architectures logicielles

Rendre adaptable un logiciel à l'exécution ne doit pas se faire au détriment de son intégrité. Pour mettre en œuvre l'adaptation, il existe donc une tension forte entre ouverture (flexibilité, dynamicité) et sûreté de fonctionnement (fiabilité, disponibilité). En s'inspirant des travaux sur la sûreté de fonctionnement [ALRL04], nous avons proposé une « chaîne de validation » permettant d'assurer la reconfiguration dynamique et fiable des architectures Fractal à partir d'un script FScript [DLG+08]. L'idée principale est de lever en amont les risques d'erreur dans le script de reconfiguration pour maximiser la disponibilité de l'application. Notre chaîne de validation repose sur deux concepts majeurs : en amont, la *prévention de fautes* (réduire le plus possible les occurrences de fautes) ; en aval, la *tolérance aux fautes* (délivrer un service en dépit des fautes).

La partie amont distingue deux phases consécutives dans la chaîne de validation : une phase d'analyse statique des scripts FScript et une phase de simulation de ces scripts sur l'architecture cible à reconfigurer. La phase d'analyse statique a pour but de vérifier si le script a un sens vis-à-vis du modèle Fractal. Une analyse sémantique reposant sur une logique du premier ordre nous permet de rejeter des scripts syntaxiquement correct mais présentant une incohérence vis-à-vis du modèle. Ensuite, a lieu une deuxième phase matérialisée par une simulation de l'exécution du script – validé à l'étape précédente – sur l'architecture cible à reconfigurer. Cette simulation détecte les incompatibilités entre le script et l'architecture à reconfigurer. Une fois cette étape de prévention de fautes terminée, l'étape de la tolérance aux fautes est assurée par un moniteur transactionnel qui assure la cohérence de l'architecture en cas de fautes matérielles ou d'erreurs non détectées précédemment [Lég09] [LLC10].

1.2.4 Eco-élasticité dans le Cloud « vert » et frugal

Au cours de ces vingt dernières années, la consommation d'électricité liée au TIC a considérablement augmenté [Koo11, GC15]. Les TIC consomment aujourd'hui près de 10% de toute l'électricité consommée dans le monde, avec un impact carbone comparable à celui du transport aérien [BSC+17]. Avec le Cloud computing, la mutualisation des ressources dans des grands centres de données a permis de rationaliser la consommation d'énergie grâce aux techniques de virtualisation ou aux serveurs basse consommation. Cependant, le Cloud est aujourd'hui confronté au *Paradoxe de Jevons*[4]. Aussi, l'*élasticité* des ressources – propriété intrinsèque du Cloud – permettant à une application

4. le paradoxe énonce qu'à mesure que les améliorations technologiques augmentent l'efficacité avec laquelle une ressource est employée, la consommation totale de cette ressource peut augmenter au lieu de diminuer (1865).

hébergée dans les nuages de s'adapter aux demandes en (dés)approvisionnant des ressources de manière automatique, comme les autres techniques d'optimisation semblent impuissantes pour enrayer la surconsommation d'énergie.

En s'inspirant du concept d'innovation frugale [APR13], nous proposons que le logiciel hébergé dans le Cloud participe lui-même à la réduction de la consommation d'énergie. L'innovation frugale consiste à créer la « valeur suffisante » pour le client. Nous avons introduit le concept d'*élasticité logicielle* [Dup16] qui permet de déformer le logiciel en vue de renvoyer une valeur suffisante et moins énergivore au client. Cette valeur peut être établie en amont via des contrats SLA (*Service Level Agreement*) [Kou13] [KADL14]. Ce nouveau type d'élasticité fait donc apparaitre de nouveaux besoins comme les contrats de service SLA mais également des besoins en coordination de l'élasticité multi-couche. Chaque type d'élasticité (i.e., élasticité du logiciel et élasticité des ressources) étant administré par une boucle autonomique différente, nous avons proposé un framework pour la coordination multi-boucle optimisant l'empreinte énergétique [ASL12][ADOJ13]. Programmer ce type d'élasticité est également un nouveau besoin, aussi nous avons conçu et développé le langage dédié ElaScript [DBAL17] et le framework associé ElaStuff [Dup16] [DLAL15] pour « coder » l'élasticité multi-couche dans le *Cloud computing*.

Enfin, plus récemment, dans le contexte des centres de données alimentées par des énergies renouvelables, nous avons proposé une utilisation intelligente de l'énergie verte – qui devient objet de première classe – à la fois au niveau de l'infrastructure et de l'application pour réduire davantage l'empreinte carbone du Cloud, et offrir des « Green SLA » [HKLP14, HALP16, HKLP17, HALP17, HAL17].

1.3 Organisation du document

1.3.1 Comment lire ce document ?

Ce document est divisé en trois grandes parties. La première *"Reconfiguration dynamique : pourquoi ?"* a pour objectif d'attester la pertinence de cette thématique de recherche dans le domaine du logiciel. La deuxième *"Reconfiguration dynamique : comment ?"* présente mes solutions pour réifier, mettre en œuvre la reconfiguration dynamique dans les architectures logicielles. La troisième *"Reconfiguration dynamique : pour qui ?"* montre l'apport de la reconfiguration dynamique pour résoudre le problème bien réel qu'est la consommation énergétique dans le domaine des TIC. Plus précisément, nous proposerons des solutions pour diminuer l'empreinte énergétique des applications logicielles construites dans les nuages.

La première partie – même si elle reprend quelques contributions du projet RNTL ARCAD – est en partie plutôt inédite (i.e., non publiée) et s'inspire de l'état de l'art sur la reconfiguration dynamique. La deuxième partie est volontairement organisée de façon originale puisque chacun de ses chapitres est construit comme un composant logiciel. Ce dernier nécessite des *interfaces requises* et expose des *interfaces fournies* : il en est de même pour chaque chapitre de contribution qui nécessite des pré-requis pour la bonne compréhension du travail réalisé et expose des résultats et perspectives. La troisième partie est volontairement plus succincte même si elle représente 8 ans de travaux [5]. Il y a au moins deux raisons à cela. Tout d'abord, une HdR a une fin et ne doit pas être un roman ! Ensuite, ces dernières contributions sont bien vivantes – contrairement à mes travaux sur Fractal – et je ne voulais pas en livrer une synthèse trop figée.

5. les parties I et II représentent 12 ans de travaux en comparaison.

Première partie

Reconfiguration dynamique : pourquoi ?

Chapitre 2

Adaptation du logiciel à l'exécution

Étudier la thématique de l'adaptation du logiciel sans interruption à l'exécution appelle en préambule un certain nombre de questions. Quelles sont les motivations principales d'un mécanisme d'adaptation dynamique ? Pourquoi l'adaptation statique ne suffit-elle pas ? Qu'entend-on par adaptatif, adaptation, auto-adaptation ? En quoi consiste un processus d'adaptation ? Ce chapitre propose un certain nombre de réponses.

2.1 Contexte et problématique

Ces 20-30 dernières années, avec la transformation numérique dans tous les secteurs d'activité, la maitrise des sciences du logiciel est devenu un enjeu majeur de notre société. Un certain nombre d'évolutions récentes sont à considérer pour mieux comprendre les défis à relever. Dans ce mémoire, nous nous intéresserons principalement à résoudre les problèmes liés aux deux évolutions suivantes.

Complexité croissante du logiciel. Jusqu'aux années 60, la distribution de logiciel était très limitée, et ceux-ci servaient essentiellement à effectuer des traitements par lots. Dans les années 70, sont apparus de nouveaux concepts tels que le multi-utilisateur, les interfaces graphiques, les bases de données relationnelles et le temps-réel. Les logiciels sont devenus beaucoup plus sophistiqués qu'auparavant. A partir des années 80, avec l'apparition des ordinateurs personnel, le logiciel devient un bien de grande distribution, nécessitant d'être maitrisé, ce qui donnera naissance au domaine du *génie logiciel*, aux ateliers de génie logiciel (AGL) permettant de produire des programmes de manière industrielle d'une certaine complexité.

Dès les années 70, les chercheurs se penchent sur ce problème de complexité et proposent le concept de programmation modulaire [Par72, DK75], puis d'architectures logicielles [GS94]. Le besoin de plus de granularité, de modularité (e.g., composants, architectures), de *frameworks* se fait ressentir pour construire des logiciels de grande taille.

Systèmes multi-échelle. Entre 1985 et le début des années 2000, avec l'avènement des systèmes distribués, des architectures client-serveur puis de l'Internet et du Web, le logiciel passe du statut de produit indépendant à celui d'élément d'un ensemble, dans lequel plusieurs ordinateurs et plusieurs logiciels travaillent en collectivité. On assiste à un changement de géométrie du logiciel, tendance encore renforcée récemment avec l'essor du *Cloud Computing* où le logiciel est hébergé par des ressources lointaines, élastiques et virtualisées.

Ce déploiement dans des systèmes multi-échelle entraine un degré de complexité qui est tel qu'il est souvent difficile pour un administrateur humain de configurer le logiciel, de résoudre les problèmes à la volée pouvant survenir pendant son exécution. De plus, pour anticiper des incidents potentiels (pannes, surcharges), les administrateurs ont souvent recours à la redondance,

à un surdimensionnement de l'infrastructure, engendrant un coût financier et énergétique non négligeable.

Pour résoudre ces problèmes (manque de réactivité, coûts induits), une voie possible est de promouvoir une auto-administration (e.g., *autonomic computing* [KC03]) du logiciel et des systèmes sous-jacents.

Profession de foi. La construction de logiciels modernes rend nécessaire la prise en compte de ce changement de géométrie du logiciel, demandant plus de modularité et d'auto-administration. Pour répondre à ces enjeux, nous proposons de voir le logiciel comme un assemblage de briques logicielles, capable de raisonner et d'agir sur lui-même pour se remodeler à l'exécution. En conclusion, un logiciel déployé dans ces systèmes multi-échelle doit être *vivant* et être capable de s'auto-administrer. Ainsi, l'adaptation du logiciel à l'exécution constitue une piste importante pour répondre aux défis du numérique.

2.2 Définitions et terminologie

> « *Self-adaptive software modifies its own behavior in response to changes in its operating environment [OGT+99].* » (Oreizy et al. - 1999)

Avant toute chose, il est nécessaire de poser un certain nombre de définitions pour comprendre le reste du manuscrit. Certaines définitions peuvent paraître triviales mais cette section a pour objectif de rassembler la terminologie du chapitre.

2.2.1 Du dictionnaire...

Pour commencer, voici quelques définitions du dictionnaire.
Adapter. (*v.*) Ajuster une chose à une autre [Lit].
Adaptable. (*adj.*) Qui peut être adapté [CNRa].
Adaptation. (*n.f.*) Action de s'adapter ou d'adapter ; résultat de cette action [CNRc].
Auto-Adaptation. (*n.f.*) Aptitude d'un système à modifier ses paramètres de structure de manière que son fonctionnement demeure satisfaisant en dépit des variations de son environnement [Lar].
Auto-adaptatif ou Adaptatif. (*adj.*) Qui est propre à s'adapter aux conditions extérieures [CNRb].
La littérature anglo-saxonne propose les termes suivants : "*self-adaptive* ou *adaptive*".

Discussion. Au regard de ces définitions, nous pouvons en déduire deux fondements de l'adaptation. En effet, ces définitions induisent « subrepticement » d'une part la notion d'une *fonction d'adéquation* [Dav05] qui est en charge d'évaluer le bon fonctionnement d'un système vis-à-vis de ses attentes pour un contexte donné, et, d'autre part, la notion d'une *stratégie d'adéquation* qui est en charge de guider, de vérifier l'ajustement d'une chose à une autre. Cette idée se retrouve surtout dans la définition de l'*auto-adaptation* avec le texte « son fonctionnement demeure satisfaisant ». C'est en effet la différence majeure qui sépare le terme *modification* du terme *adaptation* : **une adaptation est une modification évaluée par une *fonction d'adéquation* et dirigée par une *stratégie d'adéquation*.**

La définition de la *fonction d'adéquation* et/ou de la *stratégie d'adéquation* est soit de l'ordre de l'implicite car elle peut correspondre à un savoir-faire humain par exemple ; soit elle est de l'ordre de l'explicite dans le cas de l'auto-adaptation car le système doit embarquer en son sein cette connaissance pour s'ajuster.

2.2.2 ... aux types d'adaptation du logiciel

Pour aller plus loin, nous allons rapidement balayer les différents types d'adaptation liés au logiciel.

Adaptation statique *vs* dynamique.
Le moment de la réalisation de l'adaptation durant le cycle de vie d'un logiciel est un critère important [CMP06]. Une adaptation est qualifiée de *statique* lorsqu'elle intervient avant l'exécution du logiciel. Elle porte sur le système en cours de conception et/ou de développement. L'adaptation est dite *dynamique* si elle permet de modifier, totalement ou partiellement, la structure et/ou le comportement d'un logiciel (système d'exploitation, intergiciel, application, ...) pendant que celui-ci s'exécute [KM90]. Dans le cas d'une adaptation statique, le logiciel doit être arrêté, adapté puis redémarré. Son application entraine alors une rupture du service fourni. Ce cas de figure n'est pas toujours possible comme nous le verrons par la suite.

Adaptation au contexte.

> « *Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and application themselves [ADB+99].* » (Abowd et al. - 1999)

Jusqu'à très récemment, un logiciel était conçu, développé pour un environnement donné en supposant que le support d'exécution était connu à l'avance. Avec l'émergence de l'informatique ubiquitaire apparait le terme *context-awareness* pour désigner la capacité d'un logiciel à être conscient de son contexte (*context-aware*) pour s'y adapter en conséquence [ADB+99]. Ainsi, bien que le contexte soit par définition extérieur au logiciel, un système adaptatif va modifier son propre comportement en réponse aux changements survenus dans son contexte d'exécution.

Adaptation non anticipée.

> « *A system is open-adaptive if new application behaviors and adaptation plans can be introduced during runtime. A system is closed-adaptive if it is self-contained and not able to support the addition of new behaviors [OGT+99].* » (Oreizy et al. - 1999)

L'adaptation représente la capacité pour un logiciel de s'ajuster durant son cycle de vie mais cette adaptation – plus précisément le code, les artefacts qui la compose – peut-elle elle-même évoluer ? Y-a-t-il la possibilité d'intégrer dans le logiciel de nouveaux algorithmes, de nouveaux plans de reconfiguration qui n'avaient pas été prévus à la conception ? Est-ce que l'adaptation est adaptable ? Cette question fondamentale a donné lieu à des groupes de travail [KNMB02], et puise sa source dans de nombreux travaux allant des implémentations ouvertes ("open implementation" [KLL+97]) à la composition logicielle adaptative [MSKC04] en passant par la conception de langages réflexifs [RC02, RDT08] ou de méta-modèles [BBF09]. Toutes ces recherches ont étudié les propriétés du logiciel pour supporter, de manière dynamique, des « adaptations non anticipées » (cf. Section 3.1).

2.2.3 Premier bilan

> « *A wide range of autonomy might be needed, from fully automatic, self-contained adaptation to human-in-the-loop [OGT+99].* » (Oreizy et al. - 1999)

En synthétisant les différents éléments évoqués ci-dessus, un logiciel auto-adaptable possède la capacité d'évaluer son propre comportement pour un contexte donné (*fonction d'adéquation*) et de s'ajuster en suivant une *stratégie d'adéquation*. Cette dernière n'est pas forcément présente au sein du logiciel dans le cas encore où elle repose sur le savoir-faire implicite d'un humain (i.e., *"human-in-the-loop"*) ou elle est partiellement présente dans le cas des adaptations non anticipées.

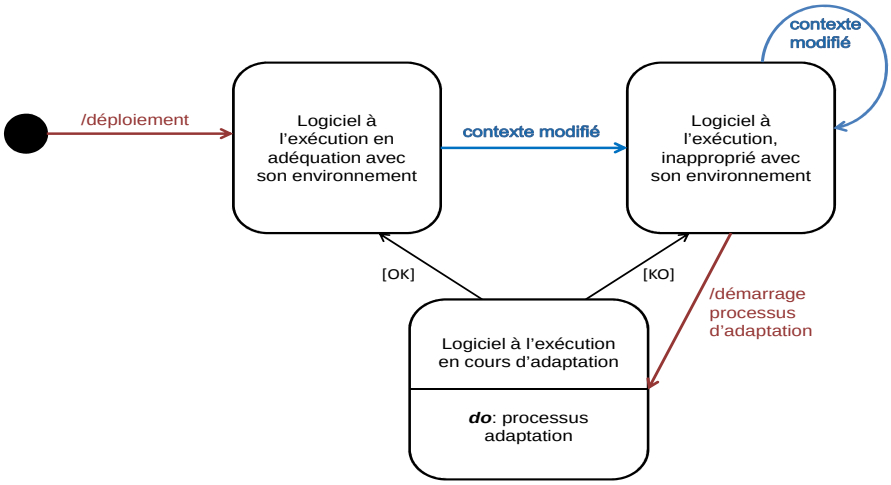


FIGURE 2.1 – Les trois états d'un logiciel (auto-)adaptable *context-aware*.

Nous pouvons schématiser l'adaptation logicielle *context-aware* par le diagramme d'états-transitions de la Figure 2.1. Une fois le logiciel déployé, le logiciel est dans un état adéquat avec son environnement d'exécution. Si l'événement "contexte modifié" se produit, le logiciel n'est peut-être plus en adéquation avec son environnement (i.e., utilisation de *fonction d'adéquation* pour le savoir). Un processus d'adaptation est alors nécessaire (*stratégie d'adéquation*) et conduit le logiciel à être dans un état instable de modification (i.e., en cours d'adaptation). Cette étape se termine soit par un échec et le logiciel est de nouveau dans un état inapproprié avec son environnement d'exécution ; soit par un succès et un retour avec une adéquation parfaite avec l'environnement d'exécution.

Si le logiciel réalise une auto-adaptation, la transition intitulée "démarrage processus d'adaptation" est réalisée automatiquement par le logiciel et non par une intervention humaine. Nous verrons par la suite (cf. Section 3.3) que ce modèle dynamique du système induit une problématique de fiabilité et de coût d'adaptation : l'état « Logiciel à l'exécution en cours d'adaptation » doit être au cœur de nos préoccupations.

2.3 Motivations de l'adaptation

Dans un premier temps, nous explicitons dans quel cadre est utilisé l'adaptation. Puis, nous revenons sur les motivations de l'adaptation dynamique.

2.3.1 Les raisons de l'adaptation

Une adaptation est nécessaire quand le fonctionnement du logiciel n'est plus satisfaisant (i.e., résultat de la *fonction d'adéquation*). Cette inadéquation avec l'objectif initial du logiciel peut être liée aux raisons suivantes.

Maintenance/évolution fonctionnelle. La *maintenance* s'intéresse essentiellement à la mise à jour du code source d'un logiciel alors que le terme *évolution* couvre non seulement l'évolution du code, mais aussi celle de l'architecture et plus généralement de tout artefact intervenant dans un système logiciel. Les raisons d'une maintenance/évolution peuvent être nombreuses (optimisation, correction, ...) mais elles sont souvent d'ordre fonctionnel (e.g., spécialisation d'un protocole, ajout d'un nouveau service, changement de signature d'une interface, ...). La *stratégie d'adéquation* est dirigée par les spécifications fonctionnelles du logiciel.

Optimisation de la qualité de service. Quand la qualité de service (QoS) du logiciel déployé est en deçà des attentes, il est nécessaire d'optimiser son fonctionnement. La qualité de service peut aussi bien concerner la performance d'un service que sa disponibilité ou encore dépendre d'un indicateur d'efficacité énergétique d'une infrastructure d'un centre de données. Ainsi, par exemple, si un composant logiciel est jugé inefficace vis-à-vis de ses performances, il peut être remplacé par un autre de manière à augmenter l'efficacité de l'application. La *stratégie d'adéquation* est dirigée par des critères d'optimisation comme la performance.

Correction/Réparation. La défaillance d'un système désigne son incapacité à rendre le service pour lequel il est conçu. Une défaillance est la conséquence d'une erreur causée par divers facteurs comme un bogue de programmation ou un problème matériel. Par exemple, le bogue de l'an 2000 lié à un problème de format de date ou encore la panne d'un serveur pour une architecture matérielle sous-dimensionnée. Selon la nature de la défaillance, le résultat de la *stratégie d'adéquation* peut prendre différentes formes : correction de programme, redémarrage de serveur, introduction de redondance, …

Protection. L'activité d'un programme malveillant ou d'un utilisateur non autorisé va provoquer un fonctionnement non désiré du logiciel. Il faut donc prévenir les intrusions en adaptant le logiciel. Selon la nature de l'intrusion, le résultat de la *stratégie d'adéquation* peut prendre différentes formes : détection d'attaques, élimination des virus, mise à jour, chiffrement, fermeture de canaux de communication, …

2.3.2 Atouts de l'adaptation dynamique

Il existe **deux spécificités majeures de l'adaptation dynamique** qui sont liés au cycle de vie du logiciel. La première est liée à l'idée que le logiciel est « vivant » et ne peut pas forcément s'arrêter. La seconde est liée à l'idée que le contexte d'exécution est « vivant » et qu'il est intéressant à écouter.

Continuité de service. La plupart des exemples d'adaptation évoqués ci-dessus pourraient être réalisés statiquement. Cependant une interruption à l'exécution n'est pas toujours possible (e.g., bogue de l'an 2000). L'intérêt principal de l'adaptation dynamique consiste à éliminer les temps d'indisponibilité [OMT08].

Des applications ayant un nombre considérable d'utilisateurs comme les serveurs mails ou les réseaux sociaux doivent fonctionner 24h/24 7j/7 et ne peuvent pas être arrêtés. Google ou Facebook introduisent de nouvelles fonctionnalités, fixent des bogues en garantissant la continuité de service. Les systèmes d'exploitation constitue une autre illustration : la plupart des mises à jour peuvent se faire de façon transparente en arrière-plan, sans redémarrer le système.

Cette continuité de service du système ne peut être possible que si le logiciel possède certaines propriétés d'adaptabilité comme une granularité assez fine permettant de bloquer le composant à adapter tout en laissant l'application s'exécuter (cf. Section 3.1).

Logiciels *context-aware*. Au début des années 2000, le contexte devient une notion centrale pour la plupart des travaux d'informatique ambiante. Lorsque l'on souhaite être sensible au contexte qui, par nature, est fortement variable, il n'est pas envisageable de stopper puis redéployer le logiciel à chaque adaptation. L'adaptation au contexte est donc par essence dynamique. Lorsque son contexte d'exécution évolue, le rôle d'une adaptation est alors non seulement de permettre au logiciel de continuer à fonctionner, mais aussi de tirer partie des nouvelles possibilités apparues dynamiquement.

Si l'on prend l'exemple du *smartphone*, celui-ci bascule automatiquement du réseau 3G au Wi-Fi quand il détecte la présence d'un réseau connu pour améliorer la qualité de la bande passante. Ou

encore, dans le cas d'un signal de batterie faible, on aimerait qu'il économise de l'énergie en arrêtant certains services facultatifs comme le GPS par exemple.

Cette adaptation au contexte n'est possible que si l'on modélise le contexte d'un système, sa variabilité et les interactions système-contexte [ADB+99].

2.3.3 Quelques illustrations d'adaptation dynamique

Nous donnons trois illustrations montrant la nécessité de l'adaptation dynamique dans le monde numérique d'aujourd'hui et nous discutons de chacun d'eux.

Mobilité et multimédia

L'exemple décrit ci-dessous est extrait du projet RNTL ARCAD [LBFB+01]. Il servait d'illustration pour motiver les travaux de recherche sur l'adaptabilité. Il n'a pratiquement pas été retouché (le mot smartphone *remplaçant le mot assistant personnel) et reste d'actualité.*

Pour motiver ces nouveaux besoins en adaptabilité, nous pouvons illustrer l'utilisation d'une application de flux vidéo sur un *smartphone* avec une connexion sans fil.

À la conception de cette application, le programmeur décide de prévoir et d'implanter deux types d'adaptation possibles dans le cas où le service ne peut être rendu de manière optimale en cas de problème de débit. La première consiste à diminuer le nombre d'images transmises par seconde, et la deuxième consiste à dégrader la qualité de l'arrière plan de la vidéo.

Une fois déployée, notre utilisateur exécute son application et plusieurs adaptations sont alors envisageables. Lors d'une diminution de la bande passante due aux interférences de la connexion sans fil, une possibilité d'adaptation est d'adopter le mode prévu de dégradation de l'arrière plan de la vidéo. Lors d'une congestion entre le serveur vidéo et le *smartphone*, une autre possibilité est de diminuer le nombre d'images transmises par seconde. Des adaptations non prévues peuvent également avoir un sens. La localisation de l'utilisateur peut être utilisée pour choisir le serveur vidéo le plus proche.

Enfin, l'application peut également évoluer en adaptant ses fonctionnalités ou en intégrant de nouvelles. Si le client utilise un décodeur MPEG-2, lors de la connexion à un serveur vidéo utilisant un encodage MPEG-4, celui-ci doit pouvoir charger et installer le décodeur correspondant. Cette intégration de nouvelles fonctionnalités peut se faire à l'initiative de l'utilisateur, par exemple si celui-ci veut intégrer un module effectuant des ralentis et des retours en arrière, ou à l'initiative d'un tiers, par exemple si le fournisseur de flux vidéo souhaite ajouter un module de télé-paiement.

Discussion. Ce scénario montre qu'il existe plusieurs types d'adaptation (*context-aware*, anticipée ou non anticipée), que celles-ci peuvent intervenir à différents moments du cycle de vie de l'application et à l'initiative de différents acteurs. Elle présuppose également de la capacité de l'application à pouvoir intégrer des adaptations non-anticipées à l'avance (cf. Section 3.1).

Le scénario de la Figure 2.2 montre un autre exemple de système adaptatif pour la mobilité issu du projet MADAM (*mobility and adaptation-enabling middleware*) [FHS+06].

Dimensionnement d'un centre de données

Les coûts de fonctionnement importants des centres de données sont liés à plusieurs facteurs comme le nombre de machines physiques démarrées pour héberger les services, le système de refroidissement, l'exploitation qui requiert la présence d'administrateurs qualifiés pour établir et maintenir l'état du système, ... Avant l'émergence des techniques de virtualisation et du Cloud computing, la solution généralement retenue pour limiter l'impact des incidents (panne, surcharge, etc.) consistait à dupliquer et à surdimensionner ces systèmes. Cela conduisait à un gaspillage des ressources, généralement sous-utilisées, et donc à une empreinte énergétique forte. Grâce au

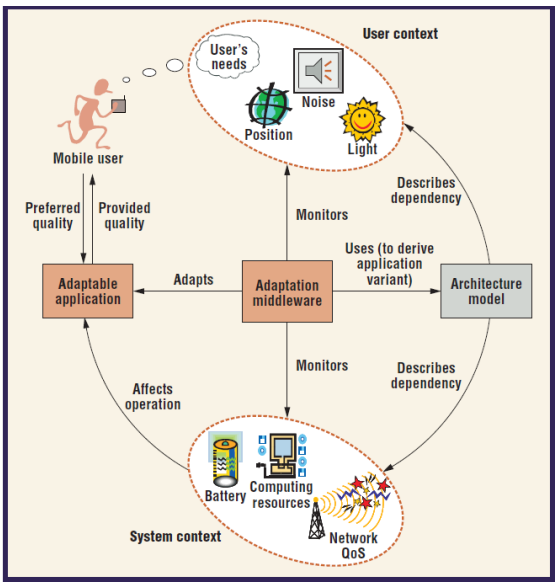


FIGURE 2.2 – Middleware adaptatif pour la mobilité [FHS+06]

concept d'élasticité du Cloud, les centres de données sont devenues moins *énergivores*. L'élasticité désigne la capacité d'un système à s'auto-redimensionner en fonction de la demande, c'est-à-dire à ajouter/retirer dynamiquement des ressources afin d'être toujours au plus proche de la demande et éviter selon les cas le sur-dimensionnement ou le sous-dimensionnement. Cette capacité d'adaptation à l'exécution associée à la consolidation dynamique de serveurs [HLM+09] permet de réduire le nombre machines physiques allumées et donc entraîne une économie d'énergie.

Discussion. Ce scénario montre l'intérêt de la reconfiguration dynamique pour un enjeu sociétal important : les transitions numérique et énergétique. Il est également l'illustration d'une auto-optimisation basée sur une administration autonomique [KC03] : le système est à l'écoute du contexte d'exécution (la demande cliente, les pannes) pour s'auto-ajuster. Cette problématique constitue une grande partie de mes travaux de recherche dans les années 2010 et sera amplement détaillée dans la Section 7.

Intelligence artificielle et prise de décision

S'illustrant début 2015 par un article dans le magazine Nature [MKS+15], le laboratoire Deep Mind de Google est un modèle du genre en intelligence artificielle. Des jeux d'arcade Atari (*breakout, space invaders*) au jeu de Go en passant par le jeu 2048, ses chercheurs ont réussi à faire en sorte qu'un ordinateur apprenne seul à jouer à des jeux. Des techniques d'intelligence artificielle comme l'apprentissage par renforcement [SB98] combinée à un réseau de neurones amélioré (nommé machine de Turing neurale) ont permis ces exploits. Concernant les jeux, le logiciel apprend en effectuant des actions et en observant les effets et conséquences. La machine de Turing neurale assimile comme un réseau neuronal classique en utilisant les entrées qu'il reçoit du monde extérieur mais il apprend aussi à stocker ces informations et à les récupérer.

Discussion. Ce scénario montre l'intérêt des adaptations dynamiques basées sur des méthodes d'apprentissage automatique permettant aux programmes d'apprendre. La prise de décision du système autonomique s'affine à l'exécution, avec le temps, à partir d'expériences acquises. Ces expériences et les informations associées peuvent être vues comme des politiques d'adaptation – non anticipées – se perfectionnant petit à petit.

2.4 Modélisation de l'adaptation

De nombreux travaux francophones [LBFB+01, Bas07, Bal10, Fer11] ou internationaux [BMZ+05, CdLG+09, ST09, KRV+15] ont abordé la question de la modélisation de l'adaptation logicielle. Il ne s'agit pas de refaire ici une n-ième modélisation mais de donner ma vision – élaborée à partir de ma propre expérience – pour ensuite revenir sur les notions développées dans cette section tout au long du mémoire.

A l'instar de [ST09], nous pensons qu'un moyen de capturer les exigences du logiciel auto-adaptatif est de s'inspirer du célèbre poème de Rudyard Kipling *"Six honest serving men"* (1902) et d'obtenir de l'aide auprès des six honnêtes serviteurs nommés *What*, *Where*, *Who*, *When*, *Why* et *How*. Ainsi, la modélisation de l'adaptation consiste à apporter des réponses aux questions suivantes : Où se situe le problème ? Que faut-il adapter ? Qui gère l'adaptation ? Quand faut-il adapter ? Comment adapter ?... Nous ne reviendrons pas sur la question « Pourquoi adapter ? » qui a été traitée précédemment (cf. Section 2.3.1).

2.4.1 Étapes du processus d'adaptation

Mais tout d'abord, quel est le patron (*template*) du processus d'adaptation ? Une fois de plus, de nombreux travaux traitant de l'adaptation dynamique évoquent ce processus. Par exemple, le cycle de vie des logiciels (auto-)adaptables est présenté sous l'appellation *"Adaptation management"* par [OGT+99], il possède 4 différentes étapes (*monitoring, detecting, deciding, acting*) dans [ST09] ou est mis en œuvre par IBM avec les boucles MAPE-K (Monitor, Analyse, Plan, Execute, Knowledge) [KC03, HM08].

Pour nous, une adaptation est une modification dirigée par une *stratégie d'adéquation* et il y a eu en amont le constat que le logiciel n'est plus en adéquation (*fonction d'adéquation*). Ainsi, ce processus est caractérisé par une *fonction d'observation* couplée à une *fonction de décision* qui raisonne sur les informations fournies par la fonction d'observation et analyse la situation afin de décider si une modification doit être réalisée ou non (fonction d'adéquation). En utilisant la stratégie d'adéquation, la fonction de décision indique à une *fonction de planification* les transformations qui doivent être apportées au logiciel. Une *fonction d'exécution* applique le plan d'actions précédemment établi pour mettre en œuvre l'adaptation [1].

2.4.2 Sujets de l'adaptation (*What*)

La première chose à définir est le sujet/la cible de l'adaptation. Pour pouvoir y répondre, nous allons tout d'abord supposer qu'un système logiciel est une architecture composée d'un ensemble de *briques* [2] reliées entre elles. Cette architecture peut être très monolithique : elle correspond alors à une seule brique ; ou très modulaire : dans ce cas, les briques sont reliées par des *liaisons*.

La cible de l'adaptation peut être soit une brique, soit une liaison (entre deux briques), soit l'architecture elle-même (i.e., la composition de *n* briques).

Pour tout sujet d'adaptation, nous allons considérer qu'il est composé de :

- un état ;
- une interface qui expose ses fonctionnalités ;
- un code qui implémente ses fonctionnalités.

1. Pour ne pas déstabiliser le lecteur, précisons que les concepts de *fonction d'adéquation* et *stratégie d'adéquation* sont d'un niveau d'abstraction supérieur aux fonctions d'observation, de décision, de planification, d'exécution qui restent des fonctions à implémenter d'un *template*.

2. Nous avons délibérément choisi ce terme – *neutre* – plutôt que module ou composant pour ne pas présupposer de sa représentation et de son implémentation.

Brique. Sa granularité peut être plus ou moins fine et sa durée de vie, dans le cycle de vie du système, plus ou moins longue. Une fonction C, une classe ou un objet Java, un composant EJB, un aspect AspectJ, un service Web sont des exemples de briques. L'adaptation de la brique peut concerner son interface, son comportement, sa sémantique, sa qualité de service, ... (cf. Section 2.3.1). L'adaptation d'une brique peut revêtir différentes formes : cela peut aller d'un simple paramétrage à une transformation complète.

Liaison. Une liaison est une relation entre deux briques. Un lien d'héritage, un bus logiciel sont des exemples de liaison de granularité différente. La modification, le changement de nature de la liaison entre deux briques sont d'autres cibles potentielles de l'adaptation. Par exemple, qui n'a jamais réalisé un travail de rétro-conception dans le cadre de la technologie objet pour utiliser un lien de délégation entre deux classes plutôt qu'un lien d'héritage ! Le changement de sémantique d'une liaison entre deux briques distantes est également un classique pour adapter les systèmes répartis [BCRP98, Led99].

Architecture. La cible de l'adaptation peut être l'architecture elle-même. Il s'agit alors de réaliser un nouvel assemblage des briques. Il peut être mis en œuvre de différentes manières : soit par retrait, remplacement, reconfiguration de certaines briques et/ou liaisons, soit par ajout de nouvelles. Par exemple, le schéma de conception Strategy [GHJV95] fournit un cadre privilégié pour ce type d'adaptation : il correspond à une reconfiguration de liaisons entre briques selon certaines conditions.

2.4.3 Mécanismes d'adaptation (*How*)

Pour aller plus loin, nous allons identifier les mécanismes pour réaliser l'adaptation d'un système logiciel (i.e., *fonction d'exécution*). Cherche-t-on à adapter la brique, la liaison entre deux briques ou l'architecture globale ? À chaque cible de l'adaptation est associée un mécanisme différent d'adaptation.

Mais avant toute chose, il faut aussi savoir si le code qui implémente les fonctionnalités du sujet est accessible ou non. Le sujet suit-il une philosophie de type *white box* (c.-à-d. il expose son implémentation) ou de type *black box* (c.-à-d. il expose son interface seulement) [Kic94] ?

Techniques pour adapter la brique. Si l'on s'intéresse à l'adaptation de la brique, il est possible de changer son état ou son code [3].

Paramétrisation. La paramétrisation est un cas notable de configuration où le système est construit avec un ensemble de paramètres prédéfinis dont les valeurs peuvent être spécifiées après construction du système. La paramétrisation de la brique est possible quand celle-ci est de type *black box*.

Transformation. Cette technique consiste à modifier directement le code de la brique. La plupart des travaux proposent une modification de la brique quand celle-ci est de type *white box*. La transformation peut être faite « à la main » (e.g., rétro-conception du code, ajout de pré/post conditions [Mey94]), mais peut aussi être réalisée par des technologies comme la spécialisation de programme [JGS93] ou le tissage d'aspects [KLM+97]. Cependant, certains travaux ont également proposé des techniques d'adaptation qui agit sur le code binaire des composants (type *black box*) [KH98].

Techniques pour adapter la liaison. L'adaptation de la liaison peut être réalisée par deux voies radicalement différentes. Le type de la liaison et celui des briques de la liaison constituent le critère de classification.

3. Le changement d'interface sans changement de code concerne les relations avec le monde extérieur et non pas l'intérieur de la brique. Elle sera traitée avec le sujet liaison.

Interposition. Cette technique est utilisée dans le cas où la liaison est de type *white box*. La communauté système propose depuis longtemps des techniques d'interposition comme mécanisme de liaison avec la volonté d'adapter la liaison elle-même [BS97]. Cette liaison peut prendre des formes très diverses comme par exemple l'association talon + squelette + protocoles de communication + réseau [OMG98]. Parallèlement, la communauté langage s'est intéressée aux liaisons de type *white box* pour contourner le problème d'adaptation des « briques noires ». En effet, dans ce cas, l'interposition peut jouer le rôle d'interception. L'idée est alors de placer une brique B′ dans la liaison devant la « brique noire » B à adapter pour réaliser une interception et utiliser cette indirection pour adapter le comportement de B. La communauté génie logiciel mentionne aussi les encapsulateurs (*Adapters ou Wrappers* [GHJV95]) qui jouent le rôle d'intercepteurs. La principale différence réside dans le fait que la brique B est incluse dans la brique B′ et donc considéré comme invisible du monde extérieur. On retrouve la même idée avec les conteneurs EJB. La modification d'interfaces des « briques noires » est également résolue par le patron de conception *Adapter* qui masque la signature des méthodes qui ne conviennent plus.

Délégation. Cette technique est utilisée quand la liaison est de type *black box* et les briques de la liaison sont de type *white box*. La délégation consiste à confier un travail d'une entité vers une autre entité. La mise en place d'une délégation dans la « brique blanche », génère une autre liaison, mais celle-ci peut être perçue comme une extension et donc une adaptation de la « liaison noire » initiale. La délégation peut être soit introduite explicitement par le concepteur, soit de façon transparente par des techniques de compilation ou de chargement à la volée.

Techniques pour adapter l'architecture. Si l'on s'intéresse à l'adaptation de l'architecture complète, il est possible d'engager une procédure de *reconfiguration* (i.e., une configuration issue d'une autre configuration). La reconfiguration de l'architecture consiste en une modification de sa topologie, c.-à-d en une recomposition des briques à l'intérieur de l'architecture. Cette reconfiguration peut donc être réalisée soit par retrait, remplacement, paramétrage de certaines briques et/ou liaisons, soit par ajout de nouvelles. L'ensemble des techniques de base présentées ci-dessus peut être utilisé pour réaliser la reconfiguration.

Discussion. Les différents mécanismes pour réaliser l'adaptation peuvent être classés en deux grandes catégories. D'une part, les ***techniques de mutation*** privilégiées par les approches *white box* (telles que la modification de code, la restructuration, ...) qui consiste à modifier le code d'une brique, d'une liaison, d'une architecture. D'autre part, les ***techniques d'habillage*** privilégiées par les approches *black box* (telles que le wrapping, la délégation, ...) qui consistent à intégrer à une liaison ou à une architecture de nouvelles fonctionnalités (par changement d'interfaces, de protocoles, ...). Signalons que la ***paramétrisation***, c.-à-d. la modification des valeurs des paramètres d'une brique n'appartient à aucune de ces catégories puisqu'elle agit sur l'état de la brique et non sur son code.

2.4.4 Moments d'adaptation (*When*)

> « *...adaptation can be described in terms of when it is designed and when it is applied [Kee04].* » (Keeney - 2004)

La notion de « moment d'adaptation » est ambiguë puisqu'il faut distinguer en réalité le moment où l'on prépare l'adaptation du moment où on l'exécute (*apply*) [Kee04]. Et il existe bien entendu une corrélation entre les deux sur l'échelle du temps comme nous le montre l'étude exhaustive de Philip K. McKinley et al. [MSKC04]. Les techniques de mutation, d'habillage ou de paramétrisation citées peuvent intervenir tout le long du cycle de vie du logiciel (i.e., développement, compilation,

déploiement, exécution). Par exemple, la conditionnelle `si-alors-sinon` est implémentée en phase de développement mais la réalisation de l'adaptation ne se fera qu'à l'exécution.

L'objectif de cette section est de faire le point sur la réalisation de l'adaptation (i.e., *when it is applied*). Il existe trois moments possibles pour la réalisation de l'adaptation : pendant le développement du logiciel, à son déploiement et pendant son exécution [BMZ+05].

Pendant le développement. Quand l'adaptation est réalisée pendant le développement, elle est dite *statique*. Dans ce cas, le concepteur adapte le logiciel selon l'environnement d'exécution visé et fixe les situations dans lesquelles il va opérer. Une adaptation statique peut être réalisée à la compilation, à l'édition des liens en fonction de la plate-forme cible. Si l'on se réfère au processus d'adaptation et ses différentes étapes (cf. Section 2.4.1), c'est comme si toutes les fonctions observation, décision, ... étaient appelées avant l'exécution.

Comme évoqué précédemment, pour préparer l'adaptation dynamique, il est nécessaire en phase de conception et développement de prévoir les possibilités d'adaptabilité du système et de faire des choix techniques en conséquence. Deux cas de figure peuvent se présenter :

1. soit la *stratégie d'adéquation* est connue à l'avance. Dans ce cas, les choix sur les sujets à adapter, les règles et les mécanismes d'adaptation peuvent être fixées à l'implémentation. Éventuellement, les règles d'adaptation peuvent être fusionnées avec le sujet de l'adaptation (e.g., une conditionnelle dans le code, une liaison tardive). L'adaptation à l'exécution est automatique, mais les fonctions du processus d'adaptation (e.g., décision) sont gravées dans le code et sont donc immuables. Les choix de l'adaptation ont été faits à la création du logiciel [CMP06] ;
2. soit la *stratégie d'adéquation* n'est pas connue à l'avance. Dans ce cas, il faut que le logiciel soit « préparé » en amont pour le support des adaptations non anticipées (cf. Section 3.1).

Au déploiement. Au moment du déploiement, l'environnement d'exécution et la topologie dans lequel va être exécuté le logiciel sont connus. Aussi, des choix pertinents d'adaptation peuvent être réalisés en conséquence (optimisation ou à l'inverse notification d'un problème).

Par exemple, comme nous le verrons par la suite (cf. Section 5.2), un script de reconfiguration qui peut être correct syntaxiquement et sémantiquement ne pourra pas être exécuté si l'architecture cible concernée par ce script est différente de ce qui était attendu.

Pendant l'exécution. L'adaptation est dite *dynamique* si elle permet de modifier la structure et/ou le comportement d'un logiciel pendant son exécution [KM90, OMT08]. Si l'on reprend l'exemple de mobilité et multimédia déroulé dans la Section 2.3.3, nous pouvons distinguer deux types d'adaptation dynamique selon le fait que la *stratégie d'adéquation* soit connue ou non à l'avance. Un problème de débit a pour conséquence une diminution du nombre d'images transmises ou une dégradation de la qualité de la vidéo. C'est un mécanisme d'adaptation qui est gravé dans le code. Par contre, l'installation du décodeur MPEG à la volée constitue un exemple d'adaptation dynamique de la liaison qui est non anticipée. Elle peut être effectuée par un tiers ou être réalisée automatiquement.

C. Krupitzer et al. [KRV+15] proposent de répondre plus précisément à la question *"When to adapt ?"* en raffinant le problème avec deux dimensions : une adaptation *réactive* et une adaptation *proactive*. La première est basée sur une fonction d'observation ou de décision relativement simple alors que la seconde adaptation va intégrer des modèles prédictifs avancés ou des modèles d'apprentissage dans les fonctions observation/décision.

2.4.5 Acteurs de l'adaptation (*Who*)

A la question « Qui gère l'adaptation ? », la réponse est multiple car un acteur différent peut intervenir à chaque étape du processus d'adaptation (cf. Section 2.4.1). Ainsi, l'acteur déclenchant

l'adaptation suite à un constat (*fonction d'observation*) peut être différent de celui qui va décider de la stratégie à suivre (*fonction de décision*). Nous distinguons deux types d'acteurs : l'acteur humain et le logiciel ; et la question principale est de savoir à quel point l'humain intervient dans le processus (*"human-in-the-loop"* [OGT+99]).

Humain. Une adaptation peut être initiée par un acteur humain qui peut être l'un des principaux acteurs du cycle de vie d'un logiciel (e.g., développeur, administrateur). Quand l'adaptation est statique, le développeur réalise une modification dirigée par la *stratégie d'adéquation* en suivant toutes les étapes du processus d'adaptation avant le redémarrage du système. Quand l'adaptation est dynamique, l'administrateur – alerté par des remontées de sondes par exemple – peut décider du déclenchement de l'adaptation, des sujets à adapter ou des mécanismes à utiliser. Dans ce cas, l'adaptation n'est pas automatique et sous le contrôle d'un acteur humain (i.e., la *stratégie d'adéquation* correspond à un savoir-faire implicite).

Logiciel. Si le concepteur du logiciel a anticipé l'adaptation, le logiciel peut initier sa propre adaptation automatiquement sans intervention humaine. Les étapes du processus d'adaptation peuvent être prises en charge automatiquement par différentes entités logicielles. Par exemple, le déclenchement de l'adaptation peut s'opérer grâce à des sondes pour l'observation ; la décision d'adaptation peut être assurée par un système expert ; la réalisation de l'adaptation peut être laissée à un intergiciel spécialisé. La boucle MAPE-K [KC03] constitue un patron pour implémenter ces systèmes auto-adaptables. Si le logiciel est à la fois le sujet et l'acteur de l'adaptation, le logiciel est un système adaptatif (cf. Section 2.2).

2.4.6 Quelques mots sur l'observation (*Where*)

Pour se rendre compte que le logiciel n'est plus en adéquation avec son environnement, il est nécessaire d'avoir une *fonction d'observation*. Cette fonction peut observer l'architecture logicielle elle-même et/ou son contexte d'exécution (i.e., adaptation *context-aware*). Soit cette fonction est implicite car assurée par l'humain (i.e., son expertise et son analyse de la situation suffit à déclencher le processus d'adaptation), soit cette fonction est explicite et correspond à un code logiciel. Si l'on s'intéresse au deuxième cas, la liaison entre la *fonction d'observation* et la *fonction de décision* (cette dernière étant soit un humain, soit un logiciel) suit un mode de communication de type *pull* (i.e., requêtes synchrones) ou de type *push* (i.e., notifications asynchrones).

Mode pull. La communication *pull* correspond à l'introspection de l'état d'une brique, d'une liaison, d'une architecture. Si l'on se trouve dans le cadre de logiciel *context-aware*, cette introspection est étendue au contexte extérieur du logiciel. Il s'agit par exemple de relevés de sondes comme un capteur de température dans une infrastructure domotique ou encore une position GPS pour un service de cartographie sur *smartphone*.

Mode push. La communication *push* est la plus utilisée dans les systèmes distribués devant manipuler un flot de données importants [CA08, HSB09, CM12] à l'instar de l'Internet des objets. Dans son implémentation basique, le mode *push* peut être mis en œuvre par le patron Observer [GHJV95] ; dans une implémentation plus sophistiquée, il peut être mis en œuvre par des systèmes réalisant du *Complex Event Processing* (CEP) [CA08] (cf. Figure 2.3).

Les domaines d'application du CEP sont divers et variés : monitoring d'infrastructures serveurs, traçabilité par le biais de puces RFID, optimisation de tournées de flottes de véhicule (via le GPS), Business Activity Monitoring (BAM), ... Schématiquement, le CEP consiste à corréler, hiérarchiser un ensemble d'événements (lien de causalité, chronologie, etc.) pour en déduire des macro-événements significatifs qu'il faut notifier à la *fonction de décision*.

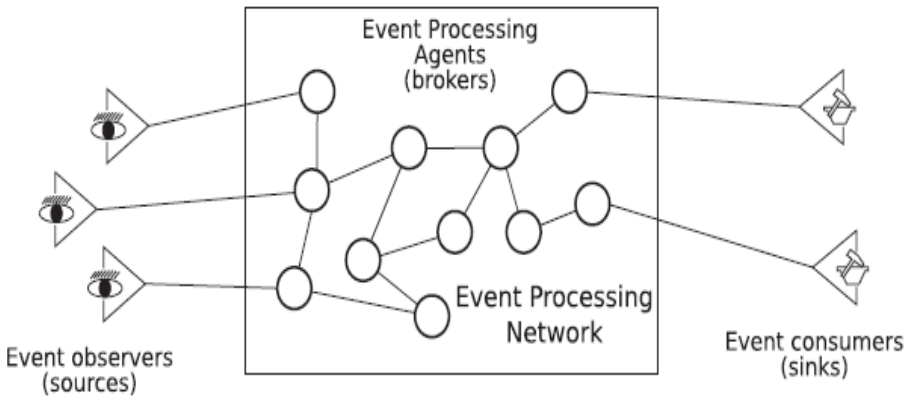


FIGURE 2.3 – Complex Event Processing [CM12]

Quel que soit le mode de communication (*pull* vs *push*), il est important de s'interroger sur la pertinence de l'information observée : est-elle significative ? est-ce un bruit, un accident ? Plusieurs propositions récentes dans le domaine du *Cloud computing* utilisent des techniques de traitement de signal (e.g., filtres passe-bas [CFF14], filtres de Kalman [BHS17]) pour réduire les irrégularités et singularités des données observées. L'objectif recherché est toujours le même : ne pas appeler la *fonction de décision* inutilement et donc éviter une reconfiguration incorrecte (e.g., redimensionnement coûteux de ressources pour le Cloud).

2.4.7 Et la prise de décision ?

Pour conclure cette section, revenons sur la *fonction de décision*. Son objectif est d'analyser la situation afin de décider si une modification doit être réalisée ou non en suivant la *stratégie d'adéquation*. Il n'existe pas de modèle de cette *fonction de décision* car elle dépend de la conception des autres fonctions connexes dans le processus d'adaptation (i.e., observation, planification). Par exemple, la règle ECA (Event-Condition-Action) – concept provenant du monde des bases de données [ANC96] – est régulièrement utilisée dans les boucles MAPE-K comme « cerveau » de l'adaptation [HM08] et adresse à la fois les fonctions d'observation, de décision et d'exécution (la planification est absente).

La prise de décision peut donc être modélisée de façon très différente en se basant sur des paradigmes variés : politiques d'adaptation [Slo94], théorie de la commande optimale [MBM11], probabilités et théorie des files d'attente [AB10], statistique et approche bayésienne [Guo03], théorie des jeux [WVZX10], apprentissage par renforcement [LRFH04, DCCC06, RBX+09], théorie du contrôle et rétroaction (*feedback*) [Roy07, BMSG+09, LBC10, BGLQ16], programmation linéaire [SH10, ZPL+12], automates et machine à états abstraits [ARS15], ... (cf. [KRV+15] pour une liste exhaustive de travaux).

2.5 Retour d'expériences : OpenCorba

OpenCorba [Led99] est un bus logiciel réflexif [Smi84] que j'ai développé pendant ma thèse. Ce travail – qualifié de précurseur [TGCB08] – a inspiré mes activités de recherche pendant de nombreuses années, notamment sur le sujet de la reconfiguration dynamique. C'est pourquoi, il me semblait important de faire un court retour d'expériences sur cette modélisation.

2.5.1 Problématique et motivations

Dans les années 90, l'architecture CORBA (*Common Object Request Broker Architecture*) [OMG98] constitue la solution industrielle la plus prometteuse pour réaliser l'interopérabilité entre des composants logiciels répartis hétérogènes. Paradoxalement, alors que le bus logiciel CORBA cherche à fédérer différents mécanismes de distribution au sein d'une même architecture, son modèle est peu flexible et inadapté aux futures évolutions.

Avec OpenCorba [4] [Led98], nous proposons la réalisation d'un bus CORBA basée sur une approche réflexive [Mae87]. Son architecture permet l'introspection et la modification des mécanismes de représentation et d'exécution du bus logiciel CORBA. Ainsi, il devient possible d'étendre aisément la sémantique de base du bus pour y intégrer – à la volée – d'autres mécanismes. Par exemple, une invocation à distance classique peut devenir un mécanisme de réplication en transformant notre système pour y intégrer la tolérance aux fautes [FP98].

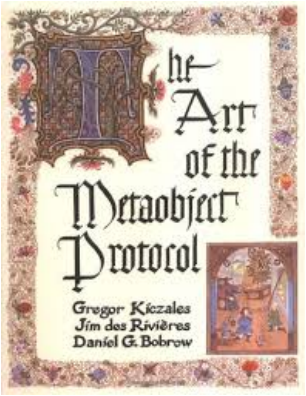


FIGURE 2.4 – Un titre intrigant, une couverture pleine de mystère qui font rêver ! [KR91]

2.5.2 Description

OpenCorba est basé sur le langage réflexif NeoClasstalk [Riv97]. Inspiré de Classtalk [BC89], ce dernier est le résultat de l'implémentation d'un MOP (*Meta Object Protocol*) [KR91] (cf. Figure 2.4) pour le langage Smalltalk [GR83]. Sa principale contribution est d'étendre les aspects dynamiques de Smalltalk en proposant d'une part, une solution efficace pour contrôler l'envoi de messages et, d'autre part, un protocole de changement dynamique de classe.

OpenCorba réifie différentes propriétés du bus CORBA par le biais des *métaclasses* (c.-à-d. une classe dont les instances sont des classes) afin de séparer les caractéristiques internes du bus du code métier de l'application (*separation of concerns* [HL95]). L'utilisation du protocole de changement dynamique de (méta)classe permet alors d'adapter le comportement du bus à l'exécution en modifiant les mécanismes de CORBA implémentés par des métaclasses.

En considérant les classes comme objet de plein droit, nous avons pu éviter l'entrelacement entre le code métier et le code décrivant une propriété/un mécanisme spécifique « *sur* » la classe métier, implémenté dans les métaclasses. Cette conception réflexive encourage la lisibilité et la réutilisabilité du code (à la fois le code métier et le code méta puisqu'il existe un découplage spatial) [FD99].

Le protocole de changement dynamique de métaclasse permet à OpenCorba de remplacer le mécanisme « *sur* » une classe par une autre, pendant son exécution, sans affecter le reste du système (découplage temporel). Pour nos travaux sur les bus logiciels adaptables, ce protocole est extrêmement important car il autorise la modification des mécanismes des architectures réparties à l'exécution et donc une adaptation dynamique de ceux-ci. Ainsi, une invocation synchrone peut devenir asynchrone,

4. Le terme "open" faisant référence à *open implementation* [KLL+97].

	Modification	Adaptation statique	Adaptation dynamique [5]	Auto-adaptation	Adaptation non anticipée
Continuité de service	N/A	non	oui	oui	oui
Context-aware	N/A	non	non	oui	oui
Open vs closed adaptive	N/A	*closed*	*closed*	*closed*	*open*
Stratégie adéquation	absente	présente	externe (H) ou intégré (L)	intégré	partiellement intégré
Fct observation	H ou L	H ou L	H ou L	L	L
Fct décision	H ou L	H ou L	H ou L	L	L
Fct planification	H ou L	H ou L	H ou L	L	L
Fct exécution	L	L		L	L

Table 2.1 – Synthèse terminologie d'adaptation

un objet volatil peut devenir persistant, un objet proxy peut décider de gérer un cache, ...(cf. [Led98] pour plus de détails).

2.5.3 Relecture de mes travaux

Il m'a semblé intéressant de faire une relecture de la conception de OpenCorba dans le prolongement de la Section 2.4. Concernant le principe d'adaptation, les métaclasses, associées au changement dynamique de métaclasse, constituent un cadre réflexif privilégié pour la construction d'architectures « ouvertes » (*open implementation* [KLL+97]). En effet, les métaclasses jouent le rôle d'interpréteur puisqu'elles définissent le comportement et la structure de l'état de ses instances (i.e., les classes). Changer de métaclasse, c'est donc changer d'interpréteur et *in fine* « adapter par interprétation » [ML01] les instances de ces métaclasses.

Si l'on s'intéresse aux sujets d'adaptation de OpenCorba et aux mécanismes d'adaptation, tout va se jouer sur les briques de type classe et métaclasse et la liaison entre elles. Le protocole de changement dynamique de métaclasse va permettre d'adapter la liaison entre une classe et sa métaclasse (i.e., technique d'habillage par délégation) et changer ainsi son interprétation.

OpenCorba propose un support pour l'adaptation dynamique et peut-être considéré comme un intergiciel adaptable. Il n'est pas auto-adaptable car OpenCorba ne définit pas la notion de contexte, n'embarque pas en son sein une *fonction d'observation* et l'acteur de l'adaptation reste un humain. Par contre, la *stratégie d'adéquation* ne peut être que partiellement embarquée au démarrage du système car de nouvelles métaclasses peuvent être chargées dynamiquement : OpenCorba supporte les adaptations non anticipées.

2.6 Synthèse

Pour conclure ce chapitre, il nous a semblé important de faire une synthèse des concepts vus dans les différentes sections. Cette synthèse consiste en à un tableau comparatif et en une analogie.

Le Tableau 2.1 présente en colonne les différentes « adaptations » possibles. Les lignes résument les notions et les fonctions liées aux différents types d'adaptation. L'intersection des lignes et des colonnes donne la caractéristique essentielle à retenir pour le type d'adaptation. Dans le tableau, la lettre 'H' désigne l'humain alors que la lettre 'L' désigne le logiciel.

Ce tableau ne prétend pas être exhaustif et montre principalement les grandes catégories d'adaptation. Par exemple, OpenCorba réalise des adaptations non anticipées alors qu'il n'intègre pas de boucle autonomique pour décrire un système auto-adaptable.

5. Il faut au moins un 'H' dans la colonne sinon on retombe sur le cas de l'auto-adaptation

Une analogie pour mieux comprendre le tableau peut être faite avec le monde de l'automobile et plus précisément avec l'émergence de la voiture autonome (*"autonomous car"*) qui n'est autre qu'un véhicule *context-aware*.

Si l'on considère la stratégie d'adéquation suivante : « adapter la vitesse et les caractéristiques du moteur aux conditions de circulation », les deux premières colonnes de notre tableau vont être hors sujet puisque d'une part pour la *modification*, il existe une stratégie d'adéquation, et d'autre part, pour l'*adaptation statique*, la voiture doit continuer à rouler tout en changeant sa vitesse (continuité de service obligatoire). L'*adaptation dynamique* peut correspondre à une voiture adaptant sa vitesse grâce à un conducteur actionnant une boite de vitesse manuelle (i.e., fonction d'observation externe) alors que l'*auto-adaptation* peut correspondre à une voiture munie d'une une boite de vitesse automatique (i.e,. prise de conscience du contexte, passages de vitesse déclenchés automatiquement en fonction de la vitesse du véhicule).

Avec l'arrivée de la voiture autonome, la modélisation du contexte s'est élargie : il ne s'agit plus seulement d'écouter le moteur mais également d'intégrer l'environnement autour de la voiture. Ainsi, la stratégie d'adéquation a évolué : « rouler sans conducteur en s'adaptant aux conditions de circulation » ! La voiture autonome est donc un exemple d'auto-adaptation avec une modélisation du contexte plus large et plus aboutie, une fonction d'observation basée sur de nombreux capteurs. Maintenant, on peut se poser la question de savoir si la *Google car*, par exemple, est capable de s'adapter à la signalisation routière hors des Etats-Unis. Quid des feux tricolore par exemple qui sont de l'autre côté de la route là-bas ? Le module de contrôle de cette voiture autonome est-il capable d'*adaptation non anticipée* pour rouler en Europe ?

Chapitre 3

Adaptation dynamique : pistes de travail

Dans le chapitre précédent, nous avons présenté *in extenso* le concept d'adaptation du logiciel à l'exécution et sa modélisation. Dans ce court chapitre, notre objectif est réaliser un bilan du travail qu'il nous semble important à mener pour intégrer durablement les principes de l'adaptation dynamique au logiciel. Car l'adaptation à l'exécution vient avec ses exigences, ses contraintes et finalement peut être évaluée qualitativement.

3.1 Propriétés d'adaptabilité du logiciel

Un logiciel dynamiquement adaptable doit posséder un certain nombre de propriétés pour être adapté.

3.1.1 Réification

> « *By means of reification, something that was previously implicit, unexpressed, and possibly inexpressible is explicitly formulated and made available to conceptual (logical or computational) manipulation. Informally, reification is often referred to as "making something a first-class citizen" within the scope of a particular system [wik].* » (Wikipedia - 2016)

La propriété fondamentale est la capacité du logiciel à être accessible à l'exécution. En effet, pour permettre l'adaptabilité à l'exécution, il faut choisir une approche permettant au logiciel d'exister explicitement à l'exécution, et pas seulement au moment de son développement. Si on veut l'observer, agir dessus, il doit être représenté à l'exécution. Qu'il s'agisse d'un bus logiciel, d'une architecture à base de composants, d'un langage de programmation, chaque concept implicite peut être explicité pour devenir citoyen de première classe (*"first-class citizen"*).

De plus, comme évoqué précédemment, la réification du contexte va être primordiale pour permettre le développement de logiciels *context-aware*. L'adaptation au contexte suppose la nécessité de modéliser la variabilité du contexte du système et les interactions système-contexte.

Réflexion, introspection, intercession

> « *Reflection : an entity's integral ability to represent, operate on, and otherwise deal with itself in the same way that it represents, operates on, and deals with its primary subject matter [Smi90].* » (Smith - 1990)

Le concept de réflexion peut être rencontré dans des disciplines aussi diverses que la philosophie, la linguistique, la logique ou l'informatique. Si le concept possède différentes connotations selon les

spécialités, il exprime, d'une manière générale, la capacité d'un système à raisonner et à agir sur lui-même. Deux aspects composent une telle manipulation [BGW93] (cf. Figure 3.1) :

— l'*introspection*, qui est la capacité d'un système à examiner son propre état ;
— l'*intercession*, qui est la capacité d'un système à modifier son propre état d'exécution ou d'altérer sa propre interprétation ou signification.

Par exemple, dans un langage à classe réflexif, des éléments du langage comme les classes, les variables d'instance ou encore les méthodes sont réifiés (*réflexion structurelle*) ainsi qu'une partie de sa propre sémantique comme l'envoi de messages, la pile d'exécution ou l'accès aux variables (*réflexion comportementale*). Par intercession, il est alors possible, par exemple, de contrôler l'envoi de messages [FJ89], d'intégrer le concept d'objets concurrents [Bri89, McA95] ou de modifier la gestion des exceptions [Don90].

Ainsi, comme en témoigne de nombreux travaux [FP98, BCRP98, Led98, TGCB08, KRV+15], la réflexion à l'exécution va être un vecteur intéressant pour mettre en œuvre l'adaptation dynamique.

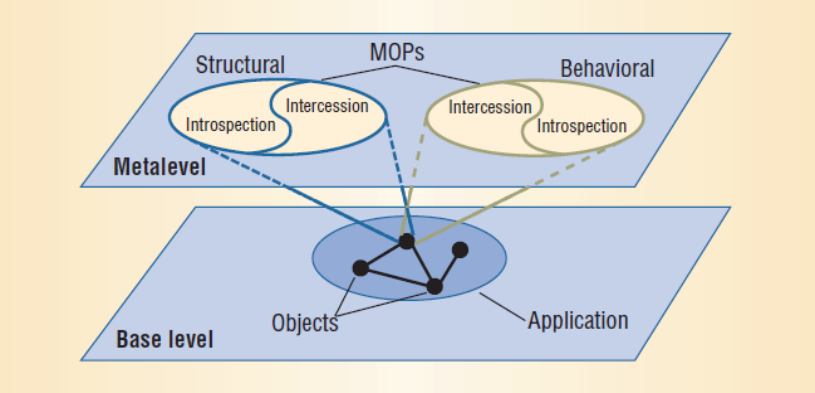


FIGURE 3.1 – MOP, introspection, intercession [MSKC04]

Models@run.time

> « *A model@run.time is a causally connected self-representation of the associated system that emphasizes the structure, behavior, or goals of the system from a problem space perspective [BBF09].* » (Blair et al. - 2009)

S'inspirant de la communauté réflexive, la communauté Models@run.time voit le jour au milieu des années 2000 [BBF07]. L'idée est d'embarquer une couche de modélisation à l'exécution, représentant le système réel et causalement connecté avec lui. Contrairement aux systèmes réflexifs où le lien causal [Mae87] induit un couplage fort entre réification et réflexion, un model@run.time présente un plus haut niveau d'abstraction, proche de l'utilisateur et permet des étapes de modification avant la synchronisation avec le système réel évitant alors le côté irréversible de la modification.

A la même époque, dans le domaine des systèmes distribués, le *framework* Jade [BDPG+09] propose un système de représentation d'une architecture n-tiers basée sur des composants Fractal [BCL+06]. Il est motivé comme une sauvegarde de l'architecture du système administré pour maintenir la cohérence globale dans le cadre de reconfiguration : toute modification passe par la représentation système qui répercute les modifications sur l'architecture du système administré. Une similitude avec les Models@run.time est plus qu'évidente.

En conclusion, les Models@run.time constituent un support séduisant pour l'adaptation logicielle à l'exécution de systèmes complexes.

3.1.2 Modularité et Composabilité

Concevoir un logiciel adaptable nécessite d'organiser sa structure afin de donner la possibilité de l'ajuster *a posteriori*.

La modularité [Par72] est une approche structurante qui découpe un logiciel en petites unités qui – assemblées – composeront l'ensemble du logiciel. Cette propriété permet de modifier ou de remplacer certaines parties (ou modules) avec un minimum d'interférences – si le couplage est lâche (*loose-coupling*) – sur les autres parties qui le composent, favorisant ainsi la continuité de service [KM90]. Le choix de la granularité du module est extrêmement impactant sur les possibilités d'adaptation : plus la granularité est fine (*fine-grained*) plus les opportunités d'adaptation sont riches mais également complexes à maitriser contrairement à une approche à gros grains (*coarse-grained*) moins flexible. Les modules sont plus ou moins réifiés selon le cycle de vie du logiciel : certains sont présents à l'exécution (classe Smalltalk) alors que d'autres disparaissent à la compilation (aspect AspectJ) autorisant ainsi ou non l'adaptabilité à l'exécution.

La composabilité est la capacité à pouvoir être composé. Une réification de cette capacité va permettre une plus grande richesse des possibilités d'adaptation. Les approches à base de composants ou d'aspects ont mis l'accent sur cette problématique, en réifiant par exemple les connections entre modules (composants, connecteurs et *Architecture Description Languages* [MT00]) ou en définissant des langages de tissage de modules (aspects [KLM+97]).

En reprenant notre hypothèse qu'un système logiciel est une architecture composée d'un ensemble de *briques* (cf. Section 2.4), nous avons vu que les mécanismes d'adaptation peuvent être classifiés en trois grandes catégories : les techniques de mutation, les techniques d'habillage et la paramétrisation (cf. Section 2.4.3). Ainsi, un système monolithique (modularité de type *coarse-grained*) ne peut être adapté qu'en le remplaçant intégralement soit par une réécriture (i.e., technique de mutation), soit par un masquage (i.e., technique d'habillage). Il peut être aussi modifié dans son état via la paramétrisation. Au contraire, OpenCorba [Led99] est l'exemple d'une modularité de type *fine-grained* où le module est la (méta-)classe et où la composabilité est permise à l'exécution par le changement dynamique de métaclasse (technique d'habillage par délégation).

3.1.3 Support des adaptations non anticipées

Un système est dit fermé à l'adaptation (*closed-adaptive*) lorsque toutes les adaptations sont définies lors de la phase de conception [OGT+99]. Or, parmi les raisons pour lesquelles l'adaptation est requise, toutes ne peuvent être anticipées dès cette phase [EVHB05]. Afin de pouvoir contourner ultérieurement cette limite, le système doit supporter, de manière dynamique, des « adaptations non anticipées » [RC02]. Comme il a été dit précédemment (cf. Section 2.2), seuls les systèmes ouverts [KLL+97] peuvent intégrer dans le logiciel de nouveaux algorithmes, de nouveaux plans de reconfiguration qui n'avaient pas été prévus à la conception.

La communauté réflexive a longtemps montré la voie pour contruire de tels systèmes comme le démontre les travaux de [RDT08] qui joue sur le « degré d'ouverture » d'un système partiellement réflexif (*"partial behavioral reflection"*). Il s'agit d'un vrai travail d'orfèvre pour définir dans l'espace et dans le temps les possibilités d'adaptation du système.

3.2 Adaptation comme préoccupation

> « *...While internal adaptation can certainly be made to work, it has a number of serious problems. First, when adaptation is intertwined with application code it is difficult and costly to make changes in adaptation policy and mechanism. Second, for similar reasons, it is hard to reuse adaptation mechanisms from one application to another. Third, it is difficult to reason about the correctness of a given adaptation mechanism, because one must also consider all of the application-specific functionality at the same time.*

An alternative approach is to develop externalized adaptation mechanisms [GSC01]. »
(Garlan et al. - 2001)

3.2.1 Externalisation

L'adaptation est une préoccupation (*concern*) transverse qui – dans les approches de programmation classiques qu'elles soient à objets, composants ou encore services – n'est pas encapsulée dans une entité réutilisable. La conditionnelle `si-alors-sinon` dans le code métier en est le parfait exemple.

Dans un logiciel adaptatif, la logique d'adaptation doit être séparée de la logique métier pour améliorer la compréhension, la maintenance, la testabilité et la modularité du logiciel. En considérant l'adaptation comme objet de première classe (*"first-class citizen"*), il est possible de raisonner sur l'adaptation, ses propriétés (fiabilité, réactivité, coût, etc.) voire de reconfigurer dynamiquement le « *concern* adaptation » pour permettre son évolution (i.e., adaptation non anticipée). Il est alors possible de définir de nombreuses stratégies d'adaptation qui s'appliquent selon le contexte. Cette séparation logique métier *vs* logique d'adaptation a été promue par de nombreux travaux [Slo94, VK03, GCH+04, FHS+06, OMT08, PBCD11].

3.2.2 Expression

L'externalisation de la logique d'adaptation constitue un pré-requis important pour développer des logiciels adaptatifs. Maintenant, comment exprimer cette préoccupation d'adaptation ? Il n'existe pas de réponse immédiate à cette question puisqu'elle va dépendre entre autres de la modélisation de la *fonction de décision* (cf. Section 2.4.7).

On va retrouver cependant deux grandes approches pour exprimer la logique d'adaptation : l'approche déclarative et l'approche impérative. Avec l'approche déclarative, on va décrire *le quoi*, c'est-à-dire le problème ; alors qu'avec l'approche impérative, on va décrire *le comment*, c'est-à-dire la structure de contrôle correspondant à la solution.

Approche déclarative. Il existe plusieurs formes d'approche déclarative comme la programmation logique ou la programmation par contraintes (PPC). Le design d'une boucle MAPE-K basée sur la programmation linéaire [BGLQ16] ou sur des *utility functions* [Koe14] constitue deux exemples récents d'auto-adaptation pour le Cloud basée sur une approche déclarative.

Si l'on prend l'exemple de la PPC [RBW06], la logique d'adaptation est modélisée à l'aide de variables de décision et de contraintes, où une contrainte est une relation entre une ou plusieurs variables qui limite les valeurs que peuvent prendre simultanément chacune des variables liées par la contrainte. Prenons l'exemple d'un cluster de machines physiques (PM) modélisé par le vecteur $P = (pm_1, pm_2, \ldots, pm_p)$. pm_i^{cpu} définit la capacité CPU de la machine $pm_i \in P$. Soit $V = (vm_1, vm_2, \ldots, vm_r)$ l'ensemble des machines virtuelles (VM) s'exécutant dans le cluster. Pour chaque $pm_i \in P$, il existe un vecteur $H_i = (h_{i1}, h_{i2}, \ldots, h_{ir})$, où $h_{ij} = 1 \iff vm_j$ est hébergée par pm_i, $h_{ij} = 0$ sinon ($1 \leq i \leq p, 1 \leq j \leq r$). Si la variable vm_j^{cpu} exprime la capacité CPU d'une machine virtuelle vm_j, la contrainte (3.1) signifie que les demandes des VM hébergées dans une PM ne doit pas dépasser sa capacité CPU.

$$pm_i^{cpu} \geq \sum_{j=1}^{r} h_{ij} * vm_j^{cpu} \tag{3.1}$$

Un solveur de contraintes est alors appelé par la *fonction de décision* pour déterminer les valeurs possibles des variables. Associée à la fonction d'objectif (3.2), il permettra de chercher la solution

optimale par rapport au critère « minimisation du nombre de PM ».

$$O = min(\sum_{i=1}^{p}(u_i)), \text{ where } u_i = \begin{cases} 1, \exists j \in [1,r] \mid h_{ij} = 1 \\ 0, \text{otherwise} \end{cases} \tag{3.2}$$

Approche impérative. L'approche impérative est la plus naturelle pour le développeur informatique car il va décrire une séquence d'instructions à exécuter pour adapter le logiciel. Garlan et al. [GCH+04] introduisent dans l'architecture Rainbow des stratégies d'adaptation basées sur la programmation impérative.

A l'instar des règles ECA (*Event-Condition-Action*) [ANC96], les règles d'*auto-scaling* pour le Cloud sont des règles à base de seuil qui suivent une approche impérative. Par exemple, la règle suivante signifie que si une métrique m dépasse un seuil *UPPER* pendant une durée $time_{upper}$ alors l'infrastructure cible verra sa capacité augmentée de k_{add} et cette action sera suivie d'une période de calme (i.e., aucun événement ne sera traité pour éviter les oscillations du système) de durée $calm_{add}$.

If m>UPPER for $time_{upper}$ *then capacity=capacity+*k_{add} *Do nothing for* $calm_{add}$

Pour permettre de relâcher l'exigence parfois trop forte sur la valeur des seuils, de récents travaux dans le Cloud [JAP14, FLK+15] ont introduit la logique floue tout en préservant une approche impérative [1].

3.3 Adaptation dynamique, garanties du processus et qualité logicielle

> « *The main concerns of interest in formalization of self-adaptation are efficiency/-performance and reliability [WIdlIA12].* » (Weyns et al. - 2012)

Contrairement à l'adaptation statique où le processus d'adaptation est – par définition – totalement maitrisé, l'adaptation du logiciel à l'exécution va introduire un certain nombre d'interrogations. Comment s'assurer que l'adaptation dynamique va garantir l'intégrité du logiciel ? Comment être sûr que le processus d'adaptation ne va pas perturber le fonctionnement du logiciel et va garantir la continuité de service ? Nous abordons ces deux questions fondamentales dans la suite de la section.

3.3.1 Fiabilité

Problématiques

> « *Change management is a critical aspect of runtime-system evolution that identifies what must be changed; provides the context for reasoning about, specifying, and implementing change; and controls change to preserve system integrity [OGT+ 99].* » (Oreizy et al. - 1999)

L'adaptation dynamique est une opération risquée qui peut porter atteinte à l'intégrité du système. En partant de l'hypothèse qu'un système logiciel est une architecture composée d'un ensemble de *briques* (cf. Section 2.4.2), le processus d'adaptation ne doit jamais faire migrer l'architecture vers une configuration qui n'est pas *valide*, i.e., cette nouvelle configuration doit être conforme à la spécification d'un modèle d'architecture.

Plusieurs problématiques ont été soulevées à ce sujet [Ore98, OGT+99] et nous proposons de prendre en compte les considérations suivantes [Lég09] :

1. [LBMAL14] propose un état de l'art exhaustif sur techniques d'*auto-scaling* pour le Cloud adressant à la fois les approches déclarative et impérative.

- Le modèle d'architecture d'un logiciel doit rester cohérent avec son implémentation au cours de l'exécution et des reconfigurations. Les systèmes réflexifs ou Models@run.time apportent une solution à ce problème en maintenant une connexion causale entre la représentation architecturale et le système à l'exécution ;
- L'adaptation dynamique doit préserver la cohérence (*consistency*) de la structure du logiciel et la validité de son comportement, de même que certaines propriétés non-fonctionnelles (e.g., sécurité, performance, ...), et ce en dépit de possibles défaillances. La vérification de contraintes (e.g., cardinalité de liaisons) ou de styles architecturaux permet par exemple de restreindre les transformations possibles des architectures ;
- La cohérence des modifications doit être gérée à tous les niveaux de granularité du logiciel (niveau brique *vs* niveau architecture) [Gou99] ;
- L'exécution des reconfigurations dynamiques doit être synchronisée avec l'exécution métier du logiciel. Par exemple, il ne doit pas être possible pour une brique A de continuer à invoquer des méthodes de l'interface d'une brique B alors que cette dernière est en train d'être déconnectée ;
- Un problème complémentaire à la synchronisation est la gestion de l'état des briques reconfigurées et plus précisément le transfert d'état entre briques. Il peut ainsi être nécessaire de transférer l'état d'une ancienne brique vers le nouvelle brique qui la remplace dans l'architecture ;
- L'occurrence possible de plusieurs reconfigurations simultanées dans un système à l'exécution requiert une synchronisation pour éviter les conflits entre reconfigurations sous peine de mettre potentiellement le système dans un état incohérent.

Langages de description d'architecture et autres formalisations

> « *An ADL for software applications focuses on the high-level structure of the overall application rather than the implementation details of any specific source module [MT00].* »
> (Medvidovic et al. - 2000)

```
Style Client-Server
  Component Client
    Port p = request → reply → p ⊓ §
    Computation = internalCompute → p.request → p.reply → Computation ⊓ §

  Component Server
    Port p = request → reply → p [] §
    Computation = p.request → internalCompute → p.reply → Computation [] §

  Connector Link
    Role c = request → reply → c ⊓ §
    Role s = request → reply → s [] §
    Glue =  c.request → s.request → Glue
         []  s.reply → c.reply → Glue
         []  §

  Constraints
    ∃! s ∈ Component, ∀ c ∈ Component : TypeServer(s) ∧ TypeClient(c) ⇒ connected(c,s)
EndStyle
```

FIGURE 3.2 – Exemple de configuration avec l'ADL Wright

La prise en compte des concepts architecturaux au niveau des langages de programmation permet une traçabilité tout au long de la phase de développement et du déploiement des architectures. Les langages de description d'architecture (*Architecture Description Languages* ou ADLs) [MT00] sont des langages généralement déclaratifs dans lesquels le couplage entre spécification architecturale et implémentation est faible. La majorité des ADL réalisent des analyses statiques des configurations permises par la spécification du modèle (cf. Figure 3.2). Certains ADL comme Darwin [MK96] se focalisent sur la description de la dynamique du système.

Pour garantir la cohérence des (re)configurations, ces ADL sont basés sur des langages formels comme l'algèbre de processus de type CSP (*Communicating sequential processes*) [ADG98], π calcul [MK96, CPT99] ou réseaux de Pétri [ZC06, SBMP07], les systèmes de transition d'états de type LTS (Labelled Transition Systems) [KM98], la logique temporelle linéaire [ZGC09, DKL12]. La logique du premier ordre peut également être utilisé pour formaliser la spécification du modèle d'architecture (i.e., spécifier les relations architecturales et les contraintes d'intégrité ou invariants sur les configurations). Par exemple, les invariants sont exprimés en logique du premier ordre avec le langage de contraintes Armani qui étend l'ADL ACME [GMW00]. Le langage de spécification Alloy [Jac02] – qui repose sur la logique du premier ordre – est doté d'un analyseur de cohérence des spécifications et est utilisé dans plusieurs travaux pour valider les reconfigurations [TMS10, EM13].

Supports à l'exécution

Etat « quiescent ». L'adaptation dynamique doit être synchronisée avec l'exécution métier du logiciel. Kramer et al. [KM90] proposent le concept d'état dit « quiescent » pour suspendre l'exécution de la brique en question et mettre en place un protocole spécifique de communication/synchronisation entre les briques impactées (cf. Figure 3.3).

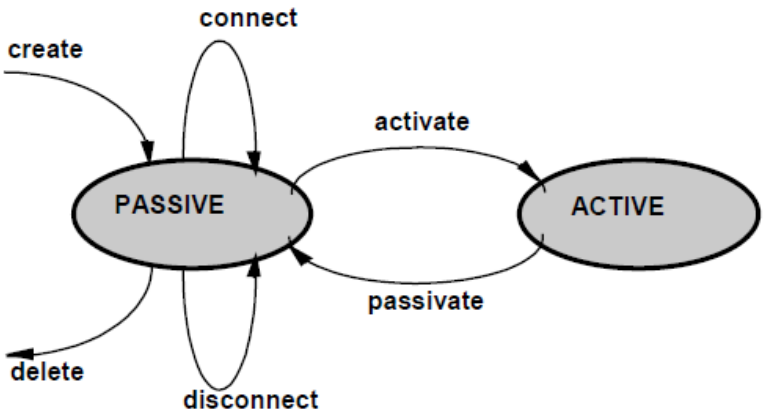


FIGURE 3.3 – Transitions d'états pour obtenir un état "quiescent" [KM98]

Adaptation transactionnelle. Dans le cas de reconfigurations simultanées et concurrentes, un gestionnaire de transactions permet d'initier, de valider ou d'annuler les opérations d'adaptation sans perturber l'exécution du système [MBC04, BJC05]. Trouvant leurs origines dans le domaine des bases de données, les transactions sont un moyen de réaliser une unité atomique d'accès à une ressource et donc de réaliser une adaptation de manière fiable en cas de concurrence d'accès ou de défaillance.

Models@run-time et connexion causale lâche. Grâce aux Models@run-time [BBF09] et leur connexion causale lâche, il est possible de vérifier la cohérence des modifications à apporter au logiciel au niveau modèle, avant de les reporter sur l'application s'exécutant [FHS+06, SXC+10, Fou13]. Cela nécessite l'utilisation d'un mécanisme de synchronisation entre l'application s'exécutant et sa représentation abstraite. Ce mécanisme peut avoir un coût sur les temps de réponse de l'adaptation : il est donc important de minimiser l'impact de ce mécanisme.

Cas du transfert d'état

Lorsque l'adaptation consiste au remplacement d'une brique avec état par une autre, la nouvelle brique doit se retrouver dans une situation cohérente qui permette la continuité de service. Le mécanisme d'adaptation doit donc assurer le transfert d'état entre les deux briques. Ce transfert revêt deux aspects [Bal10] :

1. le transfert de l'ensemble des valeurs des attributs de la brique ;
2. le transfert du point d'exécution atteint par le système.

Le transfert des valeurs est en principe réalisé par leur sérialisation/externalisation depuis l'ancienne brique suivie de leur injection dans la nouvelle brique à son instanciation (e.g., K-component [DC01]). Ce problème est régulièrement abordé dans la littérature [VB03, EVHB05, ZC06]. L'autre aspect concerne le transfert du point d'exécution. L'approche souvent adoptée consiste à n'autoriser l'adaptation que lorsque la brique est inactive (i.e., état quiescent [KM90]). Sinon, il faut instrumenter le code à la compilation afin de sauvegarder les variables de la pile d'exécution pour la reconstruire ensuite dans le nouveau contexte [Hof93].

3.3.2 Performance

> « *It should not be necessary to stop the whole of a running application system to modify part of it. The management system should, from the change specification, be able to determine a minimal set of nodes which are affected by the change. The rest of the system should be able to continue its execution normally [KM90].* » (Kramer et al. - 1990)

L'impact de l'adaptation sur le logiciel doit être court dans le temps pour que ce dernier puisse garantir la continuité de service. Il s'agit de minimiser l'*overhead* lié au processus d'adaptation, c'est à dire le temps dans le lequel le logiciel reste dans l'état « logiciel à l'exécution en cours d'adaptation » (cf. Section 2.2.3).

Deux types d'approche permettent d'aborder cette problématique : (i) minimiser la surface du logiciel à adapter ; (ii) diminuer la durée de l'adaptation. Ces deux approches peuvent être également combinées.

Minimiser l'espace

Dans une architecture composée d'un ensemble de *briques*, une granularité assez fine permet de bloquer seulement la brique – ou l'ensemble de briques – à adapter tout en laissant le logiciel s'exécuter [KM90]. L'une des conséquences de ce constat est que l'adaptation dynamique sied particulièrement aux architectures logicielles modulaires [GCH+04]. La preuve en est l'incroyable étendue des travaux menés dans ce domaine évoquée plusieurs fois dans ce manuscrit.

Minimiser le temps

Dans la Section 2.4, nous avons évoqué le patron du processus d'adaptation et également fait un point sur le moment de la réalisation de l'adaptation (le "*apply*" de [Kee04]). La durée de l'adaptation va être liée à la durée de traitement des fonctions constituant le processus d'adaptation, i.e., fonctions d'observation, de décision, de planification, d'exécution.

L'une des possibilités pour diminuer ce temps de traitement est d'anticiper le moment de l'adaptation (*adaptation proactive* [KRV+15]) en intégrant des modèles prédictifs ou des modèles d'apprentissage dans les fonctions observation/décision. Une autre possibilité est de distribuer la prise de décision sur plusieurs fonctions d'analyse/décision qui vont collaborer et se coordonner dans une configuration décentralisée pour améliorer le temps de réponse de la décision [WSG+13]. Une possibilité supplémentaire enfin est d'avoir une API efficace représentant la fonction d'exécution :

sa conception et sa richesse (variété des techniques de mutation, d'habillage) peuvent améliorer le temps de réalisation de l'adaptation.

En ce qui concerne les logiciels *context-aware*, la durée de l'adaptation par rapport aux évolutions du contexte doit être la plus faible possible, non seulement afin de ne pas perturber le fonctionnement normal du logiciel, mais aussi afin d'être compatible avec les dynamiques d'évolution de l'environnement. Des temps d'adaptation trop importants pourraient être à l'origine d'une désynchronisation entre le logiciel et son contexte. Le logiciel pourrait alors tomber dans une forme d'indéterminisme qui consisterait à ne jamais être cohérent avec son environnement [Fer11].

3.3.3 Évaluation qualitative de l'adaptation dynamique

Si l'adaptation dynamique mise en place respecte les garanties exposées ci-dessus, alors il n'y aura pas d'attitude de défiance vis-à-vis de l'adaptation à l'exécution. Rappelons que la fiabilité et la performance font partie des critères principaux en qualité logicielle [fSI11]. On peut également énumérer d'autres critères comme la *précision des résultats*, la *facilité d'utilisation*, l'*extensibilité* qui vont également concerner l'adaptation dynamique.

Les deux derniers critères seront potentiellement satisfaits si l'adaptation est vue comme une préoccupation que l'on peut développer à part. La facilité d'utilisation dépendra du type d'approche choisie (déclarative *vs* impérative). L'extensibilité sera permise si il existe un support aux adaptations non anticipées (*open-adaptive* [OGT+99]). Sinon, concernant la précision des résultats, tout dépendra du modèle adopté pour la prise de décision (cf. Section 2.4.7).

Enfin, dans [Bas07], l'auteur propose de mesurer la qualité d'un processus d'adaptation suivant les critères ci-dessous :

- le degré d'automatisation : la capacité du logiciel à réaliser sa propre adaptation, sans humain dans la boucle, en fonction de son contexte d'exécution. Est-ce qu'un degré d'autonomie de 100% est toujours synonyme de qualité ? Pas forcément, d'après les auteurs de [BD03] qui démontrent que dans certains contextes, l'humain doit garder une part de décision.
- la diversité des mécanismes d'adaptation : la richesse des techniques pour adapter la brique, la liaison entre deux briques ou l'architecture globale (cf. Section 2.4.3) va permettre d'offrir un large choix pour la mise au point d'une solution adaptée.
- la capacité de prise de décision : *last but not least*, la disposition à réaliser un ajustement le plus fin, le plus pertinent possible au nouveau contexte reste déterminant. Rappelons que la prise de décision peut être modélisée de façon très différente en se basant sur des paradigmes variés (cf. Section 2.4.7).

3.4 Bilan

Pour répondre aux défis du numérique tels que la complexité croissante du logiciel et les systèmes multi-échelles (e.g., informatique nomade, *Fog computing*), l'adaptation du logiciel à l'exécution nous semble être une piste importante à privilégier. Cependant, pour intégrer durablement les principes de l'adaptation dynamique au logiciel, il faut réaliser un certain nombre de choix.

Tout d'abord, si on veut l'observer ou le modifier, le logiciel doit être **réifié** à l'exécution. De même pour l'environnement d'exécution pour les logiciels ***context-aware***. Pour permettre d'agir dessus sans causer de rupture de service, ce logiciel doit aussi offrir une certaine **modularité** et la capacité à pouvoir être **(re)composé** à l'exécution. Pour permettre de nouveaux plans de recomposition qui n'avaient pas été prévus à la conception, il faut que le logiciel puisse offrir un support aux **adaptations non anticipées**.

Ensuite, pour améliorer la compréhension, la maintenance, la testabilité et la modularité du logiciel, il faut une séparation claire entre logique métier et logique d'adaptation (i.e., "***adaptation concern as first-class entity***" [CdLG+09]). Enfin, pour garantir la **fiabilité**, la **performance** et

plus globalement la **qualité de service** du processus adaptation logicielle, il faut considérer un certain nombre d'éléments comme la granularité de la brique logicielle, la conformité de la nouvelle configuration à la spécification d'un modèle d'architecture, la gestion des états, l'occurrence possible de plusieurs reconfigurations simultanées, …

La deuxième partie de ce manuscrit se propose de résumer, mettre en perspective mes travaux réalisés dans ce contexte au cours des années 2000. Nous allons d'une part aborder l'adaptation dynamique comme objet de première classe en proposant un patron d'auto-administration du logiciel et un langage dédié pour programmer l'adaptation comme préoccupation transverse. D'autre part, nous proposerons une solution pour la fiabilité des reconfigurations dynamiques dans les architectures logicielles adressant aussi bien une démarche pour maximiser la continuité de service du logiciel à reconfigurer que la spécification d'une (re)configuration correcte.

Deuxième partie

Reconfiguration dynamique : comment ?

Chapitre 4

Auto-reconfiguration d'architectures logicielles et langages dédiés

Dans ce chapitre, nous allons présenter nos principaux travaux ayant trait à l'adaptation logicielle comme objet de première classe (*"first-class citizen"*). Ces travaux ont été menés dans le cadre de la thèse de Pierre-Charles David [Dav05] et du projet RNTL ARCAD (2000-2004). Avant d'exposer les résultats de ces travaux, nous allons rapidement présenter les pré-requis nécessaires à la bonne compréhension de ce chapitre.

4.1 Prérequis (a.k.a. interfaces requises pour comprendre ce chapitre)

Nous avons volontairement limité ces pré-requis à trois grands thèmes que l'on retrouvera de façon récurrente dans ce manuscrit. Le lecteur averti pourra sauter la lecture de cette section !

4.1.1 Modèle de composants Fractal

Le modèle de composants Fractal [BCL+04, BCL+06] a été développé conjointement par France Telecom R&D [1] et l'INRIA au début des années 2000 et est distribué en *open source* par le consortium OW2 [2]. Il s'agit d'un modèle ouvert, réflexif et récursif (hiérarchique sur plusieurs niveaux) avec notion de partage.

Motivations et définition. L'objectif principal du modèle est la construction de systèmes répartis, hautement adaptables en proposant des techniques d'ingénierie des systèmes par assemblage de composants logiciels. Un des points forts du modèle est sa vision homogène et intégré du logiciel : tout est composant à une granularité arbitraire. En effet, grâce à son aspect hiérarchique, on peut aussi bien représenter par des composants des ressources de bas niveau (e.g., ressources matérielles) que des briques logicielles à grosse granularité telles que des serveurs d'application (e.g., serveur Java EE [BBH+05]). Ainsi, Fractal peut être utilisé pour des développements logiciels très variés : applications classiques, architectures orientées services, intergiciel, système d'exploitation. L'architecture est réifiée à l'exécution et propose des capacités d'introspection et d'intercession. Le modèle est abstrait et indépendant des langages d'implémentation. L'implémentation de référence de la spécification est un framework Java nommé Julia [BCL+04].

Description. Les concepts clés du modèle sont les composants, les interfaces et les liaisons (cf. Figure 4.1).

1. Orange Labs
2. `https://www.ow2.org`

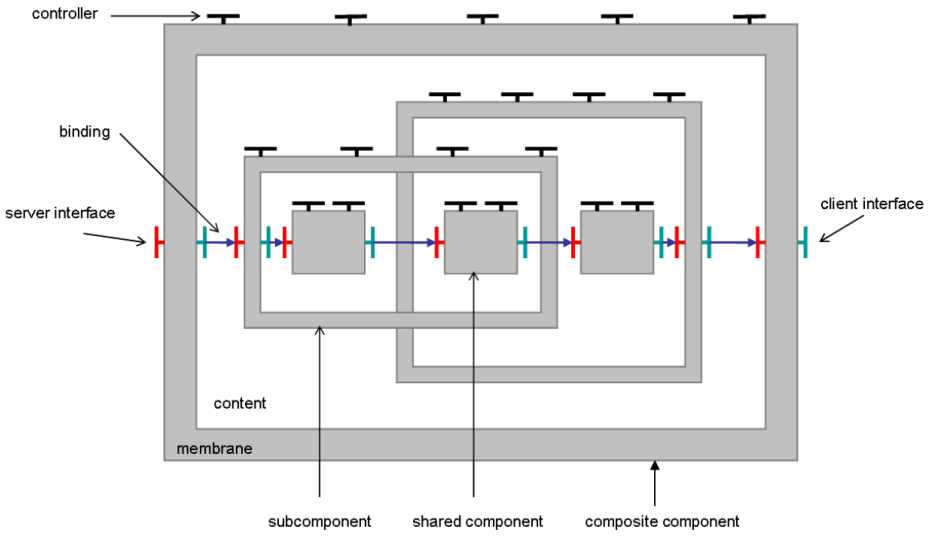


FIGURE 4.1 – Une architecture Fractal

Composants. Les composants sont les unités de compilation et d'exécution du système. Une des caractéristiques importantes du modèle Fractal est la séparation de la notion de composant en deux parties distinctes :

- un *contenu* qui définit l'intérieur d'un composant. Le contenu peut être soit un objet (au sens large) du langage de programmation sous-jacent (i.e., composant *primitif*), soit être constitué d'autres composants (composant *composite*) ;
- une *membrane*[3] qui définit l'enveloppe extérieure d'un composant et qui permet un contrôle sur ce composant.

Interfaces. Les interfaces sont les points d'accès entre les composants pour l'émission ou la réception de messages. Elles sont divisées en deux catégories : les interfaces métier et les interfaces de contrôle (ou *contrôleurs*). Les interfaces métier jouent le rôle de client ou serveur : une interface serveur reçoit des opérations d'invocation alors qu'une interface cliente peut en émettre. Les interfaces de contrôle permettent d'introspecter et de manipuler le composant en tant que tel. Dans les deux cas, la membrane peut intercepter et manipuler les messages avant de les renvoyer. Les interfaces sont typées et l'ensemble des types des interfaces métier d'un composant définit le type du composant.

Liaisons. Les liaisons en Fractal sont des canaux de communication entre interfaces de composants. Une *liaison primitive* relie deux interfaces de composants dans un même espace d'adressage et la sémantique de communication par défaut est l'invocation synchrone d'opération. Une *liaison composite* est un assemblage de composants et de liaisons primitives. Ce dernier type de liaison permet d'implémenter des communications complexes : communications réparties, asynchrones, multicast, ... Contrairement à d'autres modèles de composants (e.g., OpenCOM [CBG+08]), il n'existe pas de concept de connecteur en tant qu'entité de première classe.

Des contrôleurs comme interface réflexive. Fractal en tant que modèle réflexif propose une séparation des préoccupations entre un niveau méta – qui correspond au contrôle dans les membranes des composants – et le niveau de base – qui est l'implémentation métier du contenu de ces mêmes composants. Un certain nombre de contrôleurs sont définis par défaut dans la spécification :

3. Par analogie avec les cellules des organismes vivants.

- `BindingController` permet d'introspecter et de manipuler les liaisons entre composants ;
- `ContentController` permet de manipuler le contenu d'un composant composite par introspection de la structure ;
- `SuperController` offre la possibilité de naviguer dans la hiérarchie des composants parents ;
- `NameController` offre la possibilité de récupérer ou de changer le nom d'un composant ;
- `AttributeController` permet d'exposer les attributs configurables d'un composant ;
- `LifeCycleController` permet de gérer le cycle de vie d'un composant (démarrage, arrêt).

Cependant, de nouveaux contrôleurs peuvent être ajoutés aux composants pour étendre le modèle de base Fractal. De nombreux travaux ont montré cette possibilité en introduisant des extensions comme les contrats [CRCR05], les middleware orienté messages [LQS05], les moniteurs transactionnels [RSAM06], un service d'auto-réparation [SBDP08], la communication multi-cast [BCD+09], les architectures orientées services [SMR+12].

4.1.2 Langages dédiés

> « *A domain-specific language (DSL) is a programming language or executable specification language that offers, through appropriate notations and abstractions, expressive power focused on, and usually restricted to, a particular problem domain [vDKV00].* »
> (vanDeursen et al. - 2000)

Motivations et définition. La plupart des langages de programmation utilisés par les développeurs (e.g., C, Java, VB, Haskell, etc.) sont des langages généralistes, c'est-à-dire qu'ils sont conçus pour pouvoir créer n'importe quel type de logiciel, quel que soit le domaine. A l'inverse, les langages dédiés – ou *domain-specific language* (DSL) – sont des langages conçus spécifiquement pour un domaine d'application particulier (e.g., SQL pour les bases de données, les ADL pour la description d'architectures logicielles [MT00]). La plupart du temps (mais pas toujours), ces langages dédiés ne sont pas aussi puissants que les langages généralistes au sens où ils ne sont pas Turing-complets.

Avantages. Les avantages des langages dédiés par rapport aux langages généralistes sont les suivants :

- l'utilisation d'un niveau d'abstraction et de notations spécifiques au problème rend le code beaucoup plus lisible et concis. L'expertise du domaine est capturée dans le langage lui-même ce qui facilite pour les non-experts la compréhension et la réutilisation du code ;
- le développement de programmes est généralement beaucoup plus rapide et le résultat a plus de chances d'être correct. En effet, ces langages sont souvent plus déclaratifs qu'impératifs, et leur niveau d'abstraction élimine un grand nombre d'erreurs courantes dans des langages de plus bas niveau ;
- le pouvoir d'expression généralement limité du langage permet d'offrir des garanties et des analyses statiques plus poussées. Par exemple, le langage SQL-92 n'est pas récursif, ce qui garantit que les requêtes terminent toujours ;
- les programmes écrits peuvent souvent avoir de meilleures performances car les implémentations de ces langages dédiés peuvent tirer partie d'optimisations spécifiques à leur domaine. Par exemple, les bases de données commerciales implémentent de nombreuses optimisations sur les requêtes SQL fournies par leurs utilisateurs.

Inconvénients. Cependant, les langages dédiés ont aussi des inconvénients :

- la conception d'un langage dédié peut être difficile car il n'est pas simple de trouver le bon niveau d'abstraction et de généralité pour le langage. Un langage dédié trop ouvert perd de son intérêt face à un langage généraliste, tandis qu'un langage trop limité est certes plus efficace mais utilisable seulement sur une classe restreinte de problèmes ;

— ils sont difficiles à implémenter car ils requièrent à la fois une expertise dans le domaine concerné et dans celui de l'implémentation de langages de programmation [CM98, Mar07] ;
— à chaque fois qu'un nouveau langage dédié est créé, ses futurs utilisateurs doivent être formés à son utilisation. L'apprentissage d'un nouveau langage, surtout spécialisé, est plus long et difficile.

4.1.3 Autonomic computing

> « *Autonomic computing is the ability of an IT infrastructure to adapt to change in accordance with business policies and objectives [KC03].* » (Kephart et al. - 2003)

Motivations, définitions et propriétés. L'informatique autonomique a été introduite en 2001 par IBM comme une réponse à la difficulté d'administration des systèmes informatiques. En effet, la complexité croissante des systèmes informatiques (i.e., en termes d'envergure, d'architecture, de technologies, etc.) ainsi que le contexte hautement dynamique dans lequel ils évoluent (i.e., sollicitations fluctuantes, pannes logicielles/matérielles, etc.) a rendu la tâche d'administration difficile voire même impossible pour l'homme. L'objectif principal de l'informatique autonomique est de soulager l'opérateur humain en charge de la gestion du système, en l'assistant dans les tâches d'administration, et, ultimement, de le remplacer quand cela est possible.

Les systèmes autonomiques sont classés en quatre catégories selon les objectifs d'administration [KC03, HM08] :

— Auto-configuration (*self-configuration*) : dénote la capacité du système à se configurer dynamiquement et automatiquement aux variations de l'environnement, en suivant des objectifs de haut niveau ;
— Auto-optimisation (*self-optimization*) : dénote la capacité du système à optimiser en permanence l'utilisation de ses ressources et/ou sa qualité de service. Cette optimisation peut être guidée par des politiques définies par l'être humain (e.g., diminution empreinte énergétique, nombre de requêtes traitées par seconde, etc.) ;
— Auto-réparation (*self-healing*) : dénote la capacité du système à détecter, diagnostiquer puis compenser ou réparer des anomalies (e.g., panne matérielle) survenant dans le système au cours de son fonctionnement de manière automatique ;
— Auto-protection (*self-protection*) : dénote la capacité défensive du système qui vise à anticiper les problèmes et ajuster son comportement pour se protéger des événements qui peuvent altérer son état et ainsi le déstabiliser. Cette propriété doit lui permettre de réagir à tous types de menaces et anticiper les attaques malveillantes.

Boucle de contrôle MAPE-K. IBM a proposé un modèle de référence appelée la boucle MAPE-K (Monitor, Analyse, Plan, Execute, Knowledge) [KC03], concept issu de la théorie du contrôle en automatique. La Figure 4.2 propose l'organisation des éléments de ce modèle : un élément administré (*management element*) est en relation avec un gestionnaire autonomique (*autonomic manager*) responsable de l'observation et de l'adaptation du système administré à l'intérieur de la boucle rétroactive.

Les éléments administrés représentent le système dont on attend un comportement autonome en le couplant avec un gestionnaire autonomique. Il s'agit concrètement des ressources logicielles ou matérielles. Un gestionnaire autonomique est une brique logicielle qui doit être configurée manuellement par des administrateurs humains capables de spécifier leurs intentions sous forme de politiques de haut niveau. La communication entre l'élément administré et le gestionnaire autonomique se fait au travers de deux interfaces. La première interface – matérialisée par des capteurs (*sensors*) – va adresser la surveillance et la collecte d'informations/de métriques (introspection), tandis

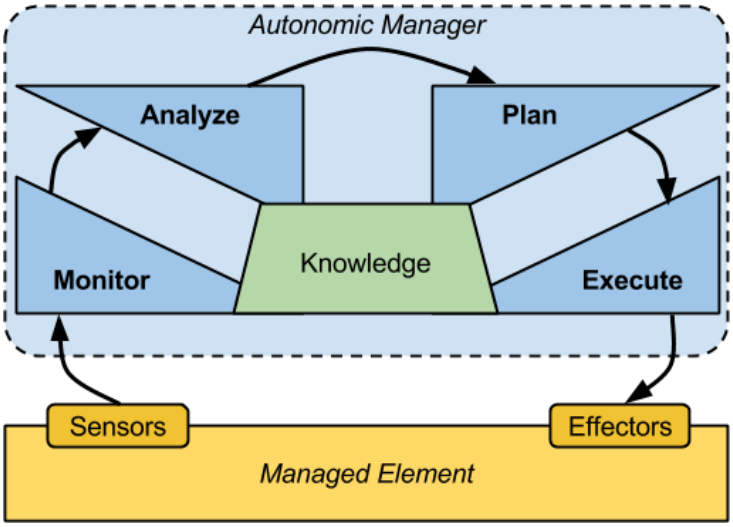


FIGURE 4.2 – Boucle MAPE-K [KC03]

que la seconde interface – matérialisée par actionneurs (*effectors*) – va concerner l'application des changements sur le système administré (intercession).

Nous détaillons ci-dessous les différentes composantes d'une boucle MAPE-K :

- Observation (*Monitor* - *M*) : collecte des informations exposées par les éléments administrés au travers des capteurs. Ces informations peuvent être pré-filtrées pour éviter de déclencher un processus d'analyse inutile (cf. Section 3.3.2) ;
- Analyse (*Analyze* - *A*) : en charge de prendre en compte les données remontées par le monitoring pour comparer l'état courant du système à l'état attendu. L'objectif final de cette phase est de prendre une décision sur le(s) changement(s) à apporter au système ;
- Planification (*Plan* - *P*) : en charge de définir une série d'adaptations à effectuer sur les éléments administrés. Il s'agit concrètement de traduire le diagnostic de l'analyse et de préparer un plan complet des actions à mettre en œuvre sur le système. Les phases d'analyse et de planification sont parfois regroupées en une unique phase de décision ;
- Exécution (*Execute* - *E*) : en charge d'appliquer le plan de reconfiguration par le biais d'appel(s) aux actionneurs ;
- Base de connaissances (*Knowledge* - K) : les différentes composantes de la boucle MAPE-K sont reliées à la base de connaissances. Elle peut être peuplée automatiquement ou manuellement par différentes sources. Par exemple, elle peut contenir des historiques de la phase d'observation, les reconfigurations passées ou encore des informations rajoutées par des experts humains pouvant aider le gestionnaire autonomique dans sa prise de décision (e.g., politiques d'adaptation). Les quatre phases MAPE ainsi que l'administrateur humain peuvent interagir avec cette base de connaissances aussi bien en lecture qu'en écriture.

4.2 Safran : une extension de Fractal pour créer des applications auto-adaptables

Au début des années 2000, c'est dans le cadre du projet ARCAD qu'une collaboration entre France Telecom R&D et l'INRIA va donner naissance au modèle de composants Fractal. Acteur de ce projet, je vois rapidement les atouts d'un tel modèle pour constituer la brique de base pour le développement d'applications auto-adaptables. Aussi, avec mon doctorant Pierre-Charles David,

nous décidons d'étudier la conception d'une boucle MAPE-K dédiée à la reconfiguration dynamique d'applications Fractal.

4.2.1 Conception et architecture générale

Safran (Self-Adaptive Fractal CompoNents) [Dav05] étend le modèle de composants Fractal afin de faciliter le développement d'applications adaptatives. La conception de Safran est basée sur les trois principes suivants :

- L'utilisation d'un **modèle de composants dynamique et réflexif**, Fractal [BCL+06], pour la construction des applications à administrer (i.e., *management element*). Avec Fractal, il est possible de reconfigurer dynamiquement l'application en modifiant son architecture, et donc indirectement son comportement, afin de l'adapter aux évolutions de son contexte d'exécution ;
- La gestion de **la logique d'adaptation comme « objet de plein droit »**, séparée de l'application elle-même (i.e., *management element*). Puisque de nouveaux contrôleurs peuvent être ajoutés aux composants pour étendre le modèle de base Fractal, nous allons spécifier un contrôleur inédit, dédié à l'adaptation.
- L'utilisation d'un **langage dédié** pour programmer la logique d'adaptation. Concrètement, ce langage permet d'écrire des *politiques d'adaptation* pour chaque composant adaptable. Ce dernier deviendra adaptatif (cf. Section 2.2) grâce à l'interprétation des politiques par le nouveau contrôleur dédié à l'adaptation.

L'externalisation de la logique d'adaptation va permettre à la fois un découplage spatial (i.e., modularisation des politiques en dehors du code métier) et un découplage temporel (i.e., possibilité de développer la stratégie une fois l'application en cours d'exécution, c'est à dire de réaliser des **adaptations non anticipées**).

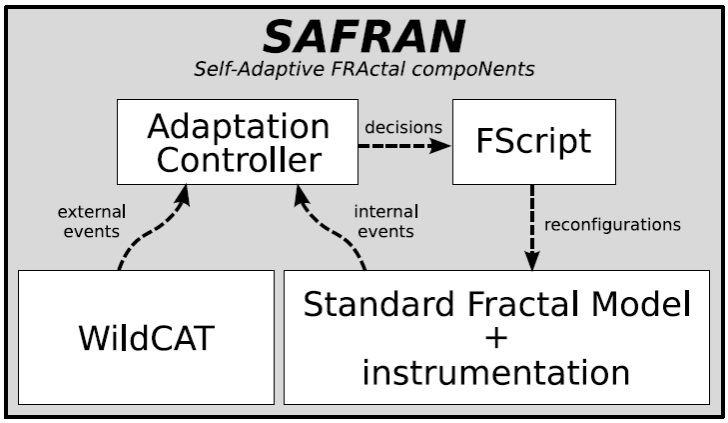


FIGURE 4.3 – Architecture générale Safran [DL06]

Par rapport aux propriétés citées à la Section 3.4, la conception de Safran remplit de nombreuses promesses. Il manque cependant l'aspect *context-aware*. Aussi, nous avons proposé l'architecture suivante (cf. Figure 4.3) pour notre framework [4] Safran :

- Le cœur du système est constitué d'une extension générique de Fractal pour le support de la logique d'adaptation via un nouveau contrôleur – `AdaptationController` – et d'une implémentation concrète de cette extension. De plus, un certain nombre de modifications ont

4. Safran est désigné par le terme *framework* puisqu'il va proposer un socle applicatif définissant les grandes lignes pour développer des applications auto-adaptables.

été apportées dans les membranes des composants pour permettre de détecter des événements *endogènes* (*internal events*) comme la création d'un composant, l'envoi et la réception de messages, ...
- En amont, le framework intègre le système WildCAT (cf. Section 4.3) pour l'observation du contexte d'exécution de l'application et la notification des événements *exogènes* (*external events*) qui s'y produisent.
- En aval, Safran utilise le langage dédié FScript (cf. Section 4.4) pour la spécification des reconfigurations.

Si on fait le lien entre l'architecture proposée et la boucle de contrôle MAPE-K, c'est comme si la composante 'M' était réalisée par la notification d'événements à la fois endogènes (dans l'application Fractal) et exogènes (contexte extérieur à l'application via WildCAT), la composante 'AP' était assurée par les politiques d'adaptations sauvegardées par le composante 'K', et enfin la composante 'E' était effectuée par FScript.

4.2.2 Mise en oeuvre d'un langage dédié à l'adaptation

L'implémentation concrète des politiques d'adaptation que fournit Safran se présente sous la forme d'un langage dédié. Dans un premier temps, nous allons voir quel est le lien entre les architectures Fractal et les politiques. Puis, nous décrirons les grandes lignes de ce langage dédié.

Un nouveau contrôleur pour l'adaptation

Safran introduit une extension au modèle Fractal qui permet d'associer dynamiquement des politiques d'adaptation aux composants métiers d'une application. Cette extension, minimale et générique, est indépendante des politiques utilisées et se traduit par la définition d'une nouvelle interface de contrôle nommée `AdaptationController`. C'est la présence de cette interface qui transforme les composants Fractal en composants adaptatifs, i.e., acteur de leur propre adaptation. L'interface `AdaptationController` possède la signature suivante :

```
public interface AdaptationController {
   void attachFcPolicy(AdaptationPolicy policy);
   void detachFcPolicy(AdaptationPolicy policy);
   AdaptationPolicy[]getFcPolicies();
}
```

Les méthodes de cette interface permettent d'introspecter et de manipuler dynamiquement l'ensemble des politiques d'adaptation qui s'appliquent au composant Fractal. Lorsqu'une politique est associée à un composant donné, elle est en charge d'adapter ce composant, et uniquement celui-ci. En effet, à l'instar de la programmation par composants qui permet un couplage lâche entre les éléments d'un logiciel, l'objectif est de respecter l'isolation des composants en limitant la portée des modifications proposée par la politique d'adaptation. Il est alors possible de raisonner localement sur les effets d'une politique sans connaitre l'intégralité de l'application. Précisons que dans le cas d'un composant *composite*, la politique d'adaptation peut agir sur ses sous-composants et ainsi de suite récursivement (modèle hiérarchique) ; mais l'ajout d'une politique à un composant ne brise pas l'encapsulation.

Structure des politiques d'adaptation

Se plaçant d'emblée dans un contexte de communication de type *push* (i.e., notifications asynchrones d'événements) (cf. Section 2.4.6), les politiques d'adaptation Safran sont structurées comme un ensemble de règles réactives de la forme :

```
when <event> if <condition> do <action>
```

Ce type de règles est similaire à ce que l'on peut trouver dans le domaine des bases de données actives sous la dénomination de règles ECA (Event- Condition-Action) [ANC96]. Une règle d'adaptation indique que lorsqu'un événement (endogène ou exogène) correspondant à l'expression `<event>` se produit, si l'expression `<condition>` concernant l'état du système est vraie, alors la reconfiguration `<action>` est exécutée (cf. Figure 4.4).

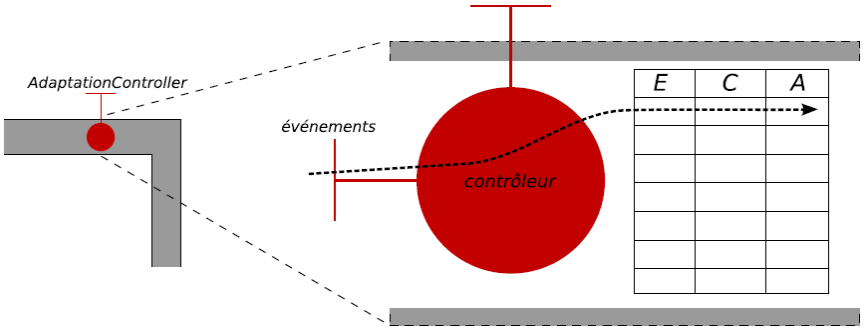


FIGURE 4.4 – Contrôleur d'adaptation Fractal [Dav05]

La règle réactive constitue le cœur du langage dédié permettant de programmer la logique d'adaptation pour Safran. Comme on peut le voir, nous avons opté pour une approche impérative pour modéliser la logique d'adaptation (cf. Section 3.2), plus facile à mettre en œuvre pour l'administrateur. Par la suite, nous décrirons comment spécifier l'expression `<event>` d'une règle réactive, qui indique le ou les événements qui doivent la déclencher. Les deux autres parties d'une règle feront l'objet d'une description complète plus loin dans ce manuscrit puisque la `<condition>` sera basée sur FPath [5] et l'`<action>` sur FScript (cf. Section 4.4), deux langages dédiés qui ont été conçus selon les règles de l'art.

Enfin, les politiques d'adaptation associées dynamiquement aux composants Fractal adaptatifs sont constituées d'une suite ordonnée de règles d'adaptation :

```
1 policy example() = {
2   rule { when <event1> if <cond1> do <action1> };
3   rule { when <event2> if <cond2> do <action2> };
4   rule { when <event3> if <cond3> do <action3> };
5 }
```

Lorsqu'une telle politique est associée à un composant Fractal – grâce à son interface `AdaptationController` – le contrôleur se met à l'écoute des événements consignés dans la politique. Dès qu'un de ces événements se produit, la ou les règles correspondantes sont activées, leurs conditions évaluées, et pour celles qui sont vraies, leurs actions sont exécutées en séquence.

Spécification et détection d'événements

L'expression `<event>` d'une règle réactive va être définie par un *descripteur d'événements* dont l'objectif est de permettre à la fois la spécification des événements significatifs pour l'adaptation, puis leur détection. De nombreuses réflexions et propositions sur la création d'événements composites ou complexes, sur la fraicheur de l'occurrence d'un événement, sur l'efficacité d'un système de remontée d'événements, ... peuvent être trouvées dans [Dav05]. Ici, nous allons seulement nous focaliser sur la création et la détection d'événement primitif endogène ou exogène.

5. voir les invariants architecturaux dans la Section 4.4.2

Spécification et détection des événements primitifs. Qu'ils soient endogènes ou exogènes, toutes les occurrences d'événement dans Safran sont représentées comme des objets de plein droit avec un ensemble de propriétés. Certaines de ces propriétés sont présentes pour chaque événement alors que certaines sont spécifiques à des types d'événements. Les propriétés communes sont le type de l'événement, sa source (un composant Fractal ou un élément du contexte d'exécution) et un *timestamp* (sa date de création).

La spécification d'événement et la détection sont réalisées par des descripteurs d'événements, pour lesquels la syntaxe exacte dépend du type d'événement, mais suit toujours la forme générale : `event-type(parameters)`. Par exemple, `changed(sys://storage/memory#free)` permet de détecter les variations de la quantité de mémoire disponible sur le système.

Pour capturer l'occurrence d'un événement primitif et le rendre accessible dans les autres parties de la règle réactive (i.e., `<condition>` et `<action>`), il faut préfixer son descripteur par un symbole ' :'. Par exemple, `mem:changed(sys://storage/memory#free)` va permettre de récupérer l'objet-événement `$mem` (utilisation du caractère '$' comme préfixe pour référencer l'objet) et accéder à ses propriétés (e.g., `$mem.new-value`).

Événements endogènes. Les événements endogènes sont notifiés soit à partir du flot d'exécution métier de l'application Fractal, soit à partir de changements dans son architecture :

- Les trois premiers types – `message-received`, `message-returned`, `message-failed` – correspondent respectivement à la réception d'un message, au retour réussi d'un message et à la levée d'une exception. Les descripteurs pour ces trois sortes d'événements partagent les mêmes propriétés, utilisent FPath pour indiquer quelles interfaces doivent être contrôlées (cf. Section 4.4.2 pour connaitre la syntaxe de FPath). Par exemple, `message-received($c/interface::logger)` détecte les invocations sur n'importe quelle méthode de l'interface `logger` du composant `$c` ;
- Les autres types d'événements endogènes correspondent aux reconfigurations possibles : création de composant, changement du cycle de vie (i.e., composant démarré ou arrêté), configuration d'attributs, manipulation de contenu (i.e., ajout/retrait de sous-composants), modification des connexions entre composants et finalement (dés)association de politiques d'adaptation au composant. Par exemple, `component-started($c/child::*)` détecte quand n'importe quel sous-composant direct de `$c` est démarré.

La mise en œuvre de ces événements est basée sur l'instrumentation des contrôleurs Fractal. Par exemple, le contrôleur `LifeCycleController` est instrumenté pour produire les événements `component-started/stopped`.

Événements exogènes. Pour détecter l'occurrence d'événements exogènes, nous devons d'abord réifier le contexte d'exécution de l'application, qui est normalement implicite. Pour cela, nous avons conçu WildCAT, un framework facilitant la construction d'applications *context-aware* (cf. Section 4.3).

WildCAT modélise le contexte d'exécution comme un ensemble de domaines contextuels, chacun représentant un aspect particulier du contexte (e.g., ressources matérielle, réseau, informations géophysiques, etc.). Chacun de ces domaines contextuels est modélisé comme un arbre de ressources décrites par un ensemble d'attributs. La syntaxe pour accéder à ces ressources est calquée sur celle des URI : `domain://path/to/resource#attribute` (`attribute` étant optionnel). Par exemple, `sys://storage/drives/hdc#removable` indique si le disque hdc est amovible.

WildCAT va notifier l'occurrence des événements exogènes qui peuvent déclencher l'exécution de reconfigurations. Les trois différents types d'événements supportés sont les suivants :

- `changed(expression)` : détecte n'importe quelle modification de la valeur de l'expression (qui peut faire référence à n'importe quel attribut ou ressource dans le contexte). Par exemple, `changed(sys://storage/memory#free)` ;

— `realized(condition)` : détecte l'occurrence d'une condition booléenne. Par exemple, `realized(sys://storage/memory#free > 2*sys://storage/swap#used)`;
— `appears(path)` *vs* `disappears(path)` : détecte l'apparition *vs* la disparition d'une ressource ou d'un attribut du contexte. Par exemple, `appears(sys://devices/input/*)` qui détecte l'apparition de n'importe quel nouveau dispositif d'entrée.

Exemple d'une règle avec événements endogènes

Pour illustrer la syntaxe d'une règle réactive, nous montrons une règle avec événement endogène [6]. Ce type de règle permet de garantir des invariants architecturaux de l'application à l'exécution.

```
1  when e:subcomponent-added($target)
2    if (count($target/child::*[started(.)]) > 3)
3    do { stop($e.sub-component); }
```

Cette règle assure que le composite `$target` ne contient pas plus de 3 composants actifs à la fois. `$target` est une variable spéciale qui désigne le composant cible sur lequel une politique d'adaptation est attachée (équivalent d'un `this` en Java). La condition est écrite en FPath alors que la partie action utilise FScript. Le corps de l'action référence la variable `$e.sub-component` qui correspond au champ `sub-component` de l'occurrence de l'événement `e` et donc au nouveau sous-composant ajouté à `$target`.

Cycle de vie des politiques d'adaptation

Le cycle de vie des politiques d'adaptation Safran est caractérisé par les étapes suivantes : (i) programmation de la politique grâce au langage dédié ; (ii) attachement de la politique au composant cible ; (iii) exécution ; (iv) détachement de la politique du composant cible. Certaines étapes méritent d'être explicitées pour mieux comprendre l'implémentation de Safran (pour plus de détails, voir [Dav05]).

Attachement/Détachement. Lors de l'attachement de la politique P au composant C, le contrôleur d'adaptation de C extrait de P l'ensemble des événements primitifs mentionnés dans les règles de P. S'il s'agit d'un événement exogène, le contrôleur s'abonne auprès de WildCAT afin d'être notifié de chaque occurrence de cet événement. S'il s'agit d'un événement endogène, le système se charge d'instrumenter le programme Fractal de façon appropriée pour que C soit notifié à chaque occurrence de cet événement.

Lors d'un détachement d'une politique d'adaptation, le contrôleur d'adaptation se désabonne de tous les événements correspondants (exogène et endogène).

Exécution. Plusieurs cas de figure peuvent se produire pendant l'exécution, ayant trait soit à l'interaction entre règles ou entre politiques pour un même composant, soit à l'interaction entre composants. Ci-dessous, voici les grands lignes de notre implémentation :

— À l'intérieur d'une politique donnée, les réactions des règles sont composées dans l'ordre textuel de leur définition et exécutées comme une unique reconfiguration. Le raisonnement suivi est qu'une politique donnée devrait mettre en œuvre une adaptation consistante et indépendante et on peut s'attendre à ce que son auteur prévoit en amont les interactions entre règles ;
— Quand plusieurs politiques sont attachées au même composant, l'ordre dans lequel les réactions des politiques sont exécutées dépend de l'ordre de leur attachement : les politiques les plus vieilles sont exécutées d'abord. Le raisonnement qui est fait est qu'une fois une politique P attachée au composant C, le composant résultant C' doit être considéré comme une boîte

6. Nous verrons un exemple complet avec événements exogènes dans la Section 4.5.

noire indépendante des politiques suivantes, et par conséquent P a une plus grande priorité sur les politiques attachées ultérieurement ;
— Quand des composants multiples doivent réagir au même événement, leurs réactions sont exécutées dans un ordre défini par les relations de composition des composants : les sous-composants sont adaptés avant leurs parents. Le raisonnement est semblable au précédent : dans une approche à base de composants, quand un composite inclut un sous-composant, il devrait le traiter comme une boîte noire.

4.3 WildCAT : modélisation et observation du contexte d'exécution

WildCAT (*Context-Awareness Toolkit*) [DL05, MLH+09] est un *framework* Java pour faciliter la création d'applications *context-aware*. Il propose un **modèle abstrait** du contexte d'exécution basé sur un modèle de données extensible et dynamique et offre une API simple aux programmeurs pour accéder à cette information de manière synchrone (*pull*) ou asynchrone (*push*) (cf. Section 2.4.6). WildCAT est hébergé par le consortium OW2 (`http://wildcat.ow2.org`).

4.3.1 Motivations et objectifs

Généralement, lorsqu'une application a besoin de s'adapter à un élément de son contexte, elle implémente cette observation de façon spécifique. Cette approche pose plusieurs problèmes :
— le code de détection correspondant doit être réécrit pour chaque application ;
— ce code nécessite souvent d'interagir directement avec le système d'exploitation, voire avec le matériel, ce qui réduit la portabilité de l'application ;
— les programmeurs d'une application ont rarement les compétences techniques pour écrire correctement ce code de bas niveau.

La solution que nous proposons consiste à fournir aux programmeurs d'application *context-aware* un service générique et extensible de modélisation et d'observation du contexte d'exécution. Ce service permet aux applications de :
— offrir une vision structurée du contexte ;
— découvrir leur contexte d'exécution dynamiquement via des sondes (de simples objets Java implémentant une interface particulière) ;
— raisonner sur celui-ci au-delà de l'acquisition de données brutes ;
— être notifié quand certains événements d'intérêt se produisent.

En résumé, notre objectif avec WildCAT est de permettre à n'importe quel développeur de modéliser l'observation d'une entité physique (e.g., mémoire vive d'un ordinateur, température ou position GPS d'une salle dans un bâtiment) ou logicielle (e.g., objets Java, évolution de design dans Eclipse [MLH+09]), et d'intégrer cette observation dans un programme Java de manière simple et homogène.

4.3.2 Modélisation du contexte d'exécution

WildCAT propose un modèle abstrait du contexte d'exécution basé sur les concepts de domaine contextuel, ressources et attributs (cf. Figure 4.5).

Domaine contextuel

WildCAT modélise le contexte d'une application sous la forme d'un ensemble de *domaines contextuels*, chacun représentant un aspect particulier du contexte. Chaque domaine contextuel est identifié par un nom globalement unique dans le contexte d'une application donnée. Chaque contexte est l'élément de racine d'une hiérarchie, semblable au système de fichiers Unix.

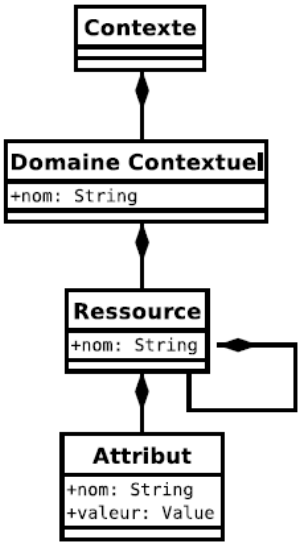


FIGURE 4.5 – Méta-modèle de WildCAT [Dav05]

La distinction entre les différents domaines contextuels permet de bien séparer les différentes préoccupations, et de réutiliser la définition de certains domaines entre plusieurs applications. Chacun peut être vu comme un *ontologie* modélisant un aspect particulier du contexte, et peut être implémenté par des experts du domaine en question.

Exemples de domaine contextuel Dans la version de base de WildCAT, le *framework* propose des domaines contextuels suivants :

- `sys` : ressources matérielles disponibles pour l'application (cf. Figure 4.6) ;
- `soft` : ressources logicielles disponibles ;
- `pref` : préférences utilisateur et autres paramètres de configuration ;
- `net` : état du réseau (topologie, performances, ...) ;
- `phys` : environnement physique de l'hôte sur lequel tourne l'application (localisation géographique, température, bruit et luminosité, ...) ;
- `user` : caractéristiques physiques de l'utilisateur (handicap, activité, état émotionnel, ...).

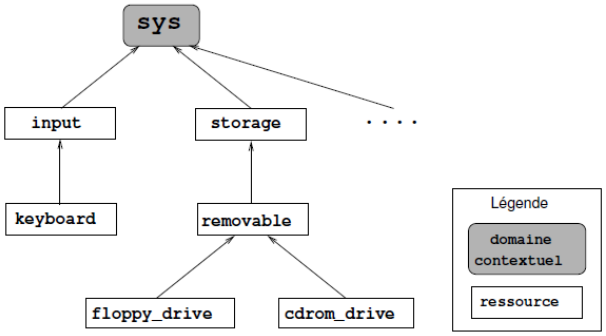


FIGURE 4.6 – Domaine contextuel `sys` [Dav05]

Ressources

Chaque domaine contextuel est organisé en une arborescence de ressources, chacune étant identifiée par son nom comme des dossiers dans un système de fichiers Unix. On distingue deux types de ressources :

- *ressources de base* : unités de structuration du modèle contextuel de type physique (e.g., `cdrom_drive`) ou logique (e.g., `storage`) (cf. Figure 4.6) ;
- *liens symboliques* : ressources spéciales qui se réfèrent à une autre ressource. Les liens symboliques sont utilisés pour créer des raccourcis – comme en Unix – dans le modèle contextuel pour avoir accès plus rapidement aux ressources importantes ou aux attributs, sans naviguer dans l'arbre entier.

Attributs

Un attribut est un simple couple (nom-valeur), les noms des attributs d'une ressource étant uniques pour cette ressource. Les attributs sont les feuilles de l'arbre de ressources (comme les fichiers dans un système de fichiers Unix). On distingue trois types d'attributs :

- *attribut de base* : possède une valeur statique (i.e., non modifiable) ;
- *attribut actif* : possède une valeur dynamique correspondant directement à une valeur observée dans le contexte d'exécution par une sonde WildCAT (e.g., mémoire vive) ;
- *attribut synthétique* (ou attribut de requête) : maintient le résultat d'expressions sur d'autres attributs. Il peut agréger et transformer les valeurs fournies par d'autres attributs, comme par exemple, le calcul de valeurs moyennes pendant 10 secondes.

Les sondes WildCAT sont conçues pour suivre un *template* POJO (*Plain Old Java Object*) qui permet l'accès aux attributs utilisant des méthodes *getter*/*setter* dont les noms sont déterminés par une convention simple. Cette caractéristique puissante permet aux développeurs de concevoir leurs propres sondes comme des classes Java et les associer à des sources d'information externes indépendamment de leur nature (e.g., réseau, thermomètre, GPS, etc.) [7].

Évolution dynamique du contexte

S'appuyant sur la modélisation précédente, WildCAT va pouvoir être à l'écoute les modifications suivantes : modification des valeurs des attributs, apparition ou disparition d'attributs au sein d'une ressource, apparition ou disparition de ressources. Chaque modification génère un événement spécifique qui sera traitée par l'interface asynchrone.

4.3.3 Interface de programmation

Du point de vue de ses utilisateurs, WildCAT présente une interface relativement simple, qui donne accès aux deux modes d'utilisation possibles du service – interrogation synchrone *pull* et notification asynchrone *push* – sans exposer la complexité du fonctionnement interne [8].

En 2008, Esper [Esp] devient le *backend* de WildCAT qui passe à la version 2.0. Loin d'être anecdotique, cette évolution permet d'adresser pleinement les atouts du *Complex Event Processing* (CEP) [CA08] depuis WildCAT. Le mode *push* de WildCAT devient plus fiable et l'interface de programmation est revue.

7. Un certain nombre de sondes (e.g., monde Java, kernel Linux) sont fournies de base par l'implémentation de WildCAT.
8. Pour savoir comment développer de nouvelles sondes, configurer WildCAT, ... voir [Dav05].

Désignation des ressources et des attributs

Comme il a été indiqué dans la Section 4.2, la syntaxe pour accéder aux ressources est calquée sur celle des URI : `domain://path/to/resource#attribute` (`attribute` étant optionnel). Voici quelques exemples :

```
sys://storage/memory                // une ressource
sys://storage/disks/hda#removable   // un attribut (de type booléen)
sys://devices/input/*               // tous les périphériques de "input"
sys://devices/input/mouse#*         // tous les attributs de la souris
```

Interrogation du contexte

La création d'un domaine contextuel et l'interrogation de celui-ci est réalisée de façon naturelle grâce à une API *Pull* relativement simple. Nous préférons donner un aperçu de cette API via un exemple plutôt que de commenter chaque interface de programmation. Pour plus de détails, voir le site Web du projet WildCAT (`http://wildcat.ow2.org`).

Exemple. Notre objectif est de construire un domaine contextuel regroupant un certain nombre de ressources informatiques tel que décrit dans la Figure 4.7. Ci-dessous le code permettant de construire cette arborescence de ressources et d'interagir avec (cf. Listing 4.1).

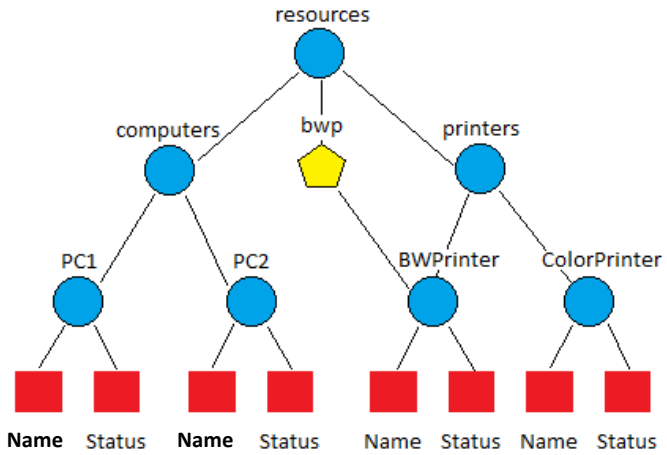


FIGURE 4.7 – Arborescence WildCAT

```
1  // Creating a context through the ContextFactory
2  Context ctx = ContextFactory.getDefaultFactory().createContext("
       TechnicalResources");
3
4  // Getting the context name
5  System.out.println("Context name = " + ctx.getName());
6
7  try {
8  // Creating some resources
9    ctx.createResource("self://computers/pc1");
10   ctx.createResource("self://computers/pc2");
11   ctx.createResource("self://printers/ColorPrinter");
12   ctx.createResource("self://printers/BWPrinter");
13
14 // Creating and initializing attributes
15   ctx.createAttribute("self://computers/pc1#name", "pc1");
16   ctx.createAttribute("self://computers/pc1#status", "ON");
```

```
17   ctx.createAttribute("self://computers/pc2#name", "pc2");
18   ctx.createAttribute("self://computers/pc2#status", "OFF");
19   ctx.createAttribute("self://printers/ColorPrinter#name", "colorprinter");
20   ctx.createAttribute("self://printers/ColorPrinter#status","Standby");
21   ctx.createAttribute("self://printers/BWPrinter#name", "bwprinter");
22   ctx.createAttribute("self://printers/BWPrinter#status", "Standby");
23
24 // Listing all printers
25   System.out.println("Printers = " + ctx.list("self://printers"));
26 //Getting pc2 status attribute
27   System.out.println("pc2 status is : " + ctx.getValue("self://computers/pc2#
        status"));
28 //Setting pc1 status attribute to OFF
29   ctx.setValue("self://computers/pc1#status", "OFF");
30   System.out.println("pc1 status is : " + ctx.getValue("self://computers/pc1#
        status"));
31 // Creating a symbolic link to the Black and White printer
32   ctx.createSymlink("self://bwp", "self://printers/BWPrinter");
33 //Getting BWPrinter status attribute using the symbolic link
34   System.out.println("BWPrinter status is : " + ctx.getValue("self://bwp#status")
        );
35  } catch (ContextException e) { e.printStackTrace();}
36
37 //OUTPUT
38 Context name = TechnicalResources
39 Printers = [ColorPrinter, BWPrinter]
40 pc2 status is : OFF
41 pc1 status is : OFF
42 BWPrinter status is : Standby
```

Listing 4.1 – WildCAT : exemple mode PULL

Notifications asynchrones avec WildCAT 2.0

WildCAT 2.0 est basé sur Esper [Esp] et utilise le langage de requêtes EPL (*Event Processing Language*) d'Esper et permet donc de profiter de la puissance d'expression de ce langage SQL-like. EPL est un langage SQL standard (avec clauses SELECT, FROM, WHERE, GROUP BY, HAVING, ORDER BY) augmenté de certaines extensions. Les flux d'événements remplacent les tables avec comme source de données les événements remplaçant les lignes. Puisque les événements sont composés de données, les concepts de SQL (corrélation, jointures, filtres, agrégation par regroupement) peuvent être utilisés efficacement. Des fenêtres glissantes sur des événements (dans le temps et/ou sur le nombre d'occurrences notifiées) ou du *pattern matching* viennent compléter le dispositif.

Concrètement, WildCAT permet aux développeurs d'inspecter le contenu d'un contexte en enregistrant des requêtes sur des événements produits par l'arborescence des ressources. Dans WildCAT, chaque événement implémente l'interface `WEvent`. Cela permet de déterminer pour chaque événement, le nœud dans la hiérarchie qui a émis l'événement. Il y a deux sortes d'événement émis par un domaine contextuel :

- les événements émis par des ressources : ils instancient la classe `WHierarchyEvent` et informent de l'opération exécutée sur la structure de la hiérarchie (e.g., nouvel attribut, création d'un lien symbolique) ;
- les événements émis par des attributs : ils instancient la classe `WAttributeEvent` et indique la modification de cet attribut et sa nouvelle valeur.

Exemple n°1. L'exemple présenté dans le Listing 4.2 fait la supposition qu'on a ajouté un attribut actif `load` à notre `PC1` de la Figure 4.7. Il va ensuite construire une attribut synthétique – intégré à l'arborescence – représentant la charge moyenne de la machine.

```
1 // Creating a query attribute holding the CPU load average for the last 15
    seconds
2  QueryAttribute LoadAVG = new QueryAttribute("select avg(value.load) as loadAVG
     from
3   WAttributeEvent(source = 'self://computers/pc1#load').win:time(15 sec)");
4
5 // Attaching the query attribute "LoadAVG" to the hierarchy
6  ctx.attachAttribute("self://computers/pc1#LoadAVG", LoadAVG);
```

Listing 4.2 – WildCAT : exemple n°1 mode PUSH

Exemple n°2. L'exemple présenté dans le Listing 4.3 montre l'aspect générique de WildCAT puisque nous allons écouter maintenant du code Java et non plus des ressources physiques. Il s'agit d'un exemple relativement classique d'une surveillance de compte bancaire. Cet exemple pourrait être programmé avec le patron de conception `Observer` mais il s'agit ici de montrer que nous restons dans la philosophie du framework WildCAT où toute ressource est modélisable et observable.

```
1 // We suppose a context has been already created with a resources tree of bank
    accounts
2 // Select all clients with balance < 0
3
4 Query query = ctx.createQuery("select * from
5   WAttributeEvent(source like 'self://bank/accounts/*', value('balance')? < 0)");
6
7 // Registering a listener to query (API herited from Esper)
8 ctx.registerListeners(query, new UpdateListener () {
9   void update (EventBean[] new_events, EventBean[] old_events) {
10    for (EventBean bean : new_events) {
11      System.out.println("Warning : The customer " + bean.get(("value('owner')?")
          + " has a negative balance ! ") };
12 }};
```

Listing 4.3 – WildCAT : exemple n°2 mode PUSH

Exemple n°3. L'exemple présenté dans le Listing 4.4 montre une utilisation avancée du *Complex Event Processing* pour identifier des événements notables. Dans un cluster de machines, il s'agit d'identifier une machine en sous-utilisation. La requête, basée sur un *pattern matching* EPL, va renvoyer les machines allumées ayant une charge de moins de 10% pendant au moins 15 minutes.

```
1 select a.source from pattern [
2   every a = WAttributeEvent(source like 'self://infra/pms#pm\%' and value('on') =
       true and value('utilization')<10)
3   --> (timer:interval (15 min) and not WAttributeEvent(source = a.source and
      value('utilization')>10)]
```

Listing 4.4 – WildCAT : exemple n°3 mode PUSH

4.4 FPath/FScript : navigation et reconfiguration dans les architectures Fractal

Dans cette section, nous présentons les langages dédiés FPath et FScript [Dav05, DLLC09]. FPath est un langage pour naviguer, introspecter et sélectionner des composants dans une architecture Fractal. FScript est un langage pour reconfigurer à l'exécution des architectures Fractal. FPath et FScript sont hébergés par le consortium OW2 (`http://fractal.ow2.org/fscript`). Plus de détails peuvent être trouvés dans la thèse de Pierre-Charles David [Dav05].

4.4.1 Problématique et motivations

Le modèle Fractal est spécifié directement sous la forme d'un ensemble d'interfaces de programmation (APIs) [9]. Utiliser directement ces interfaces pour programmer les reconfigurations, par exemple en Java, rend difficile la possibilité d'offrir des garanties quant à l'effet de ces reconfigurations sur le logiciel.

De plus, les interfaces de programmation de Fractal sont conçues pour être minimales, et leur utilisation en pratique est souvent très verbeuse. Cela est d'autant plus vrai dans le cas d'un langage typé statiquement comme Java, qui nécessite des transtypages qui rendent le code rapidement illisible. La gestion explicite des exceptions est aussi un élément qui alourdit le code. Il en résulte que des opérations mêmes simples nécessitent un code important. Par exemple, le code pour ajouter le composant `child` au composite `parent` en Java est présenté dans le Listing 4.5.

```
1 LifeCycleController lc = null;
2 try {
3   lc = (LifeCycleController) parent.getFcInterface("lifecycle-controller");
4 } catch (NoSuchInterfaceException nsie) { /* Do nothing. */ }
5 boolean started = (lc != null) ? "STARTED".equals(lc.getFcState()) : false;
6 if (started) {
7   try { lc.stopFc(); }
8   catch (IllegalLifeCycleException ilce) { /* Handle error. */ }
9 }
10 try {
11   ContentController cc = (ContentController) parent.getFcInterface("content-
        controller");
12   try { cc.addFcSubComponent(child); }
13   catch (IllegalLifeCycleException ilce) { /* Should not happen. */ }
14   catch (IllegalContentException ice) { /* Handle error. */ }
15 } catch (NoSuchInterfaceException nsie) { /* Handle error. */ }
16 finally {
17   if (started) {
18   try { lc.startFc(); }
19   catch (IllegalLifeCycleException ilce) { /* Handle error. */ }
20   }
21 }
```

Listing 4.5 – Reconfiguration Fractal en Java.

Enfin, le modèle Fractal introduit des concepts (composants, interfaces) qui n'existent pas directement dans le langage hôte (e.g., Java) et sans extension syntaxique – comme celles présentées par exemple dans ArchJava [ACN02] – il est difficile de manipuler aisément des concepts architecturaux.

Ces trois problèmes (i.e., manque de garanties, notation verbeuse et abstractions non représentées) correspondent précisément aux problèmes résolus par les langages dédiés (cf. Section 4.1.2). C'est pourquoi nous avons défini FScript, un langage dédié à la spécification et à l'exécution de reconfigurations dynamiques d'applications Fractal. Comme son nom l'indique, FScript peut être considéré comme un langage de script qui permet de programmer des scripts de reconfigurations, mais dont le pouvoir d'expression, la syntaxe et l'implémentation ont été conçus pour offrir toutes les propriétés des DSL. FScript embarque un autre langage dédié nommé FPath – inspiré de XPath – pour la navigation dans les architectures Fractal.

4.4.2 Navigation avec FPath

FPath est un langage de navigation, de sélection et de requêtes dans des architectures Fractal. Tant sur la syntaxe que sur le modèle d'exécution, FPath est inspiré de XPath [W3C10], langage de requêtes sur les documents XML du W3C. FPath est un langage d'introspection : il ne modifie pas les configurations Fractal. FPath est utilisé pour choisir des éléments architecturaux dans un

9. cf. `http://fractal.ow2.org/specification/index.html`

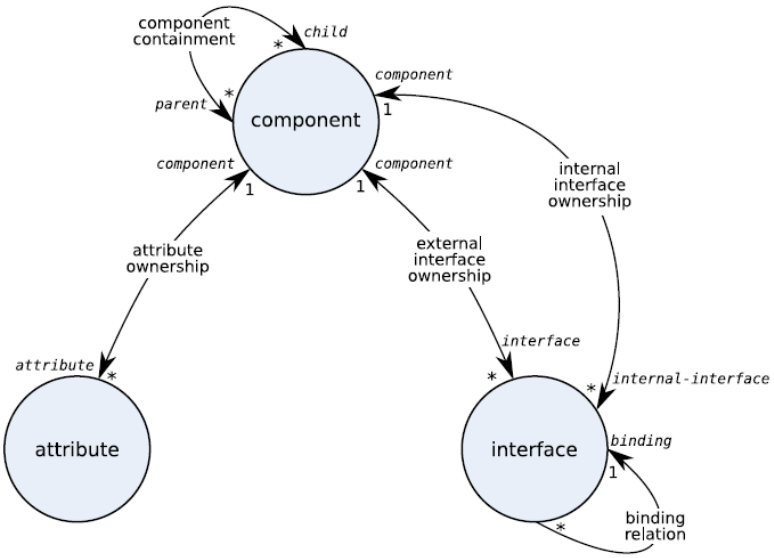


FIGURE 4.8 – Méta modèle FPath [DLLC09]

système cible selon une combinaison de leurs propriétés (e.g., l'état d'un composant) et leurs relations à d'autres (e.g., les sous-composants d'un composite).

Modélisation des architectures Fractal

Le langage FPath repose sur la modélisation d'un ensemble de composants Fractal sous la forme d'un graphe orienté, dont les nœuds représentent les composants, leurs interfaces et leurs attributs, et dont les arcs sont annotés par des labels qui dénotent le type de relation entre deux nœuds (e.g., parent *vs* enfant, connexion entre interfaces, ...). La Figure 4.8 récapitule tous les axes « primitifs » possibles définis par FPath entre les différentes sortes de nœuds. Ces axes sont appelés primitifs parce qu'ils représentent les relations principales entre les éléments du modèle. FPath définit aussi des axes « dérivés » – à partir des axes primitifs – qui sont accessibles de la même façon à l'utilisateur, mais qui permettent de réaliser des requêtes plus puissantes (e.g., tous les voisins, tous les descendants, ...).

Pour illustrer le modèle, la Figure 4.9 montre l'architecture Fractal d'un petit serveur Web nommé Comanche et sa représentation en tant que modèle FPath.

Description

Etant donnée la représentation orientée graphe décrite ci-dessus, une expression FPath est utilisée pour naviguer dans un graphe de composants, à partir d'un nœud initial (ou un ensemble de nœuds) et en suivant des axes spécifiques à l'intérieur du graphe pour atteindre d'autres nœuds. La syntaxe générale d'une telle expression est un ensemble de pas (*step*) séparés par le caractère '/' , i.e., `step1/step2/.../stepN`. Chacun de ces pas est lui-même structuré en trois éléments :

`step == axe::test[pred]` avec

- `axe`, un identifiant (parmi un ensemble fini) d'un label possible pour un arc du graphe ;
- `test`, soit l'identifiant d'un nœud, soit tous les nœuds (*wildcard* '*') ;
- `pred`, une liste facultative de prédicats pour choisir des nœuds plus précisément.

Chaque pas transforme un ensemble initial de nœuds en un nouvel ensemble de nœuds. Une expression FPath s'évalue sans effet de bord et retourne toujours un ensemble homogène de composants, d'interfaces ou d'attributs (éventuellement vide).

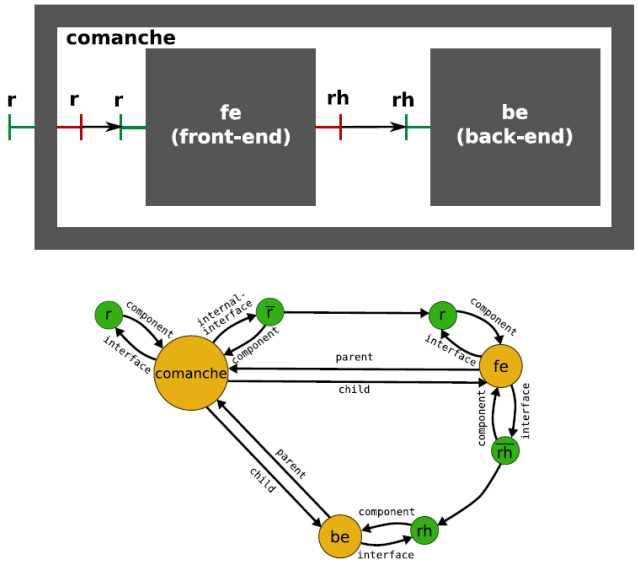


FIGURE 4.9 – Exemple modèle FPath [DLLC09]

En plus des constructions spécifiques au domaine, pour construire des prédicats élaborés, FPath supporte un ensemble de types de données réduit (nombres, chaine de caractères et booléens) et des expressions habituelles que l'on trouve dans la plupart des langages de programmation : expressions arithmétiques et booléennes, comparaisons et invocation de fonctions d'introspection [10].

Un exemple de la syntaxe générale est donné dans la Figure 4.10. L'objectif de la requête est de trouver tous les composants dans le serveur Comanche qui sont arrêtés. La première partie de l'expression, `$comanche`, référence une variable et on obtient sa valeur en la préfixant par le caractère '$'. Dans notre cas, il s'agit du composant représentant le nœud racine du serveur Web Comanche. Le premier pas utilise l'axe `descendant-or-self`, qui est la fermeture transitive et réflexive de l'axe primitif `child`. Parce que l'expression utilise une *wildcard* ('*'), ceci choisira tous les composants, y compris la racine. Puis, à la fin du pas, le prédicat invoque la fonction `stopped()` sur chaque nœud du résultat intermédiaire (le caractère '.' dénote l'argument implicite passé au prédicat). Ainsi, seuls les composants arrêtés dans l'architecture Fractal Comanche seront retournés au pas suivant.

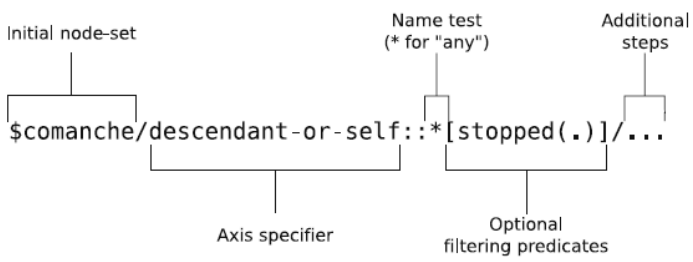


FIGURE 4.10 – Code FPath [DLLC09]

10. FPath fournit un certain nombre de fonctions prédéfinies, qui correspondent aux méthodes d'introspection de Fractal. Par exemple, `started(comp)` renvoie un booléen et permet de savoir si le composant `comp` est démarré ou non.

Quelques exemples

L'objectif de cette section est d'illustrer l'expressivité de FPath. Pour plus de détails, se reporter à [DLLC09].

Composants configurables. En Fractal, les composants pouvant être configurés possèdent un contrôleur `AttributeController`. Dans des architectures complexes avec des douzaines de composants configurables, dispersés dans de nombreux niveaux de composition du système, il n'est pas toujours évident de savoir quels paramètres de configuration sont disponibles. Avec une requête FPath très simple, il est possible de découvrir tous les attributs de configuration supportés par une telle architecture [11] :

```
$root/descendant-or-self::*/@*
```

Composants partagés. Fractal accepte le concept de partage de composants, c'est-à-dire que les composants peuvent avoir plusieurs parents directs. C'est une caractéristique puissante, mais qui peut rendre une architecture plus difficile à comprendre, comme chacun des parents essaye de contrôler le composant partagé, exigeant un peu de coordination. Encore une fois, FPath permet d'identifier très facilement tous les composants partagés dans une architecture :

```
$root/descendant-or-self::*[size(./parent::*)>1]
```

Invariants architecturaux. Fractal est un modèle très flexible et permet de reconfigurer une architecture dynamiquement. Cependant, il est important de vérifier que certaines contraintes soient respectées pour assurer la cohérence des architectures. Par exemple, avant qu'un composant ne puisse être démarré, toutes ses interfaces clientes – qui ne sont pas facultatives – doivent nécessairement être connectées à une interface serveur compatible. Une telle contrainte peut être vérifiée facilement avec une requête FPath qui trouvera tous les composants violant cette contrainte dans une architecture :

```
$root/descendant-or-self::*/interface::*[client(.)][mandatory(.)][not(./binding::*)]
```

4.4.3 Reconfiguration avec FScript

FScript est un langage dédié qui permet la définition de reconfigurations complexes d'architectures Fractal et rien d'autre. FScript intègre FPath directement dans sa syntaxe pour sélectionner les éléments à reconfigurer.

Motivations

FScript est focalisé sur la manipulation de concepts de niveau architectural. Il fournit le contrôle complet de l'architecture et de la configuration des applications Fractal. Cela signifie que tous les concepts définis par le modèle Fractal sont disponibles dans le langage, tant pour l'introspection que pour la modification (i.e., intégration des API des contrôleurs, cf. Section 4.1.1). Parce que son domaine est limité au niveau architectural, cela signifie aussi que FScript n'est pas explicitement conçu pour mettre en œuvre des composants Fractal. Enfin, le pouvoir d'expression limité du langage permet d'offrir des garanties de fiabilité (e.g., pas de boucles infinies) (cf. Section 4.1.2).

Description

FScript est un langage de script impératif avec une syntaxe surtout prise à la famille des langages C (à part les expressions FPath). FScript réalise une vérification dynamique de types à l'invocation des procédures (i.e., typage dynamique).

11. Notons pour la suite du chapitre que le caractère '@' est un « sucre syntaxique » équivalent à "`attribute::`"

Définition des procédures. Un programme FScript est constitué d'un ensemble de définitions de *fonctions* et d'*actions* et d'appels à celles-ci. Les fonctions – introduites par le mot-clé `function` – vont seulement utiliser l'introspection de l'architecture (via FPath) et sont donc sans effet de bord. Elles peuvent être appelées dans des prédicats FPath. Les actions – introduites par le mot-clé `action` – vont modifier l'architecture cible. Seules les actions peuvent invoquer d'autres actions. Cette distinction permet d'offrir certaines garanties sur le comportement des scripts de reconfiguration. Le Listing 4.6 montre deux exemples de procédures.

```
1  /* Locate the request scheduler in a Comanche instance. */
2  function scheduler(comanche) {
3    return $comanche/child::fe/child::s;
4  }
5
6  /* Replace the request scheduler of a Comanche instance. */
7  action replace-scheduler(comanche, newSched) {
8    /* Invoke user-defined function. */
9    prev = scheduler($comanche);
10   /* replace() action defined below. */
11   replace($prev, $newSched);
12   return $prev;
13 }
```

Listing 4.6 – Fonction et action en FScript.

Structures de contrôle. FScript propose un ensemble limité de structures de contrôle pour garantir la terminaison des procédures. En plus de la séquence simple d'instructions (terminé par le caractère ';'), les structures de contrôle sont : l'affectation des variables, le branchement conditionnel, l'itération, le retour explicite (à tout point du code). Pour plus de détails, se reporter à [Dav05].

Actions primitives. Toutes les opérations de (re)configuration définies dans la spécification Fractal sont disponibles dans FScript comme des actions primitives. Cela constitue la bibliothèque standard d'API. FScript supporte aussi la création de nouveaux composants par l'action appelée `new()` [12]. Le Tableau 4.1 dresse la liste des actions primitives.

Actions	Légende
set-name(component, aString)	Modifie le nom du composant
set-value(attribute, value)	Modifie la valeur de l'attribut
value(attribute)	Renvoie la valeur de l'attribut
add(composite, sub-component)	Ajoute un nouveau sous-composant à un parent
remove(composite, sub-component)	Retire un sous-composant à un parent
bind(client-itf,server-itf)	Connecte deux interfaces qui doivent être compatibles
unbind(client-itf)	Supprime la connexion sortante
start(component)/stop(component)	Démarre/stoppe le composant
new(aString)	Crée un composant à partir d'un *template*

Table 4.1 – Actions primitives FScript

En plus de ces actions prédéfinies, de nouvelles actions peuvent être facilement ajoutées pour faire écho à de nouveaux contrôleurs Fractal développés en vue d'étendre la spécification de base de Fractal. C'est le cas de Safran avec la création du contrôleur d'adaptation `AdaptationController`.

12. Il correspond au `newComponent()` de l'interface `Factory` de l'ADL Fractal.

Garanties

La fiabilité des reconfigurations FScript est à la fois permise par la structure du langage lui-même – dont le pouvoir expressif a été limité – mais aussi par la mise en œuvre d'un support à l'exécution. En effet, une première version de FScript – implémentée par Pierre-Charles David en 2005 (cf. [Dav05]) – avait déjà l'idée d'un support transactionnel comme modèle d'exécution. Ce travail a été poursuivi et enrichi par la suite [Lég09] (cf. section 5.4). Plus précisément, l'interpréteur FScript garantit :

- (i) l'interdiction d'actions récursives ; (ii) un unique type d'itération de type "for" : pour chaque pas de boucle, l'itérateur réitère sur une expression FPath, qui rend toujours un ensemble fini de nœuds. Ces contraintes garantissent la terminaison des actions en temps fini ;
- la création d'un journal complet de toutes les actions primitives exécutées. Quand une erreur se produit, l'interpréteur utilise ce journal pour annuler la reconfiguration en cours (*rollback*) et retourner à l'état initial. Étant donné que toutes les actions primitives sont atomiques et réversibles, cela garantit l'atomicité des reconfigurations ;
- l'arrêt des composants avant d'exécuter toute action de reconfiguration. Ils sont redémarrés automatiquement à la fin de la reconfiguration. Cette caractéristique simplifie beaucoup la tâche du programmeur qui n'a pas besoin de faire un appel explicite à l'action `stop()` ;
- la vérification de l'intégrité structurelle en fin d'une reconfiguration, c'est-à-dire que toutes les interfaces clientes obligatoires sont correctement liées aux interfaces serveur correspondantes. Si cela n'est pas le cas, l'interpréteur annule la reconfiguration en cours.

L'ensemble de ces garanties constitue un premier support pour le développement de reconfigurations fiables.

Exemples

Le serveur Web Comanche déjà présenté ci-avant est conçu pour être très simple. Cependant, il n'inclut pas de cache de page et doit relire le contenu des fichiers à chaque requête. Avec FScript, il est facile de créer un nouveau composant de cache et de l'intégrer dynamiquement au bon endroit dans l'architecture. Le code décrit ci-dessous (cf. Listings 4.7, 4.8) présente les deux actions FScript nécessaires pour réaliser cette reconfiguration.

```
1    action get-or-create-cache(handler) {
2      if ($handler/child::cache) {
3        return $handler/child::cache;
4    }
5    else
6    {
7      cache = new("comanche.CachingFileRequestHandler");
8      set-name($cache, "cfh");
9      add($handler, $cache);
10     return $cache;
11   }
12 }
```

Listing 4.7 – Création d'un composant cache en FScript.

```
1  action enable-cache(handler) {
2    dispatcher = $handler/child::request-dispatcher;
3    if (not($dispatcher/interface::handler/binding::*/component::cache)) {
4      unbind($dispatcher/interface::handler);
5      file-handler = $handler/child::file-handler;
6      cache = get-or-create-cache($handler);
7      bind($dispatcher/interface::handler, $cache/interface::request-handler);
8      bind($cache/interface::handler, $file-handler/interface::request-handler);
9      start($cache);}
10   }
```

Listing 4.8 – Connexion d'un composant cache en FScript.

Enfin, la concision du code est l'un des avantages des langages dédiés. Le Tableau 4.2 donne le détail, procédure par procédure, du nombre de lignes de code en FScript *vs* en Java d'un exemple issu de la publication [DLLC09]. Le nombre de lignes dans le tableau n'inclut pas de lignes de commentaire.

Procedure	*LOC in FScript*	*LOC in Java*	*Increase*
bound-to	4	7	x1.4
bindings-to	5	26	x5.2
copy-lc-state	7	10	x1.42
copy-attributes	6	7	x1.16
replace	21	37	x1.76
Total	**43**	**86**	**x2**

Table 4.2 – Comparaison de code [DLLC09]

4.5 Application adaptative avec Safran

Pour conclure ce chapitre, nous reprenons l'architecture Fractal de notre petit serveur Web nommé Comanche en faisant un focus sur le *backend* et plus particulièrement sur le composant `handler`. La Figure 4.11 montre les deux états possibles de la configuration de l'architecture après la mise en place de la politique Safran (cf. Listing 4.9) sur le composant `handler`.

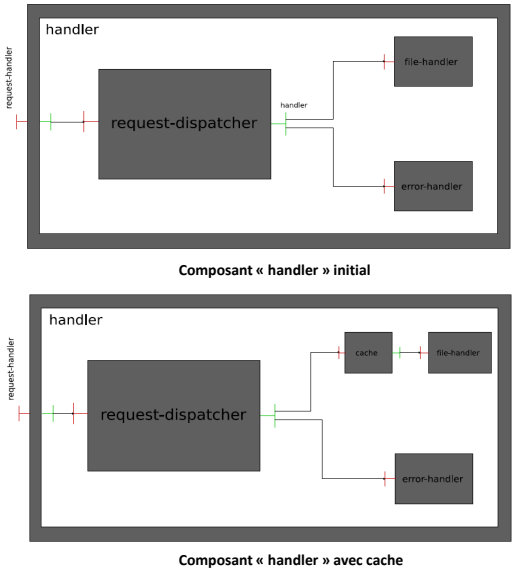


Figure 4.11 – Reconfiguration d'une architecture Fractal

Objectif. Comanche, étant extrêmement simple, n'intègre pas de mécanisme de cache. Pour améliorer ses performances, nous ajoutons ainsi un nouveau composant de cache dans le serveur Web. Les performances du cache dépendent de la quantité de la mémoire utilisée. Si cette quantité est trop basse, le système n'utilisera pas tout le potentiel du cache. Si c'est trop élevé, les performances peuvent être encore inférieures car le cache forcera le système d'exploitation à utiliser la mémoire virtuelle très lente sur disque (*swap*). La quantité de mémoire que nous devons allouer au composant cache dépend de la quantité de mémoire disponible sur le système hôte, qui varie dynamiquement et de façon imprévisible. Notre politique d'adaptation doit ainsi dynamiquement adapter la quantité

maximale de mémoire allouée au composant cache pour garantir de bonnes performances en toutes circonstances.

Description. Le composant cache expose deux paramètres – accessibles par son `AttributeController` – `currentSize` et `maximumSize`, indiquant respectivement la quantité actuelle et maximale de mémoire que le composant peut utiliser (seul `maximumSize` est en mode écriture). La politique fonctionne en ajustant la valeur de `maximumSize` selon la quantité de mémoire disponible sur le système hôte que WildCAT rend disponible grâce à l'information de contexte `sys://storage/memory#free`.

```
1  policy adaptive-cache = {
2
3  rule {
4    when realized(sys://storage/memory#free < 10*1024)
5    do {
6      toFree := 10*1024 - sys://storage/memory@free;
7      size := $target/cache/@currentSize - $toFree;
8      if ($size < 500) then {
9        set-value($target/cache/@maximumSize, 0);
10       disable-cache($target);
11     } else {
12       set-value($target/cache/@maximumSize, $size);
13       }
14   }
15 }
16
17 rule {
18   when mem:changed(sys://storage/memory@free)
19   if (sys://storage/memory@free >= 10*1024)
20   do {
21     enable-cache($target);
22     size := 0.8 * ($mem.new-value + $target/cache/@currentSize);
23     max := sys://storage/memory@used - $target/cache/@currentSize + $size;
24     if ($max < sys://storage/memory@total - 10*1024) {
25       set-value($target/cache/@maximumSize, $size);
26     }
27   }
28 }
29 }
```

Listing 4.9 – Politique d'adaptation Safran.

La politique Safran (cf. Listing 4.9) – attachée au composite `handler` (référencé dans le script par `$target`) – utilise deux actions FScript définies auparavant : `enable-cache` (cf. Listing 4.8) et `disable-cache` dont l'objectif est respectivement d'introduire le composant cache dans l'architecture ou de le retirer.

La première règle est déclenchée quand la capacité de la mémoire descend au-dessous de 10Mo (i.e., événement exogène). Quand ceci se produit, l'action de reconfiguration essaye de libérer de la mémoire en réduisant la taille de la mémoire cache, ou alors de désactiver complètement le cache en dessous d'un certain seuil.

La deuxième règle est déclenchée quand la quantité de mémoire change mais est supérieure à 10 Mo. Dans ce cas, la reconfiguration ajuste la taille de la mémoire cache pour utiliser au maximum 80% du total disponible, mais seulement si cela laisse assez de mémoire disponible au reste du système.

Cette politique Safran illustre (i) deux types différents d'événements exogènes (`realized` et `changed`) ; (ii) deux types de mécanismes d'adaptation : la paramétrisation et la technique d'habillage par interposition (cf. Section 2.4.3).

4.6 Conclusion et perspectives (a.k.a. interfaces fournies)

4.6.1 Lien avec la programmation par aspects

Dès 1997, de part la proximité intellectuelle entre Pierre Cointe (responsable d'équipe) et Gregor Kiczales (créateur de la programmation par aspects [KLM+97]), notre équipe de recherche décide d'axer une partie de ses travaux sur les liens entre méta-programmation et AOP (*Aspect-Oriented Programming*), ce qui débouchera sur la création en 2002 de l'équipe INRIA Obasco (**Ob**jets, **as**pects et **co**mposants).

Aussi, en démarrant la thèse de Pierre-Charles David en 2000, j'ai dans l'idée de tisser (sans jeu de mots !) des liens entre les applications auto-adaptables basées sur des composants et la programmation par aspects. En effet, Safran peut être vu comme une extension du modèle de composants Fractal permettant le développement d'un *aspect d'adaptation* sous la forme de politiques réactives qui détectent les évolutions du contexte d'exécution et adaptent le programme de base en le reconfigurant [DL06].

Un aspect d'adaptation

Rappel : programmation par aspects. Un aspect est un module qui regroupe et intègre des couples (*coupe, action*) qui, ensemble, implémentent une préoccupation (*concern*), en modifiant la sémantique du programme de base (i.e., métier) de façon transparente pour le programmeur du code de base [EFB01]. Les *actions* sont des fragments de code à exécuter à certains points du programme appelés *points de jonction*, qui peuvent correspondre soit à des éléments du code source du logiciel (e.g., point d'entrée d'une méthode [KHH+01]), soit à des événements se produisant lors de l'exécution (e.g., appel de méthode [DFS02]). Ces points de jonction sont regroupés en un ensemble appelés *coupes*. Le programme de base et les aspects sont ensuite *tissés* en un tout cohérent, soit statiquement [KHH+01], soit dynamiquement [DFS02].

Conception. Notre objectif est de considérer l'adaptation comme une aspect, c'est-à-dire une préoccupation transverse mais néanmoins modularisée grâce à des techniques de composition (tissage) appropriées. Il se trouve que Safran adresse la problématique de l'adaptation dynamique et semble fournir des constructions spécifiques pour exprimer des *coupes* (via des descripteurs d'événements) et des *actions* (via le langage dédié FScript). Plus précisément, nous avons effectué le raisonnement ci-dessous.

Presque tous les systèmes à aspects ont choisi de ne considérer que des points de jonction au niveau du code métier et des artefacts du langage de programmation (e.g., invocation de fonction, création d'objet, ...). Or, une application *context-aware* est affectée par le contexte dans lequel elle s'exécute et avec lequel elle interagit. Aussi, notre proposition pour les coupes de l'aspect d'adaptation est donc l'extension du domaine des points de jonction à l'ensemble du contexte du logiciel. Les événements endogènes (détectés dans Safran) correspondront ainsi aux points de jonction traditionnels (i.e., dans le code de base), alors que les événements exogènes, liés au contexte d'exécution (détectés par WildCAT), constituent un nouveau type de points de jonction qui permet de réagir aux évolutions du contexte, enrichissant par là même le pouvoir d'expression des aspects.

Concernant les actions déclenchées par la détection d'un point de jonction (i.e., un événement exogène ou endogène est notifié), elles vont correspondre aux modifications à mettre en place pour adapter le programme de base, c'est à dire la mise en œuvre d'une reconfiguration Fractal. Notons que contrairement à AspectJ [KHH+01]) qui est un langage d'aspects généraliste, nos actions seront programmées dans un langage dédié à l'adaptation (FScript) afin de pouvoir offrir des garanties de cohérence.

Concernant le tissage, le choix de s'intéresser à l'adaptation dans des domaines ouverts et non figés (e.g., informatique mobile), nous amène à penser que le tissage de l'aspect d'adaptation doit

AOP	**Safran**
Programme de base	Composants Fractal reconfigurables
Aspects	Politiques d'adaptation réactives
Coupes	Événements endogènes et exogènes
Actions	Action de reconfiguration FScript
Tisseur	Contrôleur d'adaptation

Table 4.3 – Correspondance entre les concepts AOP et les mécanismes de Safran

pouvoir être dynamique. Si le programme de base était tissé statiquement avec l'aspect d'adaptation, le code métier deviendrait inutilisable dans des contextes qui n'ont pas été prévus a priori. Nous allons donc privilégier le tissage dynamique permettant un découplage spatio-temporel entre le code métier et l'adaptation.

L'aspect lui-même (i.e., couples (*coupe, action*)) est représenté par des politiques d'adaptation modulaires de Safran tissées dynamiquement grâce à l'interface de contrôle `AdaptationController`.

Résultat

Sur le plan conceptuel, nous avons montré que l'adaptation pouvait être considérée comme une préoccupation transverse et qu'il était possible de transposer les concepts classiques de la programmation par aspects (programme de base, coupes, actions et tisseur) dans ce cas particulier pour modéliser un *aspect d'adaptation* (cf. Tableau 4.3).

Sur le plan pratique, nous avons réalisé Safran, une extension du modèle Fractal qui concrétise cette approche et permet le développement modulaire de politiques d'adaptation réactives, qui sont tissées dynamiquement dans les composants Fractal d'une architecture logicielle [DL06].

4.6.2 Généricité de l'approche FPath/FScript

Motivation

« Vers FScript/FPath 3.0 ? *"Towards a versatile and extensible query/reconfiguration language for software architectures"* » : tel était le titre (franglais) de mon exposé dans le cadre d'un séminaire interne en juillet 2010 sur le bilan du « projet FScript ». Sans doute ce bilan arrivait-il un peu tard car j'étais parti depuis 1 an déjà explorer d'autres contrées scientifiques (cf. Chapitre 6). Mais une idée avait émergé avec les projets (INRIA ADT) Galaxy, (ANR Arpege) SelfXL ou le stage de master M2 de Mayleen Lacouture [Lac08] : il serait intéressant de concevoir des langages dédiés génériques pour la navigation et la reconfiguration de n'importe quel type d'architectures logicielles.

Retour vers le futur

A la fin des années 2000, dans notre communauté génie logiciel et middleware francophone, plusieurs travaux s'inspirent de FScript/FPath pour faciliter la reconfiguration d'autres plate-formes middleware. Il s'agit principalement de GCMScript pour GCM [BHN09, IRHBJ16] à Nice, FraSCAti Script pour FraSCAti [SMF+09, SMR+12] à Lille mais également de VMScript [PLM10] à Nantes pour administrer des infrastructures virtualisées.

Même si ces langages de script (GCMScript, FraSCAti Script, VMScript) manipulent tous des architectures différentes (respectivement composants répartis pour les grilles de calcul, intergiciel orienté services, hyperviseur Xen), ils ont plusieurs points communs : la sélection d'éléments d'architectures et leurs reconfigurations, la mise en place d'invariants architecturaux. De ce constat, il nous a semblé intéressant d'explorer l'expression de reconfigurations architecturales avec une abstraction de haut niveau, indépendante des plates-formes cibles, permettant aussi de poser des contraintes d'intégrité sur les modifications.

Notre idée est de proposer un processus d'industrialisation – basé sur une approche dirigée par les modèles [Sch06] – pour la réalisation de DSLs pour la navigation et la reconfiguration d'architectures logicielles. Un premier travail est mené avec le stage de master M2 de Mayleen Lacouture [Lac08] et un co-encadrement avec Frédéric Jouault (ex-équipe INRIA Atlas). Au-delà de son expertise en ingénierie des modèles, Frédéric a permis à Mayleen de prendre en main des outils comme ATL [JABK08]. Il en résulte une preuve de concept avec une navigation dans une représentation JMX [13] de code Java et la définition au niveau modèle d'invariants architecturaux et leur projection pour validation dans FScript.

Dans le cadre de l'ADT Galaxy, des discussions ont lieu alors avec Olivier Barais (équipe INRIA Triskell) pour reprendre ces premières expérimentations, proposer un méta-modèle commun entre FScript, GCMScript et FraSCAti Script en utilisant Kermeta [14]. L'objectif étant la génération automatique des différentes instances de FPath/FScript. Malheureusement, par manque de temps, de priorité, l'idée du projet est abandonnée en 2010.

4.6.3 Bilan

Ce chapitre avait pour objectif de répondre à un certain nombre de défis identifiés dans la Section 3.4. Nos contributions ont principalement porté sur la proposition d'un framework (Safran) pour le développement d'applications auto-adaptables, *context-aware* et la proposition d'un langage dédié (FScript) pour la reconfiguration d'architectures Fractal. Au regard des critères exposés dans la Section 3.4, le modèle de composants Fractal apporte les caractéristiques de réification, modularité, composabilité, support à l'adaptation non-anticipée qui constituent un pré-requis à l'adaptation dynamique. Safran et FScript permettent la modélisation et la programmation de l'adaptation comme une préoccupation (*concern*) transverse. FScript apporte de plus une première réponse pour la fiabilité des reconfigurations. WildCAT propose un environnement pour l'adaptation au contexte.

Nos contributions ont été reconnues puisque Safran et FScript sont régulièrement cités par la communauté plus de 10 ans après leur conception. WildCAT a connu un succès qui a dépassé nos attentes (cité plus de 100 fois alors qu'il a été publié dans un petit *workshop*).

Pour en revenir à FScript, même si ce dernier offre des garanties sur un programme par construction (i.e., grâce au pouvoir d'expression du DSL), la fiabilité du langage reste perfectible. D'une part, son support d'exécution ne permet pas de reconfigurations transactionnelles. Une ébauche a été proposée dans la thèse de Pierre-Charles David [Dav05] mais il reste du travail pour arriver à des reconfigurations fiables, dynamiques, concurrentes, réparties et non anticipées. D'autre part, des analyses (statiques) peuvent être réalisées en amont de l'exécution. Soit en proposant un système de type statique à FScript, soit en proposant la vérification de modèles (*model checking*) (e.g., conformité de la nouvelle configuration à la spécification d'un modèle d'architecture). L'un des avantages majeurs des analyses en amont de l'exécution est d'éliminer les temps d'indisponibilité [OMT08] de l'architecture à reconfigurer qui est – rappelons le – l'un des atouts majeurs de l'adaptation dynamique.

13. `https://fr.wikipedia.org/wiki/JMX`
14. `http://www.kermeta.org/index_k1_k2.html`

Chapitre 5

Fiabilité des reconfigurations dynamiques dans les architectures Fractal

Dans ce chapitre, nous présentons nos principaux travaux ayant trait à la problématique de la fiabilité (*reliability*) dans les reconfigurations dynamiques. Ces travaux ont été menés dans le cadre de la thèse de Marc Léger [Lég09] et du projet RNTL Selfware (2005-2008). Avant d'exposer les résultats de ces travaux, nous introduisons les pré-requis nécessaires pour la bonne compréhension de ce chapitre.

5.1 Prérequis (a.k.a. interfaces requises pour comprendre ce chapitre)

Nous avons volontairement limité ces pré-requis à deux grands thèmes. Le lecteur averti pourra sauter la lecture de cette section !

5.1.1 Sûreté de fonctionnement et transactions

La fiabilité est un attribut de la sûreté de fonctionnement dont la garantie suppose la mise en œuvre de méthodes telles que la tolérance aux fautes [ALRL04]. Les transactions sont un moyen de rendre les systèmes tolérants aux fautes par des mécanismes de reprise sur défaillances.

Sûreté de fonctionnement

Laprie et al. proposent dans un article de référence [ALRL04] une taxonomie de la sûreté de fonctionnement (cf. Figure 5.1). La sûreté de fonctionnement (*dependability*) pour un système peut se définir comme la propriété qui permet à ses utilisateurs de placer une confiance justifiée dans le service qu'il leur délivre.

Les différents attributs liés à la sûreté de fonctionnement sont les suivants :

- disponibilité (*availability*) : le fait d'être prêt pour rendre un service correct [1] ;
- fiabilité (*reliability*) : le fait de délivrer en continue un service correct ;
- sécurité (*safety*) : l'absence de conséquences catastrophiques sur l'utilisateur et l'environnement ;
- confidentialité (*confidentiality*) : l'absence de divulgations non autorisées de l'information ;
- intégrité (*integrity*) : l'absence d'altérations incorrectes du système ;
- maintenabilité (*maintainability*) : l'aptitude aux réparations et aux évolutions.

1. Un service correct est délivré par un système lorsqu'il accomplit sa fonction [ALRL04].

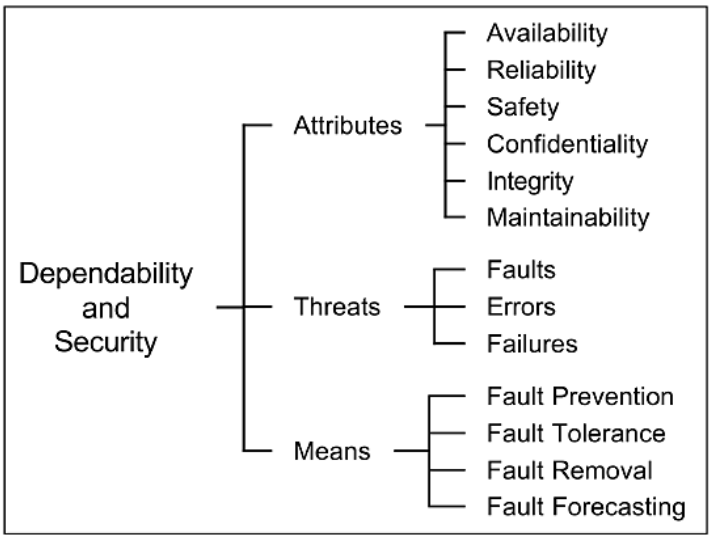


FIGURE 5.1 – Taxonomie de la sûreté de fonctionnement [ALRL04]

Il convient de relier la sûreté de fonctionnement au concept de menaces (*threats*) qui regroupe les défaillances, les fautes et les erreurs. En effet, garantir la fiabilité dans un système consiste à éviter des défaillances du service. Il existe un lien de causalité entre défaillances, erreurs et fautes [ALRL04] :

- une défaillance (*failure*) est un événement qui survient lorsque le service délivré dévie du service correct ;
- une erreur (*error*) caractérise la partie de l'état du système susceptible d'entraîner une défaillance ;
- une faute (*fault*) est la cause adjugée ou supposée d'une erreur.

Dans le contexte des reconfigurations dynamiques, les fautes considérées sont, d'une part, les fautes logicielles générées par l'exécution d'une reconfiguration dans un système donné, et d'autre part, les fautes matérielles externes aux reconfigurations.

Le développement d'un système sûr de fonctionnement passe par l'utilisation combinée d'un ensemble de méthodes qui peuvent être classées ainsi :

- la prévention des fautes (*fault prevention*) pour empêcher l'occurrence ou l'introduction de fautes ;
- la tolérance aux fautes (*fault tolerance*) pour fournir un service à même de remplir la fonction d'un système en dépit des fautes ;
- l'élimination des fautes (*fault removal*) pour réduire la présence des fautes ;
- la prévision des fautes ((*fault forecasting*) pour estimer la présence, la création et les conséquences des fautes.

Malgré des techniques de prévention de fautes dans les phases de conception du logiciel, des erreurs peuvent toujours survenir dans un système pendant son exécution. Nous nous intéressons plus particulièrement aux systèmes dits « tolérants aux fautes » qui désignent des systèmes capables de continuer à fonctionner normalement en cas d'occurrence de certaines fautes, la gestion des erreurs étant alors réalisée de manière transparente pour l'utilisateur.

Transactions

Les transactions ont été originellement utilisées dans les systèmes de gestion de base de données [TGGL82]. La notion de transaction s'est par la suite généralisée dans tous les systèmes informa-

tiques où il existe le besoin de maintenir la cohérence des informations malgré la présence d'accès concurrents et l'occurrence de défaillances. Les transactions sont donc un moyen de rendre des systèmes tolérants aux fautes.

Propriétés ACID. Une transaction consiste à effectuer une opération cohérente composée de plusieurs tâches unitaires. L'opération ne sera valide que si toutes les tâches unitaires sont effectuées correctement, on parle alors de validation (*commit*). Dans le cas contraire, l'ensemble des données traitées lors de l'opération retournent à leur état initial, la transaction est abandonnée (*rollback*). Le concept de transaction s'appuie sur la notion de point de synchronisation qui représente un état stable du système considéré, en particulier de ses données.

Les quatre propriétés fondamentales des transactions – les « propriétés ACID »[GR92] – sont définies de la façon suivante :

- *Atomicité* (A). Soit la transaction se termine et est validée, soit elle est abandonnée (sémantique du « tout ou rien ») ;
- *Cohérence* (C). Une transaction préserve la cohérence des objets modifiés ;
- *Isolation* (I). Les effets d'une transaction non validée ne sont pas visibles des autres transactions en cours d'exécution ;
- *Durabilité* (D). Les effets d'une transaction validée sont permanents.

Pour mettre en œuvre ces propriétés, il sera nécessaire de définir un protocole de validation atomique répartie (e.g., Two-Phase-Commit protocol [TGGL82]) avec contrôle de concurrence [BHG87] et mécanisme de reprise arrière (*rollback*).

5.1.2 Logique du premier ordre et Alloy

Le processus d'adaptation ne doit jamais faire migrer l'architecture vers une configuration qui n'est pas *valide*, i.e., cette nouvelle configuration doit être conforme à la spécification d'un modèle d'architecture. Aussi pour préserver l'intégrité du système et garantir la cohérence des (re)configurations, il est primordial de spécifier les modèles d'architecture.

Alloy [Jac02] est un langage déclaratif de spécifications – basé sur la logique de premier ordre – pour exprimer des contraintes structurelles complexes et un comportement dans un système logiciel.

Logique du premier ordre

> « *Tous les hommes sont mortels ; or Socrate est un homme ; donc Socrate est mortel.* »
> (Aristote)

Ce célèbre syllogisme est formalisée grâce à la logique du premier ordre (ou calcul des prédicats). Cette logique est caractérisée par l'utilisation de :

- variables comme x, y, ... pour dénoter des éléments d'un ensemble ;
- prédicats (ou relations) sur les éléments ;
- connecteurs logiques (et, ou, $\Rightarrow$, ...) ;
- deux quantificateurs, l'un « universel » (noté $\forall$) et l'autre « existentiel » (noté $\exists$).

Un prédicat décrit une propriété (vraie ou fausse) sur le système à formaliser. On va alors utiliser des systèmes de déduction qui permettent de déduire qu'un prédicat (e.g., « Socrate est mortel ») est une conséquence logique d'un ensemble de prédicats (e.g., « Tous les hommes sont mortels », « Socrate est un homme »).

La logique du premier ordre comme formalisme de spécification de modèles d'architecture a déjà montré son intérêt [GMW00] pour spécifier les relations architecturales et les contraintes d'intégrité sur les configurations.

Pour le modèle Fractal par exemple, on pourra définir un ensemble de variables (e.g., composant, interface, attribut, ...) et des relations sur ces éléments à travers de prédicats (e.g., toute interface est associée à un composant unique, seuls les composants composites peuvent contenir des sous-composants, ...), ce qui permettra de formaliser des contraintes d'intégrité structurelles.

Alloy

Alloy [Jac02] est un langage de spécification qui présente l'avantage d'être relativement simple, déclaratif et d'utiliser la logique du première ordre et la théorie des ensembles. De plus, contrairement à d'autre langages de spécifications comme UML/OCL [2], Alloy fournit un analyseur basé sur un solveur de contraintes de type SAT (« SATisfiability » en anglais) pour vérifier automatiquement la cohérence des spécifications, notamment la non contradiction entre les contraintes sur le modèle. L'analyseur de Alloy agit comme un *model checker* en générant des exemples et des contre-exemples d'instances du modèle si ils existent.

Pour revenir sur notre exemple en Fractal, les éléments architecturaux et contraintes d'intégrité sont traduits en construction du langage Alloy (ce qui est relativement direct puisque Alloy est basé sur la logique du premier ordre) puis vérifiés par son analyseur qui valide la spécification donnée en entrée ou bien signale les incohérences, notamment les contradictions entre contraintes d'intégrité.

5.2 Approche multi-étapes pour des reconfigurations dynamiques fiables

5.2.1 Motivation et problématique

Comme indiqué dans la Section 4.4, le pouvoir d'expression limité du langage FScript permet d'offrir un certain nombre de garanties (e.g., terminaison des actions en temps fini). Cependant, FScript ne traite pas un certain nombre de problèmes comme des scripts syntaxiquement corrects mais sémantiquement « saugrenus » ou encore les fautes matérielles pouvant se produire pendant une reconfiguration.

Préserver la reconfiguration des défaillances logicielles. Pour illustrer nos propos, prenons l'exemple du script suivant :

```
1  action semantically-invalid(c1, c2) {
2    add($c1, $c2);
3    add($c2, $c1);
4  }
```

Listing 5.1 – Action introduisant un cycle en FScript.

Cette action va engendrer un cycle dans l'architecture Fractal puisque le fils $c2$ d'un composant $c1$ va ensuite devenir le père de son père (cf. Listing 5.1, ligne 3). Avec la première version de FScript, il était impossible de détecter ce type d'anomalie liée à un manque de formalisation des concepts de Fractal. Spécifier les relations architecturales et les contraintes d'intégrité sur les configurations Fractal permettrait de réaliser des analyses statiques et améliorerait *de facto* la puissance du langage FScript.

Préserver la reconfiguration des fautes matérielles. Certains types d'erreurs sont soit impossibles à prévoir (e.g., fautes matérielles) ou trop coûteux à détecter. Afin de garantir des reconfigurations fiables, nous proposons un modèle et un support d'exécution pour la tolérance aux pannes.

2. http://www.omg.org/spec/OCL

Plus précisément, nous suggérons une approche transactionnelle dans laquelle une reconfiguration est considérée comme une transaction prenant en charge la récupération des erreurs, la cohérence de l'architecture et la simultanéité des reconfigurations distribuées.

5.2.2 Chaîne de validation des reconfigurations

Dans [DLG+08], nous avons proposé une démarche pour maximiser la continuité de service d'un logiciel sujet à une reconfiguration dynamique. Cette démarche nous permet d'apporter une réponse à la fois au problème de fiabilité et au problème de performance soulevés dans la Section 3.3. Notre approche est basée sur la mise en place d'une chaîne de validation (cf. Figure 5.2).

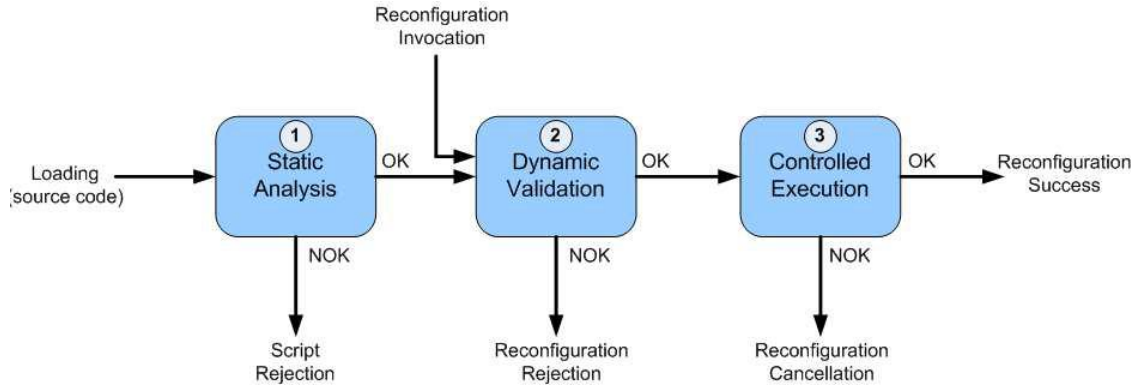


FIGURE 5.2 – Une chaîne de validation pour la reconfiguration

Cette chaîne est basée sur une décomposition du cycle de vie des reconfigurations en plusieurs étapes, de leur définition à leur exécution réelle sur le système cible. Comme les scripts de reconfiguration traversent ces étapes successives, différentes techniques sont utilisées pour « éliminer » les reconfigurations incorrectes et gérer les erreurs qui ne peuvent être évitées (e.g., panne). Les différentes étapes de la chaîne de validation se complètent pour offrir des garanties de fiabilité élevées. Dans le même temps, la chaîne reste modulaire et peut être personnalisée pour supporter différents compromis entre performances et garanties en fonction du domaine.

Dans le reste de cette section, nous présentons l'architecture globale de notre approche. Les étapes (1) et (2) vont consister à prévenir les fautes alors que l'étape (3) va tolérer les fautes.

Prévention des fautes - Etape 1

L'étape (1) de la chaîne de validation charge le code source de reconfiguration FScript dans la chaîne. Son objectif est de vérifier la validité de la reconfiguration par rapport au modèle d'architecture sous-jacent (i.e., Fractal). Le modèle de composant va définir certaines règles à satisfaire pour respecter les relations architecturales et les contraintes d'intégrité sur les configurations Fractal (cf. Section 5.3). À ce stade, les architectures réelles auxquelles le script sera appliqué sont inconnues.

L'analyse statique mis en œuvre pour détecter des scripts de reconfiguration problématiques est basée sur la logique de Hoare [Hoa69]. Cette analyse est paramétrée par un *profil* de règles du modèle d'architecture : cela permet de supporter facilement des variantes, au niveau infrastructure ou application, comme différents styles architecturaux (cf. Figure 5.3 (a)).

L'analyseur définit des formules de Hoare $\{P\}S\{Q\}$ où S est le script de reconfiguration et P et Q sont des prédicats de l'architecture à reconfigurer (exprimées en logique du premier ordre). Une

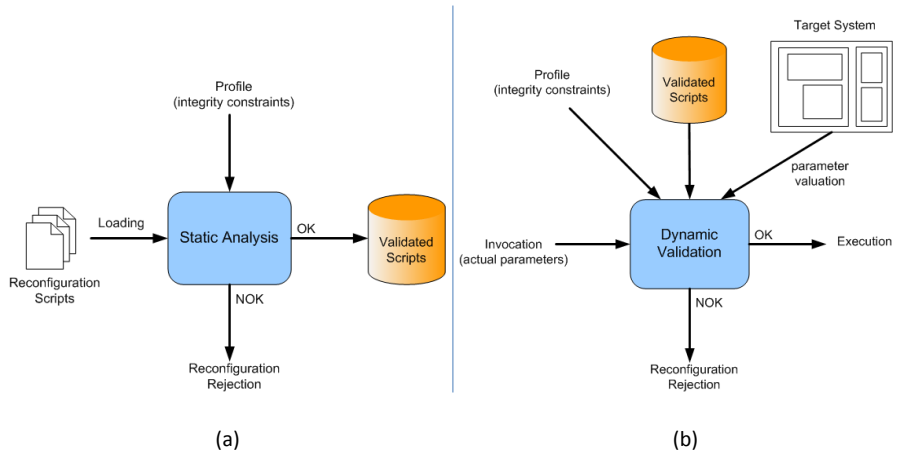


FIGURE 5.3 – Prévention des fautes : (a) Analyse statique (b) Validation dynamique

telle formule signifie que toute architecture satisfaisant la pré-condition P satisfera la post-condition Q après l'achèvement du script S. L'objectif de l'analyse sémantique est de déterminer une pré-condition P qui ne conduit pas à un état d'erreur dans lequel l'architecture viole un invariant : seules les architectures qui satisfont à la pré-condition P seront reconfigurées par le script. Par conséquent, si la pré-condition P est fausse, cela signifie que le script n'est pas applicable selon l'analyse et devra être rejeté. Sinon, le script est considéré comme potentiellement valide et passe à l'étape suivante de la chaîne de validation.

Prévention des fautes - Etape 2

L'étape (2) de la chaîne de validation est déclenchée chaque fois que l'utilisateur demande l'invocation d'un script de reconfiguration en donnant une architecture cible et des paramètres réels (cf. Figure 5.3 (b)). Cette étape exécute des validations supplémentaires grâce à la connaissance de ces informations, mais sans modifier en fait l'architecture cible puisqu'elle va travailler sur une architecture virtuelle (une image, un clone) de l'architecture cible.

Cette étape est nécessaire car à ce moment-là du processus de reconfiguration, nous avons des informations supplémentaires qui permettent d'écarter de nouveaux scripts. Pour s'en convaincre, prenons l'exemple suivant :

```
1  action conditionally-valid(c1, c2) {
2      add($c1, $c2);
3      bind($c1/internal-interface::foo, $c2/interface::bar);
```

Listing 5.2 – Action potentiellement correcte en FScript.

Cette action est sémantiquement correcte si et seulement si :
- $c1$ est un composant composite ;
- $c1$ possède une interface interne foo , $c2$ une interface bar ;
- foo et bar sont de type compatible.

Or, avec FScript, ces informations ne sont connues qu'à l'exécution.

Plus concrètement, nous proposons deux sous-étapes dans cette étape (2) de validation dynamique. La première consiste à évaluer les pré-conditions sur l'architecture cible et les paramètres réels. Cela peut se faire facilement, et il suffit d'introspecter l'architecture cible, sans la modifier (cf. exemple ci-dessus). Puis, si la vérification de compatibilité a réussi, une deuxième étape consiste à

simuler l'exécution du script sur l'architecture cible. Cette étape utilise une implémentation virtuelle (un clone) de l'architecture cible, sur laquelle le script de reconfiguration est exécuté à l'aide de FScript. L'architecture virtuelle est initialisée avec l'état initial du système cible, mais implémente la sémantique "copy-on-write" : les opérations sont appliquées à la copie virtuelle et ne modifient pas le système cible réel. Si l'un des invariants du modèle architectural est violée pendant la simulation, la reconfiguration est rejetée.

Un avantage de la simulation est que cette étape peut être plus précise (et peut donc attraper plus d'erreurs) que les analyses statiques de l'étape (1), qui peuvent être restreintes dues à une sélection des invariants à préserver. De plus, en instrumentant l'architecture virtuelle à reconfigurer, la simulation génère la trace exacte de la reconfiguration effectuée par le script, qui peut être « rejoué » avec très peu d'*overhead* pour reproduire son effet sur le système cible réel [PMSD07]. Le seul inconvénient est que cette étape peut augmenter la latence de la reconfiguration elle-même. Cependant, il faut relativiser ce surcoût en le comparant à un *rollback* nécessaire pendant la phase d'exécution liée à la violation des contraintes d'intégrité (voir ci-dessous).

Tolérance aux fautes - Etape 3

L'étape (3) de la chaîne est l'application réelle du script de reconfiguration sur l'architecture cible. Selon la configuration de l'étape précédente, elle utilise soit une forme compilée du script, soit la trace spécialisée des opérations de reconfiguration générées lors de la simulation.

Étant donné que l'objectif global de la chaîne de validation est de garantir la fiabilité de la reconfiguration, cette étape doit (i) soit appliquer le script de reconfiguration complet sans erreurs ; (ii) soit, en cas d'erreurs, restaurer le système au dernier état cohérent avant l'exécution du script grâce à un mécanisme de reprise arrière (*rollback*). Les défaillances qui se produisent lors de l'exécution réelle de la reconfiguration incluent les défaillances du logiciel (e.g., la violation du modèle d'architecture) qui n'ont pas été détectées plus tôt et certaines erreurs qui sont fondamentalement impossibles à prédire (e.g., pannes matérielles). Dans tous les cas, l'architecture résultante doit être dans un état cohérent selon la définition du modèle d'architecture sous-jacent.

Ces objectifs exigent l'utilisation de techniques de gestion des transactions, comme ils correspondent fortement aux propriétés ACID standard des transactions dans l'informatique distribuée [TGGL82]. Pour exécuter des scripts de reconfiguration à l'intérieur de transactions globales avec démarcation automatique, nous utilisons une version étendue de Fractal (cf. Section 5.4), qui fournit des sémantiques transactionnelles pour les architectures Fractal. Par conséquent, les reconfigurations peuvent bénéficier des propriétés ACID pour prendre en charge la simultanéité des reconfigurations, la tolérance aux fautes et garantir ainsi la cohérence du système.

5.3 Spécification des (re)configurations Fractal

Un modèle de composants fournit certaines règles d'assemblage pour les systèmes logiciels avec des propriétés pour leur structuration et leur modularité [OMT08]. Le modèle Fractal est essentiellement défini par une spécification textuelle informelle et par des API bas niveau écrites avec un langage de définition d'interface [BCL+04]. Aussi, nous proposons de définir une représentation plus formelle des configurations Fractal à partir du méta-modèle orienté graphe présenté en Section 4.4.2.

Dans un premier temps, nous nous attachons à spécifier une configuration Fractal correcte. Puis, nous proposons de vérifier la validité de la reconfiguration – par rapport au modèle d'architecture sous-jacent – à partir d'une configuration initiale.

5.3.1 Spécification des configurations

Vers un modèle formel pour Fractal

Une architecture Fractal est modélisée par une configuration statique d'éléments architecturaux appartenant au modèle de composant. Nous représentons une configuration Fractal comme un multi-graphe orienté et étiqueté $\mathscr{A} = (E, R)$ où l'ensemble des nœuds E représente les éléments architecturaux dans la configuration et l'ensemble des arcs R les relations entre ces éléments. Le modèle de composant Fractal est alors défini par le triplet $\mathscr{F} = (\mathscr{F}_E, \mathscr{F}_R, \mathscr{F}_P)$ où :

- $\mathscr{F}_E$ est l'ensemble des éléments architecturaux appartenant au modèle de composants ;
- $\mathscr{F}_R$ est l'ensemble des relations possibles entre éléments architecturaux ;
- $\mathscr{F}_P$ est l'ensemble des *contraintes d'intégrité* en tant que prédicats binaires $\pi(E, R)$ qui doivent être satisfaits par chaque configuration.

Ainsi, une configuration $\mathscr{A}$ est dite *conforme* au modèle Fractal $\mathscr{F}$ si et seulement si :

$$\begin{array}{rcl} E & \subseteq & \mathscr{F}_E \\ R & \subseteq & \mathscr{F}_R \\ \forall \pi & \in & \mathscr{F}_P, \pi(E, R) \end{array}$$

Nous utilisons la logique du premier ordre comme formalisme de spécification du modèle, i.e., pour spécifier les relations architecturales et les contraintes d'intégrité sur les configurations. Ce formalisme logique présente l'avantage d'être agnostique du point de vue du langage d'implémentation. Nous considérons ainsi l'utilisation des opérateurs logiques (booléens) classiques $\wedge$, $\vee$, $\neg$, et l'implication $\Rightarrow$, les quantificateurs universel $\forall$ et existentiel $\exists$. Les opérateurs ensemblistes utilisés sont les suivants : l'union $\cup$, l'intersection $\cap$, la différence $\setminus$ et l'égalité $=$. Le modèle de graphes de configuration est défini dans la suite avec ce formalisme.

Éléments architecturaux et leurs propriétés. Les éléments architecturaux réifiés dans le modèle Fractal sont les nœuds du graphe de configuration et sont au nombre de trois : les composants, les interfaces et les attributs (cf. Section 4.4.2). A chaque nœud du graphe peut être associé un certain nombre de propriétés [3] :

- $Component = Id \times Name \times State$
- $Interface = Id \times Name \times Signature \times Visibility \times Role \times Cardinality \times Contingency$
- $Attribute = Id \times Name \times Type \times Value$

Nous avons ainsi l'ensemble des éléments architecturaux du modèle Fractal :
$\mathscr{F}_E = Component \cup Interface \cup Attribute$

Les relations architecturales. Les relations entre les éléments architecturaux dans le modèle Fractal sont des relations binaires du type $x\mathscr{R}y$ où $(x, y) \in E^2$ et correspondent aux arcs dans le graphe de configuration. Les relations pourraient être généralisées à des relations n-aires mais il s'avère que dans le modèle de base, seules des relations binaires sont utilisées. Nous considérons les quatre relations primitives suivantes dans Fractal :

- *hasInterface* (définie sur $Component \times Interface$) : détermine si un composant possède une interface donnée (interne ou externe) ;
- *hasAttribute* (définie sur $Component \times Attribute$) : détermine si un composant possède un attribut donné ;

3. Pour plus de détails, voir [Lég09].

- *hasChild* (définie sur $Component \times Component$) : détermine si un composant est sous-composant direct d'un autre composant. Cette relation est non réflexive car un composant ne peut être son sous-composant direct. Elle n'est pas symétrique, le premier composant doit être le composant parent, le second le composant fils. Elle n'est pas transitive car on ne considère que les sous-composants directs ;
- *hasBinding* (définie sur $Interface \times Interface$) : détermine si une interface est liée directement à une autre interface. Cette relation est non réflexive car une interface ne peut être liée à elle-même. Elle est symétrique car elle ne fait pas de distinction entre les rôles des interfaces (*server* ou *client*). Elle n'est pas transitive car on ne considère que les liaisons directes.

Nous avons ainsi l'ensemble des relations primitives définies dans le modèle :
$\mathscr{F}_R = hasInterface \cup hasAttribute \cup hasChild \cup hasBinding$

Les relations primitives peuvent ensuite être composées pour construire de nouvelles relations. Par exemple, la fermeture transitive *hasChild*, écrite $hasChild^{trans}$, détermine si un composant est un sous-composant direct ou indirect d'un autre composant dans la hiérarchie des composants. Enfin, toutes les relations peuvent être inversées : nous pouvons, par exemple, définir une nouvelle relation *hasParent* qui est égal à $hasChild^{-1}$.

Dernier exemple : pour déterminer si deux composants sont au même niveau hiérarchique – ce qui peut être extrêmement important pour garantir une contrainte d'intégrité – nous définissons alors le prédicat suivant [4] :

$$siblings(c_1, c_2) = \exists c_3 \in E,\ hasChild(c_3, c_1) \wedge hasChild(c_3, c_2) \tag{5.1}$$

Spécification du modèle par des contraintes d'intégrité

En plus de la définition des éléments architecturaux, de leurs propriétés et des relations architecturales, nous utilisons des contraintes d'intégrité pour compléter la spécification du modèle Fractal.

Contraintes d'intégrité du modèle. Les contraintes d'intégrité sont essentiellement des invariants structurels sur les architectures de composants afin de composer un style architectural [AAG93]. Plus précisément, ce sont des prédicats binaires sur la validité d'une configuration, c'est-à-dire sur des éléments architecturaux, leurs propriétés et leurs relations. Ces contraintes sont spécifiées en utilisant la logique du premier ordre sur les relations primitives déjà définies. Nous nous intéressons d'abord aux contraintes au niveau du modèle de composant Fractal, c'est-à-dire des contraintes communes à toutes les configurations Fractal. Une contrainte d'intégrité typique dans les modèles de composants hiérarchiques est d'interdire les cycles dans la hiérarchie des composants afin d'éviter une récurrence infinie (cf. Listing 5.1). Celle-ci s'écrira ainsi :

$$noCycle(E, R) = \forall c \in E,\ \neg hasChild^{trans}(c, c) \tag{5.2}$$

Nous avons identifié l'ensemble $\mathscr{F}_P$ composé de quinze contraintes d'intégrité dans la spécification Fractal pour extraire un style architectural minimaliste qui caractérise « le » modèle de composant Fractal. L'identification des composants composites, le type de compatibilité entre les interfaces de liaison, la compatibilité des rôles entre les interfaces de liaison, ... sont des exemples de ces contraintes d'intégrité (pour plus de détails, voir [Lég09]).

4. Nous préférons utiliser une notation fonctionnelle pour représenter les relations, i.e. $hasChild(x, y)$ plutôt que la forme *x hasChild y*.

Étendre les styles architecturaux avec des contraintes d'intégrité. Une fois le noyau du modèle de composant Fractal spécifié dans notre formalisme, nous pouvons facilement étendre son style architectural en ajoutant d'autres contraintes d'intégrité. Le modèle de contrainte proposé est divisé en trois niveaux d'abstraction (cf. Figure 5.4) :

- niveau modèle : correspond au modèle de composant Fractal $\mathscr{F}_P$;
- niveau profil : extension du modèle Fractal avec de nouvelles contraintes d'intégrité communes à un ensemble d'applications partageant un style architectural commun ;
- niveau applicatif : ajoute des contraintes spécifiques à une configuration d'application donnée.

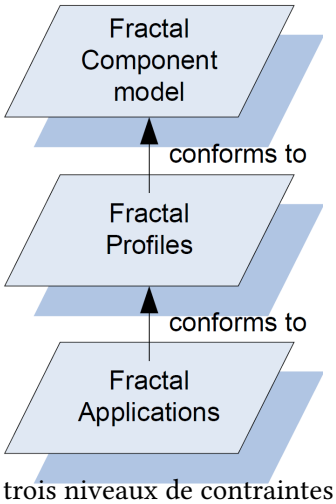


FIGURE 5.4 – Les trois niveaux de contraintes d'intégrité [Lég09].

Le niveau du profil doit être conforme au niveau du modèle, c'est-à-dire que les contraintes de profil ne doivent pas être en contradiction avec les contraintes du modèle, et le niveau applicatif doit se conformer à un profil choisi. En outre, chaque niveau de contraintes doit être cohérent avec les autres afin d'éviter des configurations sur-contraintes. Les contraintes au niveau du modèle sont immuables, elles ne peuvent ni être modifiées ni éliminées, alors que les contraintes de profil et applicatives peuvent être ajoutées ou supprimées au besoin. Cette séparation en plusieurs niveaux de contraintes autorise ainsi plus de flexibilité pour la définition de la cohérence d'une application à base de composants Fractal.

Par exemple, imaginons un profil qui voudrait ajouter une contrainte pour interdire le partage de composants entre les composants composites. Le prédicat serait défini ainsi :

$$\begin{aligned} noSharing(E,R) &= \forall (c_1,c_2,c_3) \in E^3,\ hasChild(c_2,c_1) \\ &\Rightarrow \neg hasChild(c_3,c_1) \end{aligned} \tag{5.3}$$

Les contraintes niveau applicatif sont spécifiées de manière similaire, mais elles s'appliquent uniquement à des éléments spécifiques dans une configuration donnée. Chaque élément architectural d'une configuration possédant un identifiant unique, la contrainte ci-dessus appliquée sur une instance bien précise de composant désignée par son identifiant *id* devient :

$$\begin{aligned} noSharing(E,R,id) &= \exists c \in E,\ id(c) = id \\ &\wedge \forall (c_1,c_2) \in E^2,\ hasChild(c_2,c) \\ &\Rightarrow \neg hasChild(c_3,c) \end{aligned} \tag{5.4}$$

La contrainte interdira le partage uniquement pour cette instance et pas pour les autres composants de l'application.

5.3.2 Vérification de la cohérence du modèle et des configurations

Notre objectif est de pouvoir vérifier la validité d'une configuration Fractal. Mais, tout d'abord, nous devons vérifier la cohérence globale de toutes les contraintes définies dans notre modèle de contraintes : les niveaux modèle, profil et applicatif. Nous proposons de traduire les contraintes décrites dans la logique du premier ordre dans Alloy [Jac02], un langage de spécification qui peut détecter automatiquement les incohérences entre les contraintes. Ensuite, une fois que la cohérence globale de notre modèle de contrainte a été vérifiée, il est nécessaire de traduire les contraintes d'intégrité dans un langage exécutable qui peut être facilement intégré à Fractal. Nous avons choisi FPath/FScript (cf. Section 4.4), particulièrement adapté pour spécifier des contraintes d'intégrité dans Fractal. L'ensemble du processus est illustré dans la Figure 5.5.

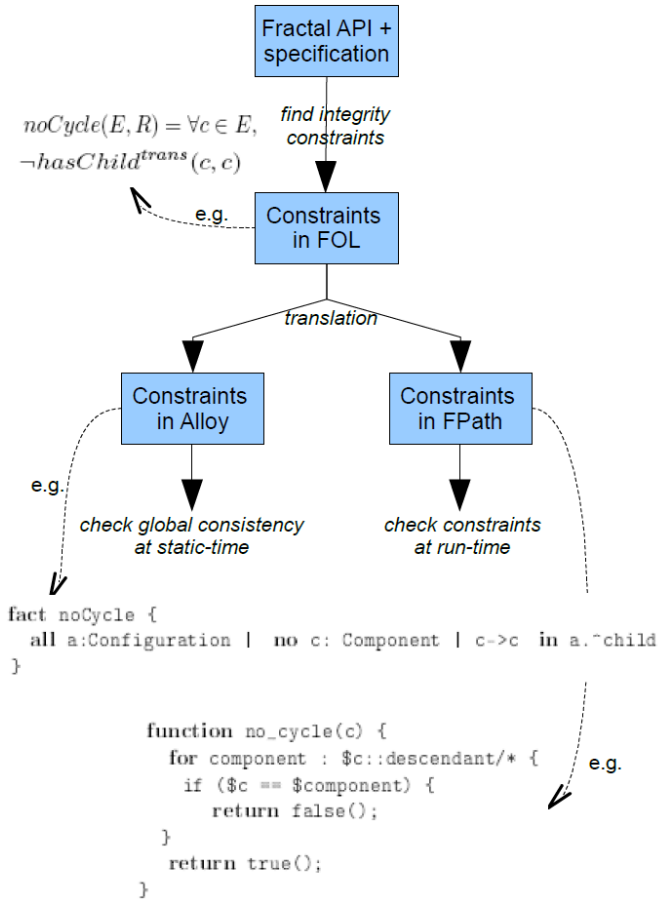


FIGURE 5.5 – Processus de vérification des contraintes.

Vérification de la cohérence globale

Alloy est un langage déclaratif basé sur la logique de premier ordre et le calcul de relation. Alloy propose un analyseur s'appuyant sur un solveur de contraintes SAT pour vérifier que les spécifications ne sont pas sur-contraintes en essayant de générer des instances du modèle. La spécification du modèle Fractal est divisée en deux modules principaux : le premier spécifie les éléments $\mathscr{F}_E$ et relations architecturales $\mathscr{F}_R$, le second spécifie les contraintes d'intégrité $\mathscr{F}_P$.

Alloy manipule des faits ($fact$) et des assertions ($assert$). Les faits sont des contraintes supposées toujours vraies dans la spécification. Ces invariants permettent ainsi de restreindre les instances possibles du modèle : lorsque l'analyseur cherche des exemples d'instance du modèle, il supprime tous ceux qui violent les faits. Les assertions servent à vérifier des propriétés sur le modèle, lorsque

l'assertion est fausse, un contre-exemple est généré par l'analyseur. Nous représentons nos contraintes d'intégrité uniquement avec des faits car elles font partie de la spécification. Par contre, les assertions permettent un développement incrémental de la spécification, avant d'être intégrées au modèle comme des faits, les contraintes peuvent en effet être traitées comme de simples assertions.

Un exemple de fait est la contrainte *noCycle* correspondant au prédicat (5.2) :

```
1 fact noCycle {
2   all a:Configuration | no c: Component | c->c  in ^(a.child)
3 }
```

Listing 5.3 – *noCycle* en Alloy.

L'opérateur ^ désigne ici la fermeture transitive de la relation *child*. La spécification de l'ensemble des contraintes d'intégrité est donnée en annexe dans la thèse de Marc Léger [Lég09].

La définition d'un profil se fait dans un nouveau module qui importe le module Fractal principal de définition des contraintes pour l'étendre avec de nouvelles contraintes sous forme de faits. Un profil doit être évidemment conforme au modèle, cette conformité est vérifiée en passant le module décrivant le profil dans l'analyseur Alloy. Par exemple, un profil qui interdirait le partage de composant (i.e., prédicat (5.3)) se définit ainsi :

```
1 fact noSharing {
2   all a: Configuration | all c1, c2, c3: Component | (c1->c3 in a.child)
3     => ! (c2->c3 in a.child)
4 }
```

Listing 5.4 – *noSharing* en Alloy.

Notons que les contraintes applicatives ne peuvent pas être intégrées dans la spécification du modèle en Alloy. En effet, ces contraintes s'appliquent à des instances d'éléments dans une configuration particulière et leur spécification en Alloy nécessiterait une modélisation au niveau instance des configurations ce qui est trop coûteux.

Vérification des contraintes dans les configurations Fractal

Le langage FPath/FScript est particulièrement adapté pour spécifier des contraintes d'intégrité dans les applications basées sur les composants Fractal programmés en Java grâce à son intégration dans ce langage. Les règles de traduction des contraintes exprimées en logique du premier ordre dans le langage FPath/FScript sont présentées en détail dans [Lég09].

Les exemples ci-dessous illustrent la mise en œuvre de la contrainte *noCycle* du modèle Fractal et la contrainte *noSharing* du profil décrit ci-dessus. Elles sont exprimées à l'aide de fonctions FScript sans effet de bord :

```
1 function no_cycle(c) {
2   for component : $c/descendant::* {
3     if ($c == $component) {
4       return false(); }
5   return true();
6 }
7
8 function no_sharing(c) {
9   return size($c/parent::*) <= 1;
10 }
```

Listing 5.5 – Contraintes exprimées en FScript.

La contrainte *noCycle* (resp. *noSharing*) est appelée par l'expression FPath suivante :

```
no_cycle(.);      // resp. no_sharing(.);
```

La syntaxe utilise le caractère '.' désignant en FPath l'élément courant. Ces fonctions s'appliqueront récursivement à tous les composants d'une configuration.

Concernant les contraintes applicatives, cette dernière est associée à une instance particulière d'un composant dans la configuration. Les contraintes sont donc spécifiées dans la définition ADL de chaque composant (cf. Section 4.1.1). La forme de la contrainte est alors exactement la même que les contraintes d'un profil hormis le caractère . qui désigne alors l'élément auquel est associé la contrainte uniquement, i.e., le composant dont la définition ADL « encapsule »la définition de la contrainte. La contrainte *noSharing* spécifique à une instance se traduit donc comme pour un profil, seule l'interprétation du caractère '.' est différente :

```
<component name="server">
  ...
  <constraint value="no_sharing(.);" />
</component>
```

5.3.3 Spécification des reconfigurations

Pour favoriser l'évolution des systèmes, certains modèles de composants comme Fractal fournissent des capacités de reconfiguration dynamique afin de modifier les systèmes lors de leur exécution. La reconfiguration dynamique est alors basée sur des adaptations architecturales, i.e., les techniques de mutation, techniques d'habillage et paramétrisation possibles sur la *brique* architecture (cf. Section 2.4).

Nous proposons d'étendre le concept de contraintes d'intégrité pour les appliquer aux architectures dynamiques. De nouveaux invariants seront définis et ne portent donc plus uniquement sur des configurations statiques mais sur les reconfigurations.

Vers un modèle formel pour la reconfiguration

En repartant de l'hypothèse qu'une configuration Fractal est un multi-graphe orienté et étiqueté, les opérations de reconfiguration sont modélisées comme des opérations de transformation de graphes. Les opérations de reconfiguration primitives correspondent à des transformations de graphe de base (e.g., ajout et/ou suppression de nœuds dans le graphe). Nous considérons les opérations de reconfiguration primitives suivantes dans le modèle de composant Fractal (correspondant aux contrôleurs de base, cf. Section 4.1.1) :

- instanciation/destruction d'un composant ;
- ajout/suppression d'un composant dans un autre composant ;
- connexion/déconnexion des interfaces d'un composant ;
- démarrage/arrêt d'un composant ;
- renommage d'un composant ;
- modification des valeurs d'attributs d'un composant.

Une opération de reconfiguration primitive op transforme une configuration $\mathscr{A} = (E, R)$ en une autre configuration $\mathscr{A}' = (E', R')$ où $\mathscr{A}$ et $\mathscr{A}'$ sont respectivement les états initiaux et finaux du système et op l'opération associée à la transition entre ces deux états :

$$\mathscr{A} \xrightarrow{op} \mathscr{A}'$$

$\mathscr{A}$ et $\mathscr{A}'$ doivent être des configurations valides toutes les deux, i.e., conforme au même modèle Fractal $\mathscr{F}$. De plus, pour que $\mathscr{A}'$ soit une évolution valide de $\mathscr{A}$, des invariants de configuration doivent être respectées par toutes les opérations primitives de reconfiguration.

Spécification de la sémantique des opérations primitives

Pour limiter les reconfigurations à des transformations de configurations seulement valides, nous spécifions les contraintes d'intégrité sous forme de *pré-conditions* et de *post-conditions* sur chaque opération primitive de reconfiguration dans Fractal. Ce sont ces nouveaux invariants qui vont être utilisés lors de la *prévention des fautes* de notre chaîne de validation de la reconfiguration (cf. Section 5.2). L'ensemble des spécifications est explicité dans la thèse de Marc Léger [Lég09]. Nous donnons seulement un exemple ci-dessous.

Exemple de l'ajout/retrait d'un composant. Les reconfigurations du contenu des composants dans l'API Fractal sont gérées au niveau du contrôleur $ContentController$.

Opération add**.** L'opération primitive $add(Component\,p, Component\,c)$ ajoute le sous-composant c dans le composant p (cf. tableau ci-dessous). Dans la pré-condition, $hasChild^{trans}(c,p)$ consiste à vérifier que le composant p n'est pas un fils indirect du composant c. Ceci répond au problème de cycle soulevé dans le Listing 5.1. Toujours dans la pré-condition, le prédicat le plus complexe teste si le composant p possède bien une interface i de type $ContentController$ ce qui lui donne le statut de composite. Ceci répond au problème potentiel dans le Listing 5.2. Enfin, dans la post-condition, on ajoute une relation *hasChild* entre p et c pour rendre $\mathscr{A}'$ conforme à la nouvelle architecture.

Ajout d'un sous-composant : $add(Component\ p, Component\ c)$
Préconditions $p, c \in E$ $\exists i \in E,\ hasInterface(p,i) \wedge contentCtrlItf(i)$ $\neg(p = c)$ $\neg hasChild^{trans}(c,p)$
Postconditions $E' = E$ $R' = R \cup \{hasChild(p,c)\}$

Opération *remove***.** L'opération primitive $remove(Component\,p, Component\,c)$ retire le sous-composant c du composant p (cf. tableau ci-dessous). Le point remarquable de la pré-condition est de remarquer que le composant p ainsi que tous ses descendants doivent être arrêtés pour cette opération.

Retrait d'un sous-composant : $remove(Component\ p, Component\ c)$
Préconditions $p, c \in E$ $\exists i \in E,\ hasInterface(p,i) \wedge contentCtrlItf(i)$ $hasChild(p,c)$ $stopped(p) \wedge \forall c \in E,\ hasChild^{trans}(p,c) \Rightarrow stopped(c)$ $\forall (i,i',c') \in E^3,\ hasInterface(c,i) \wedge hasInterface(c',i') \wedge hasBinding(i,i') \Rightarrow c' \neq p \wedge$ $\neg hasChild(p,c')$
Postconditions $E' = E$ $R' = R \setminus \{hasChild(p,c)\}$

Propriétés. Les opérations primitives de reconfiguration étant basées sur des fondements mathématiques, il est important de noter que notre formalisation apporte des propriétés intéressantes quant à leur exécution et composition. Signalons entre autres, le déterminisme du résultat, la composabilité

des opérations, l'idempotence (pour certaines opérations), l'inversibilité des opérations (à l'exception de l'opération *new*), la commutativité (pour certaines opérations), la récursivité pour *start*/*stop*.

L'inversibilité est une propriété servant à défaire les effets d'une opération de reconfiguration et est utile pour un contexte transactionnel (mécanisme de *rollback*). Ci-dessous, un exemple de formalisation de l'inverse de l'opération *remove*.

$$\forall (p,c) \in Component^2,\ remove^{-1}(p,c) = add(p,c)$$

Vérification de la reconfiguration

Pour vérifier la reconfiguration, nous avons utilisé la même approche que décrite précédemment (cf 5.3.2). Tout d'abord, les pré-conditions et les post-conditions sur les opérations de reconfiguration sont traduites dans la langage Alloy afin de vérifier qu'elles ne violent aucune autre contrainte d'intégrité dans le modèle. Une opération de reconfiguration est spécifiée dans Alloy par un prédicat (*pred*) sur deux configurations : une configuration initiale et une configuration finale. Les opérations peuvent ensuite être simulées sur toute configuration initiale. Ensuite, le langage de reconfiguration FScript est utilisé avec FPath pour spécifier les opérations de reconfiguration dans les systèmes d'exécution Fractal.

Par exemple, pour l'opération *add*, le prédicat Alloy correspondant est le suivant :

```
1  pred add [a1, a2: Configuration, p, c: Component] {
2      // Preconditions
3      (c + p) in a1.components
4    not p->c in a1.child
5      isComposite[a1, p]
6    not c->p  in *(a1.child)
7
8      // Elements
9      sameElements[a1, a2]
10     // Relations
11     a2.child = a1.child + p->c
12     a2.interface = a1.interface
13     a2.attribute = a1.attribute
14     a2.binding = a1.binding
15     // Properties
16     sameProperties[a1, a2]
17
18     // Postconditions
19     p->c in a2.child
20 }
```

Listing 5.6 – *add* en Alloy.

Nous nommerons sa translation dans l'univers FPath/Fscript $addSafe$. Nous intégrons les assertions dans FScript en définissant une nouvelle procédure *assert* qui prend en paramètre le résultat d'une expression FPath retournant obligatoirement un booléen. Ainsi, la nouvelle opération $addSafe(p,c)$ est définie comme suit :

```
1  action addSafe(p, c) {
2    //Preconditions
3    -- p must be a composite component
4    assert($p/interface::content-controller));
5    -- p must not equals c
6    assert(not($p == $c));
7    -- p must not be a descendant of c
8    assert(size(intersection($p, $c/descendant::*)) == 0);
9
10   //Operation execution
11   add($p, $c);
```

```
12
13   //Postconditions
14   -- c must be a child of p
15   assert(size(intersection($c, $p/child::*)) == 1);
16 }
```

Listing 5.7 – *addSafe* en FScript.

La traduction des contraintes d'intégrité en FPath (invariants, préconditions et postconditions) permet ainsi de valider la configuration Fractal d'un système à l'exécution ainsi que ses reconfigurations dynamiques. L'ensemble des primitives FScript (cf. Tableau 4.1) peut être rétro-conçu pour apporter plus de garanties au langage dédié FScript.

5.4 Reconfigurations transactionnelles

Afin d'assurer la fiabilité des reconfigurations dynamiques, nous avons adopté une approche transactionnelle. Une reconfiguration est alors une composition d'opérations délimitée dans une transaction. Notre modèle de transaction prend en charge la récupération des fautes pour maintenir la cohérence de l'architecture et la concurrence des reconfigurations distribuées [Lég09] [LLC10].

5.4.1 Modèle transactionnel

Motivations

Nous avons choisi un support transactionnel pour permettre des reconfigurations dynamiques fiables pour plusieurs raisons. Tout d'abord, nous pensons que des transactions bien définies associées à la vérification des contraintes décrites précédemment est un moyen de garantir la fiabilité des reconfigurations (i.e., le système reste cohérent après les reconfigurations même dans le cas d'opérations non valides). Ensuite, un support transactionnel permet une récupération des fautes : lorsqu'une panne matérielle survient pendant la reconfiguration, le système retourne dans l'état antérieur cohérent. Enfin, un support transactionnel peut gérer la concurrence entre les reconfigurations (distribuées) en évitant les conflits potentiels entre les opérations de reconfiguration.

Un modèle de transaction « à plat » pour les reconfigurations dynamiques

Pour garantir la fiabilité, nous utilisons un modèle de transactions « à plat » (*flat transaction*) [TGGL82] pour gérer les reconfigurations transactionnelles et une implémentation des propriétés ACID (Atomicité-Consistance-Isolation-Durabilité) dans le contexte des reconfigurations. Plusieurs modèles de transaction plus complexes ont été définis [WS92], mais les transactions « à plat » se sont révélées suffisantes et efficaces dans les applications où les transactions sont relativement courtes, le nombre de transactions simultanées relativement faible. Les reconfigurations dynamiques que nous considérons semblent satisfaire ces hypothèses. En effet, les reconfigurations sont essentiellement des opérations de courte durée, et le niveau de concurrence pour les reconfigurations devrait être modéré : le système n'est pas constamment reconfiguré sinon il réduirait sa disponibilité.

Etant donné un modèle de transactions, il existe différents modes de mise à jour (*update*) des transactions dans un système qui correspondent à la répercussion des effets des reconfigurations sur le système en train de s'exécuter. Deux modes ont été considérés : le mode de mise à jour immédiate (*immediate update*) et le mode de mise à jour différée (*deferred update*) [BCF+97] (cf. Figure5.6). Le mode de mise à jour a un impact important sur la performance [TGGL82]. La mise à jour immédiate applique les opérations de reconfiguration directement sur l'architecture d'exécution alors que la transaction est encore en état actif. Cette technique est rapide mais coûteuse en cas de récupération de faute car le support transactionnel doit exécuter des opérations de compensation sur le système réel pour revenir à l'état initial. La mise à jour différée réalise une copie paresseuse de la configuration

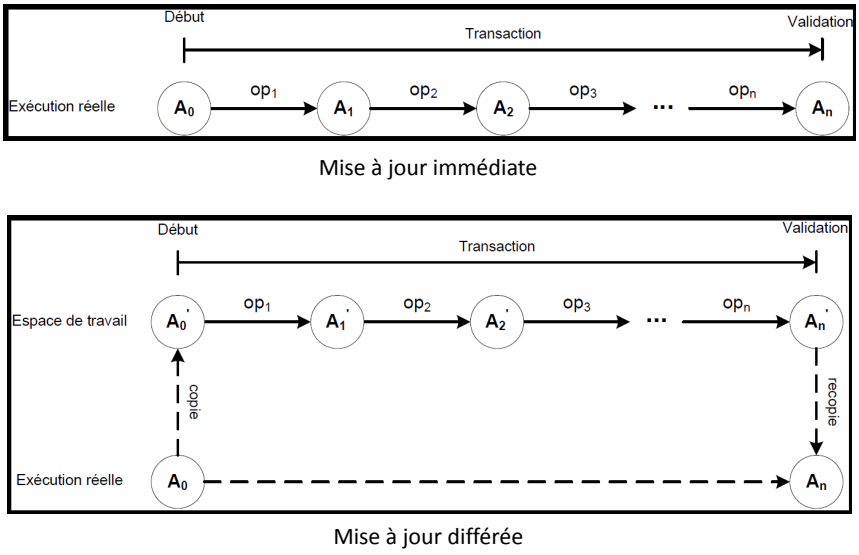


FIGURE 5.6 – Deux types de mise à jour [Lég09].

actuelle. Chaque opération de reconfiguration est ensuite appliquée à la copie de travail jusqu'à la fin de la transaction, auquel cas les modifications sont appliquées au système réel (si la transaction s'arrête, la copie est détruite). Les résultats de notre expérience ont montré que les mises à jour immédiates surpassaient les mises à jour différées à moins que la récupération des erreurs ne soit fréquente.

5.4.2 Propriétés ACID

La définition des propriétés ACID dans le contexte des reconfigurations dynamiques dans les systèmes basés sur les composants est la suivante :

- *Atomicité* (A). Soit la reconfiguration est validée et le système est reconfiguré (i.e., toutes les opérations sont exécutées), soit la reconfiguration échoue et le système revient dans un état cohérent précédent (*rollback*). Pour ce faire, nous avons défini un modèle d'annulation avec des opérations de compensation ;
- *Cohérence* (C). Une reconfiguration est une transformation valide de l'état du système, c'est-à-dire qu'elle conduit le système considéré d'un état cohérent à un autre état cohérent. Un système est dans un état cohérent si et seulement s'il est conforme à nos critères de cohérence : il ne viole pas les contraintes d'intégrité spécifiées dans la Section 5.3 ;
- *Isolation* (I). Les reconfiguration sont exécutées comme si elles étaient indépendantes. Les résultats des opérations de reconfiguration dans une transaction non achevée ne sont pas visibles par d'autres transactions jusqu'à ce que la transaction soit validée ou abandonnée. Pour gérer la concurrence entre les reconfigurations, nous avons utilisé une approche de verrouillage basée sur la sémantique des opérations de reconfiguration ;
- *Durabilité* (D). Les résultats d'une reconfiguration validée sont permanents : le nouvel état du système (à la fois la description de l'architecture et l'état des composants) est sauvegardé afin qu'il puisse être récupéré en cas de fautes majeures (e.g., pannes matérielles). Pour ce faire, nous utilisons deux mécanismes : la journalisation et les points de contrôle (*checkpointing*).

Dans les sections suivantes, nous présentons brièvement la conception qui a été retenue pour la mise en place de propriétés ACID dans Fractal. Pour plus de détails, se référer à la thèse de Marc Léger [Lég09].

Atomicité des reconfigurations transactionnelles

Pour garantir l'atomicité des reconfigurations transactionnelles dans des architectures à composants réparties, nous utilisons un protocole de validation atomique adapté. Nous définissons également un modèle pour défaire (*undo*) les opérations qui sont exécutées dans le système au cours d'une transaction invalidée.

Concernant le protocole de validation atomique, l'architecture cible des composants Fractal est gérée comme une ressource transactionnelle unique avec un seul gestionnaire centralisé de transactions dédié aux reconfigurations dynamiques. Le gestionnaire de transactions utilise en interne un protocole One-Phase-Commit [AGP98] sans phase de vote car il n'y a pas besoin de synchronisation entre plusieurs ressources transactionnelles et le gestionnaire n'a qu'à coordonner ses propres modifications, même si l'architecture est distribuée. Cela permet de s'affranchir du protocole de validation à deux phases (Two-Phase Commit ou 2PC) [TGGL82], plus difficile à implémenter.

L'exécution d'une reconfiguration transactionnelle correspond à une action ou une fonction FScript de niveau supérieur, qui aboutit à l'exécution d'opérations de reconfiguration primitives mêlant à la fois des opérations d'introspection – sans effet de bord – et des opérations d'intercession qui modifient le système. Pour assurer l'atomicité des reconfigurations, les effets des opérations exécutées dans une transaction sont défaits en cas d'abandon de la transaction. Ainsi, seules les opérations d'intercession doivent être prises en compte puisqu'elles sont les seules opérations qui modifient le système.

Dans le cas de la mise à jour immédiate des reconfigurations, deux cas se présentent :

- Quand les opérations sont réversibles, l'annulation d'une transaction est équivalent à défaire la séquence d'opérations d'intercession dans l'ordre inverse de leur exécution. Le modèle est ainsi linéaire puisqu'il ne nécessite pas de commutativité entre opérations. Chaque transaction maintient un journal en mémoire sous la forme d'un séquence d'opérations déjà exécutée. C'est ce journal qui est « défait »pour annuler une transaction ;
- Quand les opérations ne sont pas réversibles (e.g., *new*), il n'existe pas d'opération inverse (*undo*). Aussi, l'annulation de la transaction nécessite un traitement spécifique via des opérations de compensation à définir explicitement.

Dans le cas de la mise à jour différée, les effets des reconfigurations ne sont pas directement appliqués sur le système réel : l'annulation d'une transaction revient alors simplement à supprimer la copie de travail qui a servi pour la simulation de la reconfiguration.

Enfin, comme toute opération pourrait potentiellement conduire le système à un état incohérent, elle doit toujours être inclus dans une transaction (s'il n'y a pas de transaction, une nouvelle est créée). La démarcation des transactions est automatique pour les actions et les fonctions FScript (i.e., aucune démarcation explicite de langage n'est nécessaire). Comme il n'y a pas de transaction imbriquée, deux actions ou fonctions imbriquées sont toujours exécutées dans la même transaction. Une action de haut niveau est toujours composée d'une seule transaction et les actions imbriquées sont ensuite linéarisées. Cela correspond à la sémantique de FScript dans laquelle les actions encapsulées dans d'autres actions sont simplement intégrées dans les actions de plus haut niveau.

Vérification de l'exécution des contraintes d'intégrité

Une reconfiguration transactionnelle ne peut être validée que si le système résultant est cohérent, i.e., si toutes les contraintes d'intégrité sur le système sont satisfaites. Comme explicité dans la Section 5.3, l'ensemble des contraintes d'intégrité sont traduites dans le langage FPath/FScript. Au cours d'une transaction, les *pré/post conditions* des opérations primitives sont vérifiées lors de chaque exécution de l'opération, alors que les *invariants* d'un profil (e.g., contrainte *noSharing*) ne sont vérifiés qu'à la validation des transactions. Autrement dit, un système peut violer temporairement les invariants lors d'une transaction mais doit être dans un état correct après validation (cf. Figure 5.7). Lorsqu'elle est détectée, une violation de contrainte annule la transaction (*rollback*).

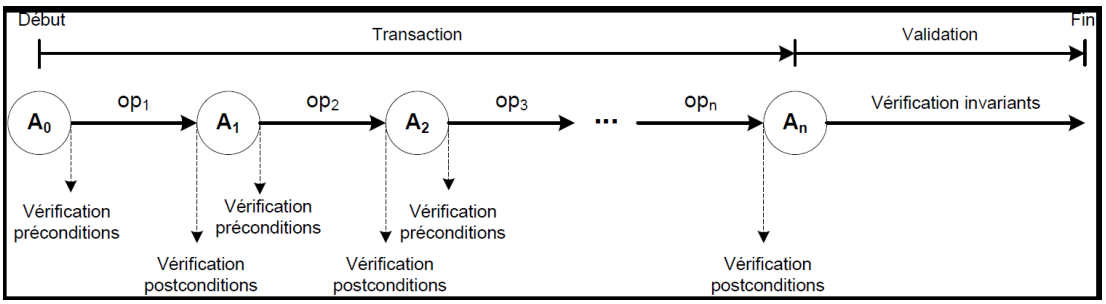


FIGURE 5.7 – Vérification des contraintes d'intégrité [Lég09].

Isolation des reconfigurations comme support à la concurrence

Les reconfigurations sont exécutées de manière concurrente et les accès aux éléments architecturaux Fractal dans le système doivent être synchronisés pour éviter les conflits. L'isolation garantit que le déroulement d'une transaction n'est pas influencé par les éventuelles exécutions d'autres transactions. Chaque transaction est isolée : elle s'exécute comme si elle était la seule à s'exécuter. Aussi, les effets d'une transaction non validée ne sont pas visibles des autres transactions.

Il existe deux principales méthodes de contrôle de concurrence :

- les méthodes dites *pessimistes* qui contrôlent les conflits entre transactions en continu, à chaque exécution d'opération. Elles sont plutôt adaptées à la mise à jour immédiate où le coût d'abandon de transaction est plus élevé en cas de conflits [BHG87]) ;
- les méthodes dites *optimistes* adaptées en cas de conflits peu nombreux entre transactions et à la mise à jour différée car les effets des transactions non validées ne sont pas directement appliqués dans le système.

Nous avons adopté une approche pessimiste avec un verrouillage en deux phases strict (2PL) [TGGL82] afin de supporter un fort niveau de concurrence. En effet, les techniques à base de verrouillage sont les plus adaptées aux transactions courtes et on peut faire raisonnablement l'hypothèse que les reconfigurations dynamiques sont en général de courte durée. Dans le protocole 2PL, les verrous acquis pendant une transaction ne sont relâchés qu'au moment de la validation.

Nous proposons un modèle de verrouillage hiérarchique (i.e., le verrouillage d'un composant par exemple entrainera également le verrouillage de toutes ses interfaces) et les verrous sont réentrants (i.e., une transaction peut acquérir plusieurs fois le même verrou). Il existe deux types de verrous : les verrous en lecture (ou partagés) – utilisés par les opérations d'introspection (e.g., accès à la liste des interfaces d'un composant, à la liste des sous-composants d'un composant) – et les verrous en écriture (ou exclusifs) pour les opérations d'intercession (e.g., modification des liaisons entre composants, ajout/retrait de composants dans un composant). Seuls les verrous en lecture sont compatibles entre eux : toute autre combinaison de demande de verrous bloque des transactions afin d'exécuter la seule la transaction qui possède tous les verrous qui lui sont nécessaires [5].

Durabilité des reconfigurations comme support pour la reprise sur défaillance

La propriété de durabilité – en rendant les effets des transactions permanents – permet la mise en place d'un mécanisme de reprise sur défaillance où le système est ramené dans un état stable

5. Pour plus de détails sur le modèle de verrouillage, la gestion des interblocages ou encore la synchronisation entre le niveau fonctionnel du système et le processus de reconfiguration, cf. [Lég09].

connu avant l'occurrence de la défaillance. La durabilité est réalisée à travers deux mécanismes : la journalisation et les points de contrôle (*checkpointing*, i.e., point de synchronisation entre le contenu du journal et l'état du système stocké sur disque).

Les transactions sont des unités de reprise sur défaillance, de sorte que pour chaque transaction, la démarcation et les opérations primitives d'intercession sont enregistrées dans un *journal*. Le journal est conservé à la fois en mémoire et persisté sur disque : il peut ainsi être rejoué en cas de panne logicielle ou matérielle. Nous utilisons la technique de "write-ahead logging" avec points de contrôle [GR92], ce qui signifie que toutes les modifications sont écrites dans le journal avant d'être appliquées dans le système et que les informations pour refaire et défaire sont stockées dans le journal. Le format du journal suit la syntaxe FPath/FScript en ce qui concerne les opérations primitives. En plus de la journalisation des transactions, l'état du système est sauvegardé sur disque périodiquement au moment du point de contrôle. L'état d'un système à base de composants qui est considéré ici est sa description architecturale (sauvegardée sous forme de définition Fractal ADL) et l'ensemble de l'état de ses composants (valeurs des attributs).

Ainsi, deux types de reprise sur défaillance sont possibles :

- les *reprises à chaud* en cas de défaillances logicielles (e.g., violation de contraintes d'intégrité) : la transaction abandonnée est défaite (*rollback*) à partir du journal ;
- les *reprises à froid* pour les fautes matérielles : le système est remis dans son dernier état cohérent à partir de l'état du système sauvegardé sur disque. Toutes les transactions qui ne sont pas validées sont annulées et toutes les transactions validées depuis le dernier point de contrôle sont refaites.

5.5 Evaluation et expérimentations de Fractal TXR

Un framework modulaire basé sur des composants Fractal a été développé pour implémenter un gestionnaire de transactions pour les reconfigurations dynamiques dans les systèmes Fractal. Ce prototype, appelé Fractal TXR (pour *Fractal Transactional Reconfiguration*) a été réalisé par Marc Léger dans le cadre de sa thèse [Lég09]. Il se compose de plusieurs sous-composants chargés des différentes fonctionnalités transactionnelles. La modularité du framework permet aux programmeurs d'activer ou de désactiver certaines propriétés transactionnelles et de modifier facilement leur implémentation. Par exemple, le sous-composant `ConcurrencyManager` implémente une approche pessimiste : nous pourrions le remplacer par une approche optimiste pour éviter le coût du verrouillage, surtout s'il y a peu de reconfigurations simultanées dans le système.

Dans les sections suivantes, nous proposons d'étudier le surcoût de cette extension transactionnelle de Fractal et de donner un exemple d'utilisation dans le cadre du projet RNTL Selfware.

5.5.1 Evaluation des performances

Contexte et motivations. Nous proposons d'évaluer le coût supplémentaire des mécanismes transactionnels lorsqu'une reconfiguration dynamique est réussie [6]. L'objectif est de déterminer le rapport entre les temps d'exécution d'une reconfiguration avec transaction et sans transaction.

Nous choisissons une application de base pour ce micro-benchmark : le composant Fractal `HelloWorld` est un composite avec deux composants primitifs (cf. Figure 5.8). L'interface `main` de `HelloWorld` renvoie un message "Bonjour" concaténé avec une valeur stockée dans un attribut du composant primitif `Server`.

Le cas d'utilisation de la reconfiguration consiste à déconnecter le composant `Server` à partir du composant `Client`, le retirer de son parent `HelloWorld`, puis l'ajouter et le reconnecter. Nous

6. Si la reconfiguration échoue *avec* notre gestionnaire de transactions, la procédure de reprise sur panne est lancée. Si la reconfiguration échoue *sans* notre gestionnaire de transactions, le système devient inutilisable. Ainsi, une comparaison de performance lorsqu'une reconfiguration est invalide n'a aucun sens.

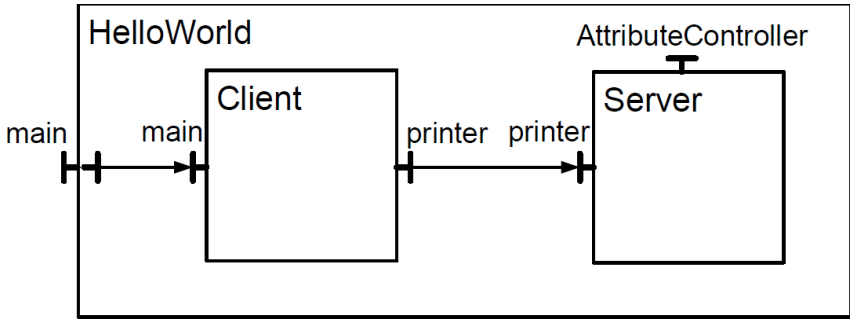


FIGURE 5.8 – Architecture Fractal du composant HelloWorld.

exécutons la reconfiguration 100 fois dans les trois contextes suivants :

1. exécution du code de reconfiguration Java dans une JVM locale ;
2. exécution du code Java distribué avec RMI dans plusieurs JVM ;
3. exécution du code de reconfiguration FScript localement.

Le code correspondant dans FScript est le suivant :

```
1 action reconfig(helloWorld) {
2   server = $helloWorld/child::server;
3   client = $helloWorld/child::client;
4   unbind($client/interface::printer);
5   remove($helloWorld, $server);
6   add($helloWorld, $server);
7   bind($client/interface::printer, $server/interface::printer);
8 }
```

Listing 5.8 – Exemple de reconfiguration en FScript.

Mise en œuvre du scénario. Pour nos tests, nous avons instancié plusieurs JVM sur la même machine pour la reconfiguration distribuée. La technique de mise à jour de la transaction utilisée pour ce micro-benchmark est une mise à jour immédiate (cf. Section 5.4). Comme expliqué précédemment, il est possible d'activer ou de désactiver certaines propriétés transactionnelles dans notre framework modulaire Fractal TXR. Cette fonctionnalité nous permet d'évaluer indépendamment les différents composants du gestionnaire de transactions.

Nous choisissons d'abord d'évaluer la performance d'un gestionnaire de transaction minimal. Nous ne conservons que le support pour la récupération : les opérations primitives de reconfiguration sont enregistrées dans un journal afin qu'il puisse être annulé en cas de *rollback*. Le Tableau 5.1 nous montre les différents résultats. Le coût supplémentaire s'explique essentiellement par l'interception des opérations primitives par le composant `Transactional Monitor` pour la journalisation. Le ratio de surcharge (*overhead*) des transactions est réduit dans le contexte de la reconfiguration distribuée (Java RMI) ou de l'interprétation du langage FScript car le coût des interceptions est presque constant tandis que les reconfigurations distribuées et reconfigurations exprimées avec un langage interprété sont plus coûteuses que les appels Java en local. Ce coût supplémentaire pourrait sembler important, mais il doit être mis en perspective avec le nombre de reconfigurations. En outre, c'est le prix à payer pour plus de fiabilité et la continuité de service.

À partir de la reconfiguration locale précédente (i.e., code Java dans une JVM locale), nous définissons deux nouveaux scenarii en introduisant d'autres propriétés transactionnelles. Chaque propriété est implémentée par un composant du gestionnaire de transactions et est présentée séparément :

Tests	Without tx support (1)	With tx support (2)	Ratio (2)/(1)
Reconfig. Java	37	112	3,03
Reconfig. Java with RMI	1297	1985	1,53
Reconfig. FScript	157	250	1,59

Table 5.1 – Comparaison du temps d'exécution (*ms*) avec/sans transaction [LLC10].

— *Vérification des contraintes d'intégrité.* Le composant `Consistency Manager` vérifie les contraintes d'intégrité du modèle et du profil (cf. Section 5.3) et une contrainte applicative spécialement définie pour le composant `HelloWorld` (spécifiée par l'ADL avec une balise de contrainte). Cette contrainte applicative interdit au composite `HelloWorld` de disposer de plus de deux sous-composants ;
— *Gestion de la concurrence.* Le composant `Concurrency Manager` implémente une approche pessimiste avec un verrouillage à deux phases strict [TGGL82] pour assurer une forte concurrence : chaque opération primitive bloque l'accès à l'élément Fractal, cible de la reconfiguration.

Le Tableau 5.2 nous montre les différents résultats où le ratio est calculé à partir du scénario local d'exécution Java sans transaction présenté dans le Tableau 5.1.

Tests	With tx support	Ratio (with tx)/(without tx)
Integrity constraints checking	161	4,35
Concurrency management	177	4,78

Table 5.2 – Comparaison du temps d'exé. (*ms*) d'une reconfiguration avec propriétés ACID [LLC10].

Le coût de la vérification des contraintes semble acceptable pour cet exemple au regard du bénéfice de cette propriété. Cependant, il convient de noter que la vérification des contraintes dépend du nombre et de la complexité des contraintes dans le système. Le coût de la gestion de la concurrence provient principalement de la stratégie pessimiste qui requiert l'acquisition de certains verrous pour les opérations de reconfiguration.

Conclusion. Cette étude réalisée à partir de micro-benchmark est difficilement généralisable. Cependant, une étude plus exhaustive n'a pas pu être réalisée, faute de benchmarks complets et reconnus par la communauté en ce qui concerne la reconfiguration dynamique. Notre expérimentation donne toutefois des tendances intéressantes (e.g., surcoût de 50% pour le *rollback* en FScript transactionnel).

5.5.2 Auto-réparation d'un serveur Java EE dans un cluster

Contexte et motivations. Nous proposons d'illustrer notre contribution avec un cas d'utilisation développé dans le projet RNTL Selfware. Ce cas d'utilisation montre une auto-réparation (*self-healing*) dans le cadre de serveurs d'applications Java EE montés en cluster [LLC10]. Ce scénario est utilisé pour réparer une faute transitoire (par exemple, une surcharge de mémoire) dans une instance Java EE en la redémarrant. Notre objectif est d'améliorer la fiabilité de l'opération « redémarrage », de sorte que nous proposons de redémarrer le serveur dans une reconfiguration transactionnelle en respectant certaines contraintes d'intégrité.

Contraintes d'intégrité. Dans la plate-forme Selfware, chaque élément architectural est enveloppé dans un composant Fractal. Nous avons spécifié des contraintes d'intégrité au niveau profil à la fois sur l'architecture globale du cluster et sur certains nœuds dans le cluster. Ci-dessous quelques exemples de ces contraintes :

— sur l'architecture globale : unicité d'un nom d'instance JOnAS [7] dans le domaine d'administration du cluster, unicité d'une instance maître dans le domaine pour gérer le cluster, séparation entre des tiers Web et EJB sur différents nœuds ;
— sur un nœud local : disponibilité de la ressource système (mémoire, CPU) pour exécuter une instance JOnAS, unicité des ports réseaux entre les instances JOnAS, nombre maximum/minimum d'instances JOnAS sur un même nœud.

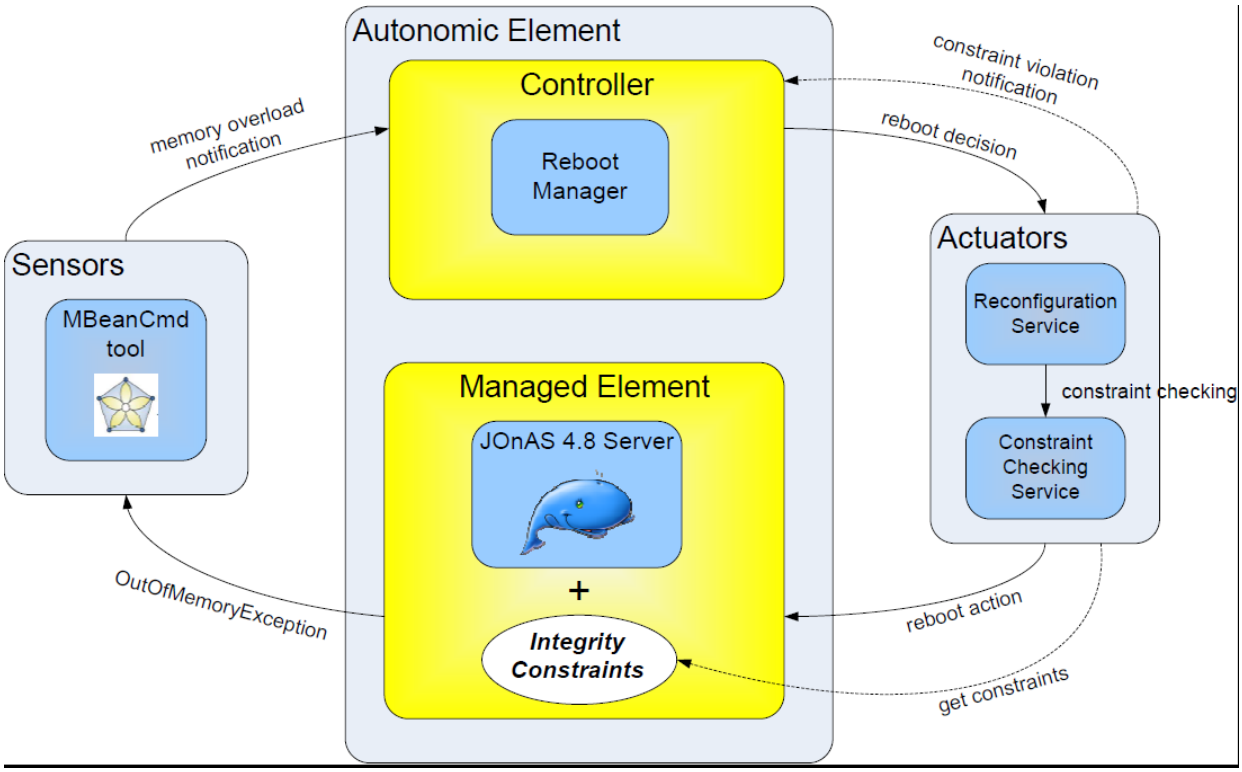


FIGURE 5.9 – Scénario de reconfiguration transactionnelle dans un cluster Java EE [Lég09].

Mise en œuvre du scénario. Nous choisissons de réparer une surcharge de mémoire dans une instance JOnAS qui peut entraîner un crash JVM en redémarrant localement le serveur. Nous mettons une contrainte locale sur chaque nœud du cluster en rapport avec la quantité minimum de mémoire disponible nécessaire pour exécuter le nœud. La boucle de contrôle MAPE-K utilisée pour réparer la faute transitoire est composée des éléments suivants (cf. Figure 5.9) :

— Sondes : un agent JMX [8] est utilisé pour surveiller les informations JOnAS dans le cluster et permet d'attraper les exceptions du type `OutOfMemoryException` levées par les JVM défectueuses (i.e., mémoire en surcharge) ;
— Contrôleur : le gestionnaire de redémarrage s'abonne aux événements JMX et quand il est notifié de la surcharge mémoire, il décide de redémarrer les serveurs d'application fautifs ;
— Actionneurs : le service de reconfiguration exécute l'opération « redémarrage » en respectant les contraintes d'intégrité dans un contexte transactionnel.

La transaction est composée des opérations suivantes :

— arrêt du composant Fractal serveur : invocation de l'opération FScript `stop` ;

7. JOnAS est un serveur d'application *open source* développé au sein du consortium OW2 qui implémente la certification Java EE.
8. `https://fr.wikipedia.org/wiki/JMX`

— arrêt de l'instance JOnAS : après l'étape précédente, le serveur est dans un état `stopped` d'où la nécessité de terminer l'exécution du processus serveur ;
— redémarrage du serveur fautif sur la même machine.

Dans la phase de validation de la transaction (*commit*), le service de reconfiguration s'assure que les contraintes d'intégrité sont satisfaites, c'est-à-dire qu'il vérifie que la mémoire disponible sur le nœud est supérieure au seuil minimal donné par la contrainte du nœud. Si c'est le cas, le scénario d'auto-réparation est exécuté jusqu'à sa fin.

Dans le cas contraire, le service de reconfiguration annule l'action de réparation et notifie le contrôleur de la violation de contrainte (et l'impossibilité de ré-instancier le serveur sur ce nœud). Pour le *rollback*, puisqu'il n'existe pas d'opération inverse immédiate à arrêt de l'instance JOnAS sur ce nœud (problème de réversibilité), il est nécessaire de prévoir une opération de compensation explicite (cf. Section 5.4.2). Cette dernière pourrait par exemple essayer de ré-instancier le serveur défaillant sur un autre nœud où plus de mémoire est disponible.

Conclusion. Ce scénario a été l'occasion d'expérimenter les reconfigurations transactionnelles dans un contexte industriel Java EE. L'apport de notre contribution sur les reconfigurations dynamiques au scénario consiste à introduire une contrainte d'intégrité applicative au niveau d'un serveur d'application et à vérifier que cette contrainte est bien respectée lors de la validation de la reconfiguration du serveur dans un cadre transactionnel.

5.6 Conclusion et perspectives (a.k.a. interfaces fournies)

5.6.1 Généralisation à d'autres modèles de composants

Notre spécification des (re)configurations est basée sur l'utilisation du modèle de composants Fractal. Pour aller au-delà du modèle Fractal, notre approche de fiabilisation des reconfigurations dynamiques peut être généralisée et appliquée à d'autres modèles de composants qui possèdent des caractéritiques similaires de reconfiguration. Les modèles réflexifs OpenCOM [CBG+08], FraSCAti [SMR+12] voire OSGi [9] sont ainsi des cibles potentielles pour appliquer/intégrer nos travaux.

Cette généralisation nécessiterait un effort de modélisation des concepts de ces différents modèles en terme d'éléments architecturaux et de relations entre ces éléments comme cela a été fait pour Fractal (cf. Section 5.3). Il faut également spécifier la sémantique des opérations de reconfiguration. Le modèle de transactions décrit dans la Section 5.4 devrait par contre rester sensiblement le même, les opérations de reconfigurations restent en effet assez similaires entre les modèles de composants (e.g., modification de l'architecture, du cycle de vie, ...).

5.6.2 De la difficulté à valoriser un prototype de recherche

Après quelques années passées au sein de ma communauté scientifique, je fais le constat que valoriser le prototype de recherche d'un doctorant reste un sujet délicat. Pourtant il existe de nombreux leviers pour y arriver. Ci-dessous je retrace quelques morceaux choisis concernant ma propre expérience. L'hypothèse de départ est que le doctorant quitte le laboratoire et que je suis incapable (par manque de temps, de compétences) de reprendre son code.

— **A travers les projets (e.g., ANR, ADT INRIA, ...)** : le prototype d'un doctorant peut avoir une nouvelle vie suite au démarrage d'un projet pour lequel le dit prototype constitue un élément clef. Il peut être soit maintenu par son propre auteur recruté par un partenaire entre-temps (cas de FScript), soit redeveloppé par un nouvel acteur (cas de WildCAT 2.0). Tout projet ayant une fin, il est important d'anticiper la suite comme le développement d'un éco-système autour du prototype ;

9. `https://www.osgi.org`

- **A travers une forge numérique (e.g., OW2, Apache, Eclipse...)** : la forge numérique va présenter l'avantage d'un support technique (centralisé, accessible à distance) pour le développement en équipe autour d'un logiciel. FScript et WildCAT sont hébergés par OW2 : cela a permis à une communauté d'acteurs intéressés par ces prototypes de contribuer à leur développement (e.g., Bull pour WilCAT ou l'équipe INRIA Adam pour FScript). Cependant, l'animation est primordiale : sans véritable leader de l'éco-système, le prototype risque l'obsolescence (e.g., FScript n'a pas été modifié depuis plus de 2500 jours) ;
- **A travers une entreprise privée** : comme nous le verrons par la suite avec le prototype Mucamac [ADOJ13], le travail d'un doctorant peut également intéressé une entreprise privée dans le cadre d'un contrat direct avec le laboratoire de recherche. Malheureusement, cette entreprise n'a pas toujours la volonté de développer (e.g., contradiction des objectifs entre RH et le pôle technique) ou ne sait pas toujours faire évoluer des prototypes souvent pointus ;
- **Le problème de la licence et/ou de la propriété intellectuelle** : *last but not least*, il est très important de choisir dès le début une licence qui permet de collaborer et non pas une licence virale (expérience de la licence GPL de WildCAT qui « a refroidi » les partenaires de l'ADT Galaxy) et d'anticiper l'éventuel cession des droits patrimoniaux du prototype du doctorant qui appartient à son employeur. Il faut remarquer en passant que l'employeur du doctorant n'est peut-être pas l'employeur de son encadrant ce qui complexifie la donne (e.g., prototype Mucamac) !

En conclusion, un ingénieur R&D en CDI dédié à une équipe peut lui permettre de valoriser plus facilement sur le long terme les prototypes les plus probants de l'équipe (cas de l'équipe INRIA Triskell avec Kermeta qui aggrège des résultats de doctorants ou encore de l'équipe INRIA Tasc avec le solveur de contraintes Choco [PFL17]). Si tel n'est pas le cas – comme pour l'équipe Ascola – il reste également la solution de transmettre/partager les idées et non le code : c'est ce qui s'est produit entre FScript et GCMScript [IRHBJ16] ou FraSCAti Script [SMR+12].

Pour revenir au chapitre courant, Fractal TXR de Marc Léger n'a pas eu la chance de connaître de seconde vie (pas de projet à la suite, code encore instable dans le *sandbox* Fractal de la forge OW2, pas de contrat industriel direct, ...). Ce qui est est vraiment dommage à la vue des résultats présentés.

5.6.3 Bilan

La fiabilité des reconfigurations dynamiques, et tout particulièrement le support de propriétés de type transactionnel associées à la reconfiguration, ont peu été traités dans la littérature (à l'exception des projets K-Component [DC01], FORMAware [MBC04], Plastik [BJC05] principalement). Pourtant l'adaptation dynamique reste un sujet d'actualité comme en témoigne le challenge scientifique "*Ever-running software*" identifié dans le plan stratégique INRIA 2018-2022 [INR].

Alors comment expliquer le peu de travaux [TS11] sur les reconfigurations fiables et dynamiques des logiciels à base de composants ? Et quid de leur dissémination ? Peut-être que la réponse est à chercher autour de la diffusion et l'adoption de la programmation par composants elle-même. Fractal n'a pas eu le succès escompté, seul OSGi a connu une large adoption grâce à l'industrie de l'informatique embarquée (et grâce à Eclipse du côté du logiciel). En effet, la grande majorité des développeurs continue de programmer *in the small* (avec le succès que l'on connait pour les langages de script comme JavaScript [10]) et la programmation *in the large* se fait essentiellement grâce à l'approche orientée services (la SOA et ses protocoles SOAP et REST). Ce constat a participé d'une part à ma démotivation dans la recherche sur les architectures à composants et d'autre part à mon interêt pour explorer les architectures à base de services [BRL07, BRL08].

Entre 2008 et 2010, je profite d'une période de détachement à INRIA pour prendre du recul sur mes travaux de recherche. Je réalise alors que j'ai consacré une grande partie de mes travaux sur

10. Pour modérer mes propos, les développeurs programment toutefois avec des frameworks qui sont des architectures logicielles (e.g., AngularJS pour JavaScript).

le « Comment » réaliser l'adaptation dynamique mais presque pas sur le « Pourquoi ». Même si certaines expérimentations ont permis de (i) modifier les mécanismes de distribution à l'exécution (OpenCorba [Led99] ; (ii) redimensionner un composant cache serveur à l'exécution et modifier la politique d'adaptation elle-même (Safran [DL03]) ; (iii) réaliser une auto-réparation d'un serveur Java EE [LLC10] ; ... je me sentais modérément motivé par ces expériences. L'émergence de l'informatique en nuages et le réveil d'une conscience citoyenne en moi liée au développement durable vont changer la donne.

Troisième partie

Reconfiguration dynamique : pour qui ?

Chapitre 6

Cloud computing et empreinte énergétique

Ce chapitre a pour objectif de motiver les travaux que je mène depuis 2009. Je me suis intéressé à la thématique du *Green computing* dans le but d'optimiser l'empreinte énergétique des centres de données qui accompagnent l'essor spectaculaire du *Cloud computing*. En partant de ma profession foi de la Section 2.1 – « l'adaptation du logiciel à l'exécution constitue une piste importante pour répondre aux défis du numérique » – j'ai voulu montrer que la reconfiguration dynamique d'une *architecture en nuages* pouvait apporter une réponse aux défis de la transition écologique.

6.1 Contexte

6.1.1 L'informatique en nuage

> « *A Cloud is a type of parallel and distributed system consisting of a collection of interconnected and virtualised computers that are* ***dynamically provisioned*** *and presented as one or more unified computing resources based on* ***service-level agreements*** *established through negotiation between the* ***service provider and consumers*** *[Buy09].* » (Buyya et al. - 2009)

Le *Cloud computing* ou l'informatique en nuage ou encore l'infonuagique (au Canada francophone) désigne l'exploitation de la puissance de calcul et/ou de stockage d'un réseau de ressources informatiques (i.e., matérielles et logicielles) virtualisées et mutualisées, accessibles à distance, à la demande et en libre-service via le réseau par le biais des technologies Internet [VRMCL08].

Même si l'on ne peut parler de révolution informatique avec le Cloud, l'informatique en nuage constitue une nouvelle façon de délivrer les ressources informatiques. Il s'agit d'accéder à des ressources, reposant sur une architecture distante gérée par une tierce partie : le fournisseur de service (i.e., *service provider*). Celui-ci assure la continuité et la maintenance du service dont il a la charge. Les utilisateurs (ou consommateurs) accèdent à la partie applicative via leur connexion Internet sans se soucier du reste. Les utilisateurs peuvent ainsi louer (i.e., paiement à l'utilisation, *pay-per-use*) les technologies de l'information et bénéficier d'une flexibilité importante avec un effort minimal de gestion.

Le concept de *Service Level Agreement* (SLA) a été introduit dans les années 80 par les opérateurs téléphoniques afin de gérer la qualité de service (*Quality of Service* ou QoS) dans leurs contrats avec les entreprises [WB10]. Un SLA, que l'on peut traduire en français par « accord de niveau de service », est un contrat dans lequel la QoS est formalisée et contractualisée entre le prestataire de service et ses utilisateurs. Dans le contexte du Cloud, les critères considérés diffèrent selon les modèles de service [ADC10] et les services proposés. Néanmoins, on retrouve deux métriques récurrentes [PSG06] : la disponibilité (*availability*) (e.g., Gmail) et le temps de réponse (*response time* ou *latency*). Pour les

applications grand public gratuites (e.g., WhatsApp), le SLA est *implicite*. Ces applications exposent des conditions générales d'utilisation (CGU) mais il n'y a pas à proprement parler d'accord sur le niveau de service (le modèle économique de ces applications n'étant pas financé par l'utilisateur mais par la publicité par exemple). Pour les applications B2B (*Business to Business*), nous sommes plutôt dans une logique d'achat et le SLA devient explicite (e.g., *Google Compute Engine Service Level Agreement*).

On dénombre actuellement trois grands modèles de service Cloud (cf. Figure 6.1) :

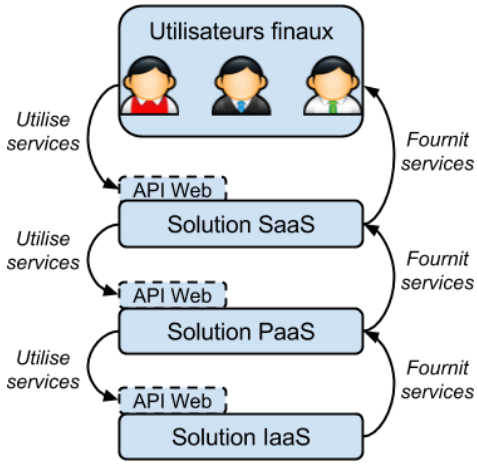


FIGURE 6.1 – Une architecture multi-couche orientée services [Dup16].

- *SaaS (Software as a Service)* : Les "logiciels en tant que service" sont des applications accessibles via Internet où le matériel informatique, l'hébergement et le logiciel sont dématérialisés. Les applications de type SaaS sont diverses et variées. On peut citer les applications grand public de type *B2C* (*Business to Consumer*) telles que la messagerie *Gmail*, le réseau social *Facebook* ou encore le service de cartographie en ligne *Google Maps*. Les applications peuvent aussi être à destination de professionnels (*B2B* ou *Business to Business*) parmi lesquelles on retrouve les *CRM* (*Customer Relationship Management*) comme *StayinFront* ou encore les solutions logicielles proposées par *Salesforce.com*.
- *IaaS (Infrastructure as a Service)* : La ressource fournie est du matériel informatique virtualisé. Physiquement, les ressources matérielles (*hardware*) proviennent d'une multitude de serveurs et de réseaux généralement distribués à travers de nombreux centres de données (*datacenters*), dont le fournisseur IaaS a la responsabilité. Une solution IaaS permet aux utilisateurs d'accéder de façon flexible à des systèmes virtuels complets en démarrant/arrêtant à la demande des *machines virtuelles* dans des *datacenters*, sans se soucier de la gestion du matériel sous-jacent. Parmi les principales solutions de IaaS, on retrouve *Amazon EC2* et *Windows Azure*.
- *PaaS (Platform as a Service)* : Les "plateformes en tant que service" offrent l'accès à un environnement de développement administré, hébergé et maintenu par les fournisseurs PaaS, facilitant le déploiement et l'exécution des applications SaaS en ajoutant une couche de services à la couche IaaS. Situé entre la couche SaaS et la couche IaaS, le PaaS abstrait la couche IaaS à ses utilisateurs et encourage ainsi les équipes de développement à se concentrer sur l'architecture et la réalisation des applications. *Cloud Foundry*, *OpenShift*, *IBM Cloud DevOps* sont des exemples de solutions de type PaaS.

L'acronyme XaaS (*Anything as a Service* [Sch09]) a été créé en raison du nombre croissant d'applications basées sur le concept d'externalisation de fonctionnalités sous forme de services (e.g., DaaS : *Data as a Service*, NaaS : *Network as a Service*, ...).

Dans ce manuscrit, nous focalisons sur le modèle de services, les SLA et sur le concept de l'*élasticité* permettant de provisionner les ressources à la demande (cf. Section 6.2.3). Pour plus de détails sur les caractéristiques principales du Cloud, voir la publication du NIST (National Institute of Standards and Technology) [MG11].

6.1.2 Transition écologique et développement durable

> « *Veiller à l'obligation des générations futures d'être une humanité véritable est notre obligation fondamentale à l'égard de l'avenir de l'humanité, dont dérivent seulement toutes les autres obligations à l'égard des hommes à venir.* » (Hans Jonas - 1979)

A l'aube du 21ème siècle, nous avons tous pris conscience que l'avenir de notre planète n'allait pas dans le bon sens. Réchauffement climatique, catastrophes naturelles historiques, épuisement des ressources naturelles, augmentation de la population mondiale, émission considérable de CO_2, ...constituent une problématique bien réelle dont se sont emparés désormais les pouvoirs publics. Comme l'affirme le philosophe allemand Hans Jonas dans son ouvrage « Le Principe responsabilité », nous sommes tous responsables et la transition écologique passe nécessairement par des changements de comportement (e.g., le tri de nos déchets ménagers, la suppression des sacs plastiques sont deux exemples de notre quotidien).

Le développement durable est l'une des réponses apportées à la transition écologique appliquée à la croissance économique et reconsidérée à l'échelle mondiale afin de prendre en compte les aspects environnementaux et sociaux d'une planète globalisée. Les anglophones vont utiliser le terme de *sustainability* et donner la définition suivante : *"the quality of not being harmful to the environment or depleting natural resources, and thereby supporting long-term ecological balance"* [1].

6.2 Problématique : enjeux énergétiques des TIC

La mondialisation des TIC (Technologies de l'Information et de la Communication), permettant un accès banalisé et 24h/24 depuis n'importe quel point du globe à un ensemble de ressources (données, puissance informatique), a des effets désastreux sur le plan de l'environnement. En effet, l'empreinte écologique des TIC grandit rapidement à cause d'une consommation très importante de métaux ; l'empreinte carbone et énergétique augmente également dans le monde car il se vend de plus en plus de terminaux informatiques (ordinateurs, *smartphone*, tablettes, objets connectés, etc.). Aussi, l'*efficacité énergétique* des TIC est l'un des enjeux majeurs de notre époque.

La Figure 6.2 montre que si les TIC était un pays, il serait le 3ème plus grand consommateur d'électricité au monde.

6.2.1 Informatique durable

L'informatique durable ou *Green computing* ou informatique verte (*Sustainable Computing* en anglais) est un concept qui vise à réduire l'empreinte écologique, économique et sociale des TIC. Elle traite notamment de la réduction de l'utilisation de matières dangereuses, de la maximisation de l'efficacité énergétique pendant la durée de vie du produit, de la recyclabilité ou la biodégradabilité des produits défunts et des déchets industriels.

Dans la suite de ce mémoire, nous traitons uniquement la problématique de *l'efficacité énergétique du Cloud.*

1. http ://www.dictionary.com/browse/sustainability

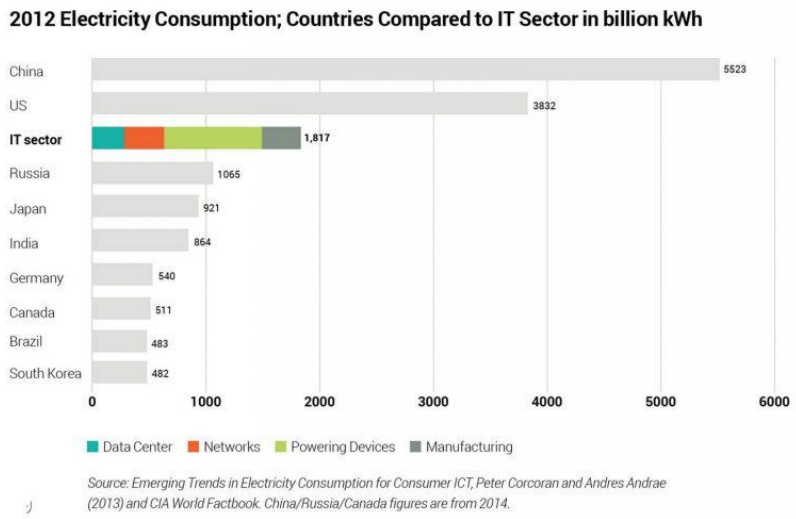


FIGURE 6.2 – Consommation d'électricité des TIC en 2012.

6.2.2 Empreinte énergétique liée à l'informatique en nuage et aux centres de données

Un concept de base du Cloud est la concentration de la production d'énergie informatique dans de grands centres de données regroupant des fermes de serveurs (*data centers*). La mutualisation des ressources induit une baisse de coût, et les technologies de virtualisation permettent de rationaliser l'usage des serveurs. Néanmoins, l'évolution de la consommation énergétique des TIC et plus précisément des centres de données reste préoccupante [Koo11]. Par exemple, en France, la consommation électrique des *data centers* aurait atteint près de 3 TWh en 2015 selon RTE, soit davantage que la consommation électrique annuelle de la ville de Lyon !

Paradoxe de Jevons

> « *As technological improvements increase the efficiency with which a resource is used, total consumption of that resource may increase, rather than decrease.* » (W.S. Jevons - The Coal Question - 1865)

Jusqu'à l'avènement du Cloud computing, l'administration système consistait à installer une architecture surdimensionnée pour limiter l'impact d'un événement (e.g., panne d'un serveur). Cela coûtait cher à l'achat et en fonctionnement (i.e., consommation énergétique) mais cette solution était rarement capable de supporter les cas exceptionnels (e.g., pic de charge imprévisible). Avec le Cloud, où la ressource est facturée à l'usage, la solution est d'activer de nouvelles machines à la demande pour absorber l'activité.

Cependant, le *Paradoxe de Jevons*, pourtant vieux de 150 ans, dit qu'*"à mesure que les améliorations technologiques augmentent l'efficacité avec laquelle une ressource est employée, la consommation totale de cette ressource tend à augmenter au lieu de diminuer"*. Bien que provenant d'un ouvrage traitant de l'utilisation du charbon, ce paradoxe reste tout à fait d'actualité et l'essor du Cloud ne permet pas la maîtrise de la consommation d'énergie et la réduction de l'empreinte carbone.

6.2.3 Un début de réponse : l'élasticité

> « *Elasticity is the degree to which a system is able to adapt to workload changes by provisioning and de-provisioning resources in an autonomic manner, such that at each point in time the available resources match the current demand as closely as possible [HKR13].* » (Nikolas Roman Herbst et al. - 2013)

L'*élasticité* – l'une des caractéristiques essentielles du Cloud – vise à gérer finement l'ajout et le retrait de ressources (*Cloud service provisioning*) afin d'être toujours au plus proche de la demande et éviter selon les cas le *sur-dimensionnement* ou le *sous-dimensionnement.*

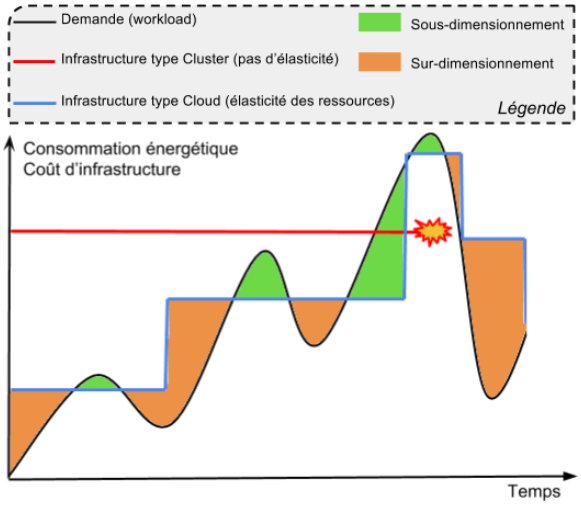


FIGURE 6.3 – Élasticité, sur-dimensionnement et sous-dimensionnement [Dup16].

En effet, le principal défi pour le fournisseur de services est de maintenir la satisfaction de ses consommateurs (i.e., éviter le sous-dimensionnement), tout en minimisant le coût de ses services (i.e., éviter le sur-dimensionnement synonyme de gaspillage énergétique et financier). Dans l'idéal, il s'agirait pour le fournisseur de service d'attribuer exactement la bonne quantité de ressources au système à chaque instant. De manière schématique sur la Figure 6.3, cela reviendrait à obtenir une courbe bleue venant épouser la courbe noire de la demande en permanence et ce quels que soient les événements pouvant survenir à l'exécution du système. On parle parfois de « dimensionnement juste à temps » (i.e., *Just-in-Time Provisioning* [YQL09]).

> « *To be perfectly elastic, the resources exactly match the demand (no more or less), there is no time delay between detecting load changes and changing resourcing levels (resourcing is instantaneous), and you only pay for what you consume (charging is fine-grain consumption based) [Bre12].* » (P. Brebner)

La définition même de l'élasticité induit deux caractéristiques essentielles attendues d'un processus de gestion de l'élasticité. Il s'agit de la *précision* du dimensionnement et de la *réactivité* du dimensionnement [WHGK14]. Il est nécessaire de traiter ces deux dimensions afin d'éviter les aspects négatifs du sur-dimensionnement et du sous-dimensionnement évoqués ci-dessus.

> « *My biggest problem is elasticity. VM spin-up time... Ten to twenty minutes is just too long to handle a spike in Yahoo's traffic when big news breaks such as the Japan tsunami or the death of Osama bin Laden or Michael Jackson [Yah11].* » (Todd Papaioannou, Vice Président des architectures Cloud de Yahoo - 2011)

Cependant, même si plusieurs états de l'art sur l'élasticité ont été récemment proposés par la communauté Cloud [LBMAL14, CdCSR+15, ADPDM18], ils n'insistent pas suffisamment sur ces deux caractéristiques et le modèle actuel du Cloud est sujet à certaines limitations (physiques, conceptuelles

et techniques) qui rendent quasiment impossible une élasticité « parfaite ». Voici un aperçu des limites que nous avons identifiées :

- *Les ressources sont limitées.* On ne peut pas passer à l'échelle de manière infinie comme le promet le Cloud dans son argumentaire [ABLS13]. En effet, la quantité maximale de ressources accessibles peut parfois être atteinte et/ou imposée par le matériel (i.e., ressources physiques elles-mêmes), par le fournisseur IaaS (e.g., Amazon limitait à 20 le nombre d'instances EC2 par utilisateur et par zone en 2015) ou encore par des restrictions provenant des clients eux-mêmes (e.g., limite budgétaire) ;
- *Le temps d'initialisation des ressources informatiques n'est pas négligeable.* Le temps nécessaire au démarrage d'une machine virtuelle et de tous les services associés (e.g., OS, Tomcat, MySQL, ...) peut prendre plusieurs minutes [MH12] [2] ;
- *La granularité du modèle de facturation est trop importante.* En effet, la majorité des fournisseurs IaaS du marché ont adopté une facturation des ressources à l'heure [3] ce qui induit parfois un gaspillage financier pour les utilisateurs ayant des besoins ponctuels et courts. Cet aspect négatif va à l'encontre de la notion de paiement à l'usage (*pay-per-use*, *pay-as-you-go*) que prône le modèle du Cloud. On retrouve le terme *partial usage waste* dans la littérature pour définir cette sur-facturation imposée par la granularité du modèle économique adopté par les grands IaaS du marché [JWW+15].

6.2.4 Etat des lieux – Bilan

Alors que la prolifération des services Cloud hébergés dans les centres de données a un impact négatif important sur notre société (e.g., augmentation de l'empreinte carbone), la « *greenitude* » de leur nature reste discutable. Nous appellerons *greenitude* d'un service Cloud une métrique informelle qui dépend à la fois des techniques d'efficacité énergétique mises en place et de la qualité de l'énergie consommée par ces centres de données (i.e., énergie décarbonée ou non).

En réponse aux enjeux énergétiques du Cloud, la plupart des efforts visant à améliorer l'efficacité énergétique se sont focalisés sur le matériel ou encore sur l'infrastructure, c'est-à-dire sur les couches basses du Cloud [BAB12, OAL14, MOC+14]. Citons entre autres les travaux sur la consolidation dynamique de serveurs [BAB12, FME12], sur du matériel avec de meilleurs compromis puissance *vs* performance [VAN08, YHS+12], et sur des techniques logicielles pour l'ordonnancement (*scheduling*) conscient du type d'énergie [WHH+13, LOM17], etc. Bien que ces efforts soient nécessaires, l'objectif de diminuer la consommation énergétique et d'atténuer l'empreinte carbone est loin d'être atteint.

6.3 Pistes de travail

Notre objectif est à la fois d'optimiser l'empreinte énergétique du Cloud et de favoriser l'apparition de services « verts » sur le marché. Par service « vert », nous entendons la capacité du Cloud à fournir un service générant une faible empreinte carbone.

6.3.1 Hypothèses de départ

Le Cloud : une architecture logicielle dans les nuages

Dans la Section 2.4.2, nous avons fait l'hypothèse qu'un système logiciel est une architecture composée d'un ensemble de *briques* logicielles reliées entre elles par l'intermédiaire de *liaisons*.

2. En théorie, le dimensionnement vertical à chaud, c'est-à-dire sans interruption de service, permet de traiter cette limitation. Néanmoins, ce type de dimensionnement est très rarement proposé par les IaaS ou alors en imposant toute une pile technologique comme le choix de l'OS ou de l'hyperviseur [TL15].

3. Cette limite est aujourd'hui à relativiser car Amazon a annoncé en octobre 2017 un modèle de facturaion à la seconde ! cf. https ://aws.amazon.com/fr/about-aws/whats-new/2017/10/announcing-amazon-ec2-per-second-billing

Cette hypothèse de travail reste vraie pour l'informatique en nuage. Le Cloud est une architecture logicielle multi-couche (SaaS, PaaS, IaaS) organisée autour du concept de fournisseurs/consommateurs de services où les contrats SLA vont établir des spécifications sur les liaisons inter-couche (voire intra-couche).

Même si la Figure 6.4 montre essentiellement les relations consommateurs–producteurs Saas–IaaS et leurs SLA, on peut voir des *briques* services SaaS (*itineraries*), des *briques* machines virtuelles IaaS (s*mall/large*) et on peut imaginer qu'il existe des *liaisons* de type *est-composé-de* ou *est-hébergé-par* entre ces briques.

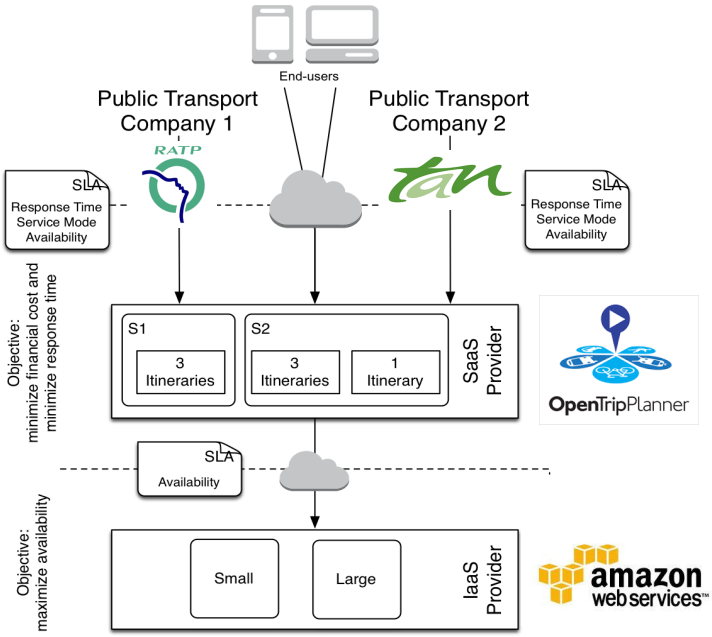


FIGURE 6.4 – Un exemple d'architecture multi-couche orientée services avec SLA [KADL14].

Ainsi, nos réflexions, notre approche et nos travaux sur la reconfiguration dynamique s'appliquent au domaine de l'informatique en nuage tout en prenant en compte les spécificités apportées par celle-ci (e.g., modèles de services, SLA). Par exemple, le concept d'élasticité et migration de machine virtuelle sont deux illustrations respectivement du principe d'auto-optimisation et d'un actionneur (cf. Section 4.1.3).

Pour peu que l'*architecture en nuages* propose les caractéristiques de réification, modularité, composabilité, support à l'adaptation non-anticipée qui constituent un prérequis à l'adaptation dynamique (cf. Section 3.4), notre objectif sera de définir un ou des *framework(s) context-aware* outillé(s) de langage(s) dédié(s) et adaptés à la problématique du Cloud. Même si ces artefacts auront pour objectif premier de répondre aux enjeux énergétiques de l'informatique en nuage, les problématiques de performance et de fiabilité resteront au cœur de nos préoccupations.

Relation logiciel – empreinte énergétique

> « *Software is getting slower more rapidly than hardware becomes faster [Wir95].* » (Niklaus Wirth - 1995)
>
> « *Bitcoin Mining Now Consuming More Electricity Than 159 Countries*[4]. » (Po-

4. `https://powercompare.co.uk/bitcoin`

wercompare.UK - 2017)

Alors que la majorité des travaux abordent la problématique sous l'angle du matériel et des couches basses des infrastructures [OAL14], force est de constater que la couche logicielle dimensionne l'infrastructure matérielle et conditionne la durée de vie des équipements. Pour s'en convaincre, nous pouvons donner quelques exemples orientés grand public :

— *Obsolescence accélérée des équipements.* Selon une étude de 2015 de l'agence fédérale allemande pour l'environnement, la durée de vie des équipements électroniques a été divisée par deux en 25 ans. La cause majeure pour les ordinateurs ou autres tablettes est liée aux logiciels installés qui sont de plus en plus gourmands en ressources. On parle d'*obésiciel.* Pour illustrer cette étude, il suffit d'analyser l'exemple des logiciels Microsoft (cf. Figure 6.5). Windows 7 + Office 2010 Pro nécessitent 15 fois plus de puissance processeur et 71 fois plus de mémoire vive que Windows 97 + Office 97 ! Autre appareil dans le collimateur de l'ADEME : les smartphones. Aujourd'hui, le taux de renouvellement des smartphones est de 20 mois alors que la durée de vie technique de ces équipements est très largement supérieure [CF16].

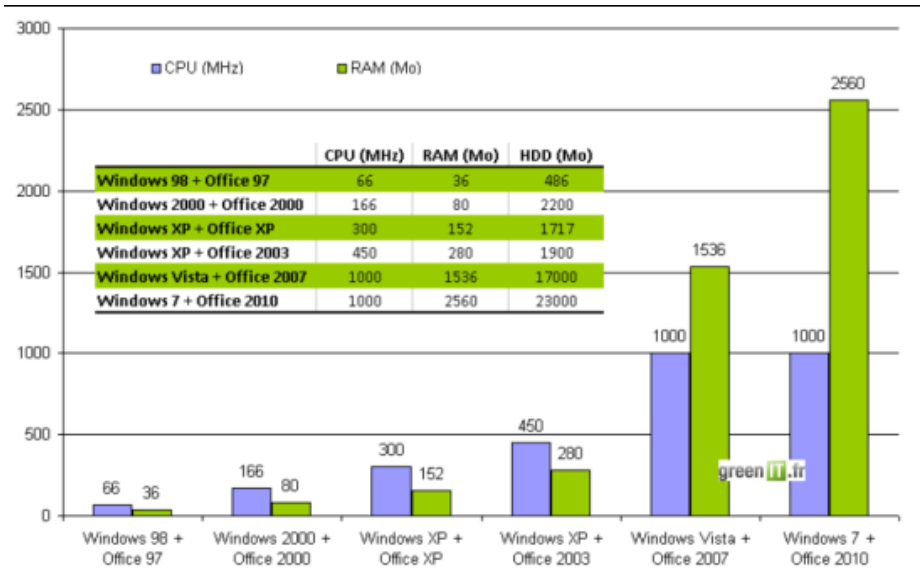


FIGURE 6.5 – Consommation en RAM et CPU de Windows dans le temps (source blog GreenIT).

— *Influence du langage de programmation.* Au début des années 2010, certains blogs français et américains affirment que le réseau social Facebook a écrit un compilateur PHP maison – nommé HipHop – qui lui permet de diviser sa consommation électrique par 2 ! A la clé, 60.000 tonnes de CO_2 en moins dans l'atmosphère et quelques millions de dollars économisés. Une information difficile à vérifier mais elle semblerait provenir au départ du site `http://developers.facebook.com`.

— *Configuration des smartphones.* Fin 2015, 1,5 milliards de smartphones étaient présents dans le monde. Malheureusement, une mauvaise configuration d'un smartphone entraine une consommation superflue de la batterie ce qui génère un gaspillage énergétique gigantesque. De plus, un smartphone nécessite au maximum 2 heures pour être pleinement chargé, inutile donc de le laisser connecté 8 heures pendant la nuit. L'ADEME estime qu'à l'échelle de toute la France, deux centrales nucléaires sont nécessaires à plein régime uniquement pour répondre au surcroît d'énergie requis par nos appareils mobiles en recharge inutile !

Ces impressionnantes illustrations indiquent bien une relation étroite entre logiciel et empreinte énergétique, et montrent la nécessité de traiter la problématique de la consommation du logiciel.

En 2018, à travers l'appel à projet Perfecto [ADE18], l'ADEME souhaite mobiliser et fédérer les entreprises et les acteurs de la recherche engagés dans l'éco-conception logicielle pour que la transition numérique converge avec la transition écologique.

Pour relever ces défis, le choix du langage de développememnt, de *green patterns* [NLS+11, PLL14], d'outils dédiés (e.g., en Java [SMM08] ou autres [5]) pour mieux programmer sont des éléments importants pour maîtriser l'empreinte énergétique [PHAH16]. Les travaux dans le domaine du *Green computing* proposent principalement des mécanismes d'adaptation basés sur la *paramétrisation* et les *techniques de mutation* alors que les *techniques d'habillage* sont peu utilisées (cf. Section 2.4.3).

— *Paramétrisation.* Les règles de bonnes pratiques (configuration, audit) permettent des gains substantiels de consommation d'énergie. Pour illustration, une entreprise nantaise [6] a récemment réalisé un audit pour une application sur smartphone Android dans le domaine militaire et cela a conduit à une correction ponctuelle de l'application. Le résultat est sans appel : il a permis un gain de 69% sur la consommation des smartphones en donnant un poids énergétique aux réglages.

— *Techniques de mutation.* Depuis longtemps, l'analyse et l'optimisation de code est la pratique consistant à améliorer l'efficacité du code informatique d'un programme. Ces améliorations permettent généralement au programme transformé de s'exécuter plus rapidement et avec moins de ressources. La ré-écriture d'algorithme de complexité inférieure avec structures de données adaptées, les optimisations à la volée réalisées par les compilateurs, ... sont des exemples d'adaptation par techniques de mutation.

Pourtant, comme il a été montré aux chapitres précédents pour les architectures à base de composants, il serait intéressant d'investiguer les techniques d'habillage pour les *architectures en nuages* pour permettre une urbanisation plus efficace du code dans l'espace et dans le temps.

6.3.2 Effacement de la consommation énergétique du logiciel à l'exécution

> « *...the energy consumption of Clouds cannot be viewed independently of the QoS that they provide, so that both energy and QoS must be managed jointly [BSC+17]* »
> (Rajkumar Buyya - 2017)

Notre première idée est de trouver un équilibre, un compromis (*trade-off*) entre le niveau de QoS fourni par le logiciel dans les nuages (e.g., performance, précision, *user experience*, maintenabilité, ...) et la quantité de ressources nécessaires à son fonctionnement.

A l'instar de l'« effacement de la consommation électrique », nous militons pour une *éco-conception* du logiciel qui implique le consommateur final. Rappelons que l'effacement de consommation électrique consiste, en cas de déséquilibre offre/demande d'électricité à réduire provisoirement (en période de pointe journalière et/ou saisonnière) la consommation physique d'un site ou d'un groupe d'acteurs donnés par rapport à sa consommation « normale ». Ces acteurs qui sont prévenus et contractuellement volontaires s'y retrouvent grâce à une incitation tarifaire. L'effacement a pris une importance nouvelle avec l'émergence des *smart grids* et de la transition énergétique.

Ainsi, nous proposons au logiciel de « s'effacer » ou plutôt de « se déformer » en cas de déséquilibre offre/demande pour ne pas solliciter de nouvelles ressources (e.g., machine virtuelle). Ceci implique de revoir la granularité des logiciels en mode SaaS pour favoriser une plus grande modularité et le choix de compromis réalisé à la volée. *Le logiciel SaaS devient lui-même élastique.* Cette approche sera dirigée par les SLA (*SLA-driven*), contractualisés en amont entre les différents acteurs (e.g., entre SaaS et IaaS).

Si l'on revient à la discussion liée à l'élasticité et à la dimension *précision* (cf. Section 6.2.3), il y a maintenant beaucoup plus de chance sur la Figure 6.3 que la courbe bleue vienne épouser la courbe noire de la demande puisque notre objectif est de réaliser une déformation intelligente du modèle

5. Par exemple, le produit Greenspector : `https://greenspector.com`
6. Non dévoilée pour des raisons de confidentialité.

de services SaaS pour qu'il puisse solliciter à bon escient les ressources tout en rendant le mieux possible le service attendu. Comme nous le verrons dans le chapitre suivant, nous résoudrons par la même occasion la dimension *réactivité* de l'élasticité puisqu'une brique logicielle plus petite (e.g., composant SaaS) se reconfigure plus vite d'une brique plus grosse (e.g., machine virtuelle IaaS).

Des démarches assez analogues à la nôtre peuvent être trouvées dans la littérature récente. Citons entre autres, [CHP+16] qui proposent une compromis QoS (performance) *vs* émission de CO_2, [KMrHR14, XDB16] qui proposent le *brownout* (délestage) pour garantir la fiabilité des services Cloud sans recourir au sur-dimensionnement mais au détriment de l'expérience utilisateur. Enfin, pour permettre la réduction de l'empreinte carbone, [ASK14] militent pour la spécification des métriques *green* dans les contrats SLA au même titre que les autres métriques de QoS.

6.3.3 Synergie entre les modèles de service

Le *Cloud computing* suit une organisation en couches, inter-dépendante les unes des autres, selon un modèle consommateur-producteur. Les décisions prises à un niveau peuvent générer des interférences sur les systèmes administrés à un autre niveau. Par exemple, une application SaaS peut demander plus de ressources alors que l'hyperviseur IaaS tente une consolidation dynamique.

Aussi la reconfiguration dynamique du Cloud doit se faire en intelligence pour éviter les interférences, les objectifs contradictoires entre prises de décision locales. Selon le modèle du Cloud privé *vs* public [MG11], une reconfiguration multi-couche (*cross-layer*) n'est pas toujours possible : chaque niveau peut être administré par sa propre boucle autonomique (cf. Figure 6.6). Cependant, nous pouvons proposer une synergie entre les niveaux grâce à des modèles de synchronisation et de coordination inter-boucle [ASPG16]. Non seulement cette synergie peut apporter une plus grande fiabilité dans la reconfiguration dynamique d'une *architecture en nuages* mais elle peut également créer des opportunités comme par exemple un nouveau modèle de *dynamic pricing* des ressources [AA15] basé sur un « dialogue » entre consommateur SaaS et fournisseur IaaS.

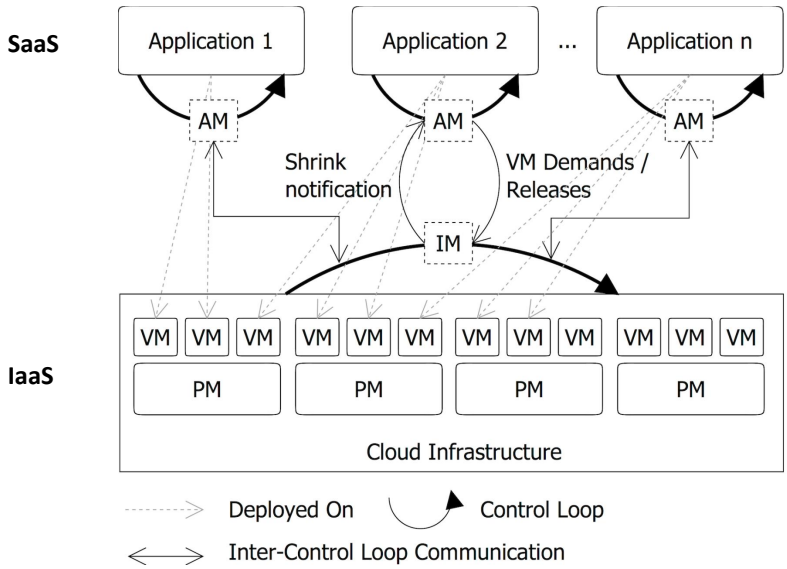


FIGURE 6.6 – Communication inter-boucle multi-couche [AdOLLM12].

6.3.4 Intégration des énergies renouvelables

L'efficacité énergétique à elle seule ne va pas réduire l'empreinte carbone car la consommation d'énergie du Cloud continuera de croître. L'intégration des énergies renouvelables aux centres

de données peut être une mesure complémentaire pour accompagner les techniques d'efficacité énergétique [MOC+14, OAL14] et ainsi réduire l'empreinte carbone et assurer la *greenitude* des services Cloud.

Cependant, la plupart des centres de données ne sont connectés qu'au réseau électrique où l'électricité est produite par la combustion d'énergies fossiles, qui sont des approches de production d'énergie à forte intensité de carbone. Ainsi, les centres de données doivent se tourner vers d'autres alternatives « vertes », c'est à dire des sources d'énergies renouvelables sur site ou hors site. Enfin, le défi considérable de l'intégration des sources d'énergie verte dans les centres de données est qu'elles sont intermittentes par nature, donc pas toujours disponibles. Aussi, comment garantir/respecter un « green SLA » pour un fournisseur de service ?

6.4 Bilan : vers un Cloud frugal

Le Cloud computing, au travers de son modèle en couche et de l'accès à ses services à la demande, a bouleversé la façon de gérer les infrastructures et la manière de distribuer les logiciels. Le Cloud, facilitant la mutualisation des ressources dans des grands centres de données, a permis de rationaliser la consommation d'énergie mais paradoxalement (Paradoxe de Jevons) n'empêche pas la prolifération des centres de données, l'augmentation de la consommation énergétique et de l'empreinte carbone.

En s'inspirant du concept d'innovation frugale (*jugaad*) [APR13], nous proposons le concept d'éco-élasticité logicielle pour le Cloud qui permet de déformer le logiciel en vue de renvoyer une « valeur suffisante » au client mais moins énergivore. Cette valeur sera négociée en amont à travers des contrats SLA à l'instar de ce qui se fait pour l'effacement de la consommation électrique. Les artefacts permettant de gérer ce type de reconfiguration pour les *architectures en nuages* restent à développer et seront l'objet du prochain chapitre.

Malgré tout, l'intégration des sources d'énergie « verte » dans les centres de données est le seul moyen pour diminuer l'empreinte carbone et augmenter la *greenitude* des services Cloud. Une contribution de recherche qui va dans ce sens conclura le prochain chapitre.

Chapitre 7

Eco-élasticité logicielle pour un Cloud frugal

Ce chapitre présente un certain nombre de nos contributions dans le domaine du Cloud frugal, notamment les contrats de service SLA (*Service Level Agreement*) dédié au Cloud, l'éco-élasticité multi-couche dans le Cloud ainsi qu'une consommation intelligente de l'énergie « verte » par le Cloud. Ces travaux ont été menés dans le cadre des thèses successives de Frederico Alvares [ADOJ13], Yousri Kouki [Kou13], Simon Dupont [Dup16] et Sabbir Hasan [Has17]. Ils ont apporté de nombreux résultats dans les projets ANR MyCloud (2010-2014), FSN OpenCloudware (2012-2014), Labex CominLabs EPOC (2013-2017) et sont à l'origine d'une vingtaine de publications dans des revues et des conférences internationales.

7.1 Vers une auto-administration fine de la gestion de ressources

Nous décrivons rapidement le *sujet* et l'*acteur* de l'adaptation sans détailler les mécanismes de l'adaptation (cf. Section 2.4) qui seront propres à chaque travail de thèse.

7.1.1 Architecture en nuages

Comme stipulé dans la Section 6.3.1, le Cloud est une architecture logicielle multi-couche organisée autour du concept de fournisseurs/consommateurs de services où les contrats SLA vont établir des spécifications sur les liaisons inter-couche et intra-couche (cf. Figure 7.1).

Couche XaaS

Si l'on observe la *pile logicielle* du Cloud, chacun de ses services à un niveau peut jouer le rôle de consommateur d'autres services situés à un niveau inférieur et être fournisseur de service pour d'autres services situés à un niveau supérieur (ou éventuellement directement pour les utilisateurs finaux). La structure *top-down* de la pile Cloud empêche d'avoir des cycles dans la relation consommateur-fournisseur (cf. Figure 7.1).

Chaque couche XaaS (*Anything as a Service* [Sch09]) du nuage est reconfigurable si elle répond aux propriétés de réification, modularité et composabilité présentées dans la Section 3.1. La couche IaaS (*Infrastructure as a Service*) est sans doute la couche répondant le mieux à ces critères puisqu'elle est réifiée et composée de machines virtuelles, de machines physiques, de clusters de machines, de serveurs de données, etc. directement accessibles via des APIs proposées par les fournisseurs IaaS (e.g., Amazon). Dès 2010, le consortium Open Grid Forum propose OCCI (*Open Cloud Computing Interface* [1]), une API standardisée de gestion à distance de l'infrastructure interne d'un fournisseur

1. http://occi-wg.org/

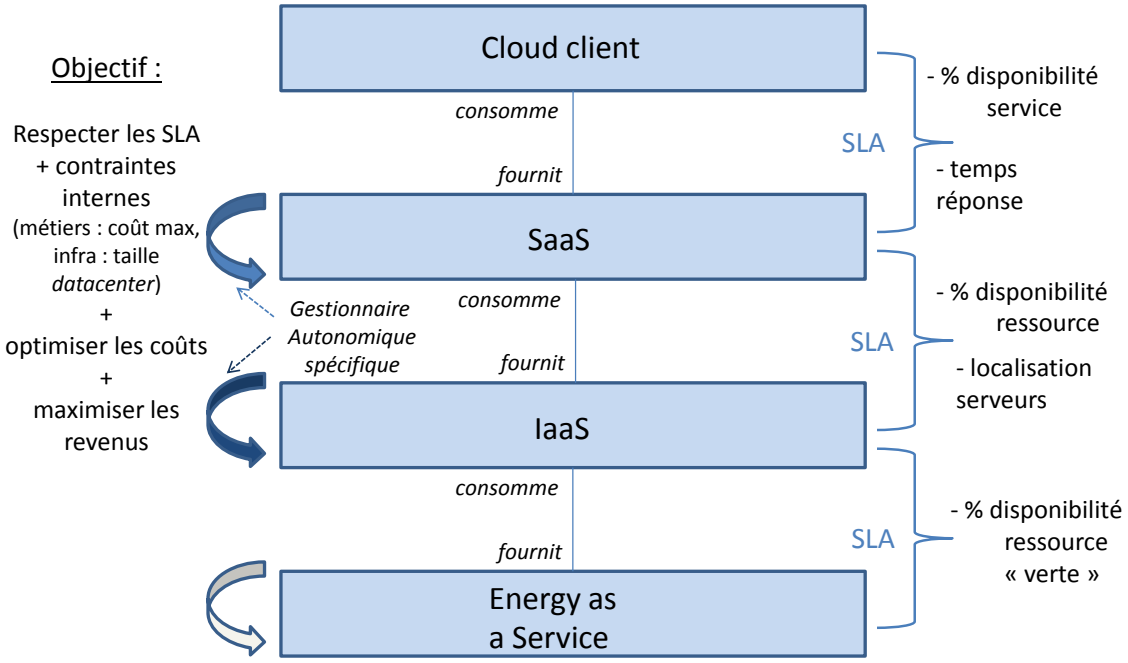


FIGURE 7.1 – Architecture en nuages.

IaaS. Concernant les autres couches du Cloud (e.g., PaaS et SaaS), de par leur spécificité, il est difficile de se positionner sur les propriétés de modularité et de composabilité. En effet, la couche PaaS (*Platform as a Service*) va regrouper des outils de développement, de gestion de base de données, des intergiciels dont la granularité, l'hétérogénéité ne permettent pas d'envisager facilement une standardisation de son administration (et donc de sa (re)configuration). La couche SaaS (*Software as a Service*) quant à elle présente des applications diverses et variées et leurs capacités d'adaptation dépendent de leur conception et du langage hôte qui a servi à leur développement.

Cependant, nous assistons depuis quelques années à une tentative d'extension [YT16] et de redesign du standard OCCI [MBP+15] pour le rendre plus facilement administrable quelque soit la couche XaaS. Nous reviendrons sur cette notion de généricité dans la Section 8.2.1.

Enfin, comme on peut le remarquer sur la Figure 7.1, nous introduisons une nouvelle couche dans la *pile logicielle* du Cloud : EaaS pour *Energy as a Service*. En effet, deux raisons nous conduisent à introduire cette nouvelle couche et l'acteur associé, i.e., l'*energy provider*. D'une part, la libéralisation du marché de l'énergie qui incite à changer de fournisseur d'énergie dans la durée, celui-ci venant avec un contrat SLA qui peut-être différent du précédent fournisseur. D'autre part – et surtout – l'émergence des énergies renouvelables et de nouveaux fournisseurs d'énergie qui peuvent proposer un « green SLA » à faible empreinte carbone.

SLA

Les SLA spécifient les relations fournisseurs-consommateurs de services. Pour le fournisseur, le contrat SLA porte une information importante car il donne des valeurs qui vont lui permettre de prévoir le dimensionnement des ressources de calcul ou de stockage nécessaire à la fourniture du service.

L'expression du contrat SLA pour le Cloud a fait l'objet de la thèse de Yousri Kouki [Kou13] et sera présentée dans le Section 7.2. Le SLA devra être capable de traiter les métriques quels que soient les niveaux XaaS impliqués dans ce contrat. Ainsi, en plus des métriques standard comme la

disponibilité et le temps de réponse, il faudra être capable de stipuler et de gérer des métriques liées au *Green Computing* comme par exemple « les émissions de CO_2 maximales autorisées d'un service » [ASK14].

7.1.2 Auto-configuration et Auto-optimisation

Une boucle autonomique MAPE-K est intégrée au niveau de chaque couche pour rendre celle-ci auto-adaptable. La couche XaaS est l'élément administré (*management element*, cf. Section 4.1.3) qu'il faut optimiser pour diminuer son empreinte énergétique.

Comme évoqué dans la Section 6.2.3, le principal défi pour le fournisseur de services est d'éviter le sous-dimensionnement synonyme de violation de SLA tout en évitant également le sur-dimensionnement synonyme de gaspillage énergétique et financier. Alors comment gérer la capacité des ressources (e.g., taille d'un *cluster*, occupation mémoire d'un serveur) et des services (e.g., disponibilité, temps de réponse) pour répondre à ce défi ? Nous avons opté pour une approche dirigée par les SLA (*SLA-driven*) pour gérer le dimensionnement, c'est-à-dire que le SLA va constituer la variable de *précision* du dimensionnement (cf. Section 6.2.3). L'auto-configuration dirigée par les SLA va être guidée par des politiques d'adaptation maximisant les revenus et minimisant les dépenses (cf. Figure 7.1) pour devenir alors une auto-optimisation.

Enfin, pour construire la boucle MAPE-K, il est nécessaire d'une part de développer les *sensors* et *effectors* adéquats qui font le lien entre l'élément administré et le gestionnaire autonomique ; puis, de réaliser les différentes composantes de la boucle, i.e., *Monitor, Analyse, Plan, Execute, Knowledge*. Dans la suite du manuscrit, nous décrirons quelques exemples de boucles autonomiques permettant de réaliser un Cloud frugal.

7.2 Etablir un contrat entre couches XaaS : le langage CSLA

Dans sa thèse [Kou13], Yousri Kouki propose une solution complète pour la gestion des contrats SLA dans les architectures en nuages.

CSLA (*Cloud Service Level Agreement*) est un langage dédié à la définition de contrat SLA pour le Cloud [KL12b]. CSLA permet de définir les SLA pour n'importe quelle couche XaaS et traite de la problématique de l'incertitude de la qualité de service (QoS) dans un environnement dynamique et imprévisible comme le Cloud. En effet, outre la définition standard des contrats de service, CSLA propose de nouvelles fonctionnalités telles que l'expression de la dégradation de la QoS, introduisant ainsi un support langage pour une gestion fine du dimensionnement des ressources, de l'élasticité dans le Cloud.

Pour évaluer CSLA, Yousri Kouki propose HybridScale un *framework* de dimensionnement autonomique dirigé par les SLA. Il implémente l'élasticité de façon hybride : approche réactive *vs* pro-active, dimensionnement vertical *vs* horizontal et multi-couche (application et infrastructure). Cette contribution est validée expérimentalement sur Amazon EC2 [Kou13].

7.2.1 Motivations et survol du langage

L'élasticité est l'élément intrinsèque qui différencie le Cloud des paradigmes informatiques traditionnels, car il permet aux fournisseurs de services d'ajuster rapidement les ressources pour absorber la demande et garantir ainsi un niveau de qualité de service (QoS) respectueux des SLA définis avec leurs clients. Cependant, en raison de limitations techniques et conceptuelles (e.g., temps d'initialisation de ressources non négligeable, charge de travail imprévisible), il devient difficile pour les fournisseurs de services de garantir des propriétés de QoS et des violations de SLA peuvent se produire. Un contrat SLA pour le Cloud doit être conçu pour des ressources hétérogènes et volatiles dans un environnement hautement imprévisible et dynamique. Les langages SLA existants tels que

WSLA [KL03] et WS-Agreement [ACD+05] ne prennent pas en charge la nature dynamique du Cloud.

Ainsi, nous proposons CSLA (*Cloud Service Level Agreement*), un langage SLA pour exprimer finement les contrats SLA et traiter les violations SLA dans le contexte des services Cloud. Outre la définition formelle standard des contrats (validité, parties impliquées, définition des services et garanties), le langage CSLA propose de nouvelles propriétés (dégradation de la QoS, dégradation de services, modèle de pénalité). En effet, CSLA permet l'expression d'objectifs de niveau de service (*Service Level Objectives* ou SLO) sophistiqués avec de nouvelles caractéristiques telles que la confiance (*confidence*) et le flou (*fuzziness*) pour traiter l'incertitude de la QoS : (i) le flou définit le degré de marge acceptable autour du seuil d'une expression SLO ; (ii) la confiance définit le pourcentage de conformité des clauses. De plus, les services Cloud peuvent fonctionner dans différents modes (e.g., affichage 2D vs 3D, différents niveaux de sécurité), chacun consommant plus ou moins de ressources. On parlera de *dégradation de service* ou d'élasticité de la partie applicative. Cette propriété permet aux fournisseurs de services de disposer de plus de flexibilité pour planifier, démarrer ou non des ressources supplémentaires. Enfin, tout en restant dans le cadre d'un contrat SLA, CSLA propose un modèle de pénalité gérant à la dégradation. Ce modèle aligne les pénalités avec la dégradation de service/QoS afin de fournir un compromis (*trade off*) entre le prix du service et la qualité rendue, ce qui est à la fois attrayant pour les consommateurs finaux et rentable pour les fournisseurs de services Cloud.

Avec CSLA, notre objectif est de rendre les contrats plus flexibles pour absorber les violations et d'augmenter les capacités d'auto-adaptation du Cloud et les possibilités d'élasticité via les nouvelles propriétés. Ainsi, CSLA permet aux fournisseurs de services de maintenir la satisfaction de leurs clients tout en minimisant les coûts des ressources. L'art du compromis va être un élément fondamental pour permettre l'effacement de la consommation énergétique du logiciel à l'exécution (cf. Section 6.3.2).

Comment évaluer les objectifs de niveau de service SLO dans CSLA ?

Un SLO est un prédicat qui a généralement l'une des formes suivantes : une métrique QoS (e.g., temps de réponse) avec une valeur supérieure/inférieure à un seuil donné (e.g., 3 s). Dans CSLA, nous enrichissons la définition SLO avec les caractéristiques de *fuzziness* et de *confidence*. Afin d'évaluer un objectif SLO, une évaluation initiale permet de classer le prédicat comme *idéal* (seuil respecté), *dégradé* (seuil respecté en marge floue) ou *inadéquat* (seuil non respecté même en marge floue) (voir la Figure 7.2).

Nous distinguons deux types d'évaluation : (i) l'évaluation par intervalle (*per-interval*), dans laquelle l'évaluation est effectuée à la fin de chaque intervalle (e.g., une fenêtre temporelle de 30 minutes) ; (ii) l'évaluation à la demande (*per-request*), dans laquelle l'objectif est évalué pour chaque demande. A la fin de la fenêtre temporelle, une évaluation finale permet de vérifier un objectif SLO en appliquant les pourcentages de flou et de confiance à l'évaluation initiale. Elle permet d'identifier la dégradation non acceptée/acceptée et les cas inadéquats. En d'autres termes, l'évaluation finale absorbe ou notifie les violations.

Syntaxe concrète : exemple CSLA

La syntaxe CSLA est définie en fonction de la grammaire générée à partir du méta-modèle CSLA (voir la thèse de Yousri Kouki [Kou13] pour plus de détails). Nous utilisons XML comme format de représentation. Le Listing 7.1 présente un extrait de fichier CSLA décrivant les termes des garanties et les pénalités pour un SLA entre un fournisseur SaaS et son client concernant le service `S1`.

Dans cet exemple, deux SLO sont composées à l'aide de l'opérateur "and" : un SLO de performance contractualisant la QoS du temps de réponse et un SLO contractualisant l'utilisation de la dégradation de service. Le SLO performance (lignes 6-11) spécifie que pour chaque intervalle de 3 min dans la fenêtre de 30 min (exprimé dans la variable `Mon-1`, non détaillée ici), le temps de réponse maximum

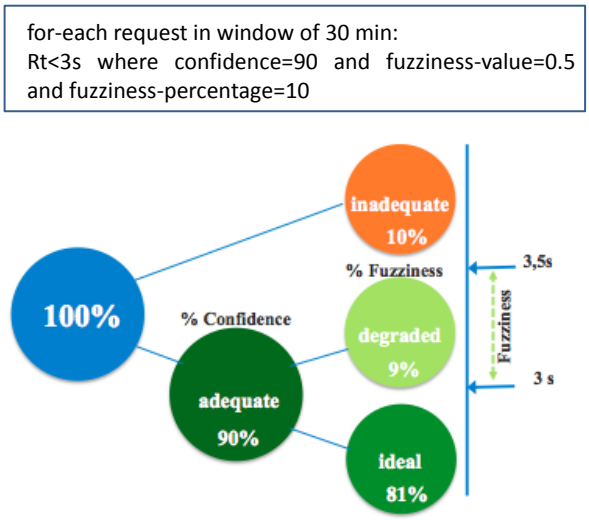


FIGURE 7.2 – Exemple d'évaluation d'un SLO dans CSLA.

(Rt) doit être inférieur à 3 s si la taille des données est inférieure à 1 To. Cet objectif devrait être atteint tous les jours entre 8h00 et 10h00 (exprimé dans la variable Sch-Morning, non détaillée ici) pendant la durée du contrat. Il garantit au moins 99% des demandes de service S1 (*confidence*) parmi lesquelles 10% peuvent être dégradées (*fuzziness-percentage*), avec une marge de 0,2 s (*fuzziness-value*). Les lignes 12-14 spécifient le SLO dégradation de service. Le mode de dégradation de fonctionnalité doit être utilisé dans 10% des demandes pour le service S1. Il est à noter que la dégradation de service est gérée comme n'importe quel autre SLO car elle définit un objectif d'utilisation.

La deuxième partie présente les pénalités (lignes 16-31). Elles sont appliquées en cas de violation des SLA pour indemniser les clients du service, c'est-à-dire que des pénalités réduisent le prix du service. La réduction peut être appliquée à taux constant ou variable. Dans ce dernier cas, le prix de la demande est modélisé comme fonction linéaire [IGC04]. Une violation du mode SLO (lignes 25-30) implique une pénalité égale à 0,1 euro / demande alors que la pénalité du SLO performance (lignes 17-24) dépend du délai. Pour chaque pénalité, une procédure indique l'acteur en charge de la notification de violation (e.g., fournisseur), la méthode de notification (e.g., courrier électronique) et la période de notification (e.g., 7 jours).

```
1  <csla:terms>
2    <csla:term id="T1" operator="and">
3      <csla:item id="responseTimeTerm"/>
4      <csla:item id="modeTerm"/>
5    </csla:term>
6    <csla:objective id="performanceSLO" priority="1" actor="provider">
7      <csla:precondition policy="Required">
8        <csla:description> Data size less than 1 TB</csla:description>
9      </csla:precondition>
10     <csla:expression metric="Rt" comparator="lt" threshold="3" unit="second"
           monitoring="Mon-1" schedule="Sch-Morning" confidence="99" fuzziness-value
           ="0.2" fuzziness-percentage="10"/>
11   </csla:objective>
12   <csla:objective id="modeSLO" priority="2" actor="provider">
13     <csla:expression metric="Mu(S1-M2)" comparator="lt" threshold="10" unit="%"
           monitoring="Mon-1" confidence="99" fuzziness-value="2"
           fuzziness-percentage="5"/>
14   </csla:objective>
15 </csla:terms>
```

```
16 <csla:penalties>
17   <csla:Penalty id="p-Rt" objective="responseTimeTerm" condition="violation"
       obligation="provider">
18     <csla:Function ratio="0.5" variable="delais" unit="second">
19       <csla:Description> ... </csla:Description>
20     </csla:Function>
21     <csla:Procedure actor="provider" notificationMethod="e-mail"
         notificationPeriod="7 days">
22       <csla:violationDescription/>
23     </csla:Procedure>
24   </csla:Penalty>
25   <csla:Penalty id="p-Mu" objective="modeTerm" condition="violation" obligation="
       provider">
26     <csla:Constant value="0.1" unit="euro/request"/>
27     <csla:Procedure actor="provider" notificationMethod="e-mail"
         notificationPeriod="7 days">
28       <csla:violationDescription/>
29     </csla:Procedure>
30   </csla:Penalty>
31 </csla:penalties>
```

Listing 7.1 – Exemple CSLA.

7.2.2 Evaluation et expérimentations

Les travaux autour de CSLA ont abordé la problématique de l'efficacité énergétique à travers la problématique de dimensionnement de ressources dirigé par les SLA. Nous avons à la fois proposé des solutions de dimensionnement statique des ressources (*capacity planning*) mais également des résultats sur l'auto-dimensionnement à l'exécution. Dans le premier cas, il s'agit de trouver la configuration initiale qui permet de dimensionner un système pour qu'il puisse rendre le service attendu en respectant les contrats SLA. Dans le second cas, il s'agit de reconfigurations à partir de la configuration initiale (garantissant toujours les SLA).

Dimensionnement statique

Une fois le langage CSLA formalisé, nos travaux ont d'abord porté sur le dimensionnement statique des ressources [KLS11, KL12a, KL13].

Planification de capacité dirigée par SLA. Dans [KL12a, KL13], nous proposons un modèle analytique, basé sur la théorie des files d'attente, qui fournit une estimation précise de la performance, la disponibilité et le coût d'un service SaaS avec une configuration donnée pour une charge de travail (*workload*) déterminée (cf. Figure7.3 (b)). La théorie des files d'attente est largement utilisée pour l'évaluation des performances des systèmes [MDA04, UPS+07]. Pour ce faire, nous modélisons les services Cloud déployés sur une architectrue n-tiers en utilisant un modèle de réseau de files d'attente fermé et nous proposons une extension d'un algorithme MVA (Mean Value Analysis) [RL80] pour prendre en compte le SLA (cf. Figure7.3 (a)). Notre méthode est suffisamment générale pour modéliser n'importe quelle application Web déployée sur le Cloud.

Puis, nous avons proposé une méthode de planification de capacité dirigée par SLA – nommée *RightCapacity* [Kou13] – qui utilise le dimensionnement multi-couche (*cross-layer*). Les actionneurs peuvent être résumés de la façon suivante :

- créer/détruire des ressources IaaS
- utiliser la dégradation de service SaaS
- ajuster le contrôle d'admission (MPL) du réseau de files d'attente

Cette méthode intègre également des politiques qui vont influencer le calcul de la capacité optimale (e.g., nombre max de serveurs côté IaaS). L'objectif de la planification de capacité est de calculer une configuration optimale d'une application Cloud pour un modèle de charge de travail

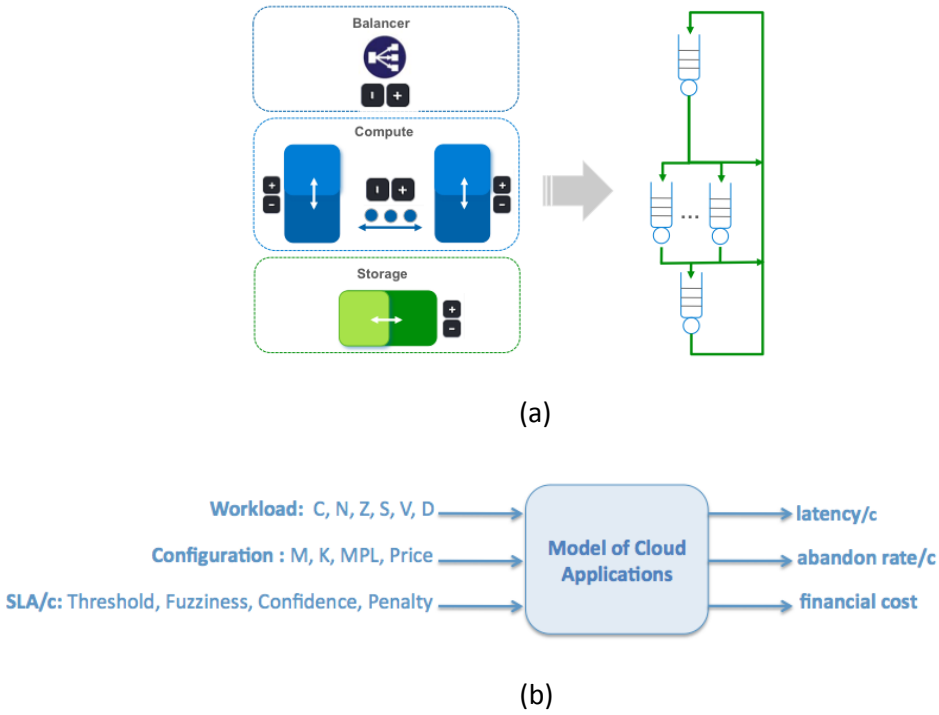


FIGURE 7.3 – Modélisation des applications Cloud

donné. L'algorithme de planification de la capacité est divisé en deux parties : une première partie qui calcule une configuration préliminaire qui garantit les contraintes SLA et une seconde partie qui minimise le coût du service via les politiques de dimensionnement (voir [Kou13] pour plus de détails).

Dans [KL13], à travers deux scénarios de charge de travail très différents, les résultats montrent que – grâce aux politiques de dimensionnement, l'utilisation des propriétés de *confidence* et *fuzziness* et l'élasticité multi-couche permise par la dégradation de services de CSLA sur l'application – on peut réaliser jusqu'à 30% d'économies tout en provoquant le minimum de violations (inférieur à 1%).

Dépendances SLA. Dans la *pile logicielle* du Cloud, la plupart des acteurs se retrouvent à jouer le rôle de consommateur-producteur (sauf pour l'acteur situé en bas et celui en haut de la pile). Ainsi, par exemple, l'acteur SaaS est impliqué dans *deux* contrats SLA (cf. Figure 7.4) :

— SLA_S entre l'utilisateur final et lui en tant que fournisseur SaaS ;
— SLA_R entre le fournisseur IaaS et lui en tant que consommateur SaaS.

Il se retrouve donc confronté au problème suivant : « Comment calibrer les bonnes valeurs de mes SLA pour déterminer les ressources appropriées (e.g., nombre, taille) requises afin de respecter les exigences de QoS de mes clients sans que cela ne me coûte trop cher ? ». Dans [KLS11], cette question est modélisée comme un problème d'optimisation multicritère. Nous proposons un frawework – basé sur le solveur de contraintes Choco [PFL17] – qui prend en entrée les préférences de l'utilisateur final (e.g., temps réponse > disponibilité > coût), le SLA_R (avec le IaaS) et qui permet de calibrer par défaut – ou sur mesure grâce aux propriétés *fuzziness* et de *confidence* – le SLA_S ainsi que la sélection exactes des ressources côté fournisseur IaaS.

En conclusion, CSLA constitue une solution de calibrage de SLA. Même si [KLS11] met l'accent sur l'acteur SaaS, l'approche est totalement générique et reproductible quelque soit la couche XaaS.

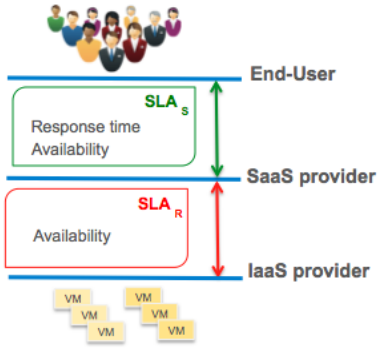


FIGURE 7.4 – Dépendances entre SLA

Auto-dimensionnement à l'exécution dirigé par les SLA

Dans [KADL14], nous présentons un exemple complet de gestion de l'élasticité du Cloud basé sur CSLA. Nous considérons une élasticité multi-couche (*cross-layer*), entre SaaS et IaaS, basée sur la fonctionnalité de dégradation de service proposée par CSLA. Afin d'évaluer l'approche proposée, nous avons utilisé l'exécution d'un scénario de cas d'utilisation réel déployé sur une infrastructure physique réelle. Ce travail a été réalisé conjointement par Yousri Kouki et Simon Dupont, autre doctorant de l'équipe [KADL14].

Cas d'utilisation. Le prototype implémente l'exemple présenté dans la Figure 6.4 à la Section 6.3.1 en suivant une architecture n-tiers (cf. Figure 7.5).

Application. Nous avons développé une application Web de transport public basée à la fois sur OpenTripPlanner [2], une plateforme *open source* pour la planification d'itinéraires de voyage multi-modaux, et l'*open data* de la ville de Nantes (liste des arrêts, horaires et parcours de tous les bus et tramways) [3]. Notre application offre des capacités d'adaptation puisque le composant de calcul d'itinéraire – très *CPU-intensive* – fournit deux niveaux de réponse : le premier niveau de réponse renvoie trois itinéraires différents alors que le second ne renvoie qu'un seul itinéraire (i.e., *dégradation de service*). L'idée de base est de faire un compromis entre les modes de service (nombre d'itinéraires) et les coûts (financier et énergétique) liés aux ressources informatiques.

L'application est organisée en architecture 2-tiers. Le premier tier correspond au niveau métier basé sur une petite machine virtuelle *vm-app*. Un serveur Apache 5 répondant aux requêtes HTTP entrantes est installé pour chaque application *vm-app*, tandis que le deuxième tier correspond au composant de calcul d'itinéraire OTP, déployé sur une machine virtuelle importante *vm-otp*, accompagné de l'*open data*. Toutes les *vm-app* sont assignées au répartiteur de charge *app-lb* tandis que *vm-otp* sont assignées au répartiteur de charge *otp-lb*. De cette façon, il est possible de mettre à l'échelle à la fois l'application métier et le composant de calcul d'itinéraire.

Infrastructure. Nous avons considéré une petite machine virtuelle (configurée avec un cœur CPU de 2 GHz et 1 Go de RAM) et une grande (configurée avec deux cœurs CPU 2 GHz et 4 Go de RAM). Deux répartiteurs de charge (LB) ont été déployés pour répartir les charges de travail sur les machines virtuelles (VM). Les VM et les LB s'appuient sur un outil de gestion de Cloud

2. `http://www.opentripplanner.org/`
3. `https://data.nantes.fr/donnees/detail/arrets-horaires-et-circuits-tan/`

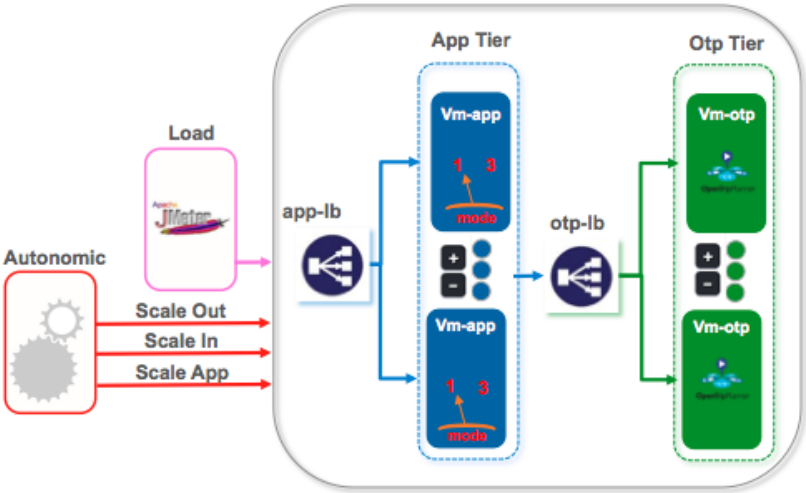


FIGURE 7.5 – Une architecture n-tiers dans le Cloud [KADL14].

largement adopté : Apache CloudStack [4]. Il utilise des hyperviseurs existants pour la virtualisation. L'hyperviseur de VMware est utilisé dans nos expériences. Nous avons préféré cette solution de Cloud privé à un Cloud public comme Amazon pour maitriser de A à Z le démonstrateur.

Deux autres petites instances VM sont ajoutées au banc d'essai sans être connectées au LB. La première, appelée *vm-autonomic*, héberge le gestionnaire autonomique du système fournissant des interfaces pour surveiller et agir sur le système. La seconde, appelée *vm-load*, est utilisée à des fins d'injection de charge. Elle contient une application Apache JMeter [5] qui simule une variation de charge sur l'application métier et permet d'analyser les performances globales sous différents types de charge de travail.

Gestionnaire autonomique. L'objectif de ce travail n'était pas de concevoir des boucles autonomiques sophistiquées. Aussi, nous avons opté pour une approche impérative pour modéliser la logique d'adaptation (cf. Section 3.2) et nous implémentons la reconfiguration via des règles à base de seuil. L'objectif n'était pas non plus de concevoir des boucles autonomiques dédiées à chaque couche XaaS. Aussi, nous réalisons la gestion autonomique avec une unique boucle regroupant toutes les fonctions MAPE-K [6] :

- *Monitor* : la surveillance repose sur une solution basée sur un journal (*log*). Lorsqu'une *vm-app* ou une *vm-otp* est déployée, elles enregistrent différentes métriques de bas niveau (e.g., charge du processeur, mémoire) correspondant à leur état de santé. *vm-app* enregistre les métriques de haut niveau (e.g., temps de réponse, codes d'état HTTP) correspondant au journal d'accès du serveur Apache Tomcat.
- *Analyze* : les métriques VM collectées sont centralisées dans *vm-autonomic*. À partir de ces messages reçus, nous avons identifié certains événements spécifiques en utilisant WildCAT (cf. Section 4.3). Avec WildCAT, il devient facile de déclencher des actions sur des conditions particulières grâce à des règles à base de seuil décrites ci-dessous ;
- *Execute* : nous avons implémenté des APIs pour agir à la fois sur les couches IaaS et SaaS. Au niveau de l'infrastructure, nous sommes en mesure de déployer (*Scale Out*) / supprimer (*Scale*

4. `https://cloudstack.apache.org/`
5. `https://http://jmeter.apache.org/`
6. La définition et la coordination de boucles autonomiques indépendantes, dédiées à chaque couche XaaS a fait l'objet de la thèse de Frederico Alvares (cf. Section 7.3).

In) des machines virtuelles à l'aide de l'API CloudStack. Au niveau du logiciel, nous pouvons commander une modification de configuration à l'application en lui demandant de modifier certaines fonctionnalités de service (*Scale App*).

SLA et règles d'adaptation. Les règles qui alimentent la partie *Analyze* de la boucle autonomique pour gérer l'infrastructure sont définies de la façon suivante [7] :

```
1 rule R1: If (avg(cpuLoad of $vm-app linked to $app-lb)) > 75% for 60s then
      ScaleOut: deploy new $vm-app and link it to $app-lb;
2 rule R2: If (avg(cpuLoad of $vm-app linked to $app-lb)) < 40% for 60s then
      ScaleIn: remove $vm-app;
```

Listing 7.2 – Règles App tier

```
1 rule R3: If (avg(cpuLoad of $vm-otp linked to $otp-lb)) > 80% for 60s then
      ScaleOut: deploy new $vm-otp and link it to $otp-lb;
2 rule R4: If (avg(cpuLoad of $vm-otp linked to $otp-lb)) < 30% for 60s then
      ScaleIn: remove $vm-otp;
```

Listing 7.3 – Règles Opt tier

Elles accompagnent le SLA écrit avec le langage CSLA et présenté sous la forme du Tableau 7.1, plus lisible que sa forme XML . La valeur de la propriété *fuzziness* a été fixée à 0 et la valeur de la propriété *confidence* à 100% pour correspondre à la réalité du marché des VMs. Par contre, le prix est le même entre les VMs pour des raisons de simplification.

Table 7.1 – SLA entre le IaaS et le SaaS

service	métrique	oper.	valeur	fuzz.	% of fuzz.	*confidence*	prix	pénalité
Small/large	Availability	$\geq$	99.95%	0	0	100	0.046$ /h	10% if 99% $\leq$ Availability $\leq$ 99.95% 30% if Availability < 99%

La règle pour gérer l'élasticité au niveau applicatif est présentée ci-dessous et le SLA correspondant dans le Tableau 7.2. En ce qui concerne la dégradation de service, seul le mode normal est accepté pour le service *S1* (i.e., 3 itinéraires), alors que pour le service *S2*, jusqu'à 30% des périodes d'observation peuvent acceptées le mode dégradé (i.e., 1 seul itinéraire).

```
1 rule R5: If (avg(responseTime of $vm-app linked to $app-lb)) > 750ms for 30s
      then ScaleApp: degrade service for 3 minutes;
```

Listing 7.4 – Règle applicative

Table 7.2 – SLA entre le SaaS et ses clients

service	métrique	oper.	valeur	fuzz.	% of fuzz.	*confidence*	pénalité
S1/S2	Response-time	$\leq$	750ms	50ms	16.66%	90%	0.003$/rqt
	Availability	$\geq$	99,5%	0,5%			0,001$/rqt
S2	Mu(degraded)	$\leq$	30%	5%			0,0005$/rqt

Enfin, notons que les règles R1 et R3 sont suivies automatiquement d'une action de dégradation de service quand c'est possible (i.e., cas *S2*) pour toutes les instances nouvelles créées. L'idée ici est d'absorber la charge croissante pendant le temps d'initialisation des ressources en utilisant le mode dégradé. Une fois les instances activées, l'application SaaS passe en mode normal.

7. Il faut préciser que la valeur des seuils a été fixée en s'appuyant sur les résultats d'une campagne de tests de calibrage effectuée au préalable pour respecter les SLA.

Expérimentation. L'objectif de notre expérimentation est de comparer le comportement des implémentations *withoutSaaSElasticity* – correspondant à un Cloud traditionnel – et *withSaaSElasticity* – correspondant à un Cloud capable d'absorber les violations SLA, de « déformer » le logiciel en vue de renvoyer une « valeur suffisante » au client mais moins énergivore. Cette comparaison revient à comparer CSLA avec n'importe quel langage SLA qui ne possède pas les bonnes propriétés pour gérer l'élasticité.

Le scénario de charge appliqué pour cette expérimentation est d'une durée de 30 min (cf. courbe hachurée noire, Figure 7.6). Nous avons étudié un scénario correspondant à un pic de charge. Ainsi, les 20 premières minutes, 5 clients par seconde (i.e., *threads/s*) sollicitent l'application, puis nous simulons un pic de charge d'environ 5 minutes en faisant tripler la demande (i.e., de 5 à 15) durant cette période. Enfin, un retour à la normale est observé sur les 5 dernières minutes. L'architecture initiale pour les deux implémentations, *withoutSaaSElasticity* et *withSaaSElasticity*, est composée d'une *vm-app* et de deux *vm-otp*. Le service de calcul d'itinéraires est paramétré en mode normal (i.e., retourne 3 itinéraires).

Résultats. La Figure 7.6 présente les résultats pour les expériences avec nos deux implémentations (*withSaaSElasticity* et *withoutSaaSElasticity*) en considérant le SLA donné dans le Tableau 7.2.

Une première observation possible, lorsque le pic de charge survient au temps $t = 20min$, est que l'implémentation *withSaaSElasticity* (cf. courbe verte) permet d'éviter l'ajout de ressources IaaS en absorbant l'augmentation de la charge de travail à l'aide de la dégradation de service (passage de 3 itinéraires à 1) (cf. Figure 7.6(c)). L'implémentation *withoutSaaSElasticity* (cf. courbe rouge), quant à elle, a nécessité l'ajout d'une nouvelle instance, c'est-à-dire une augmentation des coûts d'infrastructure, pour satisfaire la demande croissante (cf. Figure 7.6(d)).

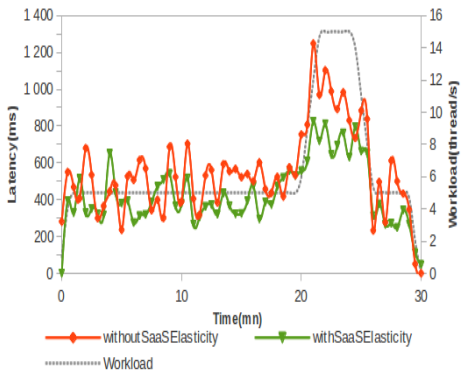
On peut aussi constater que l'élasticité au niveau applicatif a été plus à même de passer à l'échelle rapidement face à l'augmentation brutale de la demande (cf. Figures 7.6(a) et 7.6(b)). Les temps de réponse et la disponibilité observés pour l'implémentation *withoutSaaSElasticity* dépassent nettement les valeurs fixées dans le cadre des SLOs (i.e., 750 ms et 99,5%) contrairement à l'implémentation *withSaaSElasticity*. Bien que la qualité d'expérience soit dégradée dans le cas de l'implémentation *withSaaSElasticity* (i.e., 1 itinéraire au lieu de 3), la flexibilité et la réactivité offertes par notre solution permet de fournir le service avec un niveau de performance et de QoS acceptable sans nécessité l'ajout de ressource IaaS.

La Figure 7.6(e), mettant en lumière le respect des SLAs, considère uniquement la dernière fenêtre de 10 min de l'expérience. Pour rappel, l'état *Ideal* indique que le seuil du SLO a été respecté, l'état *Degraded* désigne que la requête reste dans la marge d'erreur spécifiée dans le SLA (i.e., propriété *Fuzziness*) et enfin l'état *Inadequate* indique que le SLO a été violé. Nous pouvons voir que l'implémentation *withSaaSElasticity* admet de meilleurs résultats en termes de violation de SLA. De plus, la proportion de requêtes *Ideal* de l'implémentation *withSaaSElasticity* est nettement supérieure à celle de l'implémentation *withoutSaaSElasticity*.

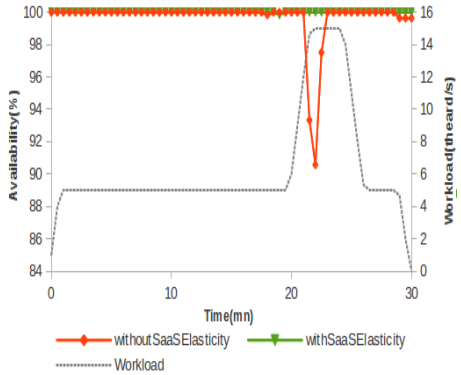
Enfin, signalons que le débit de sortie (i.e., *throughput*) supérieur de l'implémentation *withSaaSElasticity* permet au fournisseur SaaS d'augmenter son profit en traitant davantage de requêtes (i.e., facturation à la requête) sans nécessairement augmenter ses coûts d'infrastructure.

7.2.3 Conclusion

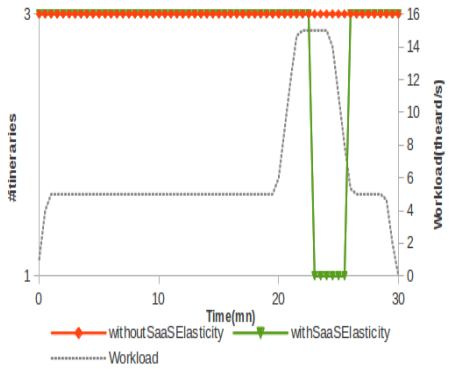
Les langages SLA existants tels que WSLA [KL03] et WS-Agreement [ACD+05] n'intègrent pas dans leurs gènes la nature dynamique du Cloud et ont donc du mal à garantir des contrats de service pour les architectures en nuage. CSLA introduit de nouveaux concepts permettant à la fois de fiabiliser les SLA dans le Cloud en minimisant les violations mais également d'intégrer des compromis QoS-énergie comme la déformation du logiciel en vue de renvoyer une « valeur suffisante » au client mais moins énergivore.



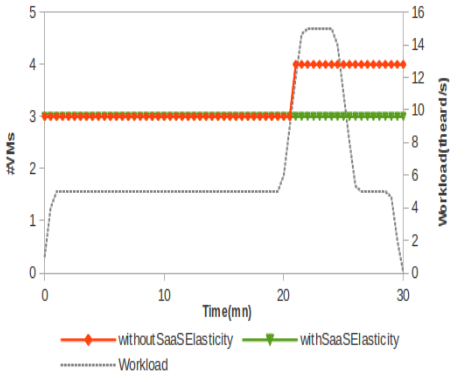
(a) Temps de réponse



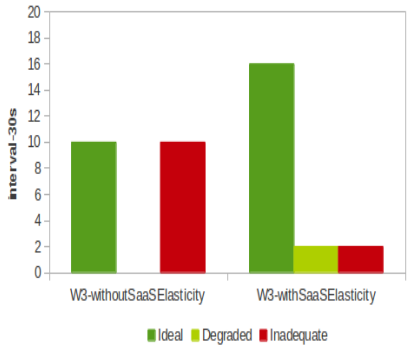
(b) Disponibilité



(c) Nb itinéraires



(d) Nb VMs



(e) Coût pénalités

FIGURE 7.6 – Résultats des expériences [KADL14].

Avec la conception de CSLA, nous avons également été confrontés à des réflexions sur le *Yield management*[8] dépassant le cadre de l'informatique. Affiner les modèles économiques autour de CSLA reste certainement une perspective intéressante pour les fournisseurs de services Cloud. Enfin, par manque de temps et sans doute d'ambition, nous ne nous sommes pas tournés vers les groupes de standardisation/normalisation pour défendre notre langage CSLA, notamment auprès de la Commission européenne (cf. "*Cloud Service Level Agreement Standardisation Guidelines*" [9]). Un rendez-vous manqué ?

7.3 Réaliser l'éco-élasticité multi-boucle dans le Cloud

Dans sa thèse [ADOJ13], Frederico Alvares développe une approche pour la gestion multi-boucle dans les architectures en nuages garantissant une meilleure *synergie* entre les différentes couches du Cloud [ASL12, ASL13]. L'approche est validée expérimentalement à la fois qualitativement (amélioration de QoS et des gains d'énergie) et quantitativement (passage à l'échelle).

7.3.1 Motivations

Le Cloud est une architecture logicielle multi-couche créant une dépendance inter-couche (cf. Section 7.1). Dans l'hypothèse où chaque couche est administrée par un fournisseur de services indépendant, les décisions prises à un niveau peuvent générer des *interférences* sur les systèmes administrés à un autre niveau. Par exemple, dans la Figure 7.7, les applications au niveau SaaS peuvent demander plus de VM pour gérer leur passage à l'échelle alors que l'infrastructure IaaS réalise une consolidation dynamique de serveurs pour optimiser l'empreinte énergétique.

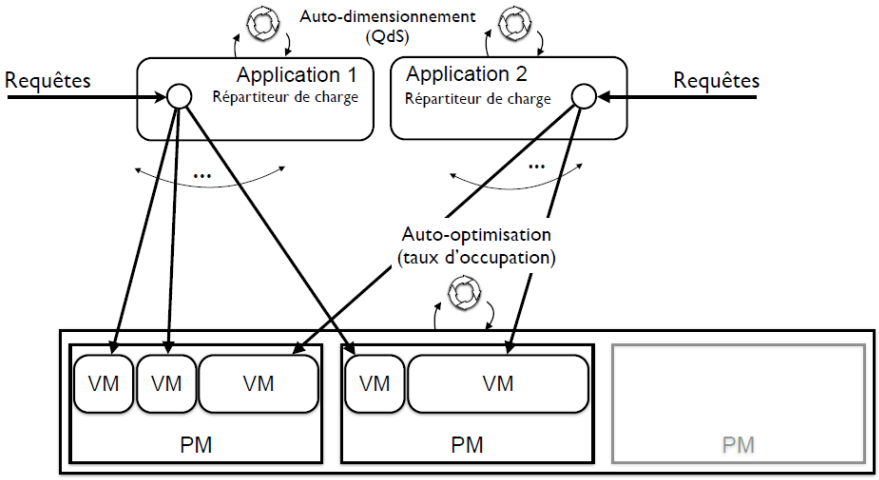


FIGURE 7.7 – Exemple d'une interférence multi-boucle.

Même si les intentions de protagonistes indépendants (i.e., les boucles de contrôle SaaS et IaaS) sont légitimes, il nous a semblé important de proposer un cadre de développement et d'exécution favorisant les synergies entre boucles de contrôle. En effet, d'une part, les décisions prises isolément par une boucle peuvent interférer avec les systèmes gérés par une autre boucle, les conduisant à des états non désirés. *Sans* coordination avec les applications SaaS, l'infrastructure IaaS peut faire face à

8. `https://fr.wikipedia.org/wiki/Yield_management`
9. https ://ec.europa.eu/digital-single-market/en/news/cloud-service-level-agreement-standardisation-guidelines

des états non optimaux (e.g., faible utilisation de ses ressources ou au contraire pénurie d'énergie). *Avec* une coordination, l'infrastructure IaaS encouragerait les applications SaaS à modifier leur comportement pour améliorer leur état ou faire face à un événement inattendu. D'autre part, en considérant les applications SaaS comme des *boîtes blanches* ayant une granularité assez fine pour se déformer et se recomposer, il est possible de rendre les applications SaaS plus sensibles à l'évolution de l'infrastructure, plus élastiques et donc de créer de nouvelles opportunités de prise de décision.

La reconfiguration dynamique du Cloud basée sur une élasticité au niveau IaaS étendue par une élasticité au niveau SaaS, les décisions d'optimisation coordonnées vont alors servir un objectif commun quand cela est possible (e.g., rééquilibrage de la consommation énergétique en cas de pénurie).

7.3.2 Coordination de boucles autonomiques

Nous proposons un modèle générique pour la synchronisation et la coordination des boucles de contrôle dans les architectures en nuage [ADOJ13, ASL12, ASL13]. La coordination entre boucles de contrôle doit respecter l'indépendance des couches XaaS qui, par défaut, n'appartiennent pas au même fournisseur. Aussi, notre solution ne peut pas reposer sur une solution centralisée (*centralized adaptation logic vs decentralized adaptation logic* [KRV+15]) et doit proposer un *couplage lâche* dans les interactions entre boucles.

Le modèle s'appuie sur un protocole de coordination basé sur des événements/actions et un protocole de synchronisation s'appuyant sur une base de connaissances (*Knowledge* - K) *partagée* entre boucles (cf. Figure 7.8). De cette façon, les décisions prises par une boucle de contrôle peuvent prendre en compte certaines informations fournies par d'autres boucles de contrôle. Ce modèle est adapté au Cloud Computing qui requiert un couplage faible entre couches et dans lequel *plusieurs* applications auto-adaptatives interagissent avec *une* infrastructure auto-adaptative commune. Un modèle pour une fédération de nuages est présenté dans [AdOLLM12].

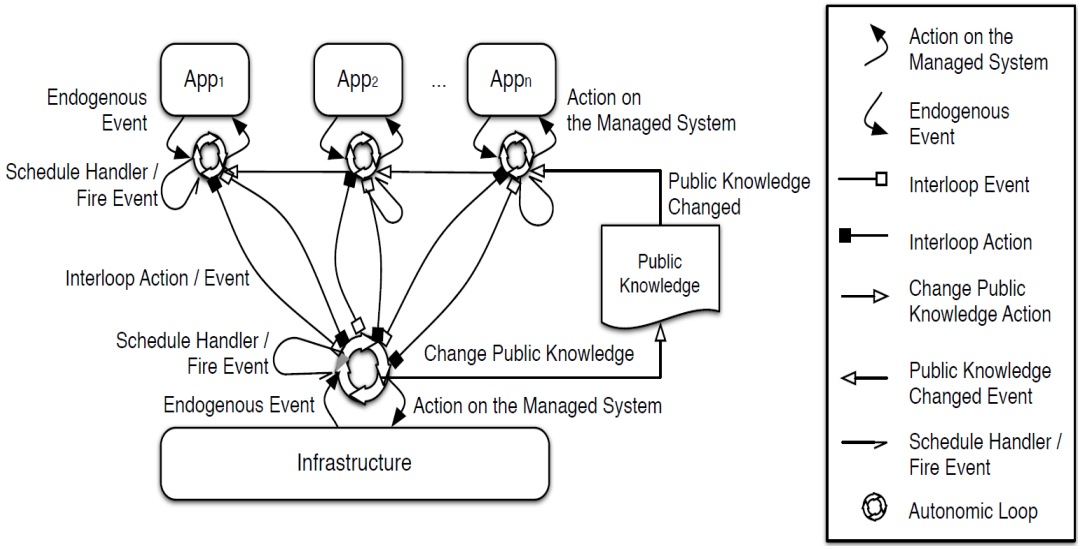


FIGURE 7.8 – Architecture multi-boucle pour le Cloud Computing [ASL12].

Modèle d'architecture des boucles

Selon l'état de l'art sur les architectures auto-adaptatives [BMSG+09, CdLG+09, VWMA11, KRV+15], les boucles de contrôle en interaction peuvent suivre un ensemble de modèles comme

présenté dans [WSG+13]. Cependant, après une étude rigoureuse [ASL13], nous avons trouvé des limites aux modèles actuels qui ne correspondaient pas à notre cadre de travail. Nous avons proposé un nouveau modèle d'architecture autonomique (cf. Figure 7.9) pour la coordination de plusieurs boucles qui répond aux contraintes architecturales du Cloud en termes de couplage lâche et de confidentialité des informations.

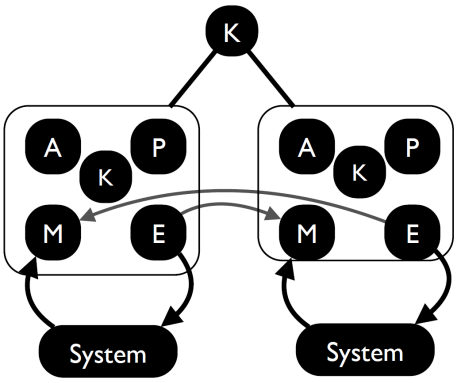


Figure 7.9 – Modèle architectural pour la coordination des boucles autonomiques [ASL13].

Dans le modèle proposé, les boucles communiquent strictement via leurs interfaces naturelles (i.e., fonctions 'M' et 'E'), ce qui limite les points d'interaction inter-boucle et augmente ainsi le couplage. En ce qui concerne la confidentialité des informations, contrairement aux modèles de l'état de l'art [WSG+13], la visibilité sur le contexte d'exécution d'un système administré donné est de la responsabilité de sa boucle autonomique. Une boucle peut partager une partie de ses connaissances (i.e., base 'K') avec d'autres boucles afin de pouvoir prendre des décisions plus appropriées. Par exemple, les fournisseurs IaaS peuvent communiquer sur l'état de leurs ressources ou utiliser une approche « agressive » sur les prix des ressources de certaines parties de leur infrastructure (e.g., régions et zones de disponibilité [10]) pour influencer le comportement des consommateurs SaaS, ce qui permet d'auto-administrer finement l'occupation de certaines parties de l'infrastructure.

Protocole de synchronisation pour la base de connaissances publique

Pour avoir à la fois une vision d'ensemble du contexte d'exécution mais également pour respecter la partie privée des boucles de contrôle, la base de connaissance 'K' de chaque boucle est scindée en deux parties : la connaissance privée, qui stocke l'information interne nécessaire aux fonctions autonomiques, et la connaissance publique, qui est partagée avec les autres gestionnaires. La base de connaissances publique doit être synchronisée si les actions exécutées par les boucles nécessitent de modifier les informations partagées (directement ou indirectement).

Pour cette raison, nous introduisons un *protocole de jeton simple* afin de synchroniser l'accès aux sections critiques. Chaque section critique est associée à un jeton, ce qui signifie que pour accéder à une section critique donnée, les boucles doivent demander le jeton correspondant. Une boucle peut obtenir le jeton soit en le demandant explicitement avec un message TOKEN REQUEST (demande active de jeton), soit elle peut recevoir le jeton d'un autre boucle via un message TOKEN TRANSFER (réception passive de jeton). Chaque fois qu'un gestionnaire n'a plus besoin du jeton, il le libère avec un message TOKEN RELEASE. Chaque fois qu'un jeton est demandé, le demandeur doit attendre un message TOKEN ACQUIRED. Chaque gestionnaire autonomique ayant une base de connaissance publique avec sections critiques implémente un gestionnaire de jetons [ASL12].

10. `https://docs.aws.amazon.com/fr_fr/AWSEC2/latest/UserGuide/using-regions-availability-zones.html`

Protocole de coordination basé sur les événements/actions

L'objectif du protocole de coordination est de fournir des moyens pour que les boucles puissent communiquer afin que les actions soient exécutées de manière coordonnée et qu'une synergie puisse être établie entre ces boucles. Dans le modèle d'architecture proposé, la communication entre les boucles est restreinte aux tâches de surveillance ('M') et d'exécution ('E') pour favoriser un couplage lâche. Cette communication est organisée à travers un bus de messages [ADOJ13].

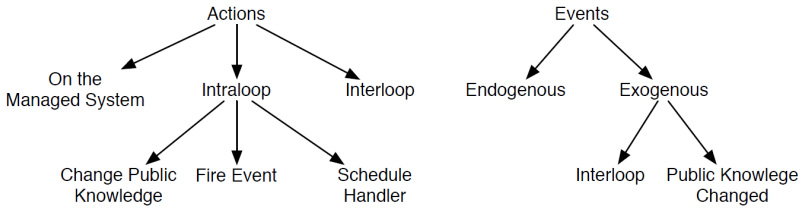


FIGURE 7.10 – Hiérarchie des événements/actions [ASL13].

Afin de clarifier les interactions entre plusieurs boucles, il est important de différencier les actions et les événements qui font partie du système administré et ceux qui font partie du modèle de coordination. La Figure 7.10 représente une hiérarchie des différents types d'événements et d'actions. Les actions peuvent être exécutées sur le système administré (i.e., *management element*) dans la boucle de contrôle MAPE-K (*intraloop*) ou sur une autre boucle de contrôle MAPE-K (*interloop*) [HGB10]. Les actions sur le système administré et les actions *interloop* sont toujours exécutées par la fonction 'E' de la boucle. Une action *interloop* peut notifier un autre gestionnaire autonomique comme si elle demandait un service et attendait la réponse. Dans ce cas, un *handler* (*Schedule Handler*) est créé et contient toutes les autres actions qui doivent être exécutées en réponse à cette action *interloop* (cf. *futures* dans [EFGK03]). Cette action *interloop* est donc une action « notificatrice », qui est considérée comme un événement *interloop* sur la boucle de contrôle cible. L'action *Change Public Knowledge* modifie le contenu public de la base de connaissance 'K'. L'action *Fire Event* est une action qui déclenche un événement endogène sur le même gestionnaire autonome.

Ensuite, un événement peut être *endogène* ou *exogène*. La source des événements endogènes est toujours le système administré. La source d'événements exogènes est une autre boucle de contrôle. Pour les événements exogènes, il y a les événements *interloop* – créés par l'action *interloop* de l'autre boucle – et le *Public Knowledge Changed* – créé par l'action *Change Public Knowledge*.

Boucles de contrôle SaaS et IaaS

Dans [ADOJ13], nous avons identifié un certain nombre d'actions et d'événements de nature différente selon que l'on s'intéresse aux systèmes administrés de type SaaS ou de type IaaS. Ainsi, nous avons proposé deux types de boucles autonomiques (SaaS *vs* IaaS). Nous avons opté pour une approche déclarative pour modéliser la logique d'adaptation (cf Section 3.2). En effet, la fonction *Analyze* modélise les problèmes de reconfiguration comme des problèmes de satisfaction et d'optimisation de contraintes (CSOP) et se base sur la programmation par contraintes (CP) [RBW06] pour les résoudre. La mise en œuvre est réalisée avec le solveur de contraintes Choco [PFL17] [11].

Illustration de contraintes pour le placement de VM. Prenons l'exemple d'une infrastructure IaaS de machines physiques (PM) modélisée par le vecteur $P = (pm_1, pm_2, \ldots, pm_p)$. pm_i^{cpu} (resp. pm_i^{ram}) définit la capacité CPU (resp. RAM) de la machine $pm_i \in P$. Soit $V = (vm_1, vm_2, \ldots, vm_v)$

11. Pour plus de détails voir la thèse de Frederico Alvares [ADOJ13].

l'ensemble des machines virtuelles (VM) s'exécutant dans l'infrastructure. Pour chaque $pm_i \in P$, il existe un vecteur $H_i = (h_{i1}, h_{i2}, \ldots, h_{iv})$, où $h_{ij} = 1 \iff vm_j$ est hébergée par pm_i, $h_{ij} = 0$ sinon $(1 \leq i \leq p, 1 \leq j \leq v)$.

On cherche à démarrer de nouvelles VM sur l'infrastructure suite à une demande provenant du SaaS (événement *VMs Requested*, voir ci-dessous). Notre objectif est de placer les VM demandées dans les PM afin de minimiser le nombre de PM nécessaires. Soit $V' = V \bigcup \{vm_j \mid j \in [v+1, v'], v' = v + nbVMrequested\}$, l'ensemble V concaténé avec les machines virtuelles demandées.

Nous définissons pour la fonction *Analyze* de la boucle autonomique les contraintes suivantes. Les contraintes 7.1 et 7.2 indiquent que les demandes CPU et RAM des machines virtuelles hébergées dans une PM ne doivent pas excéder ses capacités CPU et RAM. Notre analyseur vise à placer les nouvelles VM demandées dans le nombre minimum de PM en respectant les contraintes de ressources. Toutefois, nous imposons une contrainte supplémentaire de *non-migration* : il n'est pas prévu de regrouper les VM sur les PM en migrant les VM en cours d'exécution vers d'autres PM (cf. contrainte 7.3).

$$pm_i^{cpu} \geq \sum_{j=1}^{v'} h_{ij} * vm_j^{cpu} : \forall i \in [1, p] \tag{7.1}$$

$$pm_i^{ram} \geq \sum_{j=1}^{v'} h_{ij} * vm_j^{ram} : \forall i \in [1, p] \tag{7.2}$$

$$\forall i \in [1, p](\forall j \in [1, v] : h_{ij} = h_{ij}) \tag{7.3}$$

Enfin, l'objectif de minimiser le nombre de PM nécessaires pour héberger les VMs (V') est décrit par l'équation 7.4.

$$minimize(\sum_{i=1}^{p}(u_i)), \text{ where } u_i = \begin{cases} 1, \exists j \in [1, v'] \mid h_{ij} = 1 \\ 0, \text{otherwise} \end{cases} \tag{7.4}$$

Boucle IaaS. Ce gestionnaire autonomique est capable de traiter les événements suivants : *VMs Requested, VMs Released, Energy Shortage, Low PM Utilization, Unused PM.* Chacun de ces événements est ensuite analysé par la fonction 'A' de la boucle. Par exemple, quand l'événement *Energy Shortage* est notifié, la boucle doit trouver les machines physiques (PM) à arrêter en minimisant l'impact sur les applications SaaS hébergées. Une fois le résultat obtenu – grâce au solveur de contraintes Choco – la fonction 'P' de la boucle exécute l'algorithme décrit par le pseudo-code dans le Listing 7.5. Notons la notification de l'événement *Scale Down* vers le gestionnaire SaaS des applications concernées, créant ainsi une synergie entre les deux types de boucles.

```
1  Input: PMoff       // PMs to be switched off
2  Input: Applications // SaaS app hosted by PMOff
3  Output: plan       // the resulting plan
4
5  plan = new instance of Plan
6  fork = new instance of Fork
7  foreach app in Applications do
8    vms = vms de app sur PMoff
9    add(NotifyScaleDown(app,vms), fork)
10 done
11 add(fork, plan)
12 add(FireEvent(TimeoutExpired(PMoff),TIMEOUT),plan)
```

Listing 7.5 – Planification d'un *Energy Shortage*

Boucle SaaS. Ce gestionnaire autonomique est capable de traiter les événements suivants : *Workload Increased/Decreased, Renting Fees Changed, VMs Created, Scale Down, Promotion Expired.* Chacun de ces événements est ensuite analysé par la fonction 'A' de la boucle. Par exemple, quand l'événement *Scale Down* est notifié, il faut adapter l'application à une contrainte de ressource. Concrètement, l'événement contient la liste des machines virtuelles (VM) qui devraient être immédiatement libérées par l'application. Cet événement déclenche une tâche d'analyse pour réaffecter les composants sur un plus petit nombre de VM. À cette fin, il peut être nécessaire de modifier la configuration de l'architecture de l'application (par exemple, pour remplacer les composants qui sont trop énergivores). Comme pour la boucle IaaS, c'est le solveur Choco qui – à partir des différentes contraintes – va déterminer une solution. Il ne restera plus qu'à la fonction 'P' de la boucle de s'appuyer sur l'élasticité au niveau applicatif et de combiner les APIs classiques d'un modèle de composants (e.g., *start/stop, bind/unbind*, ...) pour reconfigurer l'application SaaS.

7.3.3 Evaluation et expérimentation

Dans sa thèse [ADOJ13], Frederico Alvares présente plusieurs illustrations de synergie entre boucles autonomiques IaaS et SaaS. Citons par exemple, un scénario d'élasticité basé sur un budget contraint ou encore un scénario de promotions (au sens solde) de machines virtuelles. Dans l'expérimentation ci-dessous, nous scénarisons la pénurie d'énergie évoqué ci-dessus.

Il est à noter que nous avons évalué l'impact de la base de connaissance (*Knowledge*) publique dans le cadre du scénario de promotions. De plus, nous avons également évalué l'évolutivité des modèles analytiques basés sur les contraintes. Pour des raisons de place, nous ne présentons pas les analyses d'évaluation qui résultent de ces expériences (voir la thèse de Frederico Alvares [ADOJ13] pour plus de détails).

Banc de tests

Le scénario consiste à déclencher un événement *Energy Shortage* au niveau de l'infrastructure IaaS et à observer ses effets sur la consommation d'énergie ainsi que sur la QoS des applications SaaS hébergés. L'objectif est de montrer un cas réel d'utilisation du protocole de coordination avec des événements et actions *interloop* [12].

Les expériences ont été effectuées sur un banc de tests composé d'applications SaaS programmées avec le modèle de composants FraSCAti [SMR+12, SMF+09] suivant la norme SCA (*Service Component Architecture*) et l'infrastructure matérielle Grid'5000 [13], équipée de Wattmètres, dans laquelle une ou plusieurs instances d'application sont déployées.

Application SaaS. Le SaaS fournit un seul service de publicité qui peut fonctionner dans deux modes (i.e, élasticité logicielle) : normal (vidéo) et dégradé (image statique). Nous distinguons ensuite deux clients du service : *AdClient 1* pour lequel seules les vues vidéo sont acceptées et *AdClient 2* pour lequel jusqu'à 20% des vues d'images peuvent être acceptées.

Infrastructure IaaS. Nous nous sommes basés sur un ensemble de 13 nœuds (PM) de Grid'5000 : deux nœuds ont été utilisés pour héberger les boucles autonomiques (une pour le SaaS et une pour le IaaS) ; un nœud a été utilisé pour héberger l'injecteur de charge Clif [14] ; dix nœuds ont été utilisés pour accueillir les VMs hébergeant les services SaaS eux-mêmes (avec deux tailles de VM, e.g., *small* et *large*).

12. Une expérimentation plus aboutie intégrant le langage CSLA est décrite dans [SBK+16].
13. `https://www.grid5000.fr`
14. `http://clif.ow2.org`

Expérimentation. La durée des expériences a été fixée à une heure, au cours de laquelle la charge de travail des deux instances SaaS passe de 0 à 140 demandes par seconde. Une pénurie d'énergie est déclenchée à 45 minutes du début de l'exécution. Pour le fournisseur IaaS, l'intervalle d'observation a été fixé à six minutes, alors que pour le fournisseur SaaS, il a été fixé à une minute. Le délai d'attente (*timeout*) avant l'arrêt des PMs par la boucle autonomique de l'infrastructure a été fixé à 10 minutes.

Résultats

La Figure 7.11 montre la consommation d'énergie totale de l'infrastructure, avant et après l'événement *Energy Shortage*. Cet événement contient le nombre de nœuds à arrêter (ici 5). L'intervalle entre la détection d'événement (ligne verticale) et la diminution de la consommation d'énergie est dû au délai entre la notification *Scale Down* du fournisseur IaaS vers le fournisseur SaaS et l'arrêt effectif des nœuds. La réduction d'échelle peut avoir un impact sur la qualité de service SaaS. La Figure 7.11 montre la QoS en termes de temps de réponse moyen du SaaS pour les clients *AdClient 1* et *AdClient 2* en fonction de la variation de charge de travail. Contrairement à *AdClient 1*, *AdClient 2* accepte une dégradation fonctionnelle pour 20% des demandes dans une fenêtre temporelle. De cette façon, l'instance SaaS de *AdClient 2* est transformée en un mode dégradé (image) de manière à absorber la même charge de travail. Il y a reconfiguration de l'architecture FraSCAti en vue de renvoyer une « valeur suffisante » au client mais moins énergivore et finalement plus performante.

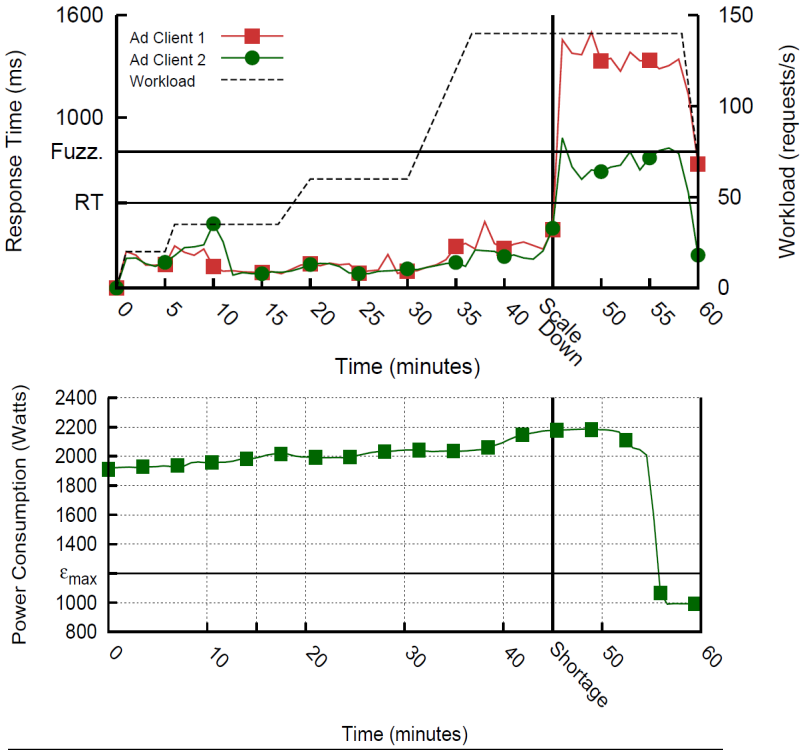


FIGURE 7.11 – Reconfiguration multi-couche suite à une pénurie d'énergie [SBK+16].

7.3.4 Conclusion

L'éco-élasticité multi-couche dans le Cloud est possible grâce au *framework* développé dans [ADOJ13] privilégiant les synergies entre boucles de contrôle SaaS et IaaS. Non seulement notre proposition permet de ne pas subir les interférences entre prises de décision indépendantes mais elle contribue à créer des opportunités de collaboration entres couches XaaS. En conclusion, avec le *framework* proposé, la fiabilité de la reconfiguration multi-couche (*cross-layer*) est garantie.

Ces travaux peuvent être étendus de plusieurs manières : pour la couche SaaS vers d'autres modèles de composants plus axés sur la grille comme GCM [BCD+09], pour la fonction *Analyze* vers la définition de contraintes de haut niveau en termes de placement de VMs qui peut être choisie au niveau applicatif et appliquée au niveau de l'infrastructure [HLM13], pour la fonction *Analyze* (encore) l'intégration d'un modèle de coûts de reconfiguration dans la formulation des problèmes d'optimisation, ...

7.4 Programmer l'éco-élasticité

Dans sa thèse [Dup16] – thèse en bourse Cifre avec la société Sigma informatique – Simon Dupont propose de modéliser l'extension du concept d'élasticité aux couches hautes du Cloud (i.e., *élasticité logicielle*), d'outiller le processus de gestion de l'élasticité en vue de simplifier la tâche d'auto-administration.

7.4.1 Motivations

Dans les travaux précédents de Yousri Kouki et de Frederico Alvares, nous avons proposé de définir un nouveau langage SLA pour le Cloud, de rendre élastique la partie logicielle hébergée par le Cloud, de reconfigurer le Cloud avec une approche dirigée par les SLA (*SLA-driven*) et ce de façon coordonnée à travers plusieurs couches XaaS (e.g., SaaS et IaaS). Toutes ces contributions, ces artefacts sont nécessaires pour traiter la problématique de l'empreinte énergétique dans les architectures en nuage par rapport à l'état de l'art focalisé sur les couches basses du Cloud [BAB12, OAL14, MOC+14].

Cependant, pour valoriser le travail réalisé auprès d'un hébergeur de services (en l'occurrence Sigma Informatique), nous nous sommes aperçus qu'il manquait des clefs pour rendre accessible ces contributions. Un acteur tiers veut des outils pour concevoir des stratégies d'élasticité, programmer des règles d'*auto-scaling* multi-couche, « apprivoiser » la boucle MAPE-K, ...

Par exemple, les règles d'*auto-scaling* pour le Cloud sont des règles qui peuvent être réalisées en suivant différents paradigmes : règles à base de seuil, théorie des files d'attente, apprentissage par renforcement, théorie du contrôle et théorie des séries chronologiques [LBMAL14]. Les modèles prédictifs et les différentes théories évoquées sont principalement utilisés dans le monde scientifique ou dans les laboratoires de R&D. En effet, les solutions industrielles préfèrent une approche impérative reposant sur les règles à base de seuil, plus simple à mettre en œuvre qui ne nécessitent pas un bagage scientifique important (e.g., maîtriser un solveur de contraintes).

Nos travaux autour de l'éco-élasticité logicielle pour le Cloud frugal doivent aussi être pensés pour être utilisés par de futurs programmeurs d'où la nécessité de modéliser l'élasticité logicielle, d'outiller le processus de gestion de l'élasticité multi-couche.

7.4.2 Gestion de l'élasticité multi-couche de bout en bout

Cette section résume les contributions principales de la thèse de Simon Dupont [Dup16].

Modèle conceptuel de l'élasticité logicielle

> « *Nous définissons l'élasticité logicielle comme la capacité d'un logiciel à s'adapter, idéalement de manière autonome, pour répondre aux changements de la demande et/ou aux limitations de l'élasticité des ressources de l'infrastructure.* » (Simon Dupont - 2016)

Par analogie avec l'élasticité de l'infrastructure, où les ressources IaaS (e.g., VM) sont ajustées dynamiquement pour garantir les contrats SLA tout en optimisant l'utilisation des ressources, l'*élasticité logicielle* repose sur une granularité d'échelle plus fine (par rapport au IaaS) pour ajuster les ressources

SaaS (e.g., composants logiciels) de manière transparente et instantanée afin de mieux répondre aux attentes des SLA.

L'élasticité logicielle n'a pas pour vocation de remplacer l'élasticité de l'infrastructure, inhérente au Cloud, mais plutôt de pallier ses manques en termes de réactivité, de flexibilité et d'empreinte énergétique. La *précision* du dimensionnement et de la *réactivité* du dimensionnement [WHGK14], critères fondamentaux de l'élasticité vont être solutionnées grâce à l'élasticité logicielle (cf. Section 6.2.3). Nous le démontrerons lors de l'expérimentation.

Comme pour l'élasticité de l'infrastructure, on distingue le dimensionnement vertical *vs* horizontal. Ainsi, l'élasticité logicielle offre de nouvelles possibilités d'adaptation, élargissant les possibilités offertes par l'infrastructure et permet de considérer une élasticité multi-couche. Par conséquent, les administrateurs Cloud peuvent créer des plans de reconfiguration plus sophistiqués en orchestrant des actions de différentes dimensions et gérer ensuite finement et efficacement les ressources Cloud. Le Tableau 7.3 résume les quatre dimensions avec les huit API sous-jacente.

Elasticity dimension	**Elasticity action**	**Description**
Infra Horizontal Scaling	Scale Out Infra (`soi`)	Add VM
	Scale In Infra (`sii`)	Remove VM
Infra Vertical Scaling	Scale Up Infra (`sui`)	Increase offering infra
	Scale Down Infra (`sdi`)	Decrease offering infra
Soft Horizontal Scaling	Scale Out Soft (`sos`)	Add soft component
	Scale In Soft (`sis`)	Remove soft component
Soft Vertical Scaling	Scale Up Soft (`sus`)	Increase offering soft
	Scale Down Soft (`sds`)	Decrease offering soft

Table 7.3 – API élasticité multi-couche [DBAL17].

Programmer des plans de reconfiguration *cross-layer* à partir de ces API sera l'objet du langage dédié ElaScript (cf. ci-après).

ElaStuff : un *framework* autonomique

Notre partenaire Sigma Informatique – en tant que hébergeur de services et éditeur de logiciel – jouait à la fois de rôle de fournisseur IaaS et de fournisseur SaaS. Cette « double casquette » a introduit un biais dans l'administration autonome des couches XaaS puisque Sigma pouvait gérer tous les API de l'élasticité dans une seule et même boucle de contrôle. Il n'y a pas eu besoin de coordination de boucles autonomiques (cf. Section 7.3). Après réflexion, ce cas est plus répandu qu'on peut le croire car il correspond au cas du Cloud privé.

Nous avons donc proposé ElaStuff, un *framework* autonomique qui correspond à une implémentation du modèle de référence de boucle MAPE-K capable de gérer des plans de reconfiguration *cross-layer* du Cloud. Comme le montre la Figure 7.12, le *framework* repose sur trois sous-modules :

- un modèle de surveillance : **PerCEPtion** est basé sur un moteur de *Complex Event Processing* (CEP) [CM12] qui permet aux administrateurs Cloud de mettre en place une observation avancée du système Cloud en décrivant les *symptômes*, les signes d'incohérence du système, détectés au moment de l'exécution, qui peuvent conduire à une reconfiguration de celui-ci [15] ;
- un modèle de décision : il est chargé de prendre des décisions de reconfiguration basées sur le contexte d'exécution et certaines préférences d'adaptation spécifiées par l'administrateur du Cloud sous la forme de *stratégies* (e.g., minimiser la consommation d'énergie). Par ailleurs, le processus décisionnel prend en compte les contrats SLA signés entre l'administrateur Cloud et ses clients (*SLA-driven*) ;

15. PerCEPtion est plus adapté au Cloud que WildCAT (cf. Section 4.3) et gère mieux la composition des événements.

— un modèle d'adaptation : **ElaScript** permet à l'administrateur Cloud de définir les adaptations à appliquer sur le système pour faire face aux symptômes détectés à l'exécution par PerCEPtion. Afin de spécifier ces adaptations potentiellement complexes et transversales, nous proposons un langage dédié [vDKV00], qui permet aux administrateurs d'exprimer simplement des plans de reconfiguration – appelés *tactiques* – en orchestrant différentes dimensions/actions d'élasticité. Un ensemble de tactiques – représentant les adaptations possibles pour un Cloud – sont rassemblées dans un catalogue et stockées dans la base de connaissances 'K' comme les définitions de symptômes et les préférences de l'administrateur du Cloud (i.e., les *stratégies*).

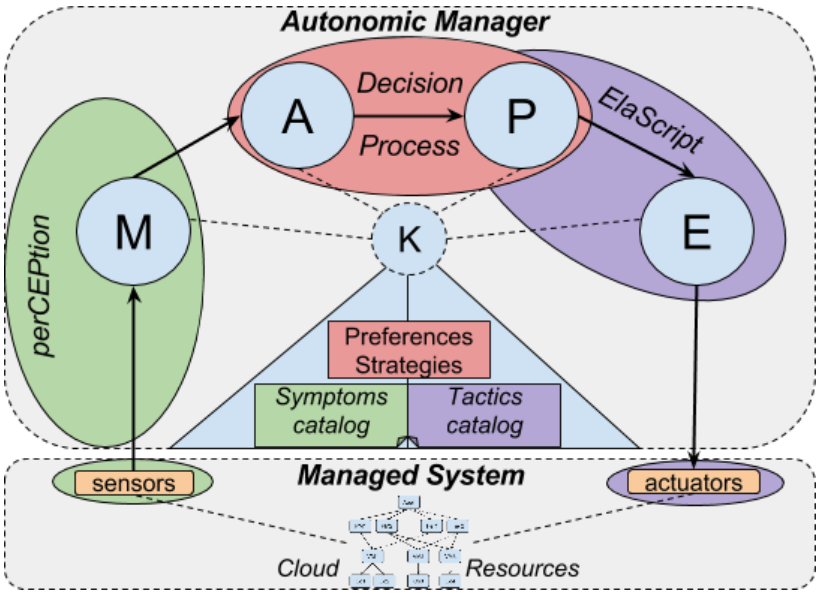


FIGURE 7.12 – Gestionnaire autonomique ElaStuff [DBAL17].

Processus de décision

Différentes dérives du système, des « maladies » identifiées/définies en amont par l'administrateur (e.g., tiers surchargé, machine physique inutilisé) vont être remontées à l'exécution sous forme de *symptômes* par perCEPtion. Par exemple, l'événement `Tier_VmsOverloaded (tier, vms)` est notifié à la fonction 'Analyze' pour traitement par perCEPtion [16].

Différents « remèdes » spécifiés par l'administrateur face à ces « maladies » vont être programmés en amont. On les appelle *tactiques* d'élasticité et elles seront codées avec ElaScript. Parfois, plusieurs tactiques adressent le même symptôme. Par exemple, quand une VM est surchargée, on peut démarrer une deuxième VM ou au contraire, dégrader tous les composants hébergés dessus. Les résultats de ces *tactiques* ne sont pas équivalents : laquelle choisir ? Selon quels critères ? Dans notre travail, nous considérons les six critères suivants :

— le *Coût d'infrastructure* (actions IaaS) ;
— la *QoS* (actions IaaS/SaaS) ;
— la *QoE* (*Quality of Experience* [17]) (actions SaaS) ;
— la *Consommation énergétique* (actions IaaS/SaaS) ;

16. Pour des raisons de place, nous ne détaillerons pas la façon dont sont générés ces événements (pour plus de précisons, voir [Dup16]).
17. `https://en.wikipedia.org/wiki/Quality_of_experience`

— le *Temps de reconfiguration* (actions IaaS/SaaS) ;
— la *Réactivité*, i.e., la capacité à réagir rapidement aux changements du système [HKR13] (actions IaaS/SaaS).

Le Tableau 7.4 dresse les tendances de l'impact de chaque action d'adaptation sur les différents *critères*. Les symboles $-$, $+$ et $\emptyset$ signifient respectivement que l'action d'adaptation a un impact négatif, positif ou nul sur le *critère* concerné tandis que le symbole $*$ indique une tendance variable.

Actions d'élasticité	**Critères d'adaptation**					
	coût	QoS	QoE	énergie	temps reconf.	réactivité
Scale Out Infrastructure (`soi`)	$-$	$+$	$\emptyset$	$-$	$-$	$-$
Scale In Infrastructure (`sii`)	$+$	$-$	$\emptyset$	$+$	$*$	$*$
Scale Up Infrastructure (`sui`)	$-$	$+$	$\emptyset$	$-$	$+$	$+$
Scale Down Infrastructure (`sdi`)	$+$	$-$	$\emptyset$	$+$	$+$	$+$
Scale Out Software (`sos`)	$\emptyset$	$-$	$+$	$-$	$*$	$*$
Scale In Software (`sis`)	$\emptyset$	$+$	$-$	$+$	$*$	$*$
Scale Up Software (`sus`)	$\emptyset$	$-$	$+$	$-$	$+$	$+$
Scale Down Software (`sds`)	$\emptyset$	$+$	$-$	$+$	$+$	$+$

Table 7.4 – Actions d'élasticité et critères d'adaptation : tendances [Dup16].

En amont, l'administrateur Cloud va définir ses préférences en termes d'adaptation (i.e., *stratégie d'élasticité*) en définissant un ensemble de *critères* strictement ordonné (e.g., réactivité > QoE > coût) qui sera utilisé par un algorithme d'optimisation combinatoire chargé de rechercher une solution optimale parmi les tactiques éligibles. On parle alors d'approche *a priori* [MA04].

A l'exécution, le processus de décision du gestionnaire autonomique va identifier la « meilleure » tactique à appliquer selon le contexte d'exécution et selon la stratégie d'élasticité adoptée.

ElaScript : un DSL pour coder l'éco-élasticité

L'objectif de ElaScript [DBAL17] est de fournir un support pour les administrateurs Cloud pour exprimer simplement et en toute sécurité des plans de reconfiguration sur des architectures en nuage. Inspiré par notre expérience FScript (cf. Section 4.4), nous avons voulu utiliser nos compétences en langages dédiés pour proposer un langage de script fiable. ElaScript n'est pas un langage de programmation complet et ne traite que de l'élasticité. Le domaine de ElaScript est limité au niveau de l'architecture : configuration des composants et tiers SaaS avec les machines virtuelles et physiques IaaS. Comme tout DSL, ElaScript offre des possibilités d'analyse, de vérification ou d'optimisation en termes de constructions DSL qu'il serait beaucoup plus difficile d'avoir dans un langage généraliste.

ElaScript : survol. Les spécifications de ElaScript sont décrites dans [Dup16, DBAL17]. ElaScript est un langage impératif avec une syntaxe de type Java (typage statique), il ne permet pas de créer de nouvelles variables (seuls les paramètres issus des symptômes sont manipulables) et possède qu'un seul type d'itérateur. ElaScript permet d'orchestrer les actions d'élasticité via des opérateurs de séquentialité, de parallélisme et de synchronisation.

Par exemple, le Listing 7.6 représente le plan de reconfiguration faisant face au symptôme `Tier_VmsOverloaded (tier, vms)`. Ce type de symptôme est diagnostiqué par perCEPtion et indique un tier admettant un temps de réponse très élevé et une liste de VM montrant des signes de surcharge. Notre script indique l'ajout de 2 machines virtuelles (définies avec l'offre "*small*") à ce tier. En parallèle, nous dégradons le niveau de service pour toutes les VM surchargées qui récursivement vont dégrader tous leurs composants hébergés. Une fois que le bloc parallèle a terminé son exécution, les VM et leurs composants retrouvent l'offre précédente.

```
1 begin
2 when Tier_VmsOverloaded(Tier tier, List<VM> vms)
3 do
4   [
5         tier.soi(2,"small");
6   ||
7       foreach vm in vms do { vm.sds;}
8   ]
9   foreach vm in vms do { vm.sus; }
10 end
```

Listing 7.6 – Exemple d'élasticité multi-couche dans ElaScript

Garanties. Le compilateur de ElaScript garantit les règles sémantiques suivantes :

- *Règles de nommage.* Il est impossible d'avoir 2 ressources avec le même nom dans le même script ;
- *Règles de portée.* Nous utilisons deux niveaux de portée : la portée globale, dans laquelle toutes les ressources passées en arguments sont visibles tout au long du script ; et l'itération dans lequel les variables liées sont seulement visibles à l'intérieur de la boucle *foreach* ;
- *Règles de typage.* L'analyse sémantique de ElaScript inclut la vérification de type, il est alors impossible d'appeler une action sur une mauvaise ressource. Par exemple, l'action `sos` (*Scale Out Software*) ne peut pas être exécutée sur une PM mais seulement sur une VM ;
- *Règles métier.* Plusieurs règles de gestion ont été introduites pour aider les administrateurs Cloud à programmer des scripts "*safe*". Nous avons distingué deux types de règles :

  1. règle "*error*" : par exemple, un `vm.sii()` ne peut pas être suivi d'une autre action sur cette `vm` (et sur ses enfants) puisque *Scale In Infrastructure* l'éteint.

  2. règle "suggestion" : par exemple, l'action *Scale Out Infrastructure* prend du temps, nous proposons de saisir l'opportunité de paralléliser d'autres actions pendant ce temps.

  Finalement, nous pouvons tirer partie d'un outillage puissant comme proposé dans l'IDE Eclipse pour aider les administrateurs Cloud à coder leurs reconfigurations (cf. Figure 7.13). Il est essentiellement basé sur Xtext [18], un *framework* Eclipse pour l'implémentation de langages de programmation.

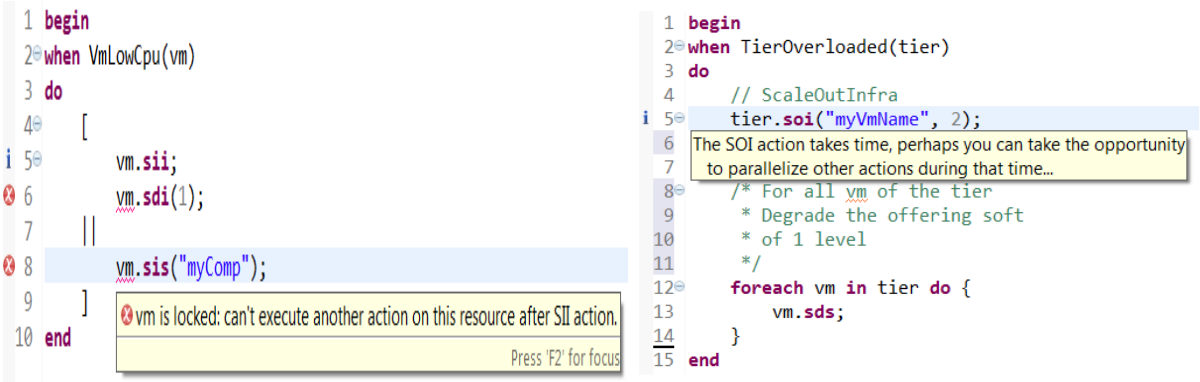


FIGURE 7.13 – Règles ElaScript dans Eclipse [Dup16].

18. https://www.eclipse.org/Xtext

7.4.3 Evaluation et expérimentation

L'évaluation suivante est décrite dans [DLAL15]. Elle a pour objectif d'éprouver le modèle d'élasticité multi-couche, d'illustrer la synergie entre l'élasticité IaaS et SaaS et d'identifier des stratégies pertinentes de gestion de l'élasticité multi-couche.

Banc de tests

Application. Nous considérons une application SaaS synthétique dont l'architecture admet 2 tiers distincts (cf. Figure 7.14). Chaque tier est composé de VM : le premier tier (*lbt*) correspond à un répartiteur de charge Nginx [19] ; le second tier (*bt*) est quant à lui constitué d'une application Java *CPU-intensive* et d'un serveur HTTP reposant sur Jetty [20]. L'application synthétique supporte l'élasticité logicielle verticale (i.e., *Scale Up Soft* et *Scale Down Soft*) au travers d'un composant du tier métier admettant 5 offres différentes se traduisant par différentes configurations de l'application, avec des niveaux de QoE distincts, plus ou moins consommateurs de ressources.

Infrastructure. Les expériences ont été menées sur Grid'5000 Nancy, en utilisant 7 machines physiques reliées par un commutateur Ethernet 20 Gbit/s dont une dédiée pour le gestionnaire autonomique. La couche de virtualisation repose sur OpenStack [21]. Dans le cadre de notre expérimentation, nous avons considéré un unique type de VM pré-configurée (1CPU, 2Go RAM, 20Go disque, ubuntu-12.04-server).

Injecteur de charge. Nous avons eu recours à Gatling [22] en tant qu'injecteur de charge pour simuler les requêtes HTTP clientes et stresser l'application.

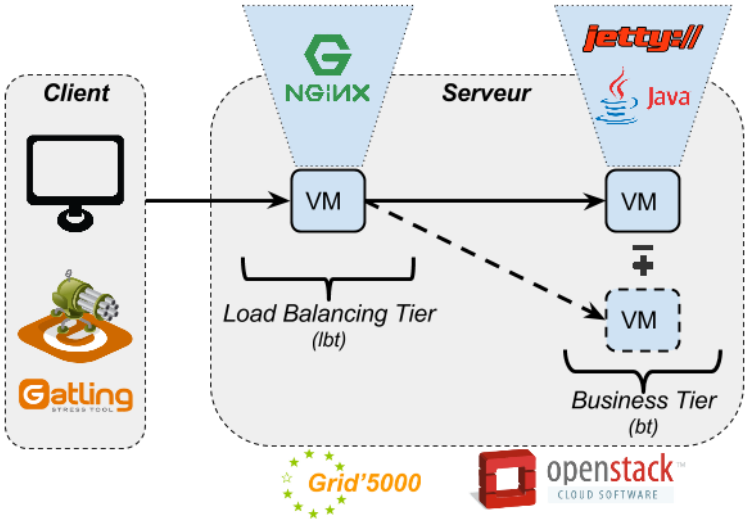


FIGURE 7.14 – Architecture globale [Dup16].

19. https://nginx.org/en
20. https://www.eclipse.org/jetty
21. https://www.openstack.org
22. https://gatling.io

Tactiques et expériences. Nous allons effectuer 3 expériences pour lesquelles nous avons mis en place des actions d'adaptation différentes en réponse aux symptômes `High_ResponseTime(tier)` et `Low_ResponseTime(tier)`.

L'adaptation de la première expérience repose uniquement sur l'élasticité horizontale IaaS (cf. Listing 7.7) alors que l'adaptation de la deuxième expérience repose uniquement sur l'élasticité verticale SaaS (cf. Listing 7.8). La dernière expérience admet une adaptation multi-couche au travers d'une tactique mettant en oeuvre à la fois l'élasticité IaaS et l'élasticité SaaS dans le cas d'un symptôme `High_ResponseTime(tier)` (cf. Listing 7.9). En revanche, un simple *Scale In Infra* est appliqué dans le cas d'un symptôme `Low_ResponseTime(tier)`.

```
1  begin
2  when High_ResponseTime(Tier businessTier)
3  do
4    businessTier.soi;
5  end
6
7  begin
8  when Low_ResponseTime(Tier businessTier)
9  do
10   businessTier.sii;
11 end
```

Listing 7.7 – Tactiques IaaS dans ElaScript

```
1  begin
2  when High_ResponseTime(Tier businessTier)
3  do
4    businessTier.sds;
5  end
6
7  begin
8  when Low_ResponseTime(businessTier)
9  do
10   businessTier.sus;
11 end
```

Listing 7.8 – Tactiques SaaS dans ElaScript

```
1  begin
2  when High_ResponseTime(Tier businessTier)
3  do
4    [
5      businessTier.soi;
6    || // Parallelism
7      businessTier.sds(4);
8    ]; // Wait until the VM is started and ready to accept request
9    businessTier.sus(4);
10 end
```

Listing 7.9 – Tactique *cross-layer* dans ElaScript

Stratégies. Nous allons évaluer les trois *stratégies d'élasticité* suivantes – chargées dans le catalogue de stratégies dans la base de connaissance 'K' – que nous appelons *Cost_First*, *Reactivity_First* et *QoE_First* :

- *Cost_First* : Coût > Énergie > QoS > TempsReconf > Réactivité > QoE ;
- *Reactivity_First* : Réactivité > QoE > QoS > TempsReconf > Coût > Énergie ;
- *QoF_First* : QoF > Coût > QoS > Réactivité > TempsReconf > Énergie.

Expérimentation et résultats

Le scénario appliqué pour nos expériences, d'une durée totale de 40 min, se décompose en 3 phases admettant des catégories de charge de travail distinctes variant en termes d'amplitude et de fréquence [KYTA12]. Notre objectif est d'évaluer l'impact des différentes tactiques considérées selon différents types de charge entrante.

Configuration initiale. La configuration initiale de l'infrastructure et de l'application est identique pour les trois expériences réalisées : chaque tier est constitué d'une unique VM pré-configurée. L'application est paramétrée avec l'offre maximale (i.e., offrant le meilleur niveau de QoE). Les temps de réponse du tier métier sont collectés toutes les 15 s et nous les confrontons à deux seuils pour *High_ResponseTime* et *Low_ResponseTime* qui correspondent respectivement à la valeur de 400 ms et de 20 ms.

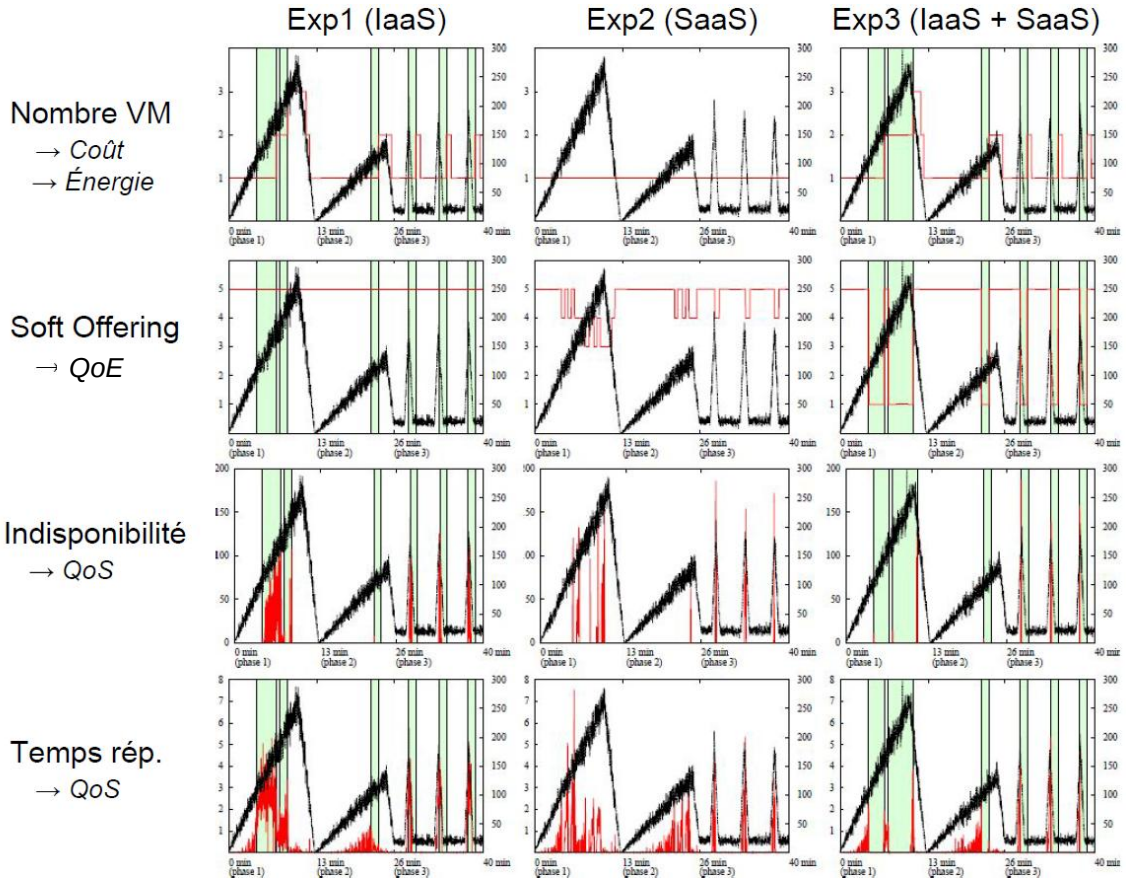


FIGURE 7.15 – Résultats des expériences [Dup16].

Résultats. Dans le cadre de notre expérimentation, nous avons collecté un certain nombre de métriques afin de comparer les différents tactiques d'élasticité. La Figure 7.15 présente les résultats obtenus pour nos trois expériences, regroupées par colonne. Nous nous sommes intéressés aux 4 métriques suivantes, constituant les lignes de notre figure :

- *Taille de l'infrastructure* : le nombre de VM utilisées par le tier métier ;
- *Offre de l'application* : l'offre des composants qui constituent le tier métier. Les offres de tous les composants sont homogènes (i.e., toutes les VM du tier ont la même offre) ;

— *Nombre de requêtes en échec* : la quantité de requêtes tombées en erreur chaque seconde. Cette métrique donne une indication quant à la disponibilité (*availability*) du service ;
— *Temps de réponse moyen* : le temps de réponse moyen des requêtes traitées (sans erreur) par seconde.

La courbe rouge dans chacune des sous-figures décrit l'évolution pour chaque métrique concernée tandis que la zone verte indique le temps d'initiation d'une VM. La courbe noire représente notre scénario de charge de travail qui se découpe en 3 phases. Les résultats détaillés de chaque expérience ne seront pas commentés par manque de place (pour plus de précisions, cf. [Dup16]).

Notre analyse expérimentale montre que chaque tactique a ses avantages et ses inconvénients en fonction de la charge de travail comme détaillé ci-dessous :

— Les tactiques IaaS favorisent la QoE au détriment de la réactivité et du coût de l'infrastructure. L'action *Scale Out Infra* est intéressante dans le cas d'une légère augmentation continue de la charge de travail car il n'est pas nécessaire que l'application soit réactive : au lieu de cela, il est nécessaire d'augmenter continuellement la quantité de ressources de calcul (VM).
— Les tactiques SaaS favorisent le coût et la réactivité de l'infrastructure au détriment de la QoE. Une action *Scale Down Soft* est logique dans le cas d'une surcharge de travail où l'ajout de ressources d'infrastructure peut prendre trop de temps pour répondre à des changements soudains.
— La tactique *cross-layer* favorise la réactivité au détriment du coût de l'infrastructure et ponctuellement la QoE. Cette tactique doit être appliquée dans le cas d'une charge de travail qui augmente modérément car il y a un besoin de réactivité et de mise à l'échelle continue des ressources d'infrastructure.

Pour conclure, les trois stratégies d'élasticité évoquées ci-avant orientent distinctement le choix de la tactique qui va être exécutée par le gestionnaire autonomique (cf. Tableau 7.5). Les *stratégies d'élasticité* offre à l'administrateur humain un moyen à la fois simple et efficace d'orienter la prise de décision ("*human-in-the-loop*" [OGT+99]).

Stratégie d'élasticité choisie	**Tactique exécutée**
Cost_First	Tactiques SaaS
Reactivity_First	Tactique *cross-layer*
QoE_First	Tactiques IaaS

Table 7.5 – Résultat des *stratégies d'élasticité* sur le choix des *tactiques*.

7.4.4 Conclusion

Avec le *framework* ElaStuff, nous proposons une solution complète et outillée pour la gestion de l'élasticité multi-couche. Pensé dès le départ comme un *framework*, il permet de contribuer aisément au développement du catalogue de symptômes ou de tactiques. Avec les stratégies, notre objectif était de déterminer le bon degré d'automatisation (cf. Section 3.3.3) pour laisser à l'administrateur Cloud un moyen d'exprimer ses préférences d'adaptation.

De nombreuses perspectives sont possibles. Par exemple, une extension possible du framework ElaStuff serait de pouvoir paralléliser plusieurs processus de reconfiguration du système afin d'accroître sa capacité d'adaptation face à un environnement hautement dynamique. Grâce à l'aspect modulaire du *management element* (i.e., les ressources Cloud), nous proposons une ébauche de cette solution dans [Dup16]. Une possibilité d'extension plus générale serait de proposer un modèle de fournisseur d'élasticité s'apparentant à une offre PaaS et permettant de configurer et piloter l'adaptation des ressources et des services du Cloud au sens large. Cela constituerait un service de gestion autonomique de l'élasticité (i.e., *Elasticity as a Service*) à la fois multi-couche et multi-nuage à destination des différents fournisseurs de Cloud.

7.5 Proposer des services sans empreinte carbone dans le SaaS

Dans sa thèse [Has17], Sabbir Hasan propose plusieurs contributions : l'intégration des énergies renouvelables via un *Cloud energy broker* dans les centres de données pour les rendre partiellement verts [HKLP14], le concept de la virtualisation de l'énergie verte [HKLP17] et enfin une gestion d'*auto-scaling* conscient de l'énergie renouvelable pour fournir des services SaaS verts [HALP16, HALP17, HAL17].

7.5.1 Motivations

Dans les sections précédentes, nous avons proposé une éco-élasticité logicielle pour le Cloud basée sur une auto-configuration multi-couche et fiable dirigée par les SLA. L'objectif était d'optimiser l'empreinte énergétique du Cloud en renvoyant une « valeur suffisante » au client mais moins énergivore.

Même si nous avons contribué à améliorer l'efficacité énergétique du Cloud, l'intégration des sources d'énergie « verte » dans les centres de données est le seul moyen pour diminuer l'empreinte carbone et augmenter la *greenitude* des services Cloud. Cependant, l'intégration de l'énergie verte dans les centres de données doit faire face à l'intermittence des énergies renouvelables ce qui rend difficile la garantie de services réellement verts.

Dans les sections suivantes, nous proposons plusieurs étapes pour atteindre un objectif final : la garantie d'un « green SLA » par un fournisseur de service.

7.5.2 Intégrer l'énergie verte dans le Cloud

Nous avons d'abord raisonné sur l'intégration de l'énergie verte dans le Cloud grâce à une phase de planification. Puis, malgré le caractère intermittent des énergies renouvelables, nous proposons une solution de dimensionnement de l'énergie verte à l'exécution.

Dimensionnement statique de l'énergie verte

L'*energy provider*. La libéralisation du marché de l'énergie et le déploiement des énergies renouvelables imposent d'ajouter une nouvelle couche dans la *pile logicielle* du Cloud : EaaS pour *Energy as a Service* (cf. Section 7.1). Cette couche est elle-même administrée par un nouvel acteur : le fournisseur d'énergie (*energy provider*).

Concrètement, dans la Figure 7.16, on peut montrer que l'approvisionneur en énergie du IaaS consiste en fait en un seul réseau de distribution d'électricité (Enedis/EDF) – agrémenté éventuellement d'une source d'énergie sur site (*on-site*) – dans lequel plusieurs fournisseurs *Energy-as-a-Service* (EaaS) vont intégrer leur énergie produite pour être distribuée jusqu'au client. Par exemple, sur le marché des REC (*Renewable Energy Certificates*) [23], les fournisseurs d'énergie renouvelable produisent de l'énergie verte et alimentent le réseau électrique mais vendent leurs crédits d'énergie renouvelable directement dans un marché aux consommateurs (e.g., fournisseurs IaaS). Autre exemple, le marché de l'énergie au comptant (*spot market*) permet aux producteurs d'énergie excédentaire de localiser instantanément les acheteurs disponibles pour cette énergie, de négocier les prix en quelques millisecondes et de fournir de l'énergie au client quelques minutes plus tard.

Comme les sources d'énergie verte sont de nature très intermittente, le prix sera très différent d'un fournisseur à l'autre en fonction de l'emplacement du site, la disponibilité des sources (éoliennes, panneaux solaires, ...) et les capacités de production de l'usine. Pour l'acteur IaaS, s'engager auprès d'un fournisseur d'énergie unique peut entraîner l'indisponibilité des besoins en énergie verte requise pour une certaine période, garantissant ainsi qu'un certain pourcentage de la disponibilité de l'énergie verte dans le centre de données ne peut pas être atteint (i.e., « green SLA » non respecté).

23. `https://www.epa.gov/greenpower/renewable-energy-certificates-recs`

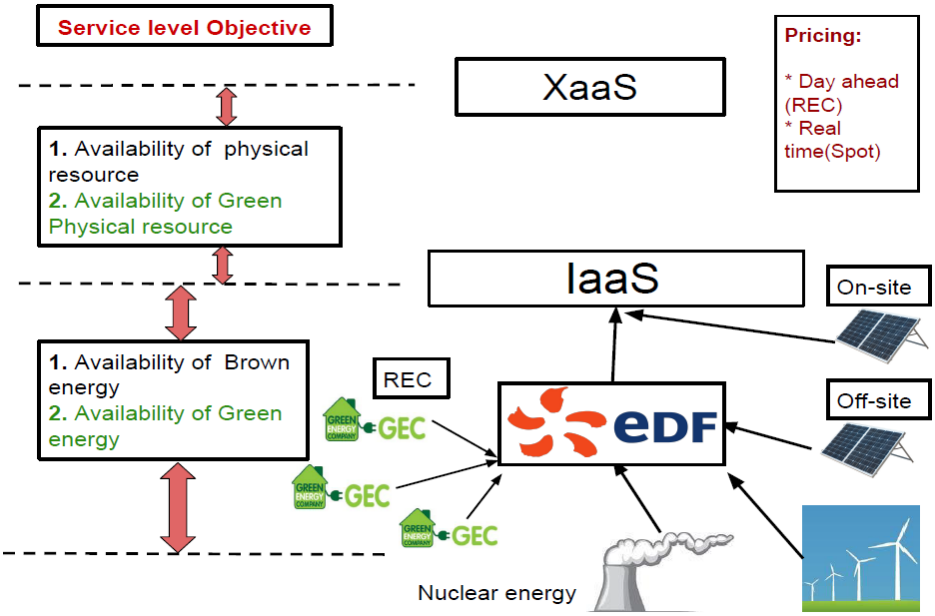


FIGURE 7.16 – *Energy as a Service* et « green SLA ».

Le *Cloud energy broker*. Afin de résoudre ce problème, nous proposons un courtier Cloud en énergie (*Cloud energy broker*) [HKLP14], qui peut calculer à l'avance la disponibilité et la combinaison de prix pour acheter de l'énergie verte sur le marché afin de rendre un centre de données partiellement vert sur la base d'un contrat SLA et dans le cadre d'une contrainte budgétaire. Nous avons pris en considération que les fournisseurs d'énergie verte publient un jour avant l'échéance la production d'énergie verte attendue et le prix par heure (*Day Ahead Pricing* (DAP)), ce qui est une pratique courante sur le marché européen de l'électricité et des *smart grids*.

La quantité d'énergie verte requise pour le IaaS pour délivrer des ressources vertes à ses clients et les informations sur les prix publiés par les fournisseurs d'énergie vont constituer les "*input*" du *Cloud energy broker*. Afin de respecter un budget prédéfini, le courtier calcule une solution Pareto-optimale pour sélectionner dynamiquement les fournisseurs EaaS en fonction des informations respectives pour chaque intervalle de temps (e.g., 1 heure). Nous abordons notre problème d'optimisation avec la programmation par contraintes (CP) [RBW06], puisque la CP accepte tout type de relations pour formuler des contraintes constituées d'inégalités linéaires. Notre courtier en énergie a été évalué avec une charge de travail réelle de PlanetLab [PP06] (cf. [HKLP14] pour plus de détails).

Dimensionnement de l'énergie verte à l'exécution

Si la planification de l'intégration de l'énergie verte dans le Cloud est un minimum, une approche statique ne nous permet pas de réagir à l'exécution à des changements dynamiques non prévisibles comme une soudaine demande de ressources vertes côté IaaS de la part de ses clients SaaS. Ainsi, nous sommes confrontés à une approche peu *élastique* et le SLA vert n'est plus respecté.

Un moyen de surmonter cette difficulté est d'utiliser le stockage d'énergie – ou la batterie – pour stocker l'énergie verte quand elle est abondante et la décharger plus tard pour réduire la demande d'électricité des centres de données ou pour satisfaire les SLA verts entre les fournisseurs IaaS et SaaS. Le stockage d'énergie entraîne des coûts supplémentaires pour les centres de données, ce qui ne constitue donc pas une solution intéressante pour les centres de données de petite échelle. De plus, les stockages ont des capacités finies pour recharger l'énergie et leur durée de vie est une fonction décroissante de la profondeur de la décharge (DoD) et des cycles de charge/décharge [DLJ+13]. De

plus, si la production d'énergie verte est supérieure à la capacité de stockage, l'énergie restante est gaspillée. En conclusion, comment gérer l'indisponibilité de l'énergie verte au moment de l'exécution si l'approche du stockage présente plusieurs inconvénients, si la production d'énergie verte *on-site* est insuffisante et si le marché *spot* est trop onéreux ?

Aussi, pour aborder le problème du dimensionnement de l'énergie verte à l'exécution, nous proposons le concept de *virtualisation* de l'énergie verte, tout en introduisant des « green SLA » entre le fournisseur de services et les utilisateurs finaux. L'énergie peut être virtuellement verte pendant une période donnée si l'abondance d'énergie verte est présente, de façon apériodique, dans un intervalle de temps avec le déficit d'énergie verte dans le reste de la période. Par conséquent, le concept de virtualisation va augmenter la *greenitude* de l'énergie, plutôt que d'augmenter la quantité d'énergie verte elle-même.

Concrètement, lorsque la disponibilité de l'énergie verte est supérieure à la demande, nous utilisons la totalité de l'énergie verte disponible mais caractérisons l'intervalle comme un *intervalle excédentaire*. Lorsque l'énergie verte est insuffisante pour répondre à la demande, nous annulons l'*intervalle dégradé* avec l'intervalle excédentaire. Nous utilisons le terme *virtualisation* parce que nous annulons un intervalle dégradé (manque d'énergie verte) avec un intervalle excédentaire (énergie verte excessive par rapport à la demande), mais du point de vue du client ou du fournisseur SaaS, ils voient l'intervalle comme *intervalle idéal*, bien que l'énergie verte ne soit pas présente instantanément mais plutôt virtuellement présente (cf. La Figure 7.17).

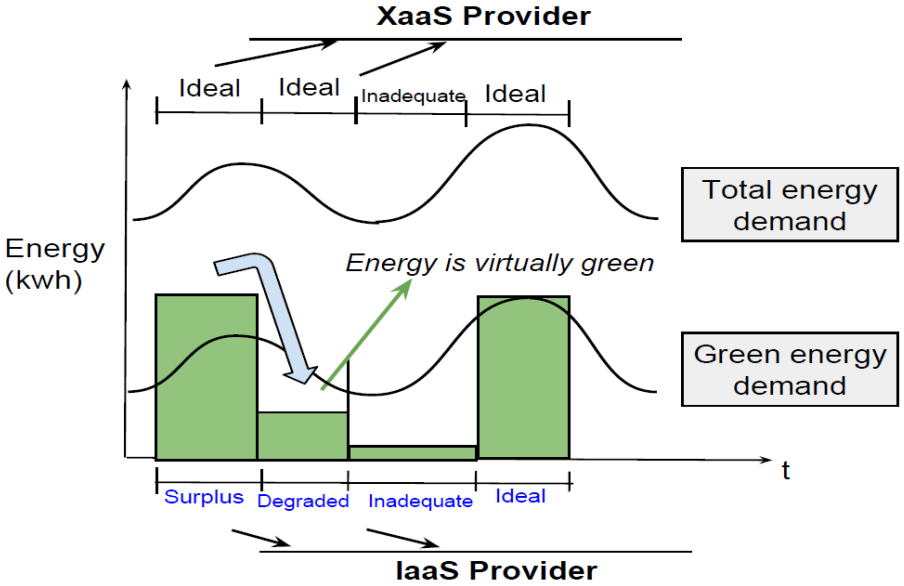


FIGURE 7.17 – Virtualisation de l'énergie verte.

Dans notre approche, le « green SLA » est observé et évalué par intervalle. Nous étendons CSLA [KL12b] afin d'évaluer un objectif de niveau de service (SLO) en utilisant deux seuils et de proposer une nouvelle catégorie d'état : *Surplus* qui est utilisé pour annuler l'intervalle dégradé [24].

Des expériences ont été menées avec le profil réel de charge de travail de PlanetLab et le modèle de puissance de serveur de SPECpower pour démontrer qu'un « green SLA » peut être établi avec succès et sans coût supplémentaire (cf. [HKLP17] pour plus de détails).

24. Rappelons qu'un SLO est évalué par CSLA en 3 catégories : *Ideal*, *Degraded*, *Inadequate* (cf. Section 7.2).

7.5.3 Vers des services verts

Dans la section précédente, notre réflexion a porté principalement sur les moyens d'intégrer l'énergie renouvelable au niveau du IaaS dans le but de proposer des SLA verts aux couches supérieures XaaS. L'étape suivante est naturellement de s'interroger sur la façon dont les applications SaaS modernes peuvent tirer profit de la présence – ou de l'absence – d'énergie verte.

Traditionnellement, les centres de données hébergent des applications hétérogènes telles que des *applications interactives* et des *applications par lots* (*batchs*). Dans ce dernier cas, les lots arrivent au *datacenter* avec des échéances, et peuvent donc être exécutés à différents moments lorsque l'énergie verte est disponible en respectant le délai prévu [GKL+13]. Au contraire, les applications interactives possèdent une flexibilité moindre, c'est-à-dire qu'elles doivent réagir avec peu ou pas de latence, sinon la qualité de service (QoS) peut être sérieusement affectée. Comme les applications interactives ne peuvent pas être planifiées à l'avance, l'adaptabilité à l'énergie verte ne peut être réalisée que si l'application hérite de la capacité d'utiliser intelligemment l'énergie verte disponible. Par conséquent, la prise de conscience (*awareness*) de l'énergie verte par les applications interactives doit être considérée comme une approche intéressante pour réduire l'empreinte carbone et augmenter la *greenitude* du service.

Alors que certains travaux abordent des modèles de référence conceptuels pour la construction d'applications logicielles vertes [NDKJ11, KDNH15], la prise de conscience de l'énergie verte par les applications Cloud est très peu traitée. Seul le monde du calcul scientifique, du HPC et des *batchs jobs* proposent des solutions [GKL+13, LOM15]. L'adaptabilité à l'énergie verte dans les applications SaaS interactives n'a pas encore été abordée.

Applications SaaS interactives *green energy-aware*

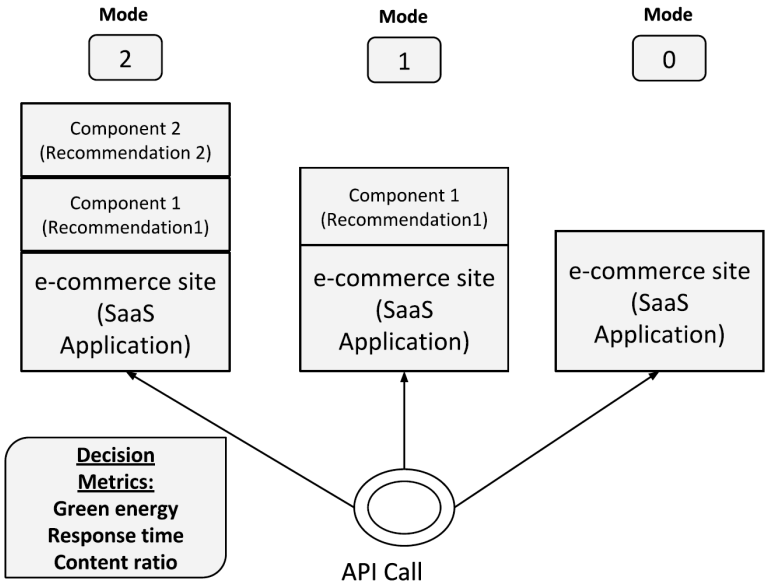


FIGURE 7.18 – SaaS *green energy-aware* [Has17].

Quand on considère une application SaaS interactive, du point de vue du client final, le temps de réponse et la disponibilité sont les indicateurs clés. Or, si l'on considère la nature de l'énergie (i.e., présence *vs* absence d'énergie renouvelable), cela peut affecter les propriétés de la QoS si les services sont liés à la quantité d'énergie disponible. Par conséquent, formuler des stratégies en

présence/absence d'énergie verte tout en respectant les paramètres de QoS traditionnels est un axe de travail important pour améliorer l'efficacité énergétique et réduire les émissions de carbone du Cloud.

La plupart des applications Cloud populaires sont conçues pour servir leurs clients avec quelques fonctionnalités supplémentaires comme par exemple plusieurs recommandations de produits dans une application de commerce électronique, qui améliorent la qualité d'expérience (QoE) de l'utilisateur mais ne constituent pas la fonctionnalité principale du service. En identifiant un composant logiciel indépendant mais gourmand en ressources (e.g., recommandation) qui peut être isolé pour être activé/désactivé, nous pourrions réduire la consommation d'énergie lorsque l'énergie verte devient rare. Par conséquent, augmenter ou diminuer progressivement l'expérience utilisateur en fonction de la présence d'énergie verte tout en atteignant des performances respectables est un moyen de rendre l'application SaaS interactive adaptable à la présence d'énergie verte (cf. Figure 7.18). Dans [HALP16, HALP17], nous proposons d'étudier les compromis entre l'énergie, la performance et l'expérience utilisateur dans les applications Cloud *interactives* basées sur les performances (temps de réponse et disponibilité) et sur la métrique de la qualité d'énergie.

SaaScaler

Nous proposons une architecture PaaS nommée SaaScaler, basée sur une boucle MAPE-K, qui hérite de la capacité de détecter des informations à partir de plusieurs couches (e.g., informations sur l'énergie verte) tandis que les actions sont effectuées uniquement sur l'application SaaS (cf. Figure 7.19). L'idée est de changer de *mode* dans l'application SaaS en fonction de l'état courant du système. Par exemple, passer de 2 recommandations (types *User-to-User* et *Item-to-Item* [KR12]) à 1 recommandation pour une application de e-commerce. On retrouve bien entendu le concept sous-jacent d'élasticité logicielle (cf. Section 7.4).

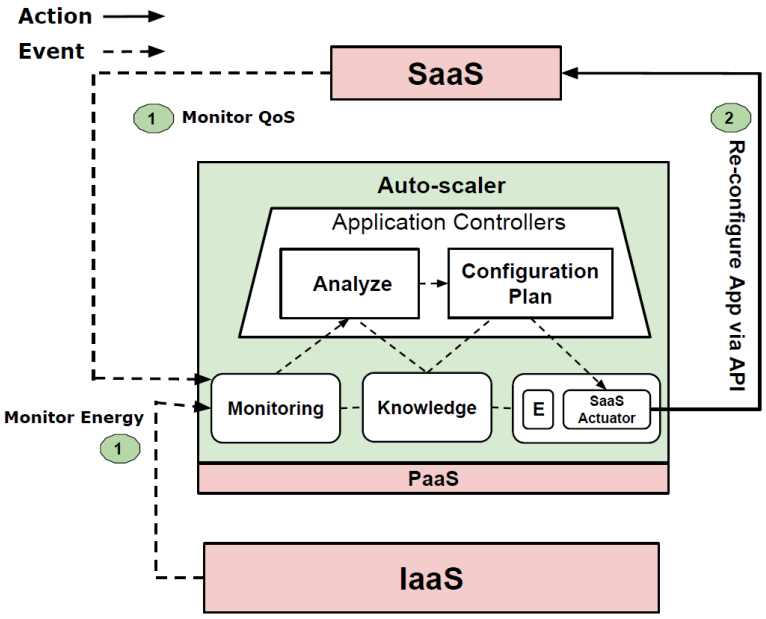


FIGURE 7.19 – Architecture SaaScaler [Has17].

Concernant les règles d'adaptation, nous avons développé plusieurs contrôleurs qui consistent en l'implémentation des fonctions "Analyze" et "Planning" de la boucle MAPE-K. Dans [HALP16, HALP17], nous présentons deux contrôleurs hybrides (multicritère) qui suivent une architecture de boucle inspirée de l'automatique et la théorie du contrôle avec des boucles de rétroaction (*feedback*

loop) et des boucles de contrôle imbriquées (*nested control loop*). L'intérêt de cette conception est de fiabiliser notre adaptation grâce aux effets de stabilisation des écarts par rapport à une consigne [FJBE15]. Il s'agit des contrôleurs *Hybrid-green controller* et *Hybrid-qoe controller* (cf. Figure 7.20). Les deux contrôleurs prennent en compte comme critère la performance de l'application mais le premier prend en plus en compte la production de l'énergie verte alors que le deuxième prend en compte l'état de l'application (i.e., les différents modes). Les détails des algorithmes des contrôleurs sont présentés dans [HALP16, HALP17].

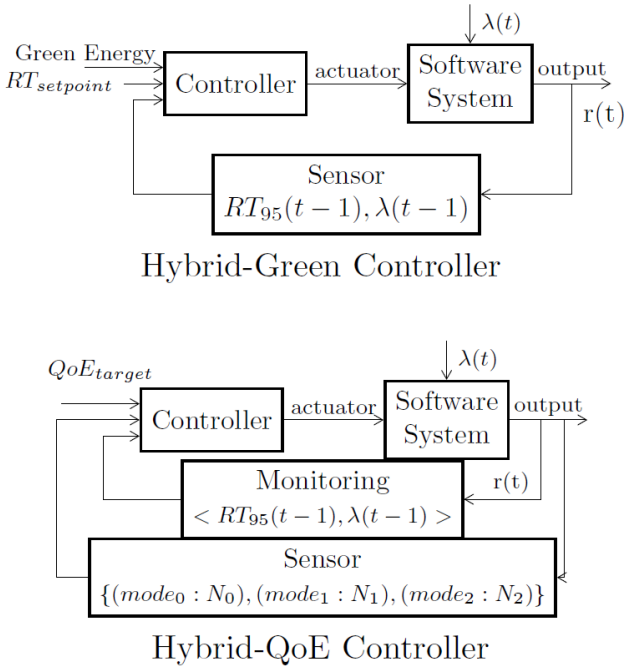


FIGURE 7.20 – Contrôleurs hybrides [Has17].

7.5.4 Evaluation et expérimentation

Cette évaluation a pour objectif d'éprouver SaaScaler et nos différents contrôleurs autonomiques.

Banc de tests

Nous supposons avoir une infrastructure IaaS statique pour valider dans quelle mesure la couche SaaS peut réduire la consommation d'énergie. Nous avons étendu RUBiS [25], un site de vente aux enchères eBay, qui est utilisé pour les expérimentations Cloud. Elle représente ici une application SaaS interactive. En outre, deux profils de charge de travail *open source* (Wikipedia et Fifa) sont utilisés pour l'expérimentation ainsi qu'une trace de production d'énergie solaire.

Infrastructure IaaS. Les expériences ont été menées sur le site de Grid'5000 Lyon, avec 2 machines physiques reliées par un switch Ethernet 10 Gbit/s et connectées à un wattmètre. Chaque machine possède deux processeurs Xeon 2,3 GHz (6 cœurs par CPU) et 16 Go de RAM, sous Linux. OpenStack Grizzly a été utilisé comme plate-forme, ce qui nécessite une machine physique dédiée pour le système de gestion de contrôleur. Par conséquent, la deuxième machine physique a été utilisée comme nœud de calcul pour héberger des machines virtuelles préconfigurées pour exécuter Ubuntu.

25. `http://rubis.ow2.org`

Application SaaS. Dans Brownout [KMrHR14], les auteurs ont fourni un moteur de recommandation de type *User-to-User* qui peut améliorer la QoE de l'utilisateur. Parallèlement à cela, nous avons mis en place une recommandation *Item-to-Item* assez simple pour offrir une meilleure expérience utilisateur. Le fonctionnement de cette dernière peut être résumé comme suit : « Récupérez 5 produits du même vendeur et de la même catégorie de produits qui ont un nombre d'enchères supérieur ou identique avec une score élevé ». Bien que les deux moteurs de recommandation manquent de sophistication, ils constituent un exemple raisonnable d'expérience utilisateur qu'une application Cloud peut isoler des fonctionnalités principales pour les activer ou les désactiver à l'exécution.

Dans la Figure 7.21, nous présentons le résultat d'une courte expérimentation montrant les variations de consommation d'énergie de l'application RUBiS avec ou sans recommandation.

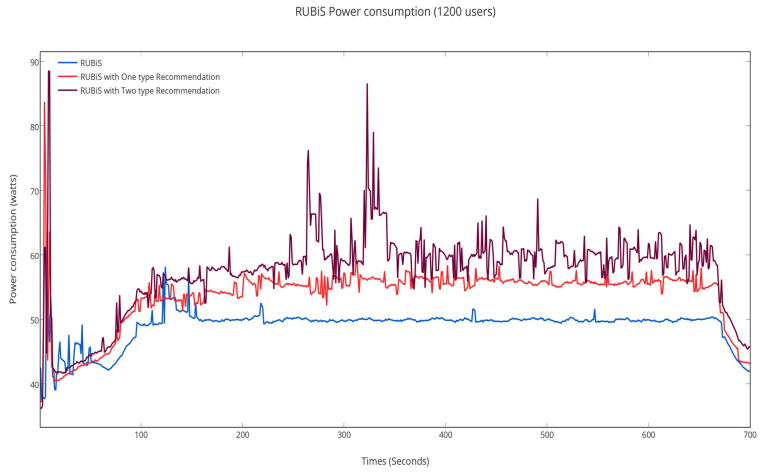


FIGURE 7.21 – Consommation énergétique de RUBiS avec 2 recommandations.

L'application RUBiS étendue a été déployée en mode SaaS et organisée en architecture 3-tiers (cf. Figure 7.22). Notre nœud de calcul est constitué d'une VM avec équilibreur de charge Nginx [26] qui distribue les requêtes sur trois machines virtuelles du serveur d'application Nginx, chacune ayant 1 CPU et 2 Go de RAM et une seule VM de serveur MySQL de 8 CPU et 16 Go de RAM.

Workloads. Nous avons pris la vraie trace de la page allemande Wikipedia d'un jour et le trafic du site Web de la coupe du monde de Fifa 1998 sur un mois et demi [FJLB15] et nous avons mis à l'échelle l'ensemble de données pour l'adapter à notre expérience. Alors que la trace Wikipedia a un modèle régulier et incrémentiel de requêtes au cours du temps, la trace de la Fifa possède de forts pics temporels. Nous avons utilisé Gatling [27] comme injecteur de charge pour générer la charge de travail souhaitée. Enfin, nous avons obtenu la production d'énergie solaire qui a été ajoutée au réseau pendant un jour (12 avril 2016) auprès de RTE et nous avons mis à l'échelle les valeurs adaptées à notre expérience.

Résultats

Notre expérimentation à consister à comparer les contrôleurs *Hybrid-green controller* et *Hybrid-qoe controller* entre eux mais également avec un approche non adaptable et aussi avec un contrôleur mono-critère basé seulement sur le temps de réponse. Pour cela, nous avons fixé en amont le SLA suivant avec CSLA : (1) temps de réponse : 2s avec *fuzziness* de 1s (cf. Section 7.2) ; QoE : 80% (30% en mode 2, 50% en mode 1).

26. `https://nginx.org/en`
27. `https://gatling.io`

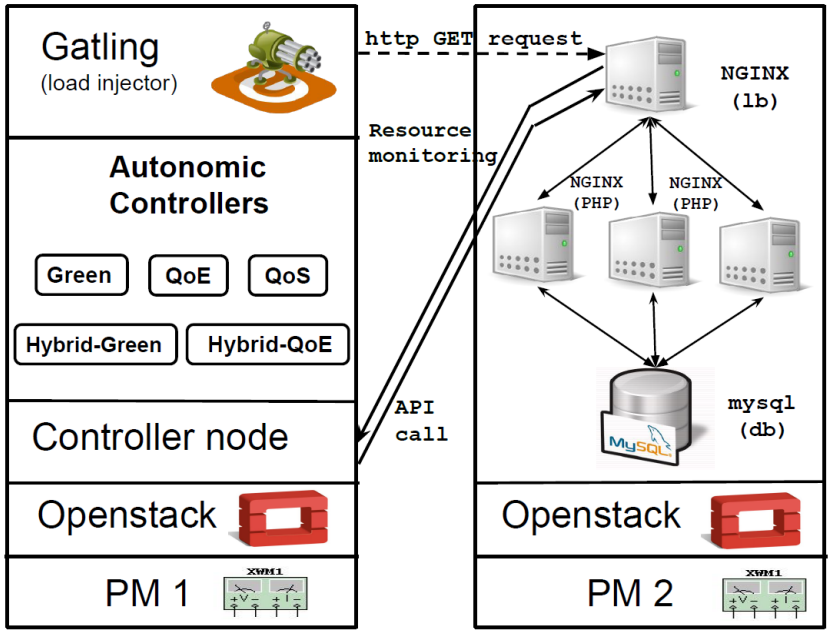


FIGURE 7.22 – Expérimentation SaaScaler [HALP17].

De nombreux résultats (courbes, tableaux comparatifs) sont présentés dans [HALP17]. Les principaux résultats peuvent être résumés comme suit :

- Les contrôleurs peuvent fournir des garanties pour maintenir le temps de réponse du 95e centile à proximité du point de consigne cible (2s), tandis que la consommation d'énergie non verte peut être réduite de 13% sans déprovisionner les ressources de l'infrastructure et les revenus augmenter de 66% ! [28]
- Notre approche ajuste également dynamiquement les besoins de capacité en soulageant les ressources existantes pour permettre à 29% d'utilisateurs en plus d'accéder à l'application SaaS par rapport à l'approche traditionnelle non adaptative.
- Les résultats vérifient que la consommation d'énergie dévie aussi peu que 0,07% lorsque notre approche est mise à l'échelle en utilisant plusieurs machines physiques. Cela garantit l'augmentation linéaire de la consommation d'énergie et des économies alors que le nombre de machines physiques augmente.

7.5.5 Conclusion

Avec sa thèse [Has17], Sabbir Hasan relève un véritable défi à savoir rendre possible le « green SLA » dans l'informatique en nuage. Pour cela, il s'attaque à la fois à la relation EaaS – IaaS et à la relation IaaS – XaaS, permettant du même coup de remonter une information énergétique au sommet de la *pile logicielle* du Cloud. Il ne reste plus qu'à proposer des services verts conscients de leur source d'énergie. Ces services SaaS avec un *label green* correspondent à une demande citoyenne de plus en plus forte.

Parmi les perspectives, nous avons présenté dans [HAL17] une extension des travaux précédents en utilisant une infrastructure élastique au niveau IaaS (alors qu'elle était statique avec SaaScaler). Tandis que les applications sont adaptées en reconfigurant dynamiquement leur niveau de service en fonction des performances et/ou de la disponibilité de l'énergie verte, l'infrastructure prend en

28. Chaque demande servie sans recommandation correspond à 1 unité monétaire, avec 1 recommandation à 0,25 unité, avec 2 recommandations à 0,5 unité seulement si ces demandes ont été servies au-dessous de 2s et déduit 1 unité pour les demandes échouées.

charge l'ajout/suppression de ressources en fonction de la demande en ressources de l'application. A l'instar des travaux de la Section 7.3, une véritable coordination de boucles de rétro-action reste à proposer.

7.6 Bilan

En s'inspirant à la fois du concept d'innovation frugale (*jugaad*) [APR13] et du mécanisme d'« effacement de la consommation électrique », nous avons proposé des contributions dans le domaine du *Cloud computing* permettant d'une part de diminuer l'empreinte énergétique des centres de données mais également de favoriser l'émergence de services verts pour le client final.

Parmi les artefacts développés, nous pouvons citer : une langage SLA pour le Cloud [Kou13], la coordination de boucles autonomiques entre SaaS et IaaS [ADOJ13], le concept d'éco-élasticité logicielle, un DSL pour coder l'éco-élasticité multi-couche [Dup16], le modèle de services *Energy as a Service* (EaaS) et ses *energy provider*, la virtualisation de l'énergie verte, le « green SLA » [Has17], etc.

Pour rendre possible l'adaptation dynamique dans les architectures en nuage, tous ces résultats utilisent des boucles autonomiques MAPE-K. Elles sont parfois basées sur de simples règles à base de seuil [Kou13, Dup16], mais également sur des modèles plus sophistiqués comme les files d'attente [Kou13], des solveurs de contraintes [ADOJ13], des boucles de rétro-action [Has17].

Comme nous l'avions promis, nous avons gardé en tête et montré que la performance et la fiabilité restent des critères essentiels dans le processus de reconfiguration dynamique, éléments indispensables à son adoption plus large (cf. Section 3.3).

- Concernant la fiabilité, nous avons fait plusieurs propositions. Le langage CSLA de par sa conception intègre dans ses gènes la nature dynamique du Cloud pour minimiser la violation de contrats SLA. La coordination de boucles autonomiques atténue très fortement les interférences entre prises de décision locales des différentes couches de services pour éviter des états globaux non optimaux. Le langage dédié ElaScript permet à l'administrateur Cloud de lever rapidement des incohérences dans ses plans de reconfiguration en encadrant l'écriture de ses scripts. Enfin, les boucles de rétro-action utilisées plus récemment [HALP17] dans la conception de nos gestionnaires autonomiques améliorent la régulation et la stabilité de la reconfiguration.
- Concernant la performance, notre contribution majeure consiste en la modélisation de l'*élasticité logicielle* qui change la granularité de la reconfiguration et de ce fait diminue la surface à reconfigurer et donc le temps de reconfiguration. Notons que l'élasticité logicielle vient également avec une API efficace qui fournit des « armes » pour améliorer le temps de réalisation de l'adaptation (e.g., opérateur de parallélisme de ElaScript).

Chapitre 8

Bilan des contributions et perspectives

Dans ce court chapitre, nous dressons le bilan de ces vingt ans de travaux et proposons quelques perspectives.

8.1 Bilan

8.1.1 Résumé des travaux

Ce mémoire présente une synthèse des travaux de recherche que j'ai réalisés autour de la reconfiguration dynamique d'architectures logicielles. Il a volontairement été organisé en 3 parties distinctes.

La première partie décrit une exploration de l'adaptation dynamique du logiciel avec la volonté de poser des concepts et d'identifier des verrous à lever. Après avoir motivé l'intérêt de l'adaptation dynamique et modélisé son processus au cœur du cycle de vie du logiciel, nous établissons un certain nombre de caractéristiques indispensables à l'adaptation (e.g., réification, composabilité). Nous proposons de considérer l'adaptation dynamique comme un *objet de première classe* afin de favoriser un découplage spatio-temporel entre code métier et logique d'adaptation. Nous rappelons également que l'adaptation dynamique ne peut se faire au détriment de la qualité du logiciel et de son intégrité et nous énumérons un certain nombre de propriétés à respecter, de pistes à étudier dans le processus d'adaptation dynamique.

La deuxième partie présente nos contributions pour mettre en œuvre l'adaptation dynamique dans les logiciels à base de composants. D'une part, nous proposons un patron d'auto-administration pour les architectures à composants (Safran), des langages dédiés pour la navigation et la reconfiguration dans les architectures logicielles Fractal (FPath/FScript), un *framework* pour faciliter la création d'applications *context-aware* (WildCAT). D'autre part, nous fournissons une approche multi-étape pour des reconfigurations dynamiques fiables comprenant une partie prévention de fautes basée sur une analyse statique et une partie tolérance aux fautes assurée par un moniteur transactionnel. Cette « chaine de validation » de l'adaptation dynamique fiable s'appuie sur une spécification des configurations et des reconfigurations du modèle Fractal basée sur une logique du premier ordre.

La troisième partie a pour objet de montrer que la reconfiguration dynamique dans les « architectures en nuages » (*Cloud computing*) peut apporter une réponse à un enjeu sociétal important, à savoir les transitions numérique et énergétique. Contrairement aux travaux actuels qui visent à améliorer l'efficacité énergétique des centres de données en proposant des solutions au niveau du matériel ou encore de l'infrastructure – c'est-à-dire sur les couches basses du Cloud – nous proposons une approche d'éco-élasticité logicielle sur les couches hautes du Cloud. En s'inspirant à la fois du concept d'innovation frugale (*jugaad*) et du mécanisme d'« effacement de la consommation électrique » – qui permettent de *déformer* le logiciel en vue de renvoyer une « valeur suffisante » au client mais moins énergivore – nous décrivons un certain nombre de contributions dans le domaine du Cloud

permettant d'une part de diminuer l'empreinte énergétique mais également de favoriser l'émergence de services « verts » générant une faible empreinte carbone.

En conclusion, l'ensemble de ces travaux soulève des questions et apporte des réponses pour modéliser et mettre en œuvre l'adaptation dynamique dans les architectures logicielles. Plus spécifiquement pour architectures de type Cloud, nous fournissons des artefacts originaux (e.g., Cloud SLA, « effacement de la consommation du logiciel », virtualisation de l'énergie, SaaS *green energy-aware*) pour diminuer l'empreinte carbone des architectures en nuages.

8.1.2 Contributions dans le domaine du Cloud computing

> « *Another frequent issue is that resource management policies tend to focus on optimizing specific metrics and resources, often lacking a systematic approach to co-existence in the same environment of multiple control loops and cycles, multi-resource fairness, and holistic optimization across layers of the Cloud stack. [BSC+17]* » (Rajkumar Buyya - 2017)

> « *The future work should also focus at maximising the usage of green energy while meeting the QoS expectations of an application, both for the Fog and the Cloud. [BSC+17]* » (Rajkumar Buyya - 2017)

Dans l'article intitulé "*Manifesto for Future Generation Cloud Computing : Research Directions for the Next Decade*" [BSC+17], Rajkumar Buyya et une trentaine de co-auteurs réalisent un état des lieux de l'informatique dans les nuages et proposent un ensemble de perspectives. Parmi les challenges identifiés, nous traitons dans ce manuscrit la moitié d'entre eux et nous apportons une solution, à savoir :

- *Scalability and Elasticity* : l'élasticité logicielle (cf. Section 7.4) vient améliorer la *précision* et la *réactivité* du dimensionnement [WHGK14] ;
- *Resource Management and Scheduling* : la coordination de boucles autonomiques (cf. Section 7.3) pour administrer les niveaux IaaS et SaaS favorise une synergie pour la gestion des ressources du Cloud dans son ensemble ; la virtualisation de l'énergie (cf. Section 7.5.2) apporte une solution au problème du dimensionnement de ressources vertes à l'exécution ;
- *Reliability* : la garantie de SLA avec le langage CSLA (cf. Section 7.2), la coordination de boucles autonomiques (cf. Section 7.3) comme les scripts *"safe"* avec ElaScript (cf. Section 7.4.2) participent à la fiabilité du processus de dimensionnement ;
- *Sustainability* : les compromis QoS-énergie directement intégré dans le langage SLA (cf. Section 7.2) permet de renvoyer une « valeur suffisante » au client mais moins énergivore ; l'énergie vue comme *objet de première classe* et le développement d'applications *green energy-aware* (cf. Section 7.5.3) permet de réduire l'empreinte carbone du Cloud ;
- *Economics of Cloud Computing* : CSLA (cf. Section 7.2) est une preuve de concept pour proposer de nouveaux modèles économiques autour du Cloud : gestion de l'incertitude, du compromis ressources (vertes) *vs* QoS, des violations, etc. ;
- *Application Development and Delivery* : CSLA (cf. Section 7.2), ElaScript, ElaStuff (cf. Section 7.4) permettent aux administrateurs Cloud de contrôler par programmation les ressources et le dimensionnement de la plate-forme Cloud.

8.2 Perspectives

Dans la continuité de nos travaux, nous avons identifié un certain nombre de perspectives.

8.2.1 Vers un gestionnaire autonomique générique pour les couches XaaS

L'une des perspectives les plus prometteuses de nos travaux consiste en la définition d'un gestionnaire autonomique *générique* pour le Cloud.

Motivations

Comme nous avons pu le voir tout au long du Chapitre 7, concevoir/implémenter des boucles autonomiques, les outiller n'est pas à la portée du premier venu. Avec l'expérience de plusieurs années de recherche et de développement, nous affirmons que les systèmes Cloud, quelle que soit la couche XaaS de la *pile logicielle* du Cloud, partagent de nombreuses caractéristiques et objectifs communs, qui peuvent servir de base à un modèle plus homogène. En fait, chaque couche XaaS peut assumer le rôle de consommateur/fournisseur dans la pile de services du Cloud, et les interactions entre ces couches sont régies par les SLA (cf. Figure 8.1 à comparer à la Figure 7.1).

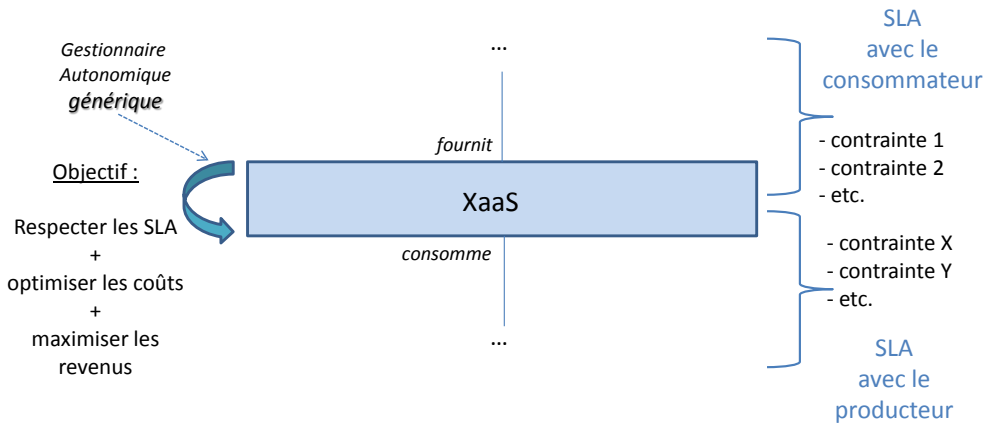


FIGURE 8.1 – Boucle autonomique générique XaaS.

Ainsi, après une année de travail préliminaire avec Jonathan Lejeune [1] et une première contribution intéressante [LAL17], nous obtenons un financement régional sur l'appel à projet Atlanstic2020 [2] pour le projet collaboratif CoMe4ACloud (*Constraints and Model Engineering for Autonomic Clouds*) [3]. Ce projet réunit 3 équipes du laboratoire portant les compétences complémentaires suivantes : ingénierie des modèles (équipe AtlanMod), optimisation et programmation par contraintes (équipe Tasc), Cloud et informatique autonomique (Ascola). Son objectif principal est de fournir une solution générique et extensible pour la gestion de l'exécution des services Cloud, indépendamment de la couche de Cloud à laquelle ils appartiennent.

Une approche dirigée par les modèles et une spécification par contraintes

Le modèle générique est basé sur un graphe orienté acyclique (DAG) avec des contraintes – devant être maintenues à l'exécution – formalisant les relations entre fournisseurs de services et leurs consommateurs. Par exemple, une machine virtuelle (i.e., consommateur) est cliente d'une machine physique (i.e., fournisseur) qui vient avec ses contraintes de capacité (e.g., nombre max de CPU/RAM). À partir de ce modèle, nous générons automatiquement un modèle de programmation par contraintes [RBW06] qui est ensuite utilisé comme outil de prise de décision et de planification des reconfigurations au sein de la boucle MAPE-K.

1. Post-doc dans l'équipe à l'époque, MdC à l'UPMC dans le laboratoire LIP6 depuis.
2. `http://atlanstic2020.fr/`
3. `https://come4acloud.github.io/`

Nous avons proposé une version initiale d'un modèle dans [LAL17]. Cependant, ce dernier ne vient pas avec une architecture basée sur un modèle approprié et un langage de modélisation réutilisable directement applicable à toutes les couches du nuage. Il faut notamment être expert des contraintes pour pouvoir manipuler le système. Aussi, nous avons l'intention de traiter ce problème via les contributions suivantes :

- une architecture basée sur *l'ingénierie des modèles* [Béz06, Sch06] pour la modélisation XaaS afin de favoriser la gestion autonomique générique. Cette architecture doit permettre la connexion avec un solveur de contraintes (Choco [PFL17]) et une transformation partielle vers/depuis le standard Cloud TOSCA [4] pour l'interopérabilité avec des solutions externes ;
- un langage de modélisation XaaS associé, pouvant prendre en charge n'importe quelle couche de Cloud possible. Pour fournir une syntaxe qui semble familière aux utilisateurs du Cloud, nous avons considéré YAML et proposé un langage dédié permettant à la fois de spécifier une topologie XaaS et d'initialiser les configurations associées ;
- un support d'outils basé sur Eclipse.

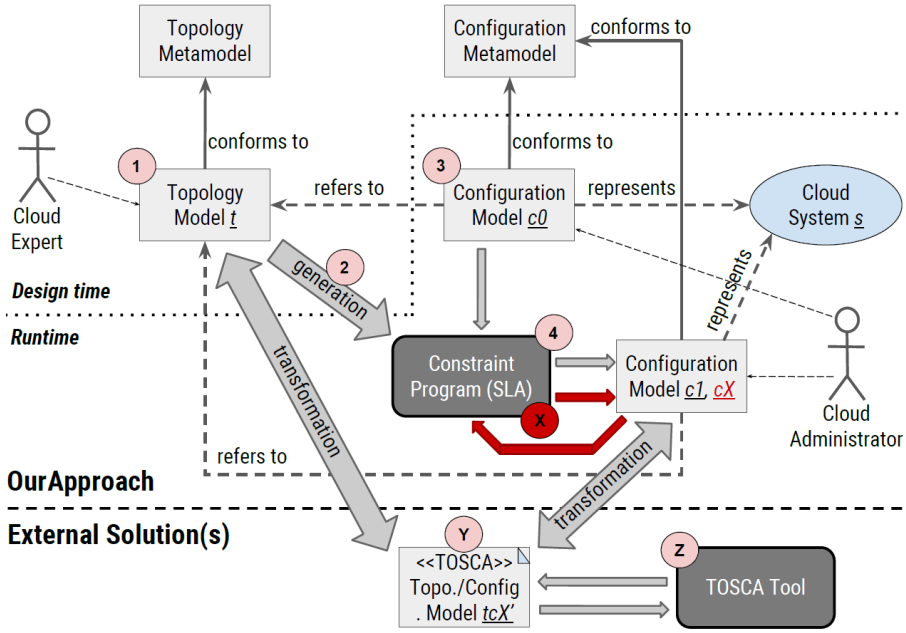


FIGURE 8.2 – Architecture CoMe4ACloud [BASA+18].

Les premiers résultats du projet CoMe4ACloud sont publiés cette année [BASA+18, ASAB+18]. L'architecture générale CoMe4ACloud regroupant interopérabilité TOSCA, génération vers Choco, modélisation de la topologie d'une couche XaaS, modélisation des configurations XaaS, boucle de reconfiguration est représentée schématiquement dans la Figure 8.2.

Dans le scénario de la vidéo-démonstration accessible en ligne [5], une architecture SaaS modélisée grâce à notre approche est reconfigurée dynamiquement. Il montre 1) la conception de la couche SaaS avec notre langage de modélisation XaaS ; 2) l'interopérabilité entre les modèles XaaS obtenus et un outil basé sur TOSCA (Eclipse Winery [6]) et 3) une illustration d'une adaptation donnée via notre architecture de décision (basée sur le solveur Choco).

Plus récemment, nous avons étudié comment la couche de modèle générique se connecte de façon transparente à la couche d'exécution avec une approche Models@run-time [BBF09] (cf.Section 3.1).

4. https://www.oasis-open.org/committees/tosca/
5. http://hyperurl.co/come4acloud
6. https://projects.eclipse.org/projects/soa.winery

Concrètement, nous avons avons étudié comment les données de surveillance d'un système IaaS en cours d'exécution sont reflétées dans le modèle et comment les modifications du modèle (effectuées par la boucle autonomique ou l'administrateur Cloud) sont propagées au système IaaS en cours d'exécution. Cette synchronisation *models—run-time* est évaluée sur une infrastructure IaaS basée sur OpenStack.

Une extension au Fog computing

Le *Fog Computing* [MB16] est un nouveau paradigme qui a pour but de distribuer les ressources/services de calcul et de stockage depuis les systèmes Cloud centralisés jusqu'à la périphérie des réseaux. D'après leurs promoteurs, ce nouveau paradigme offre de meilleures performances en termes de latence de service, de consommation d'énergie, de trafic réseau, de distribution de contenu, ...

Les principaux gains attendus sont doubles : 1) éviter les goulots d'étranglement du réseau et 2) réduire les coûts associés au mouvement des données. Cependant, la gestion des ressources dans de tels systèmes est difficile et nécessite des solutions entièrement décentralisées : il faut repenser la façon dont les différents nœuds de l'infrastructure peuvent collaborer les uns avec les autres.

Notre idée est donc de repartir des résultats de notre projet CoMe4ACloud et de les projeter dans le monde du Fog pour proposer une nouvelle architecture générique pour la gestion autonomique des ressources du Fog. Malheureusement, passer du *Cloud computing* au *Fog Computing* implique de résoudre des problèmes de passage à l'échelle, de gestion de fautes. En effet, le Fog utilise de nombreuses ressources hautement dynamiques et volatiles qui entrent en conflit avec la vision centralisée requise par un gestionnaire autonomique unique du Cloud. Pour prendre une décision, cela nécessiterait de collecter et d'agréger une trop grande quantité d'informations parfois peu stables.

Afin de s'adapter à l'approche du *Fog Computing*, nous proposons une combinaison de programmation distribuée [Ray13] et de programmation par contraintes [RBW06]. Notre nouvelle proposition sera basée sur une flotte de petits gestionnaires autonomiques répartis dans le Fog. Ils nous permettront d'aborder les questions de réactivité et de qualité de service, à la fois en limitant la collecte de données à une localité réduite et en simplifiant la combinatoire que doit résoudre le noyau du gestionnaire (i.e., le solveur) [YH98, MVV14].

8.2.2 Vers le développement de microservices verts pour le Cloud

Une autre perspective importante de nos travaux est de faciliter (encore) le développement de services verts et de « green SLA » pour les architectures en nuage.

Motivations

Dans le Chapitre 7, nous avons déjà proposé des langages dédiés (e.g., CSLA, ElaScript), les *frameworks* associés (e.g., ElaStuff) permettant la description et l'exécution des applications Cloud qui sont capables d'adapter à la fois l'infrastructure (élasticité de l'infrastructure) et l'architecture logicielle (élasticité logicielle) aux changements environnementaux. Cependant, aujourd'hui, les applications SaaS sont souvent conçues comme un bloc monolithique. Par conséquent, les applications SaaS sont difficilement élastiques et auto-reconfigurables à l'exécution.

Le paysage est train de changer car depuis peu, nous assistons à une nouvelle façon de produire du logiciel. D'une part, l'adoption rapide de la culture et des pratiques DevOps nous amène à reconsidérer le cycle de vie du logiciel et sa mise en production dans des architectures conteneurisées qui gagnent du terrain dans le Cloud (e.g., Google Container Engine, Amazon EC2 Container Service). D'autre part, l'apparition des architectures microservices (e.g., AWS Lambda) ré-interroge la construction du logiciel en proposant des structures plus modulaires, plus lâches, plus autonomes [7]. Ces deux élans constituent une opportunité inédite pour 1) rendre les applications SaaS élastiques ; 2) intégrer

7. `https://martinfowler.com/articles/microservices.html`

une dimension Green IT dans la fabrication du logiciel. En effet, d'une part, l'ingénieur DevOps a conscience des infrastructures et donc de leur consommation en ressources ; d'autre part, cet ingénieur peut architecturer son code pour organiser, classer les micro-services selon leur consommation en ressources. Cette capacité des microservices capables d'avoir conscience de leur consommation énergétique (un label Green) aura un impact important sur le Cloud mais surtout sur des infrastructures de petite taille comme le *Fog Computing* qui sont encore plus sensibles à la limitation des ressources ou l'Internet des objets, sensibles à la pénurie des sources d'énergie.

Objectifs

Le pilotage d'architectures microservices conteneurisées éco-responsables dirigé par la performance et l'énergie n'a pas été étudié pour le moment dans l'état de l'art. Notre objectif est d'affiner nos travaux existants [Dup16, Has17] pour créer des microservices verts et *in fine* proposer un socle pour le pilotage d'architectures microservices conteneurisées pour les « nuages verts ».

Ce socle sera composé à la fois d'outils, de *frameworks* et de langages dédiés pour aider l'ingénieur DevOps et/ou l'administrateur à piloter son Cloud ou Fog frugal. Chaque microservice sera piloté par un SLA propre (par exemple, l'un favorisant la performance, l'autre l'empreinte énergétique).

Même si notre objectif premier est d'expérimenter dans le domaine du Cloud, une expérimentation du socle hors de son périmètre initial (Cloud mono-site) vers le *Fog computing* (maillage de ressources hétérogènes, de micro-datacenters) sera initié. En effet, le plan de travail envisagé est le suivant :

- validation du socle logiciel avec des données énergétiques simulées en s'appuyant sur des traces réelles de production énergétique, notamment offertes sous forme d'*Open Data* par Enedis ;
- utilisation de la plateforme SeDuCe [8] qui propose une infrastructure dédiée à l'étude scientifique des problématiques croisées du Cloud Computing et des sources d'énergies renouvelables. Il s'agit d'un petit centre de données alimenté par des panneaux photovoltaïques dans les locaux de IMT Atlantique (campus de Nantes). Par conséquent, nous serons en mesure d'utiliser non seulement de données réelles liées à l'énergie, mais aussi d'observer comment l'énergie est consommée en fonction de la production d'énergie sur place et d'autres paramètres à l'exécution comme la charge client, le contrat SLA, la QoS fournie, ...
- expérimentation de la projection du socle Cloud pour le *Fog computing* : refonte d'un *use case* utilisé précédemment pour l'utiliser dans le Fog et pour observer *de facto* la problématique de la gestion des ressources limitées.

8.2.3 Vers des patterns de coordination de boucles de contrôle hétérogènes pour l'auto-adaptation dans le Fog

Dans nos travaux, nous avons utilisé différentes approches pour concevoir et implémenter des boucles autonomiques. Par exemple, certaines contributions sont basées sur une approche déclarative avec le solveur de contraintes Choco [PFL17] (cf. Section 7.3) alors que d'autres sont basées sur la théorie du contrôle avec des boucles de rétroaction (cf. Section 7.5.3). De récentes discussions fructueuses avec Eric Rutten [9] [BRPG16] nous ont motivés à explorer la complémentarité des deux approches. Par exemple, les boucles basées sur les contraintes raisonnent sur les relations entre les variables à un instant donné et ne prennent pas en compte les valeurs de retour (contrairement aux *feedback loops*), l'historique des valeurs ou des décisions passées, la dynamique du système, la décision prédictive ou l'apprentissage.

L'objectif est de travailler sur la définition de boucles autonomiques plus générales, en combinant plusieurs approches de modélisation et de décision, dans un cadre de coordination de plusieurs boucles. Notre motivation provient du fait que les systèmes Fog complexes présentent une variété

8. `http://seduce.menaud.fr`
9. chercheur INRIA et responsable de l'équipe Ctrl-A à INRIA Grenoble Rhône-Alpes.

de problèmes de régulation de différentes natures, à résoudre ensemble, et pour lesquels différentes techniques de formalisation et de résolution sont nécessaires (par exemple, comment aborder les oscillations du système, la défaillance de nœud en périphérie).

Nous considérerons la combinaison de contrôleurs impliquant des approches basées sur les contraintes ou sur des boucles de rétroaction. Par exemple, la résolution de contraintes peut être appliquée sur des variables, avec certaines valeurs calculées en retour par une boucle de rétroaction. A l'instar de nos travaux [ASL12], nous définirons des *patterns* de coordination assurant l'absence d'interférences entre les boucles. Le cas d'utilisation sera un scénario de type *"Crisis management system"* bien adapté au contexte du Fog. L'application sera basée sur le paradigme des micro-services.

Bibliographie

[AA15] F. Alzhouri and A. Agarwal. Dynamic pricing scheme : Towards cloud revenue maximization. In *2015 IEEE 7th International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 168–173, Nov 2015.

[AAG93] Gregory Abowd, Robert Allen, and David Garlan. Using style to understand descriptions of software architecture. In *Proceedings of the 1st ACM SIGSOFT Symposium on Foundations of Software Engineering*, SIGSOFT '93, pages 9–20, New York, NY, USA, 1993. ACM.

[AB10] Jean Arnaud and Sara Bouchenak. Adaptive internet services through performance and availability control. In *Proceedings of the 2010 ACM Symposium on Applied Computing*, SAC '10, pages 444–451, New York, NY, USA, 2010. ACM.

[ABLS13] Vasilios Andrikopoulos, Tobias Binz, Frank Leymann, and Steve Strauch. How to adapt applications for the cloud environment. *Computing*, 95(6) :493–535, Jun 2013.

[ACD+05] Alain Andrieux, Karl Czajkowski, Asit Dan, Kate Keahey, Heiko Ludwig, Toshiyuki Nakata, Jim Pruyne, John Rofrano, Steve Tuecke, and Ming Xu. Web Services Agreement Specification (WS-Agreement). Technical report, Global Grid Forum, Grid Resource Allocation Agreement Protocol (GRAAP) WG, September 2005.

[ACN02] Jonathan Aldrich, Craig Chambers, and David Notkin. Archjava : Connecting software architecture to implementation. In *Proceedings of the 24th International Conference on Software Engineering*, ICSE '02, pages 187–197, New York, NY, USA, 2002. ACM.

[ADB+99] Gregory D. Abowd, Anind K. Dey, Peter J. Brown, Nigel Davies, Mark Smith, and Pete Steggles. Towards a better understanding of context and context-awareness. In *Proceedings of the 1st International Symposium on Handheld and Ubiquitous Computing*, HUC '99, pages 304–307, London, UK, UK, 1999. Springer-Verlag.

[ADC10] M. Alhamad, T. Dillon, and E. Chang. Conceptual sla framework for cloud computing. In *4th IEEE International Conference on Digital Ecosystems and Technologies*, pages 606–610, April 2010.

[ADE18] ADEME. « perfecto » amélioration de la performance environementale des produits et éco-conception logicielle, 2018. Appel à projets de recherche et développement.

[ADG98] Robert Allen, Rémi Douence, and David Garlan. Specifying and analyzing dynamic software architectures. In Egidio Astesiano, editor, *Fundamental Approaches to Software Engineering*, pages 21–37, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg.

[ADOJ13] Guilherme Alvares De Oliveira Junior, Frederico. *Multi Autonomic Management for Optimizing Energy Consumption in Cloud Infrastructures.* PhD thesis, Université de Nantes, April 2013.

[AdOLLM12] Frederico Alvares de Oliveira, Adrien Lèbre, Thomas Ledoux, and Jean-Marc Menaud. *Achieving Federated and Self-Manageable Cloud Infrastructures : Theory and Practice*, chapter Self-Management of Applications and Systems to Optimize Energy in Data Centers. IGI Global, 2012.

[ADPDM18] Y. Al-Dhuraibi, F. Paraiso, N. Djarallah, and P. Merle. Elasticity in cloud computing : State of the art and research challenges. *IEEE Transactions on Services Computing*, 11(2) :430–447, March 2018.

[AGP98] Maha Abdallah, Rachid Guerraoui, and Philippe Pucheral. One-phase commit : Does it make sense ? In *ICPADS '98 : Proceedings of the 1998 International Conference on Parallel and Distributed Systems*, page 182, Washington, DC, USA, 1998. IEEE Computer Society.

[ALRL04] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr. Basic concepts and taxonomy of dependable and secure computing. *Dependable and Secure Computing, IEEE Transactions on*, 1(1) :11–33, Jan 2004.

[ANC96] Corporate Act-Net Consortium. The active database management system manifesto : A rulebase of adbms features. *SIGMOD Rec.*, 25(3) :40–49, September 1996.

[APR13] Simone Ahuja, Jaideep Prabhu, and Navi Radjou. *L'Innovation Jugaad.* Diateino, 2013.

[ARS15] P. Arcaini, E. Riccobene, and P. Scandurra. Modeling and analyzing mape-k feedback loops for self-adaptation. In *2015 IEEE/ACM 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, pages 13–23, May 2015.

[ASAB+18] Zakarea Al-Shara, Frederico Alvares, Hugo Bruneliere, Jonathan Lejeune, Charles Prud'Homme, and Thomas Ledoux. Come4acloud : An end-to-end framework for autonomic cloud systems. *Future Generation Computer Systems*, 86 :339 – 354, 2018.

[ASK14] C. Atkinson, T. Schulze, and S. Klingert. Facilitating greener it through green specifications. *IEEE Software*, 31(3) :56–63, May 2014.

[ASL12] Frederico Alvares De Oliveira Jr, Remi Sharrock, and Thomas Ledoux. Synchronization of Multiple Autonomic Control Loops : Application to Cloud Computing. In Marjan Sirjani, editor, *Coordination 2012*, volume 7274 of *Lecture Notes in Computer Science*, pages 29–43, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[ASL13] Frederico Alvares De Oliveira Jr, Remi Sharrock, and Thomas Ledoux. A framework for the coordination of multiple autonomic managers in cloud environments. In *2013 IEEE 7th International Conference on Self-Adaptive and Self-Organizing Systems*, pages 179–188, Sept 2013.

[ASPG16] Rim Abid, Gwen Salaün, Noel Palma, and Soguy Mak-Kare Gueye. Asynchronous coordination of stateful autonomic managers in the cloud. In *Revised Selected Papers of the 12th International Conference on Formal Aspects of Component Software - Volume 9539*, FACS 2015, pages 48–65, Berlin, Heidelberg, 2016. Springer-Verlag.

[BAB12] Anton Beloglazov, Jemal Abawajy, and Rajkumar Buyya. Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. *Future Gener. Comput. Syst.*, 28(5) :755–768, May 2012.

[Bal10] Cyril Ballagny. *MOCAS : a state-based component model for self-adaptation.* Theses, Université de Pau et des Pays de l'Adour, March 2010.

[Bas07] Gautier Bastide. *Scorpio : an Approach for Software Component Structural Adaptation - Usage for adaptation in Ubiquitous Environment.* Theses, Université de Nantes ; Ecole Centrale de Nantes, December 2007.

[BASA+18] Hugo Bruneliere, Zakarea Al-Shara, Frederico Alvares, Jonathan Lejeune, and Thomas Ledoux. A model-based architecture for autonomic and heterogeneous cloud systems. In *Proceedings of the 8th International Conference on Cloud Computing and Services Science - Volume 1 : CLOSER,*, pages 201–212. INSTICC, SciTePress, 2018.

[BBF07] Nelly Bencomo, Gordon Blair, and Robert France. *Summary of the Workshop Models@run.time at MoDELS 2006*, pages 227–231. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[BBF09] G. Blair, N. Bencomo, and R. B. France. Models@ run.time. *Computer*, 42(10) :22–27, Oct 2009.

[BBH+05] S. Bouchenak, F. Boyer, D. Hagimont, S. Krakowiak, A. Mos, N. de Palma, V. Quema, and J. B. Stefani. Architecture-based autonomous repair management : an application to j2ee clusters. In *24th IEEE Symposium on Reliable Distributed Systems (SRDS'05)*, pages 13–24, Oct 2005.

[BC89] Jean-Pierre Briot and Pierre Cointe. Programming with explicit metaclasses in Smalltalk-80. In *Conference on Object-Oriented Programming Systems, Languages and Applications (OOPSLA'89). Sigplan Notices*, 24(10) :419–432, October 1989.

[BCD+09] Françoise Baude, Denis Caromel, Cédric Dalmasso, Marco Danelutto, Vladimir Getov, Ludovic Henrio, and Christian Pérez. Gcm : a grid extension to fractal for autonomous distributed components. *annals of telecommunications - annales des télécommunications*, 64(1) :5–24, 2009.

[BCF+97] Jérôme Besancenot, Michèle Cart, Jean Ferrié, Rachid Guerraoui, Philippe Pucheral, and Bruno Traverson. *Les systèmes transactionnels : concepts, normes et produits.* Hermes collection Informatique, 1997.

[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 Stafford, Heinz Schmidt, and Kurt Wallnau, editors, *Component-Based Software Engineering*, volume 3054, pages 7–22. Springer Berlin / Heidelberg, 2004.

[BCL+06] Eric Bruneton, Thierry Coupaye, Matthieu Leclercq, Vivien Quéma, and Jean-Bernard Stefani. The fractal component model and its support in java. *Software : Practice and Experience*, 36(11-12) :1257–1284, 2006.

[BCRP98] G. S. Blair, G. Coulson, P. Robin, and M. Papathomas. An architecture for next generation middleware. In *Proceedings of the IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing*, Middleware '98, pages 191–206, London, UK, UK, 1998. Springer-Verlag.

[BD03] Louise Barkhuus and Anind Dey. Is context-aware computing taking control away from the user ? three levels of interactivity examined. In Anind K. Dey, Albrecht Schmidt, and Joseph F. McCarthy, editors, *UbiComp 2003 : Ubiquitous Computing*, pages 149–156, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[BDF+03] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the art of virtualization. *SIGOPS Oper. Syst. Rev.*, 37(5) :164–177, October 2003.

[BDPG+09] Fabienne Boyer, Noel De Palma, Olivier Gruber, Sylvain Sicard, and Jean-Bernard Stefani. *Architecting Dependable Systems VI*, chapter A Self-repair Architecture for Cluster Systems, pages 124–147. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[Béz06] Jean Bézivin. *Generative and Transformational Techniques in Software Engineering : International Summer School, GTTSE 2005, Braga, Portugal, July 4-8, 2005. Revised Papers*, chapter Model Driven Engineering : An Emerging Technical Space, pages 36–64. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.

[BGLQ16] Luciano Baresi, Sam Guinea, Alberto Leva, and Giovanni Quattrocchi. A discrete-time feedback controller for containerized cloud applications. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE 2016, pages 217–228, New York, NY, USA, 2016. ACM.

[BGW93] Daniel G. Bobrow, Richard P. Gabriel, and Jon L. White. Object-oriented programming. chapter CLOS in Context : The Shape of the Design Space, pages 29–61. MIT Press, Cambridge, MA, USA, 1993.

[BHG87] Philip A. Bernstein, Vassos Hadzilacos, and Nathan Goodman. *Concurrency Control and Recovery in Database Systems.* Addison-Wesley, 1987.

[BHN09] F. Baude, L. Henrio, and P. Naoumenko. Structural reconfiguration : An autonomic strategy for gcm components. In *2009 Fifth International Conference on Autonomic and Autonomous Systems*, pages 123–128, April 2009.

[BHS17] Michael Borkowski, Christoph Hochreiner, and Stefan Schulte. Moderated resource elasticity for stream processing applications. In *International Workshop on Autonomic Solutions for Parallel and Distributed Data Stream Processing (Auto-DaSP 2017)*, 2017.

[BJC05] Thais Batista, Ackbar Joolia, and Geoff Coulson. Managing dynamic reconfiguration in component-based systems. In *Proceedings of the 2Nd European Conference on Software Architecture*, EWSA'05, pages 1–17, Berlin, Heidelberg, 2005. Springer-Verlag.

[BMSG+09] Yuriy Brun, Giovanna Marzo Serugendo, Cristina Gacek, Holger Giese, Holger Kienle, Marin Litoiu, Hausi Müller, Mauro Pezzè, and Mary Shaw. Software engineering for self-adaptive systems. chapter Engineering Self-Adaptive Systems Through Feedback Loops, pages 48–70. Springer-Verlag, Berlin, Heidelberg, 2009.

[BMZ+05] Jim Buckley, Tom Mens, Matthias Zenger, Awais Rashid, and Günter Kniesel. Towards a taxonomy of software change : Research articles. *J. Softw. Maint. Evol.*, 17(5) :309–332, September 2005.

[Bre12] Paul C Brebner. Is your cloud elastic enough ? : performance modelling the elasticity of infrastructure as a service (iaas) cloud applications. In *Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering*, pages 263–266. ACM, 2012.

[Bri89] Jean-Pierre Briot. Actalk : A testbed for classifying and designing actor languages in the smalltalk-80 environment. In *ECOOP '89 : Proceedings of the Third European Conference on Object-Oriented Programming, Nottingham, UK, July 10-14, 1989.*, pages 109–129, 1989.

[BRL07] Fabien Baligand, Nicolas Rivierre, and Thomas Ledoux. *Service-Oriented Computing – ICSOC 2007 : Fifth International Conference, Vienna, Austria, September 17-20, 2007. Proceedings*, chapter A Declarative Approach for QoS-Aware Web Service Compositions, pages 422–428. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[BRL08] Fabien Baligand, Nicolas Rivierre, and Thomas Ledoux. QoS Policies for Business Processes in Service Oriented Architectures. In *Service-Oriented Computing - ICSOC 2008*, Lecture Notes in Computer Science, pages 483–497. Springer-Verlag, 2008.

[BRPG16] N. Berthier, É. Rutten, N. De Palma, and S. M. K. Gueye. Designing autonomic management systems by using reactive control techniques. *IEEE Transactions on Software Engineering*, 42(7) :640–657, July 2016.

[BS97] Gordon Blair and Jean-Bernard Stefani. *Open distributed processing and multimedia.* Addison-Wesley, 1997.

[BSC+17] Rajkumar Buyya, Satish Narayana Srirama, Giuliano Casale, Rodrigo N. Calheiros, Yogesh L. Simmhan, Blesson Varghese, Erol Gelenbe, Bahman Javadi, Luis Miguel Vaquero Gonzalez, Marco Aurélio Stelmar Netto, Adel Nadjaran Toosi, Maria Alejandra Rodriguez, Ignacio Martín Llorente, Sabrina De Capitani di Vimercati, Pierangela Samarati, Dejan S. Milojicic, Carlos A. Varela, Rami Bahsoon, Marcos Dias de Assunção, Omer F. Rana, Wanlei Zhou, Hai Jin, Wolfgang Gentzsch, Albert F. Zomaya, and Haiying Shen. A manifesto for future generation cloud computing : Research directions for the next decade. *CoRR*, abs/1711.09123, 2017.

[BSL04] Noury M. Bouraqadi-Saadani and Thomas Ledoux. Supporting AOP Using Reflection. In Mehmet Aksit, Siobhán Clarke, Tzilla Elrad, and Robert E. Filman, editors, *Aspect-Oriented Software Development*, pages 261–282. Addison-Wesley, 2004.

[BSLR98] Noury M. N. Bouraqadi-Saâdani, Thomas Ledoux, and Fred Rivard. Safe metaclass programming. In *Proceedings of the 13th ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications*, OOPSLA '98, pages 84–96, New York, NY, USA, 1998. ACM.

[Buy09] R. Buyya. Market-oriented cloud computing : Vision, hype, and reality of delivering computing as the 5th utility. In *2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, pages 1–1, May 2009.

[CA08] Sharma Chakravarthy and Raman Adaikkalavan. Events and streams : Harnessing and unleashing their synergy ! In *Proceedings of the Second International Conference on Distributed Event-based Systems*, DEBS '08, pages 1–12, New York, NY, USA, 2008. ACM.

[CBG+08] Geoff Coulson, Gordon Blair, Paul Grace, Francois Taiani, Ackbar Joolia, Kevin Lee, Jo Ueyama, and Thirunavukkarasu Sivaharan. A generic component model for building systems software. *ACM Trans. Comput. Syst.*, 26(1) :1 :1–1 :42, March 2008.

[CdCSR+15] Emanuel Ferreira Coutinho, Flávio Rubens de Carvalho Sousa, Paulo Antonio Leal Rego, Danielo Gonçalves Gomes, and José Neuman de Souza. Elasticity in cloud computing : a survey. *annals of telecommunications - annales des télécommunications*, 70(7) :289–309, Aug 2015.

[CdLG+09] Betty H. C. Cheng, Rogério de Lemos, Holger Giese, Paola Inverardi, Jeff Magee, Jesper Andersson, Basil Becker, Nelly Bencomo, Yuriy Brun, Bojan Cukic, Giovanna Di Marzo Serugendo, Schahram Dustdar, Anthony Finkelstein, Cristina Gacek, Kurt Geihs, Vincenzo Grassi, Gabor Karsai, Holger M. Kienle, Jeff Kramer, Marin Litoiu, Sam Malek, Raffaela Mirandola, Hausi A. Müller, Sooyong Park, Mary Shaw, Matthias Tichy, Massimo Tivoli, Danny Weyns, and Jon Whittle. *Software Engineering for Self-Adaptive Systems : A Research Roadmap*, pages 1–26. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[CF16] Clément Chauvin and Erwann Fangeat. Allongement de la durée de vie des produits. Technical report, ADEME, 2016.

[CFF14] Antonio Corradi, Mario Fanelli, and Luca Foschini. Vm consolidation : A real case based on openstack cloud. *Future Generation Computer Systems*, 32 :118 – 127, 2014. Special Section : The Management of Cloud Systems, Special Section : Cyber-Physical Society and Special Section : Special Issue on Exploiting Semantic Technologies with Particularization on Linked Data over Grid and Cloud Architectures.

[CHP+16] C. Cappiello, N. T. T. Ho, B. Pernici, P. Plebani, and M. Vitali. Co2-aware adaptation strategies for cloud applications. *IEEE Transactions on Cloud Computing*, 4(2) :152–165, April 2016.

[CM98] Charles Consel and Renaud Marlet. Architecturing software using a methodology for language development. In *Proceedings of the 10 th International Symposium on Programming Language Implementation and Logic Programming, number 1490 in Lecture Notes in Computer Science*, pages 170–194, 1998.

[CM12] Gianpaolo Cugola and Alessandro Margara. Processing flows of information : From data stream to complex event processing. *ACM Comput. Surv.*, 44(3) :15 :1–15 :62, June 2012.

[CMP06] Carlos Canal, Juan Manuel Murillo, and Pascal Poizat. Software Adaptation. *L'objet*, 12(1) :9–31, 2006.

[CNRa] Définition de l'adjectif adaptable. `http://www.cnrtl.fr/definition/adaptable`. Accessed : 2016-02-22.

[CNRb] Définition de l'adjectif adaptatif. `http://www.cnrtl.fr/definition/Adaptatif`. Accessed : 2016-02-22.

[CNRc] Définition du nom adaptation. `http://www.cnrtl.fr/definition/adaptation`. Accessed : 2016-02-22.

[CPT99] Calos Canal, Ernesto Pimentel, and José M. Troya. *Software Architecture : TC2 First Working IFIP Conference on Software Architecture (WICSA1) 22–24 February 1999, San Antonio, Texas, USA*, chapter Specification and Refinement of Dynamic Software Architectures, pages 107–125. Springer US, Boston, MA, 1999.

[CRCR05] Philippe Collet, Roger Rousseau, Thierry Coupaye, and Nicolas Rivierre. *Component-Based Software Engineering : 8th International Symposium, CBSE 2005, St. Louis, MO, USA, May 14-15, 2005. Proceedings*, chapter A Contracting System for Hierarchical Components, pages 187–202. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.

[Dav05] Pierre-Charles David. *Développement de composants Fractal adaptatifs : un langage dédié à l'aspect d'adaptation.* PhD thesis, Université de Nantes, July 2005.

[DBAL17] Simon Dupont, Salma Bouri, Frederico Alvares, and Thomas Ledoux. Elascript : A dsl for coding elasticity in cloud computing. In *Proceedings of the Symposium on Applied Computing*, SAC '17, pages 392–398, New York, NY, USA, 2017. ACM.

[DC01] Jim Dowling and Vinny Cahill. The k-component architecture meta-model for self-adaptive software. In *Proceedings of the Third International Conference on Metalevel Architectures and Separation of Crosscutting Concerns*, REFLECTION '01, pages 81–88, London, UK, UK, 2001. Springer-Verlag.

[DCCC06] Jim Dowling, Raymond Cunningham, Eoin Curran, and Vinny Cahill. Building autonomic systems using collaborative reinforcement learning. *Knowl. Eng. Rev.*, 21(3) :231–238, September 2006.

[DFS02] Rémi Douence, Pascal Fradet, and Mario Südholt. *A Framework for the Detection and Resolution of Aspect Interactions*, pages 173–188. Springer Berlin Heidelberg, Berlin, Heidelberg, 2002.

[Dij76] Edsger Wybe Dijkstra. *A Discipline of Programming.* Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edition, 1976.

[DK75] Frank DeRemer and Hans Kron. Programming-in-the large versus programming-in-the-small. In *Proceedings of the international conference on Reliable software*, pages 114–121, New York, NY, USA, 1975. ACM.

[DKL12] Julien Dormoy, Olga Kouchnarenko, and Arnaud Lanoix. Using Temporal Logic for Dynamic Reconfigurations of Components. In *FACS 2010 - 7th International Symposium on Formal Aspects of Component Software*, pages 200–217. Springer Verlag, 2012.

[DL03] Pierre-Charles David and Thomas Ledoux. Towards a framework for self-adaptive component-based applications. In *Distributed Applications and Interoperable Systems (DAIS'03)*, pages 1–14. Springer, 2003.

[DL05] Pierre-Charles David and Thomas Ledoux. WildCAT : a generic framework for context-aware applications. In *3rd international workshop on Middleware for pervasive and ad-hoc computing - MPAC '05*, number 3, pages 1–7, New York, New York, USA, 2005. ACM Press.

[DL06] Pierre-Charles David and Thomas Ledoux. An aspect-oriented approach for developing self-adaptive fractal components. In Welf Löwe and Mario Südholt, editors, *5th international conference on Software Composition*, volume 4089 of *Lecture Notes in Computer Science*, pages 82–97, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[DLAL15] S. Dupont, J. Lejeune, F. Alvares, and T. Ledoux. Experimental analysis on autonomic strategies for cloud elasticity. In *Cloud and Autonomic Computing (ICCAC), 2015 International Conference on*, pages 81–92, Sept 2015.

[DLBs01] Pierre-Charles David, Thomas Ledoux, and Noury M. N. Bouraqadi-saâdani. Two-step weaving with reflection using aspectj. In *in OOPSLA 2001 Workshop on Advanced Separation of Concerns in Object-Oriented Systems*, pages 14–18, 2001.

[DLG+08] Pierre-Charles David, Marc Léger, Hervé Grall, Thomas Ledoux, and Thierry Coupaye. A Multi-stage Approach for Reliable Dynamic Reconfigurations of Component-Based Systems. In René Meier and Sotirios Terzis, editors, *Distributed Applications and Interoperable Systems (DAIS'08)*, volume 5053 of *Lecture Notes in Computer Science*, pages 106–111, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[DLJ+13] Wei Deng, Fangming Liu, Hai Jin, Chuan Wu, and Xue Liu. Multigreen : Cost-minimizing multi-source datacenter power supply with online control. In *Proceedings of the Fourth International Conference on Future Energy Systems*, e-Energy '13, pages 149–160, New York, NY, USA, 2013. ACM.

[DLLC09] Pierre-Charles David, Thomas Ledoux, Marc Léger, and Thierry Coupaye. FPath and FScript : Language support for navigation and reliable reconfiguration of Fractal architectures. *Annales des Telecommunications/Annals of Telecommunications*, 64(1-2) :45–63, 2009.

[dOL12] Frederico Alvares de Oliveira, Jr. and Thomas Ledoux. Self-management of cloud applications and infrastructure for energy optimization. *SIGOPS Oper. Syst. Rev.*, 46(2) :10–18, July 2012.

[Don90] Christophe Dony. Exception handling and object-oriented programming : Towards a synthesis. In *Proceedings of the European Conference on Object-oriented Programming on Object-oriented Programming Systems, Languages, and Applications*, OOPSLA/ECOOP '90, pages 322–330, New York, NY, USA, 1990. ACM.

[Dup16] Simon Dupont. *Crosslayer elasticity management for Cloud : towards an efficient usage of Cloud resources and services.* Theses, Ecole des Mines de Nantes, April 2016.

[EFB01] Tzilla Elrad, Robert E. Filman, and Atef Bader. Aspect-oriented programming : Introduction. *Commun. ACM*, 44(10) :29–32, October 2001.

[EFGK03] Patrick Th. Eugster, Pascal A. Felber, Rachid Guerraoui, and Anne-Marie Kermarrec. The many faces of publish/subscribe. *ACM Comput. Surv.*, 35(2) :114–131, June 2003.

[EM13] M. C. Eddin and Z. Mammeri. Preserving the global consistency of dynamic reconfiguration. In *2013 14th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing*, pages 71–76, July 2013.

[Esp] Esper. `http://www.espertech.com/esper/`. Accessed : 2016-02-22.

[EVHB05] Peter Ebraert, Yves Vandewoude, Theo D Hondt, and Yolande Berbers. Pitfalls in unanticipated dynamic software evolution. In *RAM-SE'05 – ECOOP'05 Workshop on Reflection, AOP, and Meta-Data for Software Evolution*, pages 41–49, 2005.

[FD99] Ira R. Forman and Scott H. Danforth. *Putting Metaclasses to Work : A New Dimension in Object-oriented Programming.* Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 1999.

[Fer11] Nicolas Ferry. *Dynamic adaptation to context in ambient intelligence : logical and temporal properties.* Theses, Université Nice Sophia Antioplis, December 2011.

[FHS+06] J. Floch, S. Hallsteinsen, E. Stav, F. Eliassen, K. Lund, and E. Gjorven. Using architecture models for runtime adaptability. *IEEE Software*, 23(2) :62–70, March 2006.

[FJ89] B. Foote and R. E. Johnson. Reflective facilities in smalltalk-80. In *Conference Proceedings on Object-oriented Programming Systems, Languages and Applications*, OOPSLA '89, pages 327–335, New York, NY, USA, 1989. ACM.

[FJBE15] S. Farokhi, P. Jamshidi, I. Brandic, and E. Elmroth. Self-adaptation challenges for cloud-based applications : A control theoretic perspective. In *10th International Workshop on Feedback Computing*, 2015.

[FJLB15] S. Farokhi, P. Jamshidi, D. Lucanin, and I. Brandic. Performance-based vertical memory elasticity. In *2015 IEEE International Conference on Autonomic Computing*, pages 151–152, July 2015.

[FLK+15] S. Farokhi, E. B. Lakew, C. Klein, I. Brandic, and E. Elmroth. Coordinating cpu and memory elasticity controllers to meet service response time constraints. In *2015 International Conference on Cloud and Autonomic Computing*, pages 69–80, Sept 2015.

[FME12] E. Feller, C. Morin, and A. Esnault. A case for fully decentralized dynamic vm consolidation in clouds. In *4th IEEE International Conference on Cloud Computing Technology and Science Proceedings*, pages 26–33, Dec 2012.

[Fou13] François Fouquet. *Model@Runtime for continuous development of heterogeneous distributed adaptive systems.* Theses, Université Rennes 1, March 2013.

[FP98] Jean-Charles Fabre and Tanguy Pérennou. A metaobject architecture for fault-tolerant distributed systems : The friends approach. *IEEE Trans. Comput.*, 47(1) :78–95, January 1998.

[fSI11] International Organization for Standardization (ISO). Iso/iec 25010 :2011, 2011. Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models.

[GC15] Erol Gelenbe and Yves Caseau. The impact of information technology on energy consumption and carbon emissions. *Ubiquity*, 2015(June) :1 :1–1 :15, June 2015.

[GCH+04] D. Garlan, S.-W. Cheng, A.-C. Huang, B. Schmerl, and P. Steenkiste. Rainbow : architecture-based self-adaptation with reusable infrastructure. *Computer*, 37(10) :46–54, October 2004.

[GHJV95] Erich Gamma, Richard Helm, Ralph E. Johnson, and John Vlissides. *Design Patterns : Elements of Reusable Object-Oriented Software.* Addison-Wesley, Reading, MA, 1995.

[GKL+13] Íñigo Goiri, William Katsak, Kien Le, Thu D. Nguyen, and Ricardo Bianchini. Parasol and greenswitch : Managing datacenters powered by renewable energy. In *Proceedings of the Eighteenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '13, pages 51–64, New York, NY, USA, 2013. ACM.

[GMW00] David Garlan, Robert T. Monroe, and David Wile. Foundations of component-based systems. chapter Acme : Architectural Description of Component-based Systems, pages 47–67. Cambridge University Press, New York, NY, USA, 2000.

[Gou99] K.M. Goudarzi. *Consistency Preserving Dynamic Reconfiguration of Distributed Systems.* PhD thesis, University of London, 1999.

[GR83] Adele Goldberg and David Robson. *Smalltalk-80 : The Language and Its Implementation.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1983.

[GR92] Jim Gray and Andreas Reuter. *Transaction Processing : Concepts and Techniques.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1992.

[GS94] David Garlan and Mary Shaw. An introduction to software architecture. Technical report, Pittsburgh, PA, USA, 1994.

[GSC01] David Garlan, Bradley Schmerl, and Jichuan Chang. Using gauges for architecture-based monitoring and adaptation. *Working Conference on Complex and Dynamic Systems Architecture*, (December), 2001.

[Guo03] H. Guo. A bayesian approach for autonomic algorithm selection. In *IJCAI Workshop on AI and autonomic computing : developing a research agenda for self-managing computer systems*, 2003.

[HAL17] MD Sabbir Hasan, Frederico Alvares, and Thomas Ledoux. Gpaascaler : Green energy aware platform scaler for interactive cloud application. In *Proceedings of the10th International Conference on Utility and Cloud Computing*, UCC '17, pages 79–89, New York, NY, USA, 2017. ACM.

[HALP16] M. S. Hasan, F. Alvares de Oliveira, T. Ledoux, and J. L. Pazat. Enabling green energy awareness in interactive cloud application. In *2016 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 414–422, Dec 2016.

[HALP17] M. S. Hasan, F. Alvares, T. Ledoux, and J. L. Pazat. Investigating energy consumption and performance trade-off for interactive cloud application. *IEEE Transactions on Sustainable Computing*, 2(2) :113–126, April 2017.

[Has17] MD Sabbir Hasan. *Smart management of renewable energy in clouds : from infrastructure to application.* Theses, INSA de Rennes, May 2017.

[HGB10] Regina Hebig, Holger Giese, and Basil Becker. Making control loops explicit when architecting self-adaptive systems. In *Proceedings of the Second International Workshop on Self-organizing Architectures*, SOAR '10, pages 21–28, New York, NY, USA, 2010. ACM.

[HKLP14] Md Sabbir Hasan, Yousri Kouki, Thomas Ledoux, and Jean-Louis Pazat. Cloud Energy Broker : Towards SLA-driven Green Energy Planning for IaaS Providers. In *IEEE Internatonal Conference on High Performance Computing and Communications, HPCC 2014*, pages 1–8, France, August 2014.

[HKLP17] M. S. Hasan, Y. Kouki, T. Ledoux, and J. L. Pazat. Exploiting renewable sources : When green sla becomes a possible reality in cloud computing. *IEEE Transactions on Cloud Computing*, 5(2) :249–262, April 2017.

[HKR13] Nikolas Roman Herbst, Samuel Kounev, and Ralf Reussner. Elasticity in cloud computing : What it is, and what it is not. In *Proceedings of the 10th International Conference on Autonomic Computing (ICAC 13)*, pages 23–27, San Jose, CA, 2013. USENIX.

[HL95] Walter L. Hürsch and Cristina Videira Lopes. Separation of concerns. Technical report, 1995.

[HLM+09] Fabien Hermenier, Xavier Lorca, Jean-Marc Menaud, Gilles Muller, and Julia Lawall. Entropy : A consolidation manager for clusters. In *Proceedings of the 2009 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, VEE '09, pages 41–50, New York, NY, USA, 2009. ACM.

[HLM13] F. Hermenier, J. Lawall, and G. Muller. Btrplace : A flexible consolidation manager for highly available applications. *IEEE Transactions on Dependable and Secure Computing*, 10(5) :273–286, Sept 2013.

[HM08] Markus C. Huebscher and Julie A. McCann. A survey of autonomic computing – degrees, models, and applications. *ACM Comput. Surv.*, 40(3) :7 :1–7 :28, August 2008.

[Hoa69] C. A. R. Hoare. An axiomatic basis for computer programming. *Commun. ACM*, 12(10) :576–580, October 1969.

[Hof93] Christine Ruth Hofmeister. *Dynamic Reconfiguration of Distributed Applications.* PhD thesis, College Park, MD, USA, 1993. UMI Order No. GAX94-07643.

[HSB09] Annika Hinze, Kai Sachs, and Alejandro Buchmann. Event-based applications and enabling technologies. In *Proceedings of the Third ACM International Conference on Distributed Event-Based Systems*, DEBS '09, pages 1 :1–1 :15, New York, NY, USA, 2009. ACM.

[IGC04] D. E. Irwin, L. E. Grit, and J. S. Chase. Balancing risk and reward in a market-based task service. In *Proceedings. 13th IEEE International Symposium on High performance Distributed Computing, 2004.*, pages 160–169, June 2004.

[INR] Préparation du plan stratégique inria 2018-2022. `https://intranet.inria.fr/Inria/Instances/Instances-nationales/Cellule-Veille-Prospective-CVP`. Accessed : 2018-01-10.

[IRHBJ16] M. Ibáñez, C. Ruz, L. Henrio, and J. Bustos-Jiménez. Reconfigurable applications using gcmscript. *IEEE Cloud Computing*, 3(3) :30–39, May 2016.

[JABK08] Frédéric Jouault, Freddy Allilaire, Jean Bézivin, and Ivan Kurtev. Atl : A model transformation tool. *Science of Computer Programming*, 72(1) :31 – 39, 2008. Special Issue on Second issue of experimental software and toolkits (EST).

[Jac02] Daniel Jackson. Alloy : A lightweight object modelling notation. *ACM Trans. Softw. Eng. Methodol.*, 11(2) :256–290, April 2002.

[JAP14] Pooyan Jamshidi, Aakash Ahmad, and Claus Pahl. Autonomic resource provisioning for cloud-based software. In *Proceedings of the 9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, SEAMS 2014, pages 95–104, New York, NY, USA, 2014. ACM.

[JGS93] Neil D. Jones, Carsten K. Gomard, and Peter Sestoft. *Partial Evaluation and Automatic Program Generation.* Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1993.

[JWW+15] H. Jin, X. Wang, S. Wu, S. Di, and X. Shi. Towards optimized fine-grained pricing of iaas cloud platform. *IEEE Transactions on Cloud Computing*, 3(4) :436–448, Oct 2015.

[KADL14] Yousri Kouki, Frederico Alvares de Oliveira, Simon Dupont, and Thomas Ledoux. A Language Support for Cloud Elasticity Management. In *14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pages 206–215. IEEE, May 2014.

[KC03] J.O. Kephart and D.M. Chess. The vision of autonomic computing. *Computer*, 36(1) :41–50, 2003.

[KDNH15] Eva Kern, Markus Dick, Stefan Naumann, and Tim Hiller. Impacts of software and its engineering on the carbon footprint of ict. *Environmental Impact Assessment Review*, 52 :53 – 61, 2015. Information technology and renewable energy - Modelling, simulation, decision support and environmental assessment.

[Kee04] John Keeney. *Completely Unanticipated Dynamic Adaptation of Software.* PhD thesis, University of Dublin, Trinity College, 2004.

[KH98] Ralph Keller and Urs Hölzle. Binary component adaptation. In *Proceedings of the 12th European Conference on Object-Oriented Programming*, ECCOP '98, pages 307–329, London, UK, UK, 1998. Springer-Verlag.

[KHH+01] Gregor Kiczales, Erik Hilsdale, Jim Hugunin, Mik Kersten, Jeffrey Palm, and William G. Griswold. An overview of aspectj. In *Proceedings of the 15th European Conference on Object-Oriented Programming*, ECOOP '01, pages 327–353, London, UK, UK, 2001. Springer-Verlag.

[Kic94] Gregor Kiczales. Why are black boxes so hard to reuse ? In *Proceedings of the 9th Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA)*, volume 29, Portland, oct 1994. ACM. Invited talk.

[KL03] Alexander Keller and Heiko Ludwig. The wsla framework : Specifying and monitoring service level agreements for web services. *Journal of Network and Systems Management*, 11(1) :57–81, Mar 2003.

[KL12a] Y. Kouki and T. Ledoux. Sla-driven capacity planning for cloud applications. In *4th IEEE International Conference on Cloud Computing Technology and Science Proceedings*, pages 135–140, Dec 2012.

[KL12b] Yousri Kouki and Thomas Ledoux. Csla : A language for improving cloud sla management. In *Proceedings of the 2nd International Conference on Cloud Computing and Services Science - Volume 1 : CLOSER,*, pages 586–591. INSTICC, SciTePress, 2012.

[KL13] Yousri Kouki and Thomas Ledoux. Rightcapacity : Sla-driven cross-layer cloud elasticity management. *IJNGC*, 4(3), 2013.

[KLL+97] Gregor Kiczales, John Lamping, Christina Videira Lopes, Chris Maeda, Anurag Mendhekar, and Gail Murphy. Open implementation design guidelines. In *Proceedings of the 19th International Conference on Software Engineering*, ICSE '97, pages 481–490, New York, NY, USA, 1997. ACM.

[KLM+97] Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Lopes, Jean-Marc Loingtier, and John Irwin. Aspect-oriented programming. In Mehmet Akşit and Satoshi Matsuoka, editors, *ECOOP'97 — Object-Oriented Programming*, pages 220–242, Berlin, Heidelberg, 1997. Springer Berlin Heidelberg.

[KLS11] Yousri Kouki, Thomas Ledoux, and Remi Sharrock. Cross-layer sla selection for cloud services. In *Proceedings of the 2011 First International Symposium on Network Cloud Computing and Applications*, NCCA '11, pages 143–147, Washington, DC, USA, 2011. IEEE Computer Society.

[KM90] J. Kramer and J. Magee. The evolving philosophers problem : dynamic change management. *Software Engineering, IEEE Transactions on*, 16(11) :1293–1306, Nov 1990.

[KM98] J. Kramer and J. Magee. Analysing dynamic change in software architectures : a case study. In *Proceedings. Fourth International Conference on Configurable Distributed Systems (Cat. No.98EX159)*, pages 91–100, May 1998.

[KMrHR14] Cristian Klein, Martina Maggio, Karl-Erik Årzén, and Francisco Hernández-Rodriguez. Brownout : Building more robust cloud applications. In *Proceedings of the 36th International Conference on Software Engineering*, ICSE 2014, pages 700–711, New York, NY, USA, 2014. ACM.

[KNMB02] Günter Kniesel, Joost Noppen, Tom Mens, and Jim Buckley. Unanticipated software evolution. In Juan Hernández and Ana Moreira, editors, *Object-Oriented Technology ECOOP 2002 Workshop Reader*, volume 2548 of *Lecture Notes in Computer Science*, pages 92–106. Springer Berlin Heidelberg, 2002.

[Koe14] Martin Koehler. An adaptive framework for utility-based optimization of scientific applications in the cloud. *Journal of Cloud Computing*, 3(1) :4, 2014.

[Koo11] Jonathan Koomey. Growth in data center electricity use 2005 to 2010. *A report by Analytical Press, completed at the request of The New York Times*, page 9, 2011.

[Kou13] Yousri Kouki. *SLA-driven cloud elasticity anagement approach.* PhD thesis, Ecole des Mines de Nantes, December 2013.

[KR91] Gregor Kiczales and Jim Des Rivieres. *The Art of the Metaobject Protocol.* MIT Press, Cambridge, MA, USA, 1991.

[KR12] J. A. Konstan and J. Riedl. Recommended for you. *IEEE Spectrum*, 49(10) :54–61, October 2012.

[KRV+15] Christian Krupitzer, Felix Maximilian Roth, Sebastian VanSyckel, Gregor Schiele, and Christian Becker. A survey on engineering approaches for self-adaptive systems. *Pervasive Mob. Comput.*, 17(PB) :184–206, February 2015.

[KYTA12] A. Khan, X. Yan, S. Tao, and N. Anerousis. Workload characterization and prediction in the cloud : A multiple time series approach. In *2012 IEEE Network Operations and Management Symposium*, pages 1287–1294, April 2012.

[Lac08] Mayleen Lacouture. A generic version of fscript : A model-driven engineering approach. Master's thesis, Vrije Universiteit Brussel, aug 2008. In Collaboration with Ecole des Mines de Nantes.

[LAL17] Jonathan Lejeune, Frederico Alvares, and Thomas Ledoux. Towards a generic autonomic model to manage cloud services. In *Proceedings of the 7th International Conference on Cloud Computing and Services Science - Volume 1 : CLOSER,*, pages 175–186. INSTICC, SciTePress, 2017.

[Lar] Définition du nom auto-adaptation. `http://www.larousse.fr/dictionnaires/francais/autoadaptation/6574`. Accessed : 2016-02-22.

[LBC10] Harold C. Lim, Shivnath Babu, and Jeffrey S. Chase. Automated control for elastic storage. In *Proceedings of the 7th International Conference on Autonomic Computing*, ICAC '10, pages 1–10, New York, NY, USA, 2010. ACM.

[LBFB+01] Thomas Ledoux, Mireille Blay-Fornarino, Eric Bruneton, Denis Caromel, Thierry Coupaye, Daniel Hagimont, J-M Menaud, Jacques Noyé, and Michel Riveill. D1.1 - Etat de l'art sur l'adaptabilité. Technical report, Ecole des mines de Nantes, Nantes, France, December 2001.

[LBMAL14] Tania Lorido-Botran, Jose Miguel-Alonso, and Jose A. Lozano. A review of auto-scaling techniques for elastic applications in cloud environments. *Journal of Grid Computing*, 12(4) :559–592, 2014.

[LC96] Thomas Ledoux and Pierre Cointe. Explicit metaclasses as a tool for improving the design of class libraries. In *Proceedings of the Second JSSST International Symposium on Object Technologies for Advanced Software*, ISOTAS '96, pages 38–55, London, UK, UK, 1996. Springer-Verlag.

[Led98] Thomas Ledoux. *Réflexion dans les systèmes répartis : application à CORBA et Smalltalk*. PhD thesis, Université de Nantes, March 1998.

[Led99] Thomas Ledoux. OpenCorba : A Reflective Open Broker. In *Meta-Level Architectures and Reflection*, volume 1616, pages 197–214, Berlin, Heidelberg, 1999. Springer-Verlag.

[Lég09] Marc Léger. *Fiabilité des reconfigurations dynamiques dans les architectures á composants* . PhD thesis, École Nationale Supérieure des Mines de Paris, May 2009.

[Lit] Définition du verbe adapter. `http://littre.reverso.net/dictionnaire-francais/definition/adapter/708`. Accessed : 2016-02-22.

[LLC10] Marc Léger, Thomas Ledoux, and Thierry Coupaye. Reliable dynamic reconfigurations in a reflective component model. In Lars Grunske, Ralf Reussner, and Frantisek Plasil, editors, *13th international conference on Component-Based Software Engineering (CBSE'10)*, volume 6092 of *Lecture Notes in Computer Science*, pages 74–92, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[LOM15] Y. Li, A. C. Orgerie, and J. M. Menaud. Opportunistic scheduling in clouds partially powered by green energy. In *2015 IEEE International Conference on Data Science and Data Intensive Systems*, pages 448–455, Dec 2015.

[LOM17] Y. Li, A. C. Orgerie, and J. M. Menaud. Balancing the use of batteries and opportunistic scheduling policies for maximizing renewable energy consumption in a cloud data center. In *2017 25th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP)*, pages 408–415, March 2017.

[LQS05] M. Leclercq, V. Quema, and J. Stefani. Dream : A component framework for constructing resource-aware, configurable middleware. *IEEE Distributed Systems Online*, 6(9) :1–1, Sept 2005.

[LRFH04] M. L. Littman, N. Ravi, E. Fenson, and R. Howard. Reinforcement learning for autonomic network repair. In *International Conference on Autonomic Computing, 2004. Proceedings.*, pages 284–285, May 2004.

[MA04] R Timothy Marler and Jasbir S Arora. Survey of multi-objective optimization methods for engineering. *Structural and multidisciplinary optimization*, 26(6) :369–395, 2004.

[Mae87] Pattie Maes. Concepts and experiments in computational reflection. In *Conference Proceedings on Object-oriented Programming Systems, Languages and Applications*, OOPSLA '87, pages 147–155, New York, NY, USA, 1987. ACM.

[Mar07] Renaud Marlet. Spécialiser les programmes, spécialiser les langages, 2007. Habilitation à diriger les recherches (Université de Bordeaux 1).

[MB16] Redowan Mahmud and Rajkumar Buyya. Fog computing : A taxonomy, survey and future directions. *CoRR*, abs/1611.05539, 2016.

[MBC04] Rui S. Moreira, Gordon S. Blair, and Eurico Carrapatoso. Supporting adaptable distributed systems with formaware. In *Proceedings of the 24th International Conference on Distributed Computing Systems Workshops - W7 : EC (ICDCSW'04) - Volume 7*, ICDCSW '04, pages 320–325, Washington, DC, USA, 2004. IEEE Computer Society.

[MBM11] L. Malrait, S. Bouchenak, and N. Marchand. Experience with conser : A system for server control through fluid modeling. *IEEE Transactions on Computers*, 60(7) :951–963, July 2011.

[MBP+15] P. Merle, O. Barais, J. Parpaillon, N. Plouzeau, and S. Tata. A precise metamodel for open cloud computing interface. In *2015 IEEE 8th International Conference on Cloud Computing*, pages 852–859, June 2015.

[McA95] Jeff McAffer. Meta-level programming with coda. In *Proceedings of the 9th European Conference on Object-Oriented Programming*, ECOOP '95, pages 190–214, London, UK, UK, 1995. Springer-Verlag.

[MDA04] Daniel A. Menasce, Lawrence W. Dowdy, and Virgilio A. F. Almeida. *Performance by Design : Computer Capacity Planning By Example.* Prentice Hall PTR, Upper Saddle River, NJ, USA, 2004.

[Mey94] Bertrand Meyer. *Eiffel, le langage.* Dunod, 1994.

[MG11] Peter M. Mell and Timothy Grance. Sp 800-145. the nist definition of cloud computing. Technical report, Gaithersburg, MD, United States, 2011.

[MH12] Ming Mao and Marty Humphrey. A performance study on the vm startup time in the cloud. In *Proceedings of the 2012 IEEE Fifth International Conference on Cloud Computing*, CLOUD '12, pages 423–430, Washington, DC, USA, 2012. IEEE Computer Society.

[MK96] Jeff Magee and Jeff Kramer. Dynamic structure in software architectures. In *Proceedings of the 4th ACM SIGSOFT Symposium on Foundations of Software Engineering*, SIGSOFT '96, pages 3–14, New York, NY, USA, 1996. ACM.

[MKS+15] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei a Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, Stig

Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540) :529–533, 2015.

[ML01] Noury MN Bouraqadi-Saâdani and Thomas Ledoux. How to weave ? In *ECOOP 2001 Workshop on Advanced Separation of Concerns*, 2001.

[MLH+09] B. Morin, T. Ledoux, M. B. Hassine, F. Chauvel, O. Barais, and J. M. Jezequel. Unifying runtime adaptation and design evolution. In *2009 Ninth IEEE International Conference on Computer and Information Technology*, volume 1, pages 104–109, Oct 2009.

[MOC+14] Toni Mastelic, Ariel Oleksiak, Holger Claussen, Ivona Brandic, Jean-Marc Pierson, and Athanasios V. Vasilakos. Cloud computing : Survey on energy efficiency. *ACM Comput. Surv.*, 47(2) :33 :1–33 :36, December 2014.

[MSKC04] P. K. McKinley, S. M. Sadjadi, E. P. Kasten, and B. H. C. Cheng. Composing adaptive software. *Computer*, 37(7) :56–64, July 2004.

[MT00] N. Medvidovic and R. N. Taylor. A classification and comparison framework for software architecture description languages. *IEEE Transactions on Software Engineering*, 26(1) :70–93, Jan 2000.

[MVV14] Gabriele Mencagli, Marco Vanneschi, and Emanuele Vespa. A cooperative predictive control approach to improve the reconfiguration stability of adaptive distributed parallel applications. *ACM Trans. Auton. Adapt. Syst.*, 9(1) :2 :1–2 :27, March 2014.

[NDKJ11] Stefan Naumann, Markus Dick, Eva Kern, and Timo Johann. The greensoft model : A reference model for green and sustainable software and its engineering. *Sustainable Computing : Informatics and Systems*, 1(4) :294 – 304, 2011.

[NLS+11] Alexander Nowak, Frank Leymann, Daniel Schleicher, David Schumm, and Sebastian Wagner. Green business process patterns. In *Proceedings of the 18th Conference on Pattern Languages of Programs*, PLoP '11, pages 6 :1–6 :10, New York, NY, USA, 2011. ACM.

[OAL14] Anne-Cecile Orgerie, Marcos Dias de Assuncao, and Laurent Lefevre. A survey on techniques for improving the energy efficiency of large-scale distributed systems. *ACM Comput. Surv.*, 46(4) :47 :1–47 :31, March 2014.

[OGT+99] Peyman Oreizy, Michael M. Gorlick, Richard N. Taylor, Dennis Heimbigner, Gregory Johnson, Nenad Medvidovic, Alex Quilici, David S. Rosenblum, and Alexander L. Wolf. An architecture-based approach to self-adaptive software. *IEEE Intelligent Systems*, 14(3) :54–62, May 1999.

[OMG98] OMG. The common object request broker : Architecture and specification. Technical report, Object Management Group, feb 1998. Revision 2.2 - Document formal/98-07-01.

[OMT08] Peyman Oreizy, Nenad Medvidovic, and Richard N. Taylor. Runtime software adaptation : Framework, approaches, and styles. In *Companion of the 30th International Conference on Software Engineering*, ICSE Companion '08, pages 899–910, New York, NY, USA, 2008. ACM.

[Ore98] Peyman Oreizy. Issues in modeling and analyzing dynamic software architectures. In *International Workshop on the Role of Software Architecture in Testing and Analysis*, June 1998.

[Par72] D. L. Parnas. On the criteria to be used in decomposing systems into modules. *Commun. ACM*, 15(12) :1053–1058, December 1972.

[PBCD11] Carlos Parra, Xavier Blanc, Anthony Cleve, and Laurence Duchien. Unifying design and runtime software adaptation using aspect models. *Sci. Comput. Program.*, 76(12) :1247–1260, December 2011.

[PFL17] Charles Prud'homme, Jean-Guillaume Fages, and Xavier Lorca. *Choco Documentation.* TASC, LS2N CNRS UMR 6241, COSLING S.A.S., 2017.

[PHAH16] C. Pang, A. Hindle, B. Adams, and A. E. Hassan. What do programmers know about software energy consumption ? *IEEE Software*, 33(3) :83–89, May 2016.

[PLL14] G. Procaccianti, P. Lago, and G. A. Lewis. A catalogue of green architectural tactics for the cloud. In *2014 IEEE 8th International Symposium on the Maintenance and Evolution of Service-Oriented and Cloud-Based Systems*, pages 29–36, Sept 2014.

[PLM10] Rémy Pottier, Marc Léger, and Jean-Marc Menaud. A reconfiguration language for virtualized grid infrastructures. In Frank Eliassen and Rüdiger Kapitza, editors, *Distributed Applications and Interoperable Systems*, volume 6115 of *Lecture Notes in Computer Science*, pages 42–55. Springer Berlin Heidelberg, 2010.

[PMSD07] Juraj Polakovic, Sebastien Mazare, Jean-Bernard Stefani, and Pierre-Charles David. *Component-Based Software Engineering : 10th International Symposium, CBSE 2007, Medford, MA, USA, July 9-11, 2007. Proceedings*, chapter Experience with Safe Dynamic Reconfigurations in Component-Based Embedded Systems, pages 242–257. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[PP06] KyoungSoo Park and Vivek S. Pai. Comon : A mostly-scalable monitoring system for planetlab. *SIGOPS Oper. Syst. Rev.*, 40(1) :65–74, January 2006.

[PSG06] Adrian Paschke and Elisabeth Schnappinger-Gerull. A categorization scheme for sla metrics. In *Service Oriented Electronic Commerce*, 2006.

[Ray13] Michel Raynal. *Distributed Algorithms for Message-Passing Systems.* Springer Publishing Company, 2013.

[RBW06] Francesca Rossi, Peter van Beek, and Toby Walsh. *Handbook of Constraint Programming (Foundations of Artificial Intelligence).* Elsevier Science Inc., New York, NY, USA, 2006.

[RBX+09] Jia Rao, Xiangping Bu, Cheng-Zhong Xu, Leyi Wang, and George Yin. Vconf : A reinforcement learning approach to virtual machines auto-configuration. In *Proceedings of the 6th International Conference on Autonomic Computing*, ICAC '09, pages 137–146, New York, NY, USA, 2009. ACM.

[RC02] Barry Redmond and Vinny Cahill. Supporting unanticipated dynamic adaptation of application behaviour. In *Proceedings of the 16th European Conference on Object-Oriented Programming*, ECOOP '02, pages 205–230, London, UK, UK, 2002. Springer-Verlag.

[RDT08] David Röthlisberger, Marcus Denker, and Éric Tanter. Unanticipated partial behavioral reflection : Adapting applications at runtime. *Comput. Lang. Syst. Struct.*, 34(2-3) :46–65, July 2008.

[Riv97] Fred Rivard. *Évolution du comportement des objets dans les langages à classes réflexifs.* PhD thesis, Université de Nantes, Ecole des Mines de Nantes, juin 1997.

[RL80] M. Reiser and S. S. Lavenberg. Mean-value analysis of closed multichain queuing networks. *J. ACM*, 27(2) :313–322, April 1980.

[Roy07] Peter Van Roy. Self management and the future of software design. *Electronic Notes in Theoretical Computer Science*, 182 :201 – 217, 2007.

[RSAM06] Romain Rouvoy, Patricia Serrano-Alvarado, and Philippe Merle. *Software Composition : 5th International Symposium, SC 2006 Vienna, Austria, March 25-26, 2006 Revised Papers*, chapter A Component-Based Approach to Compose Transaction Standards, pages 114–130. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.

[SB98] Richard S. Sutton and Andrew G. Barto. *Introduction to Reinforcement Learning.* MIT Press, Cambridge, MA, USA, 1st edition, 1998.

[SBDP08] Sylvain Sicard, Fabienne Boyer, and Noel De Palma. Using components for architecture-based management : The self-repair case. In *Proceedings of the 30th International Conference on Software Engineering*, ICSE '08, pages 101–110, New York, NY, USA, 2008. ACM.

[SBK+16] Damián Serrano, Sara Bouchenak, Yousri Kouki, Frederico Alvares de Oliveira Jr., Thomas Ledoux, Jonathan Lejeune, Julien Sopena, Luciana Arantes, and Pierre Sens. Sla guarantees for cloud services. *Future Generation Computer Systems*, 54 :233 – 246, 2016.

[SBMP07] Christophe Sibertin-Blanc, Philippe Mauran, and Gérard Padiou. Safe adaptation of component coordination. *Electronic Notes in Theoretical Computer Science*, 189 :69 – 85, 2007.

[Sch06] D. C. Schmidt. Guest editor's introduction : Model-driven engineering. *Computer*, 39(2) :25–31, Feb 2006.

[Sch09] H. E. Schaffer. X as a service, cloud computing, and the need for good judgment. *IT Professional*, 11(5) :4–5, Sept 2009.

[SH10] W. Shi and B. Hong. Resource allocation with a budget constraint for computing independent tasks in the cloud. In *2010 IEEE Second International Conference on Cloud Computing Technology and Science*, pages 327–334, Nov 2010.

[Slo94] Morris Sloman. Policy driven management for distributed systems. *Journal of Network and Systems Management*, 2(4) :333–360, 1994.

[SMF+09] L. Seinturier, P. Merle, D. Fournier, N. Dolet, V. Schiavoni, and J. B. Stefani. Reconfigurable sca applications with the frascati platform. In *2009 IEEE International Conference on Services Computing*, pages 268–275, Sept 2009.

[Smi84] Brian C. Smith. Reflection and semantics in lisp. In *Proceedings of the 11th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*, POPL '84, pages 23–35, New York, NY, USA, 1984. ACM.

[Smi90] Brian C. Smith. What do you mean, meta ? In *In Workshop on Reflection and MetaLevel Architectures in OO Programming, ECOOP/OOPSLA'90*, oct 1990.

[SMM08] Chiyoung Seo, Sam Malek, and Nenad Medvidovic. Component-level energy consumption estimation for distributed java-based software systems. In *Proceedings of the 11th International Symposium on Component-Based Software Engineering*, CBSE '08, pages 97–113, Berlin, Heidelberg, 2008. Springer-Verlag.

[SMR+12] Lionel Seinturier, Philippe Merle, Romain Rouvoy, Daniel Romero, Valerio Schiavoni, and Jean-Bernard Stefani. A component-based middleware platform for reconfigurable service-oriented architectures. *Softw. Pract. Exper.*, 42(5) :559–583, May 2012.

[ST09] Mazeiar Salehie and Ladan Tahvildari. Self-adaptive software : Landscape and research challenges. *ACM Trans. Auton. Adapt. Syst.*, 4(2) :14 :1–14 :42, May 2009.

[SXC+10] Hui Song, Yingfei Xiong, Franck Chauvel, Gang Huang, Zhenjiang Hu, and Hong Mei. *Models in Software Engineering : Workshops and Symposia at MODELS 2009, Denver, CO, USA, October 4-9, 2009, Reports and Revised Selected Papers*, chapter Generating Synchronization Engines between Running Systems and Their Model-Based Views, pages 140–154. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[Szy02] Clemens Szyperski. *Component Software : Beyond Object-Oriented Programming*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2nd edition, 2002.

[TGCB08] François Taïani, Paul Grace, Geoff Coulson, and Gordon Blair. Past and future of reflective middleware : Towards a corpus-based impact analysis. In *Proceedings of the 7th Workshop on Reflective and Adaptive Middleware*, ARM '08, pages 41–46, New York, NY, USA, 2008. ACM.

[TGGL82] Irving L. Traiger, Jim Gray, Cesare A. Galtieri, and Bruce G. Lindsay. Transactions and consistency in distributed database systems. *ACM Trans. Database Syst.*, 7(3) :323–342, 1982.

[TL15] Marian Turowski and Alexander Lenk. Vertical scaling capability of openstack. In Farouk Toumani, Barbara Pernici, Daniela Grigori, Djamal Benslimane, Jan Mendling, Nejib Ben Hadj-Alouane, Brian Blake, Olivier Perrin, Iman Saleh Moustafa, and Sami Bhiri, editors, *Service-Oriented Computing - ICSOC 2014 Workshops*, pages 351–362, Cham, 2015. Springer International Publishing.

[TMS10] Alban Tiberghien, Philippe Merle, and Lionel Seinturier. *Abstract State Machines, Alloy, B and Z : Second International Conference, ABZ 2010, Orford, QC, Canada, February 22-25, 2010. Proceedings*, chapter Specifying Self-configurable Component-Based Systems with FracToy, pages 91–104. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[TOHS99] P. Tarr, H. Ossher, W. Harrison, and S. M. Sutton. N degrees of separation : multi-dimensional separation of concerns. In *Proceedings of the 1999 International Conference on Software Engineering (IEEE Cat. No.99CB37002)*, pages 107–119, May 1999.

[TS11] Kirti Tyagi and Arun Sharma. Reliability of component based systems : A critical survey. *SIGSOFT Softw. Eng. Notes*, 36(6) :1–6, November 2011.

[UPS+07] Bhuvan Urgaonkar, Giovanni Pacifici, Prashant Shenoy, Mike Spreitzer, and Asser Tantawi. Analytic modeling of multitier internet applications. *ACM Trans. Web*, 1(1), May 2007.

[VAN08] Akshat Verma, Puneet Ahuja, and Anindya Neogi. pmapper : Power and migration cost aware application placement in virtualized systems. In *Proceedings of the 9th ACM/IFIP/USENIX International Conference on Middleware*, Middleware '08, pages 243–264, New York, NY, USA, 2008. Springer-Verlag New York, Inc.

[VB03] Yves Vandewoude and Yolande Berbers. A meta-model driven methodology for state transfer in component-oriented systems. In *Proceedings of the 2nd International Workshop on Unanticipated Software Evolution (USE)*, 2003.

[vDKV00] Arie van Deursen, Paul Klint, and Joost Visser. Domain-specific languages : an annotated bibliography. *SIGPLAN Not.*, 35 :26–36, June 2000.

[VK03] Giuseppe Valetto and Gail Kaiser. Using process technology to control and coordinate software adaptation. In *Proceedings of the 25th International Conference on Software Engineering*, ICSE '03, pages 262–272, Washington, DC, USA, 2003. IEEE Computer Society.

[VRMCL08] Luis M. Vaquero, Luis Rodero-Merino, Juan Caceres, and Maik Lindner. A break in the clouds : Towards a cloud definition. *SIGCOMM Comput. Commun. Rev.*, 39(1) :50–55, December 2008.

[VWMA11] Pieter Vromant, Danny Weyns, Sam Malek, and Jesper Andersson. On interacting control loops in self-adaptive systems. In *Proceedings of the 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, SEAMS '11, pages 202–207, New York, NY, USA, 2011. ACM.

[W3C10] W3C. Xml path language (xpath) 2.0. `https://www.w3.org/TR/xpath20/`, Dec 2010. Accessed : 2017-01-16.

[WB10] Linlin Wu and Rajkumar Buyya. Service level agreement (SLA) in utility computing systems. *CoRR*, abs/1010.2881, 2010.

[WHGK14] Andreas Weber, Nikolas Herbst, Henning Groenda, and Samuel Kounev. Towards a resource elasticity benchmark for cloud environments. In *Proceedings of the 2Nd International Workshop on Hot Topics in Cloud Service Scalability*, HotTopiCS '14, pages 5 :1–5 :8, New York, NY, USA, 2014. ACM.

[WHH+13] J. Wang, C. Huang, K. He, X. Wang, X. Chen, and K. Qin. An energy-aware resource allocation heuristics for vm scheduling in cloud. In *2013 IEEE 10th International Conference on High Performance Computing and Communications 2013 IEEE International Conference on Embedded and Ubiquitous Computing*, pages 587–594, Nov 2013.

[WIdlIA12] Danny Weyns, M. Usman Iftikhar, Didac Gil de la Iglesia, and Tanvir Ahmad. A survey of formal methods in self-adaptive systems. In *Proceedings of the Fifth International C* Conference on Computer Science and Software Engineering*, C3S2E '12, pages 67–79, New York, NY, USA, 2012. ACM.

[wik] Définition de réification. `https://en.wikipedia.org/wiki/Reification_(computer_science)`. Accessed : 2016-12-22.

[Wir95] Niklaus Wirth. A plea for lean software. *Computer*, 28 :64–68, 1995.

[WS92] Gerhard Weikum and Hans-J. Schek. Database transaction models for advanced applications. chapter Concepts and Applications of Multilevel Transactions and Open Nested Transactions, pages 515–553. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1992.

[WSG+13] Danny Weyns, Bradley Schmerl, Vincenzo Grassi, Sam Malek, Raffaela Mirandola, Christian Prehofer, Jochen Wuttke, Jesper Andersson, Holger Giese, and Karl M. Göschka. *Software Engineering for Self-Adaptive Systems II : International Seminar, Dagstuhl Castle, Germany, October 24-29, 2010 Revised Selected and Invited Papers*, chapter On Patterns for Decentralized Control in Self-Adaptive Systems, pages 76–107. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[WVZX10] Guiyi Wei, Athanasios V. Vasilakos, Yao Zheng, and Naixue Xiong. A game-theoretic method of fair resource allocation for cloud computing services. *The Journal of Supercomputing*, 54(2) :252–269, 2010.

[XDB16] M. Xu, A. V. Dastjerdi, and R. Buyya. Energy efficient scheduling of cloud application components with brownout. *IEEE Transactions on Sustainable Computing*, 1(2) :40–53, July 2016.

[Yah11] Interview de Todd Papaioannou par Julie Bort. `http://www.networkworld.com/article/2179359/cloud-computing/yahoo-builds-ultimate-private-cloud.html`, July 2011.

[YH98] Makoto Yokoo and Katsutoshi Hirayama. Distributed constraint satisfaction algorithm for complex local problems. In *Third Int. Conf. on Multiagent Systems (ICMAS)*, pages 372–381, 1998.

[YHS+12] Y. Yao, L. Huang, A. Sharma, L. Golubchik, and M. Neely. Data centers power reduction : A two time scale approach for delay tolerant workloads. In *2012 Proceedings IEEE INFOCOM*, pages 1431–1439, March 2012.

[YQL09] J. Yang, J. Qiu, and Y. Li. A profile-based approach to just-in-time scalability for cloud applications. In *2009 IEEE International Conference on Cloud Computing*, pages 9–16, Sept 2009.

[YT16] Sami Yangui and Samir Tata. An occi compliant model for paas resources description and provisioning. *The Computer Journal*, 59(3) :308–324, 2016.

[ZC06] Ji Zhang and Betty H. C. Cheng. Model-based development of dynamically adaptive software. In *Proceedings of the 28th International Conference on Software Engineering*, ICSE '06, pages 371–380, New York, NY, USA, 2006. ACM.

[ZGC09] Ji Zhang, Heather J. Goldsby, and Betty H.C. Cheng. Modular verification of dynamically adaptive systems. In *Proceedings of the 8th ACM International Conference on Aspect-oriented Software Development*, AOSD '09, pages 161–172, New York, NY, USA, 2009. ACM.

[ZPL+12] Han Zhao, Miao Pan, Xinxin Liu, Xiaolin Li, and Yuguang Fang. Optimal resource rental planning for elastic applications in cloud market. In *Proceedings of the 2012 IEEE 26th International Parallel and Distributed Processing Symposium*, IPDPS '12, pages 808–819, Washington, DC, USA, 2012. IEEE Computer Society.

Quatrième partie

Annexe : Curriculum Vitae détaillé

I. Curriculum Vitae

A. Informations personnelles

- Nom : Thomas Ledoux
- Date de naissance : né le 15/09/1967 aux Sables d'Olonne (85)
- Nationalité : française
- Situation de famille : marié, 3 enfants
- Adresse personnelle : 18 avenue Guillemet, 44000 Nantes
- Affiliation : Enseignant-chercheur IMT Atlantique campus de Nantes[1], équipe Stack[2] (INRIA Rennes-Bretagne Atlantique, LS2N)
- Adresse professionnelle : 4 rue Alfred Kastler, 44307 Nantes cedex 3
- Contact : Thomas.Ledoux@imt-atlantique.fr, Tel : 02.51.85.82.19, web : http://www.emn.fr/ledoux

B. Expérience professionnelle

- Depuis septembre 1998 : Maître-assistant IMT Atlantique campus de Nantes avec 2 années de détachement à l'INRIA (2008-2010)
- 1997-1998 : Maître-assistant associé (équivalent ATER) à l'Ecole des Mines de Nantes
- 1994-1997 : Doctorant en informatique des Mines de Nantes en bourse Cifre avec IBM France
- 1992-1994 : Ingénieur d'études en informatique à IBM France (technologie objet, Smalltalk)

C. Formation

- 1998 : Doctorat en Informatique de l'Université de Nantes (bourse Cifre avec IBM)

« Réflexion dans les systèmes répartis : application à CORBA et Smalltalk » – Université de Nantes, École des Mines de Nantes, France, Mars 1998.

Jury :

- Rapporteurs : Jean-Marc Geib (Université de Lille), Rachid Guerraoui (EPFL)
- Examinateurs : Jean Bézivin (Université de Nantes), Jean-Pierre Briot (Université Paris 6), Michel Riveill (Université de Savoie)
- Directeur : Pierre Cointe (École des Mines de Nantes)
- Membre invité : Philippe Gatet, IBM Global Services

- 1991 : DEA Informatique de l'Université de Paris-Dauphine
- 1990 : Maîtrise en Informatique de l'Université de Nantes
- 1989 : Licence en Informatique de l'Université de Nantes
- 1987 : DEUG Sciences de l'Université de Nantes
- 1985 : Baccalauréat, série C

D. Distinction

Chevalier dans l'ordre des Palmes Académiques (promotion 1er janvier 2016)

[1] Nouveau nom de l'Ecole des Mines de Nantes suite à la fusion avec Telecom Bretagne au 1er janvier 2017
[2] Stack est une *spin-off* de l'équipe-projet Ascola créée en 2017, elle-même issue de l'équipe-projet Obasco en 2008

II. Activités de recherche

A. Domaine de recherche

- H-Index : 23 (d'après Google Scholar 08/06/2018)
- Profil : génie logiciel, langages de programmation (à objets, par aspects, à composants, méta-programmation), architectures logicielles et patrons de conception, systèmes autonomiques et reconfiguration dynamique
- Domaines d'application : systèmes distribués plus particulièrement intergiciels (*middleware*), architectures orientées services (SOA), informatique dans les nuages (*Cloud computing*) et informatique verte (*green computing)*

B. Contexte scientifique

En rejoignant en thèse Pierre Cointe en 1994 à l'Ecole des Mines de Nantes, j'ai intégré l'équipe en construction autour de la programmation par objets. Ma communauté scientifique était celle du futur GDR GPL et mes conférences cibles ECOOP, OOPSLA pour l'international et LMO pour le national. Avec l'équipe de Pierre Cointe, nous avons participé à l'émergence de la programmation par aspects (et sa conférence internationale AOSD, puis le réseau d'excellence européen AOSD), à la diffusion de la technologie Java (conférences OCM, compilateur Eclipse). J'ai ensuite été membre de l'équipe-projet INRIA/LINA Obasco (dir. Pierre Cointe, 2002-2008), puis membre de l'équipe-projet INRIA/LINA Ascola (dir. Mario Südholt, 2008-2017). Tout récemment (nov. 2017), je suis devenu membre de l'équipe INRIA/LS2N Stack, une *spin-off* de l'équipe-projet Ascola (dir. Adrien Lèbre). Au cours des années 2000, j'ai élargi le champ de ma communauté scientifique en me rapprochant du GDR ASR notamment du fait de mes travaux sur les composants/services répartis. Cette évolution s'est accélérée depuis avec mes travaux autour du *Cloud Computing* et je publie maintenant régulièrement dans des conférences de la communauté systèmes distribués.

C. Synthèse de mes activités de recherche

Depuis 1994, je m'intéresse au paradigme de l'approche à composants et au principe de « séparation des préoccupations » (*separation of concerns*) qui proposent de mieux structurer les systèmes logiciels pour favoriser leur réutilisation et leur évolution. Mais plus spécifiquement, mes travaux ont pour fil conducteur la problématique de la *reconfiguration dynamique d'architectures logicielles* que j'ai initiée lors de ma thèse avec la contribution OpenCorba [39, 50].

En effet, dans un premier temps, j'ai utilisé comme moyen d'expression la réflexion pour proposer une taxonomie de métaclasses pour rétro-concevoir des bibliothèques de classes Smalltalk justifiant ainsi l'intérêt de la réflexion pour la conception de langages à objets [41]. C'est également à cette époque que j'ai proposé avec deux autres doctorants (N. Bouraqadi, F. Rivard) un nouveau modèle pour organiser les architectures à méta-niveaux [40, 51]. Finalement, l'application de ces techniques au domaine des intergiciels a été le sujet de ma thèse et m'a permis de concevoir et d'implémenter OpenCorba un bus logiciel ouvert basé sur un protocole à méta-objets en Smalltalk [39, 50]. J'ai pendant un temps collaboré avec S. Danforth et I. Forman de IBM Austin (du fait de ma bourse Cifre avec IBM) sur leur produit Corba nommé SOM/DSOM qui possédait un noyau réflexif issu des travaux de Pierre Cointe avant de me consacrer à OpenCorba. Ensuite, intéressé par les liens entre la réflexion et la programmation par aspects, j'ai montré

que la méta-programmation pouvait être un substrat possible pour la conception de langages d'aspects [3, 14], offrant ainsi de nouvelles propriétés comme le tissage dynamique dans AspectJ [63].

Puis, avec le projet RNTL ARCAD (2000-2004) – qui m'a permis de fréquenter des personnalités aussi diverses que M. Riveill (ESSI), D. Caromel (I3S), T. Coupaye (France Telecom R&D), J.-B. Stefani (INRIA Rhône-Alpes) – je me suis concentré sur le concept d'adaptation dynamique dans les architectures logicielles à base de composants. ARCAD a participé à l'émergence du modèle de composants Fractal (fractal.ow2.org), à la définition d'un patron d'auto-administration pour logiciel à base de composants avec le *framework* Safran [37], à la réalisation d'un intergiciel pour le monitoring (WidCAT [60]) mais également à la proposition d'un aspect d'adaptation pour la reconfiguration dynamique de composants Fractal [12, 36, 47].

Avec le projet RNTL Selfware (2005-2008) – et ses partenaires en particulier D. Hagimont (ENSEEIHT), N. De Palma (INRIA Rhône-Alpes) et encore T. Coupaye – j'ai abordé un nouveau domaine qui était déjà un domaine cible mais implicite de mes précédents travaux : celui de *l'autonomic computing*. Mon idée était d'utiliser les approches langages dédiés (au sens *Domain Specific Language*) pour développer des langages sur mesure pour l'administration autonomique de systèmes répartis. L'intérêt du langage dédié est d'être spécifiquement conçu pour un domaine d'application particulier, celui de la reconfiguration dynamique dans notre cas. J'ai notamment contribué à la définition du langage de reconfiguration FScript [11, 59] pour le modèle de composants Fractal ; j'ai participé à l'intégration d'un moniteur transactionnel dans l'interpréteur FScript pour rendre les reconfigurations fiables [31, 57] ; enfin, en parallèle avec mes partenaires de France Telecom R&D (N. Rivierre), intéressé par la gestion de la qualité de service (QoS) dans les orchestrations de Web services, j'ai contribué à la conception d'un langage dédié permettant de spécifier des règles d'adaptation dynamique pour garantir la QoS lors de fluctuations de l'environnement d'exécution [33, 35, 58].

Entre 2008 et 2010, je profite d'une période de détachement à INRIA pour prendre un peu de recul sur mes travaux de recherche. Je réalise alors que j'ai focalisé une grande partie de mes travaux sur le *Comment* réaliser la reconfiguration dynamique mais presque pas sur le *Pourquoi ?* Même si certaines évaluations ont permis de (i) modifier les mécanismes de distribution à l'exécution (OpenCorba [39]) ; (ii) redimensionner un composant cache serveur à l'exécution et modifier la politique d'adaptation elle-même (Safran [37]) ; (ii) réaliser une auto-réparation d'un serveur Java EE [31] ; etc. je me sentais modestement motivé par ces expérimentations. L'émergence du *Cloud Computing* et le réveil d'une conscience citoyenne liée au développement durable vont changer la donne.

A partir de 2010, je me suis rapproché de mon collègue Jean-Marc Menaud, et je me suis orienté vers la thématique du *Green computing* dans le but d'optimiser l'empreinte énergétique des centres données qui accompagnent l'essor spectaculaire du *Cloud computing* [10]. L'idée phare est que le logiciel hébergé par un centre de données doit lui-même participer à la réduction de la consommation énergétique. D'où des problématiques d'optimisation multi-niveaux, multi-critères, de coordination de boucles autonomiques entre l'application et l'infrastructure virtuelle l'hébergeant [28]. La reconfiguration dynamique d'architectures Cloud multi-niveaux sera dirigée par des compromis qualité service (QoS)/empreinte énergétique (un composant gourmand en énergie rendant en général un service de meilleure qualité) [23]. Ces compromis seront possibles car les critères de QoS seront spécifiés en amont dans un contrat de service SLA (*Service Level Agreement*). C'est dans le cadre du projet ANR MyCloud (2010-2013) -- et ses partenaires, en particulier Pierre Sens (LIP6) et Sara Bouchenak (INRIA Rhône-Alpes) -- que nous avons abordé cette problématique et défini le langage CSLA [8,25,29], prenant finement en compte les caractéristiques du Cloud. Pour conclure, la gestion autonomique de l'élasticité multi-couche des applications dans le Cloud pour une utilisation efficiente des ressources devient ainsi un axe majeur de ma recherche ces dernières années [17,19].

Avec le projet EPOC (2013-2017) du Labex CominLabs, l'idée est d'intégrer des sources d'énergie renouvelable comme objet de première classe dans la prise de décision pour optimiser l'empreinte carbone du Cloud (*green energy vs brown energy*) [7,21]. Mon objectif actuel est de relier ces résultats à nos travaux précédents en proposant des contrats de type « Green SLA » à l'utilisateur final du Cloud [5,6,15,18].

D. Production scientifique

	International	National	Total
Livres ou chapitres de livres	4	0	4
Revues	8	3	11
Conférences	28	9	37
Ateliers	16	3	19
TOTAL	56	15	71

1. En terme d'impact

D'après Google Scholar, 1807 citations au 08/06/2018. Référencement/papier :

> 200 fois :

- Thomas Ledoux. *OpenCorba: a Reflective Open Broker*. In Proceedings of the Second International Conference on Meta-Level Architectures and Reflection (Reflection'99), Springer-Verlag, LNCS Vol. 1616, Saint-Malo, France, July 1999.

> 100 fois :

- Pierre-Charles David and Thomas Ledoux. *Towards a Framework for Self-Adaptive Component-Based Applications*. In Distributed Applications and Interoperable Systems (DAIS'03), Springer-Verlag, LNCS Vol. 2893, Paris, France, November 2003.
- Pierre-Charles David and Thomas Ledoux. *WildCAT: a generic framework for context-aware applications.* In Proceedings of the 3rd international workshop on Middleware for pervasive and ad-hoc computing (MPAC 2005), ACM Digital Library, Grenoble, France, November 2005.
- Pierre-Charles David and Thomas Ledoux. *An Aspect-Oriented Approach for Developing Self-Adapting Fractal Components*. In Software composition (SC 2006), Satellite event of ETAPS 2006, Springer-Verlag, LNCS Vol. 4089, Vienna, Austria, March 2006.

2. En terme de classement de conférences

Rang conférence	Nombre
A* - flagship conference, a leading venue in a discipline area	1 (OOPSLA)
A - excellent conference, and highly respected in a discipline area	6 (CCGridx2, Coordination, ICSOCx2,Reflection[3])
B - good conference, and well regarded in a discipline area	8 (CBSE, DAISx2, HPCC, SACx2, SC, Services)
C - other ranked conference venues that meet minimum standards	4 (CIT, CloudComx2, ISOTAS)

D'après CORE 2017 Conference Portal[4] :

[3] Rebaptisée AOSD puis Modularity
[4] CORE2017 Summary: A* - 4%, A - 14%, B - 26%, C - 49%, Other - 8%

3. Les cinq plus importantes selon moi (ordre : chronologie inversée)

Ces publications m'ont donné – pour des raisons différentes – une grande satisfaction intellectuelle.

- Md Sabbir Hasan, Frederico Alvares de Oliveira , Thomas Ledoux and Jean-Louis Pazat. *Investigating Energy consumption and Performance trade-off for Interactive Cloud Application*. In IEEE Transactions on Sustainable Computing, vol. 2, no. 2, pp.113-126, April-June 2017
- Damián Serrano, Sara Bouchenak, Yousri Kouki, Frederico Alvares de Oliveira Jr., Thomas Ledoux, Jonathan Lejeune, Julien Sopena, Luciana Arantes, Pierre Sens. *SLA guarantees for Cloud Services*. In Future Generation Computer Systems (FGCS), pages 233-246, Volume 54, January 2016.
- Yousri Kouki, Frederico Alvares de Oliveira Jr., Simon Dupont and Thomas Ledoux. A *Language Support for Cloud Elasticity Management*. In IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), Chicago, USA, May 26-29, 2014.
- Pierre-Charles David, Thomas Ledoux, Marc Léger and Thierry Coupaye. *FPath and FScript: Language support for navigation and reliable reconfiguration of Fractal architectures*. In Special issue on Component-based architecture: the Fractal Initiative. Annals of Telecommunications, Volume 64, n°1/2, Springer, January/February 2009.
- Pierre-Charles David and Thomas Ledoux. *An Aspect-Oriented Approach for Developing Self-Adapting Fractal Components*. In Software composition (SC 2006), Satellite event of ETAPS 2006, Springer-Verlag, LNCS Vol. 4089, Vienna, Austria, March 2006.

4. Best paper award

Hugo Bruneliere, Zakarea Al-shara, Frederico Alvares, Jonathan Lejeune and Thomas Ledoux. *A Model-based Architecture for Autonomic and Heterogeneous Cloud Systems*. In the 8h International Conference on Cloud Computing and Services Science (CLOSER 2018), Funchal, Madeira, Portugal, 19-21 March 2018.

Jonathan Lejeune, Frederico Alvares and Thomas Ledoux. *Towards a generic autonomic model to manage Cloud Services*. In the 7th International Conference on Cloud Computing and Services Science (CLOSER 2017), Porto, Portugal, 24-26 April 2017

E. Encadrements

1. Encadrement doctoral

1. **Sabbir Hasan** (origine : Kyung Hee University, Corée du Sud – soutenance 03/05/2017) : *Smart management of renewable energy in Clouds: from infrastructure to application*. Directeur : Jean-Louis Pazat. Financement : Labex CominLabs. Maintenant Post-doc chez Orange Labs
2. **Simon Dupont** (origine : Univ. Nantes – soutenance : 26/04/2016) : Gestion autonomique de l'élasticité multi-couche des applications dans le Cloud. Directeur : Jean-Marc Menaud. Financement : bourse Cifre Sigma informatique. Maintenant CDI chez Sigma.
3. **Yousri Kouki** (origine : ENSI Manouba, Tunisie – soutenance : 09/12/2013) : Approche dirigée par les contrats de niveaux de service pour la gestion de l'élasticité du "nuage". Directeur : Pierre Cointe. Financement : ANR. Maintenant CDI chez Linagora.
4. **Frederico Alvares** (origine : Technological Institute of Aeronautics, Brésil – soutenance : 09/04/2013) : *Multi Autonomic Management for Optimizing Energy Consumption in Cloud Infrastructures*. Directeur : Pierre Cointe. Financement : MESR. Maintenant CDI chez EasyVirt

5. **Marc Léger** (origine : Mines Nantes – soutenance : 19/05/2009) : Fiabilité des Reconfigurations Dynamiques dans les Architectures à Composants. Directeur : Pierre Cointe. Financement : Orange Labs (co-encadrant : Thierry Coupaye). Maintenant CDI chez Dassault Systèmes.
6. **Fabien Baligand** (origine : Mines Nantes – soutenance : 25/06/2008) : Une approche déclarative pour la gestion de la qualité de service dans les compositions de service. Directeur : Pierre Cointe. Financement : Orange Labs (co-encadrant : Nicolas Rivierre). Maintenant en CDI chez Aldebaran Robotics.
7. **Pierre-Charles David** (origine : Univ. Nantes – soutenance : 01/07/2005) : Développement de composants Fractal adaptatifs: un langage dédié à l'aspect d'adaptation. Directeur : Pierre Cointe. Financement : MESR. Maintenant en CDI chez Obeo.

2. Stages de Master

Langage dédié pour l'élasticité multi-couche dans le Cloud (**A. Benabadji**, 2017), Industrialisation de briques logicielles pour la gestion autonomique de l'élasticité multi-couche dans le Cloud (**S. Bouri**, 2016), Eco-conception de logiciel (**G. Delamare**, 2012), *A Generic Version of FScript: a Model-Driven Engineering Approach* (**M. Lacouture**, 2008), Intégrité structurelle dans les architectures à composants (**M. Léger**, 2005), Une infrastructure pour middleware adaptable (**P.C. David**, 2001), Vers un bus logiciel adaptable (**F. Sarradin**, 2000*), Transparent Strong Mobility using Reflective Smalltalk* (**G. Casarini**, 2000), *Mobile Agents* (**J. Windmuller**, 1999)

3. Post-docs/ingénieurs

Zakarea Alshara (oct. 2016-oct.2017), **Jonathan Lejeune** (oct. 2014-août 2015), **Yousri Kouki** (dec. 2013-sept.2014), **Frederico Alvares** (oct. 2012-oct. 2013), **Mahmoud Ben Hassine** (nov. 2008-nov. 2010, partagé avec INRIA Triskell), **Nicolas Loriant** (nov. 2007-août 2008), **Pierre-Charles David** (sept. 2006-fev.2008), **Patricio Salinas** (mars-nov. 2003), **Zahi Jarir** (oct.-dec. 2001)

F. Rayonnement scientifique

1. Jurys de thèse

Xuan Sang Le – Univ. Bretagne Occidentale (31/05/2017), **Zakarea Al Shara** – Univ. Montpellier (17/11/2016), **Erwan Daubert** – Univ. Rennes 1 (24/05/2013), **Loris Bouzonnet** – Univ. Grenoble (16/09/2011), **Roméo Said** – Univ. Bretagne Sud (23/02/2011), **Franck Chauvel** – Univ. Bretagne Sud (19/09/2008), **Nagapraveen Jayaprakash** – INP Grenoble (27/06/2008), **Romain Rouvoy** – Univ. Sciences et Technologies de Lille (08/12/2006), **Olivier Barais** – Univ. Sciences et Technologies de Lille (29/11/2005), **Victor Budau** – INT Evry (19/11/2003).

2. Comités de sélection

- **Université de Nantes** : membre du comité de sélection concours Maitre de conférences UFR Sciences (2010)
- **Université Rennes 1** : membre du comité de sélection concours Maitre de conférences IFSIC (2010)

3. Expertise

- **ANR** : expert pour l'Appel générique 2015, Arpège 2010, Technologies Logicielles 2007
- **INRIA Rennes-Bretagne Atlantique** : membre de la Commission Post-docs (2010)

4. Présidence de conférences/workshops/tutoriels

- **ComPAS 2013** : co- responsable (avec Sara Bouchenak, LIG) de l'atelier « SLA pour le Cloud Computing » (Grenoble, 15 janvier 2013). Keynote speaker : Ivona Brandic, Vienna University of Technology
- **ECOOP 2006** : co-tutorial chair (avec Antoine Beugnard, ENSTB) de la conférence ECOOP (Nantes, 3-4 juillet 2006) : une centaine de participants aux tutoriels
- **Journées Composants 2005** : président du Comité de Programme des Journées Composants (Le Croisic, 6-8 avril 2005) : une cinquantaine de participants
- **ECOOP 2000** : co- responsable (avec Shigeru Chiba, Univ. Tokyo – Walter Cazzola, Univ. Milan) du workshop "Reflection and Meta-level Architectures", ECOOP (Nice, 13 juin 2000) : une trentaine de participants

5. Comités de programme et comités éditoriaux

International

Livres

Achieving Federated and Self-Manageable Cloud Infrastructures: Theory and Practice (IGI Global, 2012)

Revues

Journal of Systems Architecture (2018), Future Generation Computer Systems (2018), IEEE Transactions on Services Computing (2017), IEEE Communications Letters (2016), Journal of Parallel and Distributed Computing - Elsevier (2015) ; Journal of Grid Computing - Springer (2013) ; Annals of Telecoms - special issue: Component-based architecture, the Fractal initiative - Springer (Volume 64, n°1/2-2009) ; IEEE Transactions of Software Engineering (2001)

Conférences

ACM Symposium On Applied Computing (SAC'13, SAC'14, SAC'15) - track Software Engineering Aspects of Green Computing ; International Conference on Eco-friendly Computing and Communication Systems (ICECCS'14, ICECCS'15) ; International Conference on ICT as Key Technology against Global Warming (ICT-GLOW'11, ICT - GLOW'12) ; IEEE International Conference on Green Computing (ICGREEN 2011)

Workshops

Workshop (CrossCloud'16, CrossCloud'17, CrossCloud'18) @ EuroSys ; Int. Workshop on Green and Sustainable Software (Greens'14, Greens'15, Greens'16, Greens'18) @ ICSE ; Workshop on Adaptive and Reflective Middleware (ARM'08, ARM'09, ARM'10, ARM'11, ARM'12, ARM'13, ARM'14, ARM'15, ARM'16, ARM'17) @ Middleware; Workshop CrossCloud'14 @INFOCOM ; Workshop CrossCloud Brokers '14 @ Middleware ; Int. Workshop on Green Computing Middleware (GCM'2010, GCM'11, GCM'12, GCM'13) @ UCC ; 1st Int. Workshop on Green In Software Engineering, Green By Software Engineering (GIBSE'13) @ AOSD ; ACM 1st Workshop on Middleware and Architectures for Autonomic and Sustainable Computing (MAASC'11) ; IEEE Int. Workshop on Methodologies for Non-functional Properties in Services Computing (2008)

National

Revues

Technique et Science Informatiques (TSI) - Hermès/Lavoisier (2013, 2014) ; Ingénierie des systèmes d'information (ISI) - n° spécial "Adaptation et gestion du contexte" – Hermès/ Lavoisier (vol.11/5-2006) ; Technique et Science Informatiques (TSI) - n° spécial "Systèmes à composants adaptables et extensibles", Hermès/Lavoisier (vol. 23/2 -2004) ; l'Objet - n° spécial "Coopération et systèmes à objets" – Hermès/Lavoisier (vol. 8/3 -2002)

Conférences

Compas 2018 (*track* Système) ; Conférence Francophone en Systèmes d'Exploitation (CFSE) 2006, 2008 ; Langages et Modèles à Objets (LMO) 2003 ; Journées Composants 2002, 2004, 2005, 2006.

G. Responsabilités scientifiques

1. Responsabilités collectives

- **Laboratoire LS2N** : co-animateur du thème transverse « Gestion de l'énergie et maîtrise des impacts environnementaux » (2017-…)
- **Association ADN Ouest (**http://www.adnouest.org) : membre de la commission Green (2017-…)
- **Association Green Lab Center** (http://www.greenlabcenter.fr) : membre du bureau (2013-2017)
- **Ecole doctorale STIM Pays de Loire** (ED 503) : membre du Conseil et de la cellule de site Nantes (2008-2013) ; responsable des manifestations de l'ED pour les doctorants (2008-2013)
- **Laboratoire LINA** (UMR 6241) : membre de la Commission des thèses (2009-2011) ; membre élu du Conseil du labo (2004-2007)
- **INRIA Rennes- Bretagne Atlantique** : membre du Comité de Centre (2009-2011)

2. Comités d'organisation

- **AOSD Summer School 2009** : co-organisation (avec Rémi Douence, Mines Nantes) de l'école d'été AOSD (Nantes, 24-28 août 2009) : une quarantaine de participants de plusieurs pays
- **Journée Thème Emergent de l'ASF :** co-organisation (avec Daniel Hagimont, ENSEEIHT) du JTE « Systèmes Automomes » (Toulouse, 16 nov. 2007)
- **Journées Composants 2002** : co-organisation (avec Michel Riveill, ESSI) des Journée composants : Systèmes à composants adaptables et extensibles (Grenoble, 17-18 oct. 2002)

H. Médiation scientifique

1. Présentation invitée

- « Projet CoMe4ACloud », 4th Grenoble Workshop on Autonomic Computing and Control (Grenoble, 23 octobre 2017)
- « Eco-élasticité logicielle pour un Cloud frugal », Conférence CNRS EcoInfo « Impact des logiciels sur l'environnement, quid de l'éco-conception ? » (Grenoble, 3 février 2017)
- Projet de recherche "TUBA" mené en collaboration avec Sigma Informatique. Train de la Nouvelle France Industrielle 2014 (22 avril à Nantes et 23 avril à Rennes)

- « Eco- élasticité logicielle» – Nantes Atlanticiels « Nantes, green IT is ? » (Nantes, 9 juillet 2013)
- « Elasticité logicielle pour optimiser l'empreinte énergétique » – colloque « Eco-conception » (Nantes, 18 oct. 2012)
- « Reconfigurations dynamiques fiables » – Séminaire au vert équipe INRIA Triskell (Le Tronchet, 14 juin 2010)
- « Pour un aspect d'adaptation dans le développement d'applications à base de composants » – séminaire groupe Polair, France Telecom R&D (Meylan, 8 juin 2004)

2. Organisation de manifestations de vulgarisation scientifique

- Cycle de conférences « **Les Jeudis de l'Objet** » (www.emn.fr/jeudis_objet) (Nantes) : responsable scientifique entre 1999 et 2012. Conférence regroupant une quarantaine de participants à chaque session et dont l'objectif était d'être un lieu d'échanges avec les industriels de la région autour des architectures logicielles : +1000 participants en cumul !
- **Eclipse Day** (15/03/2005, Nantes) : membre du comité d'organisation. Conférence consacrée à la plate-forme Eclipse : 200 participants environ
- **Objet 99** (Nantes), **OCM 2000** (18/05/2000, Nantes), **OCM 2002** (21/03/2002, Nantes) : membre du comité d'organisation. Conférences consacrées à la diffusion des résultats de la technologie des objets et des modèles auprès des académiques et des industriels des régions Bretagne et Pays de Loire : 150 participants en moyenne

III. Activités de valorisation

A. Contrat de recherche

Atlanstic 2020 CoMe4ACloud (2016-2017)

- Rôle : coordinateur Mines Nantes [budget Mines Nantes : 60 k€]
- Coordinateur : Thomas Ledoux
- Partenaires : Equipes Ascola, Tasc, AtlanModels du LINA
- Thème : *Constraints and Model Engineering for Autonomic Clouds*
- Objectif : proposer une architecture générique basée sur une approche modèle pour la gestion autonomique du Cloud. Nous dérivons un gestionnaire autonomique (AM) unique et générique capable de gérer n'importe quel service Cloud, quelle que soit la couche XaaS. L'AM générique est basé sur un solveur de contraintes qui tente de trouver la configuration optimale pour le XaaS modélisé et le meilleur équilibre entre les coûts et les revenus tout en respectant les contraintes concernant le SLA établi.
- Contribution : best paper award Closer'17 et Closer'18

Labex CominLabs EPOC (2013-2016)

- Rôle : participant Mines Nantes [budget Mines Nantes : 240 k€]
- Coordinateur : Jean-Marc Menaud (Ascola)
- Partenaires : ENIB, Telecom Bretagne, INSA-IRISA (Myriads), Mines Nantes-LINA (Ascola , Tasc), Univ. de Nantes-LINA (Aelos), Ecole Centrale de Nantes-IRCCyN (STR)
- Thème : *Energy proportional and opportunistic computing systems*
- Objectif : proposer des modèles d'optimisation, de gestion de ressources adaptée pour réaliser l'exécution d'une tâche informatique (e.g., appel de service Cloud, batch) consciente de l'énergie, sur une infrastructure qui va du matériel à l'application dans le contexte d'un centre de données mono-site qui est connecté au réseau électrique et à des sources d'énergie renouvelable (comme des éoliennes ou des panneaux solaires).
- Contribution : thèse Sabbir Hasan

FSN OpenCloudware (2012-2014)

- Rôle : participant Mines Nantes [budget Mines Nantes : 210 k€]
- Coordinateur : Daniel Stern (Orange Labs)
- Partenaires : Bull, Orange Labs, Thalès Communications, Thalès Services, ActiveEon, eNovance, eXo, Linagora, UShareSoft, ,Armines/Mines Nantes, IRIT, Télécom Paris Tech, Télécom Saint- Etienne, Univ. Joseph Fourier, Univ. Savoie, INRIA, OW2
- Thème : *Think to PaaS for Multi-IaaS Cloud Computing*
- Objectif : fournir une plate-forme d'ingénierie logicielle ouverte permettant des développements collaboratifs d'applications Cloud, ainsi que leur déploiement et administration, en visant une portabilité sur des infrastructures Cloud IaaS multiples. Disponible sur le nuage, elle intègre l'ensemble des briques d'ingénierie du logiciel pour gérer le cycle de vie *d'appliances* virtuelles complexes, disponible à la carte.
- Contribution : diffusion CSLA

ANR MyCloud (2010-2013)

- Rôle : coordinateur pour Mines Nantes [budget Mines Nantes : 190 k€]
- Coordinateur : Sara Bouchenak (Sardes)
- Partenaires : INRIA Rhône-Alpes (Sardes), LIP6, Mines Nantes (Ascola), WeAreCloud
- Thème : SLA et qualité de service pour Cloud Computing
- Objectif : proposer le premier modèle de cloud SLAaaS (SLA aware Service), qui intègre la qualité de service et le contrat SLA (Service Level Agreement) comme éléments à part entière du Cloud. Deux questions principales sont traitées : (i) comment construire un Cloud qui soit capable de se reconfigurer dynamiquement et de manière autonomique pour garantir le niveau de service et le SLA établi avec l'utilisateur ; (ii) comment fournir à l'utilisateur un moyen de gouvernance de SLA pour qu'il soit partie prenante du SLA et qu'il ait un retour sur l'état du Cloud au cours du temps (tel que l'empreinte énergétique du Cloud ou la violation de SLA).
- Contribution : langage CSLA, thèse Yousri Kouki

ADT INRIA Galaxy (2008-2010)

- Rôle : coordinateur pour Mines Nantes [budget total : 43 k€ hors personnel]
- Coordinateur : Alain Boulze (Tuvalu)
- Partenaires : INRIA Adam, Ascola, Ecoo, Oasis, Sardes, Triskell, Tuvalu
- Thème : une plateforme R&D SOA (Service Oriented Architecture) ouverte et agile
- Objectif : fournir un environnement intégré pour piloter l'agilité dans des architectures orientées services et offrir en ce sens diverses fonctionnalités: (i) intégration des paradigmes de service, composant et processus ; (ii) pilotage SOA de bout en bout intégrant la modélisation, le déploiement, l'exécution et la surveillance de systèmes et applications ; (iii) fourniture des capacités d'orchestration dynamique de services, et de distribution de services sur une grille de calcul qui repose sur des architectures logicielles hautement distribuées, dynamiquement adaptables et reconfigurables.
- Contribution : diffusion logiciel FScript & WildCAT

RNTL Selfware (2005-2008)

- Rôle : coordinateur pour Mines Nantes [budget Mines Nantes : 222 k€]
- Coordinateur : Jean-Bernard Stéfani (Sardes)
- Partenaires : Bull, France Telecom R&D, INRIA Rhône-Alpes (Sardes), IRIT-ENSEEIHT, Scalagent, Mines Nantes (Obasco)
- Thème : plate-forme répartie sous administration autonomique
- Objectif : développer une plate-forme logicielle pour la construction de systèmes informatiques répartis sous administration autonomique, et son application à deux domaines particuliers : (i) l'administration de serveurs d'applications J2EE de grande taille (serveurs en grappe), (ii) l'administration d'un bus d'information d'entreprise. Les résultats du projet avaient vocation à venir consolider la base de code OW2 (www.ow2.org).
- Contribution : consolidation logiciel FScript & WildCAT, thèse Marc Léger

RNTL ARCAD (2000-2004)

- Rôle : coordinateur pour Mines Nantes [budget Mines Nantes : 107 k€]
- Coordinateur : Michel Riveill (ESSI)

- Partenaires : France Telecom R&D, INRIA Rhône-Alpes (Sardes), INRIA Sophia-Antipolis (Oasis), laboratoire I3S (ESSI), Mines Nantes (Obasco)
- Thème : adaptation dynamique de composants répartis
- Objectif : définir une plate-forme logicielle extensible adaptée à l'exécution d'une application construite par assemblage de composants répartis. Ce projet a donné lieu à plusieurs expérimentations avec le serveur d'applications JOnAS et a permis l'émergence du modèle de composants Fractal (fractal.objectweb.org).
- Contribution : logiciel FScript & WildCAT , thèse Pierre-Charles David

B. Collaboration industrielle directe

1. Sigma Informatique

Collaboration à travers une thèse co-encadrée :

- Thèse de **Simon Dupont** (2012-2015), co-encadrant : Steven Morvan

  Thème : Eco-élasticité du logiciel pour le Cloud

  Objectif : proposer un ensemble de patrons de conception, un langage dédié pour gérer finement l'élasticité du Cloud qui adresse à la fois les applications et l'infrastructure. Une attention particulière est donnée sur des patrons permettant de limiter l'empreinte énergétique [19, 23].

2. Orange Labs (France Telecom R&D, équipe MAPS/AMS)

Collaboration à travers deux thèses co-encadrées :

- Thèse de **Marc Léger** (2006-2009), co-encadrant : Thierry Coupaye

  Thème : reconfiguration fiable de composants et auto-réparation

  Objectif : fournir une infrastructure pour réaliser des reconfigurations fiables d'architectures de composants Fractal. Les reconfigurations proposées sont des reconfigurations transactionnelles et suivent le modèle ACID [11, 31, 34, 57].

- Thèse de **Fabien Baligand** (2005-2008), co-encadrant : Nicolas Rivierre

  Thème : qualité de service dans les orchestrations de Web services

  Objectif : produire à un langage dédié et une plate-forme permettant de gérer l'adaptation de la qualité de service dans les orchestrations de Web services aussi bien en phase de déploiement qu'à l'exécution [33, 35, 58].

C. Logiciels

CSLA (Cloud Service Level Agreement)

Issu de la thèse de Yousri Kouki (2013), CSLA est un langage dédié à la définition de contrat de service pour les architectures en nuage [8, 23, 25, 29, 42]. Il adresse finement les violations SLA via des dégradations fonctionnelles ou de qualité de service et propose un modèle de pénalité avancé.

Diffusion/impact

CSLA (http://www.emn.fr/z-info/csla) a été utilisé dans le projet FSN OpenClouware et le projet CominLabs EPOC.

FPath/FScript : langages de navigation et de reconfiguration d'architectures logicielles

Issus de la thèse de Pierre-Charles David (2005), FPath et FScript sont des langages dédiés (au sens *Domain Specific Language*) pour les architectures à base de composants Fractal [11]. FPath est un langage dédié à la navigation dans les architectures Fractal (introspection) alors que FScript est un langage dédié à la reconfiguration dynamique d'architectures Fractal (intercession). En plus de leur syntaxe concise et dédiée à leur domaine, FPath et FScript garantissent par construction un certain nombre de propriétés comme la terminaison (requête s'exécutant en temps borné dans FPath, pas de boucle infinie dans FScript).

Diffusion/impact

FPath et FScript sont diffusés sous licence GPL par le consortium OW2 http://fractal.ow2.org/fscript. FPath et FScript ont été utilisés par les équipes INRIA Adam, Sardes et Triskell ainsi que par France Telecom R&D et l'INT EVRY. FScript a influencé la conception de FraSCAti Script (frascati.ow2.org), une extension SCA (Service Component Architecture) de FScript développé à l'INRIA Lille- Nord Europe (équipe Adam) et celle de GCM-script, une extension pour GCM (Grid Component Model) de FScript développé à l'INRIA Sophia Antipolis (équipe Oasis).

WildCAT : un framework pour le monitoring

Issu également de la thèse de Pierre-Charles David (2005), WildCAT est un outil et un framework facilitant la construction d'applications sensibles au contexte (*context-aware*) [60]. Grâce à sa capacité de « monter » des systèmes hétérogènes de monitoring et de présenter de manière unifiée les données à observer, WildCAT peut être vu comme un intergiciel pour le monitoring.

Diffusion/impact

WildCAT est diffusé sous licence GPL par le consortium OW2 (http://wildcat.ow2.org). WildCAT – comme FPath/FScript – a été utilisé dans l'ADT Galaxy et est régulièrement cité –plus de 120 citations – par une communauté qui est devenu progressivement internationale.

IV. Activités de formation

A. Responsabilité pédagogique IMT Atlantique (Mines Nantes)

- Responsable pédagogique de la formation d'ingénieurs par apprentissage **FIL** (Formation Ingénierie Logicielle) en partenariat à l'ITII Pays de Loire (depuis 2011)
    - Domaine cible : logiciels et services
    - Thèmes : génie logiciel, langages de programmation, systèmes distribués
    - Volume horaire : 1800 h sur 3 ans
    - Flux étudiants : 24 élèves / année
    - + 100 intervenants
    - + 50 entreprises partenaires
    - +100 d'alumnis
- Responsable de l'option **GSI** (Génie des Systèmes Informatiques) (2002-2007)
    - Thèmes : génie logiciel, systèmes distribués
    - Niveau Bac+5
    - Volume horaire : 360h sur 1 an
    - Intervenants : 2/3 réalisé par experts du monde de l'entreprise ou de l'INRIA
    - Flux étudiants : plus d'une centaine d'alumni

B. Enseignement

1. Enseignement IMT Atlantique (Mines Nantes)

Maitre-assistant depuis 1998

- **Cours** et TD/TP (≈180 eq.TD/an) :
    - Année 1 : Algorithmique, Structures de données, Langages à objets (Smalltalk, C++ puis Java), Programmation Modulaire, C**omposants logiciels**
    - Année 2 : Base de données, Génie logiciel (UML)
    - Année 3 : **Middleware (CORBA puis Java EE)**
- Tuteurs de stages de projet de fin d'études (PFE)
- Tuteur de plusieurs projets d'élèves (sous statut étudiant et apprenti)
- Participation aux instances de formation : comités pédagogiques, conseil de profs, jurys des études

2. Enseignement de troisième cycle

- Master 2 ALMA : « Reconfiguration dynamique d'architectures logicielles » (6-9h/an depuis 2008)
- Master de recherche (DEA Informatique (1999-2003), puis Master 2 ALD (2004-2007)) : « Intergiciels réflexifs » (3h/an)
- Master européen EMOOSE : course « *Reflective languages* » (1999-2001), encadrement de nombreux projets (1998-2009)

3. Autres enseignements

- ENSEIRB : « Middleware (CORBA, J2EE) » : cours /TP (10h/an) en 2003, 2004
- IBM Formation : « VisualAge for Smalltalk » : cours/TP (12h/an) en 1995, 1996

C. Autres responsabilités

1. Responsabilités pédagogiques

- Président des jurys des soutenances des stages ingénieurs (depuis 2002)
    - GSI de 2002 à 2007
    - FIL depuis 2014 (date de sortie de la première promotion)
- Responsable des Unités de Valeurs (UVs) (depuis 2002)
    - UVs « Middleware » (90h), « Informatique Nomade » (90h) en GSI de 2002 à 2007
    - UVs « Qualité logicielle et méthodes agiles » (45h), « Projet et outils de développement » (45h) en FIL A1 depuis 2011
    - UVs « Infrastructures d'intégration » (45h), « Projet Innover avec les startups » (45h) en FIL A2 depuis 2012
    - UVs « Informatique mobile » (45h), « Développement pour le Cloud » (45h) en FIL A3 depuis 2013

2. Responsabilités collectives

- Responsable du dossier de renouvellement de la formation FIL pour 6 ans auprès de la Commission des Titres Ingénieurs (CTI) (hiver 2013-2014)
- Co-responsable du dossier de création de la formation d'ingénieurs par apprentissage FIL auprès de la Commission des Titres Ingénieurs (CTI) (hiver 2010-2011)
- Responsable de l'actualisation du programme de l'option GSI (hiver 2004-2005)
- Participation au Comité de Pilotage du Projet Compétences de l'Ecole des Mines de Nantes (2003-2005) dont l'objectif était de réfléchir à la mise en place de l'approche compétence au cœur du dispositif de formation
- Co-pilote du projet « Portail Métiers Anciens » de l'Ecole des Mines de Nantes (2007) dont l'objectif était de réaliser un canal privilégié de tous les flux d'informations entre l'école et ses anciens élèves.

V. Annexe : liste des publications

1. Livres ou chapitres de livres

1. Yousri Kouki, Frederico Alvares de Oliveira Jr. and Thomas Ledoux. *Cloud Capacity Planning and Management*. In San Murugesan and Irena Bojanova (Ed.), Encyclopedia of Cloud Computing, Wiley-IEEE, July 2016
2. Frederico Alvares de Oliveira Jr., Adrien Lèbre, Thomas Ledoux and Jean-Marc Menaud. *Self-management of applications and systems to optimize energy in data centers*. In Ivona Brandic, Massimo Villari and Francesco Tusa (Ed.), Achieving Federated and Self-Manageable Cloud Infrastructures: Theory and Practice, IGI Global , May 2012
3. Noury Bouraqadi and Thomas Ledoux. *Supporting AOP Using Reflection*. In book "Aspect-Oriented Software Development", p. 261-282, M. Aksit, S. Clarke, T. Elrad, R. E. Filman editors, Addison-Wesley, 2005.
4. Walter Cazzola, Shigeru Chiba and Thomas Ledoux. *Reflection and Meta-Level Architectures: State of the Art and Future Trends*. ECOOP'2000 - Workshop "Reflection and Meta-level Architectures", Workshop Reader, Jacques Malenfant, Sabine Moisan, Ana Moreira editors, p. 1-15, Springer-Verlag, LNCS 1964, 2000.

2. Revues internationales avec comité de lecture

5. Zakarea Al-Shara, Frederico Alvares, Hugo Bruneliere, Jonathan Lejeune, Charles Prud'Homme, Thomas Ledoux. *CoMe4ACloud: An End-to-end Framework for Autonomic Cloud Systems*. In Future Generation Computer Systems (FGCS), 2018
6. Md Sabbir Hasan, Frederico Alvares de Oliveira , Thomas Ledoux and Jean-Louis Pazat. *Investigating Energy consumption and Performance trade-off for Interactive Cloud Application*. In IEEE Transactions on Sustainable Energy, vol. 2, no. 2, pp.113-126, April-June 1 2017 (available online 13 june 2017).
7. Md Sabbir Hasan, Yousri Kouki, Thomas Ledoux and Jean Louis Pazat. *Exploiting Renewable sources: when Green SLA becomes a possible reality in Cloud computing*. In IEEE Transactions on Cloud Computing, Vol. 5, No 2, pp. 249-262, April-June 2017 (available online 22 july 2015).
8. N. Beldiceanu, B. Dumas Feris, P. Gravey, S. Hasan, C. Jard, T. Ledoux, Y. Li, D. Lime, G. Madi-Wamba, J-M. Menaud, P. Morel, M. Morvan, M-L. Moulinard, A-C. Orgerie, J-L. Pazat, O. Roux, A. Sharaiha. *Towards energy-proportional clouds partially powered by renewable energy*. In Computing, Springer Verlag, January 2017, Volume 99, Issue 1, pp 3–22.
9. Damián Serrano, Sara Bouchenak, Yousri Kouki, Frederico Alvares de Oliveira Jr., Thomas Ledoux, Jonathan Lejeune, Julien Sopena, Luciana Arantes, Pierre Sens. *SLA guarantees for Cloud Services*. In Future Generation Computer Systems (FGCS), Pages 233-246, Volume 54, January 2016.
10. Yousri Kouki, Thomas Ledoux. *RightCapacity: SLA-driven Cross-Layer Cloud Elasticity Management*. In International Journal of Next-Generation Computing (IJNGC), Vol 4, No 3, November, 2013.
11. Frederico Alvares de Oliveira Jr. and Thomas Ledoux. *Self-management of cloud applications and infrastructure for energy optimization*. SIGOPS Operating Systems Review, 46(2), 2012.
12. Pierre-Charles David, Thomas Ledoux, Marc Léger and Thierry Coupaye. *FPath and FScript: Language support for navigation and reliable reconfiguration of Fractal architectures*. In Special issue on Component-based architecture: the Fractal Initiative. Annals of Telecommunications, Volume 64, n°1/2, Springer, January/February 2009.

3. Revues nationales avec comité de lecture

13. Pierre-Charles David et Thomas Ledoux. *Une approche par aspects pour le développement de composants Fractal adaptatifs*. RSTI - L'Objet. Volume 12 - n°2-3/2006, Hermès Sciences/Lavoisier, 2006.
14. Pierre Cointe, Jacques Noyé, Rémi Douence, Thomas Ledoux, Jean-Marc Menaud, Gilles Muller and Mario Südholt. *Programmation post-objets : des langages d'aspects aux langages de composants*. RSTI L'Objet. Volume 10 - n°4, p.119-143, Hermès, 2004.

15. Noury M. N. Bouraqadi-Saâdani et Thomas Ledoux. *Le point sur la programmation par aspects*. Technique et Science Informatiques. Volume 20 - n°4/2001, Hermès, 2001.

4. Conférences internationales avec comité de lecture

16. Hugo Bruneliere, Zakarea Al-shara, Frederico Alvares, Jonathan Lejeune and Thomas Ledoux. *A Model-based Architecture for Autonomic and Heterogeneous Cloud Systems*. In the 8h International Conference on Cloud Computing and Services Science (CLOSER 2018), Funchal, Madeira, Portugal, 19-21 March 2018.
17. Md Sabbir Hasan, Frederico Alvares de Oliveira and Thomas Ledoux. *GPaaScaler: Green Energy aware Platform Scaler for Interactive Cloud Application*. In the 10th IEEE/ACM International Conference on Utility and Cloud Computing (UCC 2017), Austin, Texas, USA, Dec 5-8, 2017
18. Jonathan Lejeune, Frederico Alvares and Thomas Ledoux. *Towards a generic autonomic model to manage Cloud Services*. In the 7th International Conference on Cloud Computing and Services Science (CLOSER 2017), Porto, Portugal, 24-26 April 2017
19. Simon Dupont, Salma Bouri, Frederico Alvares de Oliveira and Thomas Ledoux. *ElaScript: a DSL for Coding Elasticity in Cloud Computing*. In 32nd ACM Symposium on Applied Computing - Track on Cloud Computing (SAC 2017 - CC), Marrakesh, Morocco, April 3-7, 2017.
20. Md Sabbir Hasan, Frederico Alvares de Oliveira, Thomas Ledoux and Jean-Louis Pazat. *Enabling Green Energy awareness in Interactive Cloud Application*. In 8th IEEE International Conference on Cloud Computing Technology and Science (CloudCom 2016), Luxembourg, December 12-15, 2016.
21. Simon Dupont, Jonathan Lejeune, Frederico Alvares and Thomas Ledoux. *Experimental Analysis on Autonomic Strategies for Cloud Elasticity*. In IEEE International Conference on Cloud and Autonomic Computing (ICCAC 2015), Cambridge, Massachusetts, USA, September 21-25, 2015.
22. Yousri Kouki, Md Sabbir Hasan and Thomas Ledoux. How resources scalability/termination can be taken place Economically? In 11th World Congress on Services (SERVICES), New York, USA, June 27-July 2, 2015.
23. Nicolas Beldiceanu, Barbara Dumas Feris, Philippe Gravey, Sabbir Hasan, Claude Jard, Thomas Ledoux, Yunbo Li, Didier Lime, Gilles Madi-Wamba, Jean-Marc Menaud, Pascal Morel, Michel Morvan, Marie-Laure Moulinard, Anne-Cécile Orgerie, Jean-Louis Pazat, Olivier Roux and Ammar Sharaiha.*The EPOC project: Energy Proportional and Opportunistic Computing system*. In the 4th International Conference on Smart Cities and Green ICT Systems (SMARTGREENS), Lisbon, Portugal, May 20-22, 2015.
24. Md Sabbir Hasan, Yousri Kouki, Thomas Ledoux and Jean Louis Pazat. *Cloud Energy Broker: Towards SLA-driven Green Energy Planning for IaaS Providers*. In the 16th IEEE Internatonal Conference on High Performance Computing and Communications (HPCC), Paris, France, August 20-22, 2014.
25. Yousri Kouki, Frederico Alvares de Oliveira Jr., Simon Dupont and Thomas Ledoux. *A Language Support for Cloud Elasticity Management*. In IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), Chicago, USA, May 26-29, 2014.
26. Frederico Alvares de Oliveira Jr., Thomas Ledoux and Remi Sharrock. *A framework for the coordination of multiple autonomic managers in cloud environments*. In 7th IEEE International Conferences on Self-Adaptive and Self-Organizing Systems (SASO), Philadelphia, USA, September 9-13, 2013.
27. Damian Serrano, Sara Bouchenak, Yousri Kouki, Thomas Ledoux, Jonathan Lejeune, Luciana Arantes, Julien Sopena, and Pierre Sens. *Towards QoS-Oriented SLA Guarantees for Online Cloud Services*. In IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid 2013), Delft, the Netherlands, May 13-16, 2013.
28. Yousri Kouki, Thomas Ledoux. *SCAling : SLA-driven Cloud Auto-scaling* (poster paper). In the 28thACM Symposium on Applied Computing (SAC 2013) - Track on Cloud Computing, Coimbra, Portugal, March 18-22, 2013.
29. Yousri Kouki and Thomas Ledoux. *SLA-driven Capacity Planning for Cloud applications*. In 4th IEEE International Conference on Cloud Computing Technology and Science (CloudCom 2012), Taipei, Taiwan, December 3-6, 2012.
30. Frederico Alvares de Oliveira Jr., Remi Sharrock and Thomas Ledoux. *Synchronization of Multiple Autonomic Control Loops: Application to Cloud Computing*. In the 14th International Conference on Coordination Models and Languages (Coordination 2012), Stockholm, Sweden, June 14-15, 2012.

31. Yousri Kouki and Thomas Ledoux. *CSLA : a Language for Improving Cloud SLA Management*. In the 2nd International Conference on Cloud Computing and Services Science (CLOSER 2012), Porto, Portugal, April 18-21, 2012.
32. Yousri Kouki, Thomas Ledoux and Rémi Sharrock. *Cross-layer SLA Selection for Cloud services*. In IEEE First Symposium on Network Cloud Computing and Applications (NCCA), Toulouse, France, November 21-23, 2011.
33. Marc Léger, Thomas Ledoux and Thierry Coupaye. *Reliable Dynamic Reconfigurations in a Reflective Component Model*. In 13th International Symposium on Component Based Software Engineering (CBSE 2010), Springer-Verlag, LNCS Vol. 6092, Prague, Czech Republic, June 2010.
34. Brice Morin, Thomas Ledoux, Mahmoud Ben Hassine, Franck Chauvel, Olivier Barais and Jean-Marc Jezequel. *Unifying Runtime Adaptation and Design Evolution*. In IEEE 9th International Conference on Computer and Information Technology (CIT'09), IEEE Computer Society, Xiamen, China, October 2009.
35. Fabien Baligand, Nicolas Rivierre and Thomas Ledoux. *QoS Policies for Business Processes in Service Oriented Architectures*. In 6th International Conference on Service Oriented Computing (ICSOC 2008), Springer-Verlag, LNCS Vol. 5354, Sydney, Australia, December 2008.
36. Pierre-Charles David, Marc Léger, Hervé Grall, Thomas Ledoux and Thierry Coupaye. *A Multi-Stage Approach for Reliable Dynamic Reconfigurations of Component-Based Systems*. In Distributed Applications and Interoperable Systems (DAIS'08), Springer-Verlag, LNCS Vol. 5053, Oslo, Norway, June 2008.
37. Fabien Baligand, Nicolas Rivierre and Thomas Ledoux. *A Declarative Approach for QoS-Aware Web Service Compositions*. In 5th International Conference on Service Oriented Computing (ICSOC 2007), Springer-Verlag, LNCS Vol. 4749, Vienna, Austria, September 2007.
38. Pierre-Charles David and Thomas Ledoux. *An Aspect-Oriented Approach for Developing Self-Adapting Fractal Components*. In Software composition (SC 2006), Satellite event of ETAPS 2006, Springer-Verlag, LNCS Vol. 4089, Vienna, Austria, March 2006.
39. Pierre-Charles David and Thomas Ledoux. *Towards a Framework for Self-Adaptive Component-Based Applications*. In Distributed Applications and Interoperable Systems (DAIS'03), Springer-Verlag, LNCS Vol. 2893, Paris, France, November 2003.
40. Pierre-Charles David and Thomas Ledoux. *An Infrastructure for Adaptable Middleware*. In DOA'02, Springer-Verlag, LNCS Vol. 2519, Irvine, California, USA, October 2002.
41. Thomas Ledoux. *OpenCorba: a Reflective Open Broker*. In Proceedings of the Second International Conference on Meta-Level Architectures and Reflection (Reflection'99), Springer-Verlag, LNCS Vol. 1616, Saint-Malo, France, July 1999.
42. Noury M. N. Bouraqadi-Saâdani, Thomas Ledoux and Fred Rivard. *Safe Metaclass Programming*. In Proceedings of the 13th ACM SIGPLAN conference on Object-Oriented Programming, Systems, Languages and Applications (OOPSLA'98), ACM Sigplan Notices, Vancouver, Canada, October 1998.
43. Thomas Ledoux and Pierre Cointe. *Explicit Metaclasses as a Tool for Improving the Design of Class Libraries*. In Proceedings of the Second JSSST International Symposium on Object Technologies for Advanced Software (ISOTAS'96), Springer-Verlag, LNCS Vol. 1049, Kanazawa, Japan, March 1996.

5. Conférences nationales avec comité de lecture

44. Yousri Kouki, Thomas Ledoux, Damian Serrano, Sara Bouchenak, Jonathan Lejeune, Luciana Arantes, Julien Sopena, and Pierre Sens. *SLA et qualité de service pour le Cloud Computing*. In Conférence d'informatique en Parallélisme, Architecture et Système ComPAS 2013, Grenoble, France, January 15-18, 2013.
45. Jean-Marc Menaud, Adrien Lèbre, Thomas Ledoux, Jacques Noyé, Pierre Cointe, Rémi Douence and Mario Südholt. *Vers une réification de l'énergie dans le domaine du logiciel*. In Journées du GDR Génie de la Programmation et du Logiciel (GDR GPL 2010), Pau, mars 2010.
46. Fabien Baligand, Thomas Ledoux, Pierre Combes. *Une Approche pour Garantir la Qualité de Service dans les Orchestrations de Services Web*. In NOTERE 2007, Marrakech, Maroc, Juin 2007.
47. Marc Léger, Thierry Coupaye et Thomas Ledoux. *Contrôle dynamique de l'intégrité des communications dans les architectures à composants*. In LMO 2006, Hermès Sciences/Lavoisier, Nîmes, France, Mars 2006.

48. Pierre-Charles David et Thomas Ledoux. *Une approche par aspects pour le développement de composants Fractal adaptatifs*. In JFDLPA 2005, Lille, France, Septembre 2005.
49. Pierre-Charles David et Thomas Ledoux. *Pour un aspect d'adaptation dans le développement d'applications à base de composants*. In Actes Journée de l'AS 150, Systèmes répartis et réseaux adaptatifs au contexte. Paris, Avril 2004.
50. François Sarradin et Thomas Ledoux. *Adaptabilité dynamique de la sémantique de communication dans Jonathan*. In LMO 2001, Hermès Science, L'objet-7/2001, Le Croisic, France, Janvier 2001.
51. P. Cointe et T. Ledoux. *Pour des architectures logicielles ouvertes et adaptables. La réflexion : pourquoi et comment ?* In Séminaire Systèmes distribués et Connaissances, Sophia-Antipolis, France, Novembre 2000.
52. Thomas Ledoux. *OpenCorba : un bus logiciel réflexif adaptable*. In LMO'99, Hermès Science, Villefranche sur Mer, France, Janvier 1999.
53. M.N. Bouraqadi-Saâdani, F. Rivard et T. Ledoux. *Composition de métaclasses*. In JFLA'98, INRIA collection, Côme, Italy, Février 1998.
54. Thomas Ledoux. *Adaptabilité dynamique des aspects pour la construction d'applications réparties* ouvertes. In NOTERE'98, Montréal, Canada, Octobre 1998.
55. Thomas Ledoux et Pierre Cointe. *Les métaclasses explicites comme outil pour améliorer la conception des bibliothèques de classes*. In Actes GDR'95, Grenoble, France, Novembre 1995

6. Ateliers internationaux avec comité de lecture

56. Frederico G. Alvares de Oliveira Jr. and Thomas Ledoux. *Self-management of applications QoS for energy optimization in datacenters*. In 2nd International Workshop on Green Computing Middleware (GCM 2011), held at the ACM/IFIP/USENIX 12th International Middleware Conference, Lisboa, Portugal, December 12, 2011.
57. Frederico G. Alvares de Oliveira Jr. and Thomas Ledoux. *Self-optimisation of the energy footprint in Service-Oriented Architectures*. In 1st International Workshop on Green Computing Middleware (GCM 2010), held at the ACM/IFIP/USENIX 11th International Middleware Conference, Bangalore, India, November 29, 2010.
58. Mayleen Lacouture, Hervé Grall and Thomas Ledoux. *CREOLE: a Universal Language for Creating, Requesting, Updating and Deleting Resources*. In 9th International Workshop on the Foundations of Coordination Languages and Software Architectures (FOCLASA 2010), A Satellite Workshop of CONCUR 2010, Paris (France), September 4, 2010.
59. Marc Léger, Thomas Ledoux, Thierry Coupaye. *Reliable Dynamic Reconfigurations in the Fractal Component Model*. In Proceedings of the 6th international workshop on Adaptive and Reflective Middleware (ARM'07), held at the ACM/IFIP/USENIX International Middleware Conference, Newport Beach, California, November 2007.
60. Fabien Baligand, Didier Le Botlan, Thomas Ledoux, Pierre Combes. *A Language for Quality of Service Requirements Specification in Web Services Orchestrations*. In WESOA'06 (In conjunction with ICSOC'06), Springer-Verlag, LNCS Vol. 4652, Chicago, USA, December 2006.
61. Pierre-Charles David and Thomas Ledoux. *Safe Dynamic Reconfigurations of Fractal Architectures with FScript*. In ECOOP'06 on Fractal CBSE Workshop, Nantes, France, July 2006.
62. Pierre-Charles David and Thomas Ledoux. *WildCAT: a generic framework for context-aware applications*. In Proceedings of the 3rd international workshop on Middleware for pervasive and ad-hoc computing (MPAC 2005), ACM Digital Library, Grenoble, France, November 2005.
63. Pierre-Charles David and Thomas Ledoux. *Dynamic Adaptation of Non-Functional Concerns*. In ECOOP'02 Workshop on "Unanticipated Software Engineering", Malaga, Spain, June 2002.
64. Zahi Jarir, Pierre-Charles David and Thomas Ledoux. *Dynamic Adaptability of Services in Enterprise JavaBeans Architecture*. In ECOOP'02 Workshop on "Component-Oriented Programming", Malaga, Spain, June 2002.
65. Pierre-Charles David, Thomas Ledoux and Noury M. N. Bouraqadi-Saâdani. *Two-step Weaving with Reflection using AspectJ*. In OOPSLA'01 Workshop on "Advanced Separation of Concerns in Object-Oriented Systems", Tampa Bay, USA, October 2001.

66. Noury M.N.Bouraqadi-Saâdani and Thomas Ledoux. *How to weave?* In ECOOP'01 Workshop on "Advanced Separation of Concerns", Budapest, Hungary, June 2001.
67. Thomas Ledoux and Noury M.N.Bouraqadi-Saâdani. *Adaptability in Mobile Agent Systems using Reflection*. In Middleware'2000 Workshop on "Reflective Middleware", New-York, USA, April 2000.
68. Thomas Ledoux. *Implementing Proxy Objects in a Reflective ORB*. In ECOOP'97 Workshop on "CORBA: Implementation, Use and Evaluation", Jyvaskyla, Finland, June 1997.
69. Noury Bouraqadi-Saâdani, Thomas Ledoux, Fred Rivard and Pierre Cointe. *Providing Explicit Metaclasses in Smalltalk*. In OOPSLA'96 Workshop on "Extending the Smalltalk Language", San Jose, California, October 1996.
70. Noury Bouraqadi-Saâdani, Thomas Ledoux and Fred Rivard. *Metaclass Composability*. In ECOOP'96 Workshop on "Composability Issues in OO", Linz, Austria, July 1996.
71. Philippe Mulet, Thomas Ledoux, Denis Barbaron, Fred Rivard and Pierre Cointe. *Importing SOM Libraries into Classtalk*. In OOPSLA'94 Workshop on "Multi-Language Object Models", Portland, Oregon, October 1994.

Reconfiguration dynamique d'architectures logicielles : des métaclasses aux « nuages verts »

Thomas Ledoux

Résumé. Cette habilitation à diriger des recherches présente une synthèse de mes travaux ayant trait à la reconfiguration dynamique d'architectures logicielles.

En première partie, je propose une analyse de l'adaptation dynamique du logiciel avec la volonté de poser les concepts de base et d'identifier les verrous à lever. Je motive l'intérêt de l'adaptation dynamique et modélise son processus de développement. Je présente un nombre de caractéristiques indispensables à l'adaptation dont la réification et la modularité. Je traite alors l'adaptation dynamique du logiciel comme un objet de première classe introduisant un découplage spatio-temporel entre code métier et logique d'adaptation. L'adaptation dynamique ne pouvant se faire au détriment de la qualité du logiciel et de son intégrité, j'énumère un nombre de propriétés indispensables à respecter, de pistes à étudier pour réaliser à l'exécution ce processus d'adaptation du logiciel.

En deuxième partie, je présente ma contribution à la mise en œuvre de l'adaptation dynamique pour les logiciels à base de composants. Dans un premier temps, je propose un patron d'auto-administration pour les architectures à composants, des langages dédiés pour la navigation et la reconfiguration dans les architectures logicielles Fractal, un framework pour faciliter la création d'applications sensibles au contexte. Dans un second temps, je développe une approche multi-étape pour des reconfigurations dynamiques fiables, qui contient à la fois une partie prévention de fautes basée sur une analyse statique et une partie tolérance aux fautes assurée par un moniteur transactionnel. Cette approche s'appuie sur une spécification des (re)configurations du modèle Fractal basée sur une logique du premier ordre.

En troisième partie, je montre que la reconfiguration dynamique dans les « architectures en nuages » (*Cloud computing*) peut apporter une réponse à un enjeu sociétal important, à savoir les transitions numérique et énergétique. Contrairement aux travaux actuels qui visent à améliorer l'efficacité énergétique des centres de données en proposant des solutions dans les couches basses du Cloud, je préconise une approche d'éco-élasticité logicielle sur ses couches hautes. En m'inspirant à la fois du concept d'innovation frugale (*Jugaad*) et du mécanisme d'effacement de la consommation électrique, un certain nombre d'artefacts originaux comme le Cloud SLA, l'effacement de la consommation du logiciel, la virtualisation de l'énergie ou encore le SaaS *green energy-aware*, sont suggérés pour diminuer l'empreinte carbone des architectures en nuages.

En conclusion, je dresse un bilan final de ma contribution et présente mes perspectives de recherche centrées d'abord sur une généralisation de l'approche précédente à l'aide de méta-modèles, puis sur la conception de micro-services « verts ».

Mots-clés : architectures logicielles, composants logiciels, reconfiguration dynamique, informatique en nuage, informatique durable