Une approche autonome pour la gestion logicielle des espaces intelligents

par

Charles Gouin-Vallerand

Thèse en cotutelle présentée au Département d'Informatique en vue de l'obtention du grade de docteur de Telecom SudParis et de l'Université Pierre et Marie Curie, et docteur ès sciences (Ph.D.) de l'Université de Sherbrooke

FACULTÉ DES SCIENCES
UNIVERSITÉ DE SHERBROOKE

Sherbrooke, Québec, Canada, juin 2011

Library and Archives Canada

Published Heritage Branch

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque et Archives Canada

Direction du Patrimoine de l'édition

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence
ISBN: 978-0-494-83333-9

Our file Notre référence
ISBN: 978-0-494-83333-9

NOTICE:

The author has granted a non-exclusive license allowing Library and Archives Canada to reproduce, publish, archive, preserve, conserve, communicate to the public by telecommunication or on the Internet, loan, distrbute and sell theses worldwide, for commercial or non-commercial purposes, in microform, paper, electronic and/or any other formats.

The author retains copyright ownership and moral rights in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

AVIS:

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque et Archives Canada de reproduire, publier, archiver, sauvegarder, conserver, transmettre au public par télécommunication ou par l'Internet, prêter, distribuer et vendre des thèses partout dans le monde, à des fins commerciales ou autres, sur support microforme, papier, électronique et/ou autres formats.

L'auteur conserve la propriété du droit d'auteur et des droits moraux qui protege cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

In compliance with the Canadian Privacy Act some supporting forms may have been removed from this thesis.

While these forms may be included in the document page count, their removal does not represent any loss of content from the thesis.

Conformément à la loi canadienne sur la protection de la vie privée, quelques formulaires secondaires ont été enlevés de cette thèse.

Bien que ces formulaires aient inclus dans la pagination, il n'y aura aucun contenu manquant.

Canada

Le 29 juillet 2011

le jury a accepté la thèse de Monsieur Charles Gouin-Vallerand
dans sa version finale.

Membres du jury

Professeur Sylvain Giroux
Directeur de recherche
Département d'informatique
Université de Sherbrooke

Professeur Mounir Mokhtari
Codirecteur de recherche
Département Réseau et Services de Télécommunications
Télécom SudParis et CNRS-IPAL

Professeur Bessam Abdulrazak
Codirecteur de recherche
Département d'informatique
Université de Sherbrooke

Professeur Jean-François Perrot
Membre interne
Laboratoire d'Informatique de Paris 6 - LIP6
Université Pierre et Marie Curie

Professeur Daniel Racoceanu
Membre externe
CNRS-IPAL
Université National de Singapour

Professeur Bruno Bouchard
Membre externe
Département d'informatique
Université du Québec à Chicoutimi

Professeur Gabriel Girard
Président rapporteur
Département d'informatique
Université de Sherbrooke

"On va s'aimer encore, après nos bons coups, après nos déboires et de plus en plus fort. On va s'aimer encore au bout de nos doutes au bout de la route [...]" On va s'aimer encore - Vincent Vallières

À mes inspirations Sarah, Alice et Éloi.

Sommaire

Depuis une vingtaine d'années, les développements dans les technologies de l'information ont fait évoluer les paradigmes de l'informatique. L'arrivée d'approches telles que l'informatique diffuse et l'intelligence ambiante ont fait émerger de nouvelles technologies permettant d'améliorer la qualité des interactions avec les systèmes informatisés. Entre autres, l'application de l'informatique diffuse et de l'intelligence ambiante à des environnements tels que des habitats, les espaces intelligents, offre des milieux où une assistance contextualisée est offerte aux utilisateurs dans la réalisation de leurs activités quotidiennes. Toutefois, la démocratisation de l'informatique diffuse et la mise en place des espaces intelligents rencontrent un bon nombre de problèmes. Le nombre important de composantes matérielles et logicielles, les dépendances entre celles-ci et leurs natures hétérogènes contribuent à la complexité de déploiement et de gestion de ces milieux, entraînant des coûts élevés.

Cette thèse vise à contribuer à la gestion logicielle des espaces intelligents par la réduction de la complexité des tâches de gestion. Notre proposition consiste en une approche autonome de la gestion logicielle, fondée sur l'approche de l'informatique diffuse autonome. L'objectif est de fournir un ensemble de fonctionnalités et de mécanismes permettant de rendre autonome la majeure partie des tâches de gestion des logiciels déployés dans des espaces intelligents. Dans le cadre de ce travail, nous proposons une solution permettant l'organisation autonome des logiciels des espaces intelligents. Ainsi, cette solution utilise les informations contextuelles des milieux afin de déterminer quelle répartition des logiciels parmi les appareils des milieux correspond le mieux aux besoins des applications, aux caractéristiques propres des environnements et, non le moindre, aux modalités et préférences d'interaction des utilisateurs de ces milieux. La solution proposée a été implémentée et évaluée à l'aide d'une série de tests et de mises en situation d'organisation logicielle.

Les contributions de ce travail à l'état de l'art de la gestion des espaces intelligents sont multiples avec comme principales innovations la présentation d'une vision de l'informatique diffuse autonome, l'implémentation d'un intergiciel d'organisation logicielle autonome basé

sur une sensibilité au contexte macroscopique et microscopique et l'intégration des modalités d'interaction des utilisateurs dans le raisonnement portant sur l'organisation des logiciels.

Enfin, ce travail de thèse se déroule dans le contexte des recherches du laboratoire DOMUS de l'Université de Sherbrooke, Canada, et du laboratoire Handicom de Telecom SudParis, France. Ces deux laboratoires travaillent à la conception de solutions permettant d'améliorer la qualité de vie et l'autonomie de personnes dépendantes, atteinte par exemple de troubles cognitifs ou de handicaps physiques.

Mots-Clés : Gestion logicielle, Informatique diffuse autonome, Auto-organisation, Espace Intelligent, Informatique diffuse, Intelligence Ambiante

Remerciements

Je voudrais remercier mes directeurs de thèse Sylvain Giroux, Mounir Mokhtari et Bessam Abdulrazak pour leur support à mon travail de recherche. Leurs conseils et commentaires ont été de précieuses informations dans la conduite de mes études doctorales. J'aimerais également remercier le Fonds québécois de recherche sur la nature et les technologies pour leur support financier. J'aimerais également remercier les membres du jury de cette thèse : Bruno Bouchard, Daniel Racoceanu, Gabriel Girard et Jean-François Perrot pour leur lecture méticuleuse et leurs questions justes et intéressantes.

Une distinction toute spéciale à mes collègues des laboratoires DOMUS et Handicom, qui ont aidé sur plusieurs aspects, à l'avancement de cette thèse. Enfin, j'aimerais remercier ma conjointe Sarah, mes enfants Alice et Éloi, ainsi que mes parents Jeannine et Claude, mes beau-parents Valois et Dorothée, pour le courage et la motivation qu'ils m'ont apportés. Sans eux, ce travail n'aurait pas été possible.

Publications et communications reliées à la thèse

Journaux internationals avec comité de lecture

- Bessam Abdulrazak, Belkacem Chikhaoui, Charles Gouin-Vallerand, Benoit Fraikin, "A standard ontology for smart spaces", International Journal of Web and Grid Services (IJWGS), Inderscience Publishers, Vol. 6, No. 3, 2010
- Charles Gouin-Vallerand, Bessam Abdulrazak, Sylvain Giroux, Mounir Mohktari, "A self-configuration middleware for smart spaces", International Journal of Smart Home, Vol. 3, No. 1, Janvier 2009
- Sylvain Giroux, Matthieu Castebrunet, Charles Gouin-Vallerand, Bessam Abdulrazak, "A Pervasive Framework for Multi-Agent Personalization in Smart Homes," Communications of SIWN, Vol. 5, Août 2008, pp. 57-61

Conférences internationales avec actes et comité de lecture

- Bessam Abdulrazak, Patrice Roy, Charles Gouin-Vallerand, Sylvain Giroux, Yacine Belala, "Macro and Micro Context-Awareness for Autonomic Pervasive Computing", 12^{th} International Conference on Information Integration and Web-based Applications Services, iiWAS2010, Paris, France, Novembre 2010.
- Charles Gouin-Vallerand, Bessam Abdulrazak, Sylvain Giroux, Mounir Mokhtari, "A software self-organization middleware for smart spaces based on fuzzy logic". 12^{th} IEEE International Conference on High Performance Computing and Communications (HPCC 2010), Melbourne, Australie, Septembre 2010
- Charles Gouin-Vallerand, Sylvain Giroux, Bessam Abdulrazak, Mounir Mohktari, "Toward a self-configuration middleware for smart spaces," In proceeding of 3^{rd} International Symposium on Smart Home, SH 2008, Sanya, China, Décembre 2008

- Charles Gouin-Vallerand, Bessam Abdulrazak, Sylvain Giroux, Mounir Mohktari, "Toward Autonomic Pervasive Computing," 10^{th} International Conference on Information Integration and Web-based Applications & Services, iiWAS 2008, Linz, Austria, Novembre 2008

Affiches et communications

- Charles Gouin-Vallerand, Sylvain Giroux, Bessam Abdulrazak, Mounir Mohktari, "Gestion des services et des logiciels dans le contexte des habitats intelligents," Journée de la recherche de l'Université de Sherbrooke, Février 2008

Table des matières

Sommaire iv

Remerciements vi

Publications et communications reliées à la thèse vii

Table des matières ix

Liste des figures xii

Liste des tableaux xiv

Introduction 1

1 La gestion logicielle dans les espaces intelligents 15

1.1 Définition de la gestion logicielle dans les espaces intelligents 15

1.2 Problématique de la gestion logicielle dans les espaces intelligents 18

1.2.1 L'hétérogénéité et la complexité des espaces intelligents 19

1.2.2 Le profil des utilisateurs/gestionnaires et la sécurité 21

1.2.3 Les méthodes de développement et de conception 22

1.2.4 Les fautes logicielles et les bris matériels 23

1.3 Méthodes et stratégies de gestion logicielle 25

1.3.1 Les méthodes *Pull* et *Push* 25

1.3.2 Les stratégies d'Orchestration et de Chorégraphie 27

1.4 Revue des solutions de gestion logicielle existantes 29

1.4.1 Travaux de recherche en gestion logicielle pour espaces intelligents et autres 30

1.4.2 Outils d'installation et de mise à jour 33

1.4.3 Grappes de calcul 34
1.4.4 Plateformes applicatives 35
1.4.5 Outils de gestion de flottes d'appareils 38
1.5 Discussion 39

2 L'informatique autonome **42**
2.1 L'informatique diffuse autonome 43
2.1.1 Travaux en informatique diffuse autonome 44
2.1.2 Notre vision de l'informatique diffuse autonome 50
2.2 Scénarios de gestion logicielle autonome des espaces intelligents 58
2.2.1 Scénario #1 : Auto-configuration logicielle dans un espace intelligent par un gestionnaire non expert 59
2.2.2 Scénario #2 : Auto-configuration logicielle dans un espace intelligent . 60
2.2.3 Scenario #3 : Auto-réparation par les composantes d'un espace intelligent 61
2.2.4 Scénario #4 : Auto-optimisation dans un espace intelligent 62
2.2.5 Scénario #5 : Auto-protection dans un espace intelligent 63

3 Projet Tyche, un intergiciel de gestion logicielle autonome pour espaces intelligents 65
3.1 Introduction 66
3.1.1 Objectifs du projet Tyche 66
3.1.2 Architecture et fonctionnalités 69
3.1.3 Implementation 73
3.2 Approche contextuelle macroscopique et microscopique 75
3.3 Description sémantique des espaces intelligents 78
3.3.1 État de l'art des descriptions sémantiques et ontologies 79
3.3.2 Description sémantique utilisée 80
3.4 Auto-organisation logicielle des espaces intelligents 85
3.4.1 Le Device Capability Quotient (DCQ) 87
3.5 Le Fuzzy Logic Organization Reasoning Engine 89
3.5.1 Introduction à la logique floue 91
3.5.2 L'algorithme du FLORE 94
3.5.3 Raisonnement sur le micro-contexte des appareils 98
3.5.4 Raisonnement sur les zones des espaces intelligents 102
3.5.5 Raisonnement sur le profil des utilisateurs 104
3.6 Auto-optimisation de l'organisation logicielle 110

3.7 Outils de gestion graphique 112
3.8 Discussion 115
3.8.1 Sécurité 118
3.8.2 Le temps et la sensibilité au contexte 120
3.8.3 Modalités d'interaction 121
3.8.4 Taxonomie et ressources dynamiques 122
3.8.5 Sous-espace intelligent et topologie 123

4 Évaluations, Tests et Résultats **125**
4.1 Objectifs 125
4.2 Tests de performance 128
4.3 Évaluations 131
4.3.1 Scénario #1 : Déploiement d'une application avec dépendances matérielles dans une zone de l'espace intelligent 132
4.3.2 Scénario #2 : Déploiement d'une application sans dépendances logicielles et matérielles dans une zone de l'espace intelligent, avec variation de la saturation des performances des systèmes 138
4.3.3 Scénario #3 : Évaluation des relations entre les zones des espaces intelligents dans le contexte du déploiement d'une application visant une zone 140
4.3.4 Scénario #4 : Déploiement d'une application liée à un utilisateur en particulier 143
4.3.5 Scénario #5 : Évaluation des préférences des utilisateurs envers des périphériques d'interaction lors de l'organisation logicielle 150
4.3.6 Scénario #6 : Évaluation des fonctions d'auto-optimisation de l'organisation logicielle 154
4.4 Discussion générale sur les résultats 156

Conclusion **159**

Annexe A Extrait de la description sémantique OWL des espaces intelligents **166**

Annexe B Extrait du contrôleur de logique floue du FLORE partie coordonnateur **170**

Bibliographie **184**

Liste des figures

1.1 Graphe présentant les états du cycle de vie d'une application et ses transitions . 17
1.2 Diagramme présentant la méthode *Pull* en gestion logicielle 25
1.3 Diagramme présentant la méthode *Push* en gestion logicielle 26
1.4 Diagramme présentant la stratégie d'orchestration avec la méthode Push et Pull 27
1.5 Diagramme présentant la stratégie de la chorégraphie 28

2.1 La boucle de contrôle de l'informatique autonome (Kephart 2003) 44
2.2 Illustration résumant l'informatique diffuse autonome, ces mécanismes et sa disposition dans un espace intelligent 58

3.1 Architecture générale des composantes de l'intergiciel du projet Tyche 70
3.2 Exemple sous la forme d'un diagramme de séquence, d'une requête d'auto-organisation sur l'intergiciel du projet Tyche 72
3.3 Les modules OSGi du projet Tyche et leurs dépendances fonctionnelles 74
3.4 Utilisation des informations contextuelles par les noeuds orchestrateurs et appareils 78
3.5 Concept de base de la description sémantique (tiré de [1]) 81
3.6 Concept utilisé dans le cadre du projet Tyche) 82
3.7 Aperçu des résultats d'auto-organisation en mode manuel de l'outil de gestion du projet Tyche 88
3.8 Fonction d'appartenance des ensembles flous Froid, Tiède et Chaud 92
3.9 Fonctions de *defuzzification*, le résultats des règles et le centroïde 93
3.10 Fonctions de *fuzzification* de l'utilisation des ressources du FLORE partie appareil (FLORE-A) 100
3.11 Fonctions de *defuzzification* du FLORE partie appareil (FLORE-A) vers le DCQ 101
3.12 Fonctions d'appartenance de la vitesse de déplacement des utilisateurs du FLORE 108

3.13 Capture d'écran de l'outil de gestion du projet Tyche, présentant la carte des espaces intelligents avec les appareils découverts 113
3.14 Capture d'écran de l'outil de gestion du projet Tyche, présentant la liste des applications pouvant être déployée et les options d'organisation/déploiement pas à pas (*See FLORE results*) ou bien autonome (*Tyche, I'm feeling lucky*) . . 114
3.15 Capture d'écran de l'outil de gestion du projet Tyche, présentant le service de messagerie 115
3.16 Captures d'écran de l'outil de gestion pour appareil mobile Android 116
3.17 Architecture multiniveaux du projet Tyche 124

4.1 Diagramme présentant les temps d'exécution de l'auto-organisation lors des tests de performance 129
4.2 Diagramme présentant la distribution du DCQ selon la saturation des ressources d'un appareil 131
4.3 Carte de l'appartement du Laboratoire DOMUS avec la disposition des appareils simulés lors des scénarios d'évaluation 133
4.4 Emplacement de l'utilisateur dans le Scénario #4 et les zones de modalités d'interaction correspondantes à l'itération #2 146

Liste des tableaux

3.1 Résumé des objectifs du projet Tyche 69
3.2 Exemple de DCQ pour des applications à déployer et organiser 96

4.1 Résumé des objectifs des tests et évaluations 127
4.2 Temps d'exécution du FLORE lors des tests de performance 128
4.3 Résultats de la mise en place du Scénario #1 135
4.4 Résultats de la mise en place du Scénario #1 139
4.5 Résultats de la mise en place du Scénario 3 141
4.6 Résultats de la mise en place du Scénario 4 147
4.7 Résultats de la mise en place du Scénario 5 152
4.8 Résultats de la mise en place du Scénario 6 155

Introduction

Depuis près de deux décennies, l'informatique diffuse ou *ubiquitous computing* [2] et l'intelligence ambiante ou *ambient intelligence* [3] n'ont cessé d'attirer le regard des scientifiques, des industriels et des consommateurs. En promettant des environnements intelligents où les interactions avec les objets de la vie courante sont améliorées ou adaptées au profil de la personne, ces approches ouvrent la porte à des utilisations multiples, tant du côté de la santé que du militaire, de la sécurité, du commercial, de l'écologie ou du divertissement.

L'application de l'informatique diffuse et de l'intelligence ambiante à l'intérieur d'environnements tels que des habitats, peut améliorer les expériences utilisateur aux moyens de diverses technologies. Ces technologies utilisent alors les informations propres aux environnements pour donner des services adaptés aux contextes de ces espaces (nous reviendrons plus tard sur la définition de contexte) et des utilisateurs. Autrement dit, ces environnements deviennent intelligents, d'où l'expression *espace intelligent*, qui peut être appliquée à plusieurs types d'environnements tel que les habitats, les espaces commerciaux, les environnements médicaux, les moyens de transport, etc.

L'utilisation des espaces intelligents dans le but d'aider les personnes dépendantes peut améliorer de façon significative leur qualité de vie. Par exemple, en utilisant l'information donnée par un habitat intelligent à travers des capteurs, des agents et/ou des bases de connaissances, il est alors possible d'offrir de l'assistance pour la réalisation de tâches à des personnes ayant des déficiences cognitives via une interface homme-machine et des objets de l'environnement, tel que dans le projet *Archipel* [4]. Ce type d'assistance peut alors améliorer l'autonomie des utilisateurs de ces environnements. Les espaces intelligents peuvent également améliorer l'autonomie des personnes ayant des handicaps physiques en leur permettant de mieux interagir avec les objets de la vie courante [5].

L'informatique diffuse

Afin de comprendre les problématiques reliées à la gestion logicielle dans les espaces intelligents, il est important de connaître l'approche de l'informatique diffuse. Ce chapitre introduit tout d'abord l'informatique diffuse, puis présente les principaux travaux effectués dans ce domaine.

Définition de l'informatique diffuse

Les progrès en informatique depuis plus de cinquante ans sont phénoménaux. En peu de temps à l'échelle humaine, les ordinateurs et appareils informatiques sont passés d'une taille gigantesque pesant plusieurs tonnes métriques à quelques dizaines de grammes pour les derniers appareils informatiques grands publics tels que les baladeurs ou les téléphones portables. À titre d'exemple, la plupart des téléphones portables de nos jours possèdent plus de puissance de calcul que l'ordinateur de bord d'Apollo 11 en 1969, alors que celui-ci remplissait une bonne partie de l'habitacle.

Depuis son invention, la présence de l'informatique dans notre vie quotidienne n'a cessé de croître. Il y a trente ans, l'informatique était réservée à quelques scientifiques et entrepreneurs dans les pays les plus riches de la planète alors qu'aujourd'hui des millions d'enfants ont accès à des ordinateurs, et ce, sur tous les continents.

De plus, l'arrivée des premiers protocoles télématiques a eu un impact considérable sur notre quotidien. Grâce à eux, les ordinateurs ont pu commencer à communiquer entre eux pour la réalisation de tâches, créant l'informatique distribuée. Toutefois, c'est probablement l'invention d'un protocole commun, le HTTP, et d'un standard de présentation, le HTML, qui amènera la plus grande révolution, en permettant à des gens d'un peu partout dans le monde d'échanger des informations via le *World Wide Web*. C'est dans ce contexte d'évolution rapide de l'électronique, de l'informatique et de la popularisation de certaines technologies, que Mark Weiser, chercheur au laboratoire californien Xerox Parc, a effectué ses recherches en informatique qui mèneront à la création d'un nouveau modèle technologique. Ainsi, en regroupant les différents domaines de recherche du Xerox PARC, il a créé l'*ubiquitous computing* ou informatique diffuse [2] [6].

Ce modèle consiste à distribuer le traitement de l'information dans les objets de la vie courante afin d'améliorer les interactions entre les usagers et leur environnement par la présentation de l'information plus personnalisée, plus disponible, plus près du contexte d'application. Ce modèle s'inspire des différentes avancées en électronique et en informatique afin de

rendre les objets électroniques mobiles, tels que l'augmentation de la puissance de calculs des appareils électroniques, leurs miniaturisations, l'utilisation des protocoles de télécommunication avancés afin de partager l'information entre les objets d'un environnement, l'exploitation des nouvelles piles électriques et des protocoles hertziens. Avec cette approche, l'informatique devient omniprésente dans notre quotidien ; les ordinateurs personnels et les agendas électroniques portatifs (PDA) deviennent des objets électroniques parmi tant d'autres et leur importance diminue au profit de la constellation formée par les objets informatisés de l'environnement [7].

Ainsi, on peut résumer l'histoire de l'informatique en trois phases : la première étant la période de démocratisation des ordinateurs *Mainframes*, la seconde comme étant la période de démocratisation des ordinateurs personnels (PC) et finalement la troisième, l'informatique diffuse, où l'informatique est présente dans les objets de la vie courante et dissimulée aux utilisateurs.

Les bénéfices de l'informatique diffuse sont multiples et ses contextes d'utilisation presque infinis. Grâce à ce modèle, il est possible de rendre des milieux industriels plus sécuritaires pour les travailleurs grâce aux réseaux de capteurs intégrés dans l'environnement et grâce à des interfaces homme environnement plus efficaces. Dans une situation dangereuse, l'environnement est alors capable d'informer les travailleurs rapidement et d'une façon significative, de la présence d'un danger, à travers les objets du milieu de travail [8]. En commerce, l'informatique diffuse peut être utilisée afin d'apporter un service plus personnalisé dans les magasins grâce à un environnement qui se personnalise selon le profil de l'acheteur [9]. Enfin, l'application résidentielle de l'informatique diffuse permet la création d'habitats où les interactions avec les objets sont améliorées, l'accès à l'information se fait à travers plus d'objets et l'assistance est personnalisée selon le profil de l'utilisateur. Ainsi, à l'aide de capteurs, les applications de l'habitat peuvent connaître le contexte de celui-ci et au travers d'effecteurs, il leur est possible de manipuler certaines composantes du milieu telles que l'éclairage, la plomberie, les appareils électroménagers, les flux vidéo vers les téléviseurs, etc. Dans ce dernier cas, on peut également parler, de façon plus générale, de domotique.

La notion de contexte en informatique diffuse

Dans le cadre de ses travaux, Weiser [6] proposait de ramener le traitement des données près de leurs sources et d'utiliser plus d'information provenant de l'environnement afin d'améliorer la qualité des interactions. Il proposait également d'utiliser des données sur l'utilisateur afin

de personnaliser les applications et de rendre celles-ci plus performantes pour un utilisateur donné. Ainsi, sans vraiment la définir, Weiser utilisait alors la notion de contexte. Il faudra toutefois attendre quelques années avant que d'autres scientifiques s'attardent sur cette notion et la définisse plus clairement.

Ainsi, Dey propose de définir le contexte comme étant :

> « 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 applications themselves. » [10]

Il définit ensuite la sensibilité au contexte ou *context awareness* chez un système comme étant :

> « A system is context-aware if it uses context to provide relevant information and/or services to the user, where relevancy depends on the user's task. » [10]

Quoique résolument centré utilisateur, ces deux définitions s'appliquent à la plupart des systèmes exploitants l'information contextuelle. On pourrait toutefois y substituer la notion d'utilisateur par la notion d'agent ou d'acteur, ce qui permet d'étendre ces définitions à un plus grand type de systèmes informatiques. Un autre chercheur, Ryan, donne une définition plus précise du contexte, mais qui inclut des notions importantes telles que l'emplacement d'une composante et le temps :

> « [...] describes the ability of the computer to sense and act upon information about its environment, such as location, time, temperature or user identity. This information can be used not only to tag information as it is collected in the field, but also to enable selective responses such as triggering alarms or retrieving information relevant to the task at hand. » [11]

Dans cette dernière définition, une notion essentielle, le temps, est introduite. Le contexte est à propos des informations du présent, mais puisque le présent est partiellement construit à partir d'événements du passé, l'information provenant des contextes passés peut être utilisée dans le cadre de la sensibilité au contexte. Ainsi, plusieurs données provenant du passé tel que les préférences ou l'historique d'utilisation d'un utilisateur, peuvent être utilisées pour améliorer la qualité des interactions [11]. D'un autre côté, déduire ou inférer des informations futures peut également être très utile pour un système sensible au contexte. Le contexte est donc constitué d'information provenant du passé, du présent et du futur, déduit des deux types

de données précédentes. Donc, en résumé, un système sensible au contexte utilise l'information contextuelle qui l'entoure, de l'information abstraite telle que le temps et de l'information quantitative telle que l'emplacement d'une composante, afin d'améliorer la qualité de son traitement de l'information et de ses interactions avec d'autres acteurs.

Notre travail de recherche s'inscrit résolument dans une approche de sensibilité au contexte. Comme nous l'écrivons plus en détail dans les sections suivantes traitant de notre problématique de recherche et de nos objectifs, nous utilisons la notion de contexte et de sensibilité afin d'aider à réduire la complexité et les difficultés de la gestion logicielle des espaces intelligents. De plus, nous appliquons dans le cadre de nos travaux, les premiers concepts d'une nouvelle approche dans la modélisation et l'utilisation du contexte. Celle-ci est basée sur une spécification du contexte à deux principaux niveaux : le contexte microscopique et macroscopique [12]. Cette approche est décrite plus en détail dans le Chapitre 3.

Travaux en informatique diffuse

L'informatique diffuse est l'une des approches de l'informatique qui rassemble le plus de domaines de recherche : système embarqué, informatique distribué, intelligence artificielle, architecture logicielle, interaction homme-machine, etc. Les recherches effectuées dans ce domaine sont nombreuses et souvent d'une grande qualité. En plus de rassembler plusieurs domaines de l'informatique, l'informatique diffuse attire une foule de gens provenant d'autres domaines tels que l'électronique, la médecine, la neuropsychologie, l'ergonomie, la conception industrielle, l'architecture, etc. L'informatique diffuse est donc une approche multidisplinaire.

Plusieurs équipes académiques travaillant sur l'informatique diffuse ont vu le jour depuis une dizaine d'années. Le groupe *Oxygen* du *Massachusetts Institute of Technology* [13] fut l'un des premiers groupes à se pencher sur l'approche de l'informatique diffuse après les recherches du Xerox PARC. Le *Massachusetts Institute of Technology* contribue également à l'informatique diffuse à l'aide de son groupe de recherche *House_n*, travaillant principalement à l'élaboration de solutions pour rendre les habitats plus dynamiques et afin de mieux répondre à nos besoins au quotidien [14]. Ils ont entre autres développé un appareil dénommé *BoxLab*[1], truffé de capteurs et de caméras et pouvant être déployé facilement dans un environnement. Il permet de récolter l'information contextuelle de l'environnement, de l'analyser et de le rendre public à d'autres appareils et applications.

Le laboratoire *Aware Home* du *Georgia Institute of Technology* [15] contribue également à

[1]Site web de BoxLab : http ://boxlab.wikispaces.com/

l'informatique diffuse à l'aide de ses travaux sur l'assistance aux utilisateurs dans les habitats intelligents. C'est également dans ce laboratoire que Dey a travaillé sur la notion de contexte et sur le développement du *Context Toolkit* [16], une plateforme permettant d'interfacer des capteurs, d'exploiter les données, de les analyser et de rendre les données publiques.

Enfin un autre laboratoire américain, le *Mobile & Pervasive Computing Research Lab* de l'Université de Floride [17] contribue aussi à la recherche dans ce domaine avec son projet *Gator Tech House* et sa recherche sur l'informatique mobile. Plusieurs autres chercheurs universitaires dans le monde contribuent à la recherche en informatique diffuse telle que Das [18] à l'Université du Texas, avec ses recherches sur les habitats intelligents, ou Zenn Bien [19] de l'*Ulsan National Institute of Science and Technology* (UNIST) en Corée du Sud, avec ses travaux sur l'assistance à l'aide de la robotique et la prédiction de contexte à l'aide de la logique floue.

De même, plusieurs grandes entreprises investissent dans la recherche en informatique diffuse : Microsoft avec son équipe de recherche en informatique ubiquiste et son projet *Easy-Living* [20], IBM avec son *Advance Pervasive Technology Laboratory* [21] et Intel avec ses recherches en reconnaissance d'activités humaines et en système de localisation intérieur/extérieur [22].

On ne peut finir ce rapide tour d'horizon des recherches en informatique diffuse sans présenter les deux laboratoires de recherche où est effectué notre travail, le laboratoire DOMUS de l'Université de Sherbrooke et le laboratoire Handicom de Telecom SudParis. Le laboratoire DOMUS est un groupe dédié à la recherche sur les habitats intelligents et l'informatique mobile. Ainsi, DOMUS utilise l'informatique diffuse afin de récolter l'information contextuelle de l'environnement et de donner une assistance aux activités de la vie quotidienne (AVQ) plus appropriée et plus personnalisée [23] [24]. Le laboratoire s'est d'ailleurs spécialisé dans le développement de solutions d'assistance pour les personnes ayant des déficits cognitifs tels que la schizophrénie, les traumatismes crâniens et la maladie d'Alzheimer [25]. Quant au laboratoire Handicom, il consacre ses recherches à la conception de systèmes permettant d'améliorer l'autonomie des personnes ayant des handicaps physiques à l'aide de supports tels que les espaces intelligents et les interfaces homme-machine adaptées aux handicaps des utilisateurs [26]. Par la proximité des sujets de recherche, ces deux laboratoires collaborent activement, par des échanges scientifiques et la participation à des activités conjointes.

Problématique générale

Malgré les avantages que l'informatique diffuse et les espaces intelligents peuvent apporter aux utilisateurs, ceux-ci se font encore rares. La lente démocratisation de l'informatique diffuse peut être expliquée par plusieurs causes :

- la complexité et les coûts importants reliés à l'implantation des espaces intelligents ;
- le coût élevé des technologies utilisées ;
- la difficulté d'étendre les technologies diffuses à d'autres cas, contextes ou à des technologies non-polyvalentes ;
- le peu de versatilité des applications de l'informatique diffuse ;
- la difficulté de trouver des technologies stables répondant aux besoins de l'informatique diffuse ;
- la difficulté de trouver des technologies répondant réellement aux besoins des utilisateurs.

Les quatre premières causes sont étroitement reliées, dans la plupart des cas, capacité d'extension (*scalability*) et versatilité sont synonymes de complexité systémique, qui mène à des coûts importants dans l'implémentation et la maintenance de ceux-ci. L'hétérogénéité des technologies utilisées et la coexistence de plusieurs appareils, applications et utilisateurs, augmentent encore plus la complexité des systèmes diffus. Enfin, les technologies à informatique diffuse doivent essentiellement aider les utilisateurs dans des problèmes réels et répondre à des besoins réels. Autrement, ces technologies deviennent alors de simples *gadgets*.

Encore relativement méconnue, la complexité du déploiement, de la gestion et de la maintenance des logiciels dans les espaces intelligents contribue également de façon importante aux coûts d'implantation et de maintenance de ces environnements. Cet aspect deviendra de plus en plus important au fur et à mesure que l'informatique diffuse se démocratisera. Étant le coeur de la problématique de cette thèse, nous reviendrons un peu plus loin sur les divers aspects de la gestion logicielle dans les espaces intelligents.

L'informatique diffuse n'est pas la seule approche en informatique à affronter ces problèmes. Dans les technologies de l'information, la popularité sans cesse croissante du *Web* a amené une augmentation importante du nombre de systèmes informatiques, une augmentation de la complexité des applications qui y sont déployées et, par effet de bord, une augmentation du coût d'implantation et de gestion [27] de ces systèmes. Les interconnexions existantes entre les applications et les systèmes peuvent être à ce point nombreuses et complexes qu'il est difficile pour des êtres humains de comprendre, d'utiliser et de gérer ces systèmes.

L'informatique autonome [28], une approche visant à rendre les systèmes informatiques plus autonomes vis-à-vis des interventions humaines, répond à la problématique de la gestion logicielle en proposant des mécanismes structurés autour de quatre concepts : l'auto-configuration, l'auto-protection, l'auto-optimisation et l'auto-réparation. Ainsi, les systèmes informatiques récupèrent une majeure partie des processus de gestion logicielle, libérant les utilisateurs de la complexité et du temps investi dans cette gestion. L'approche de l'informatique autonome peut être adaptée au contexte de l'informatique diffuse, i.e. l'informatique diffuse autonome, afin d'améliorer la convivialité des espaces intelligents, principalement pour les utilisateurs non-experts, de faciliter l'implantation des espaces intelligents et de réduire les coûts de gestion logicielle des espaces intelligents [29].

Dans le contexte des espaces intelligents pour personnes dépendantes, tel qu'étudiés par le laboratoire DOMUS et Handicom, l'application de l'informatique diffuse autonome vise, par exemple à :

- déployer et gérer automatiquement des applications en tenant compte du contexte des environnements ;
- entreprendre la réparation de bris ou de fautes de façon autonome ;
- optimiser de façon proactive les performances des espaces en reconfigurant automatiquement les architectures logicielles lorsque nécessaire ;
- améliorer la sécurité des milieux en appliquant des mesures de protection face aux dangers virtuels ou physiques potentiels.

Objectifs et méthodologie de la recherche

L'objectif général de ce travail de recherche est la compensation par l'environnement de la complexité et de la difficulté de la gestion logicielle. La réalisation de ce but passe par la conception de mécanismes, de stratégies, de fonctionnalités et d'une architecture permettant la gestion logicielle autonome des espaces intelligents. Ce travail de thèse vise également à démontrer les stratégies proposées en développant et en évaluant un intergiciel, offrant des mécanismes intelligents permettant de réduire l'implication des gestionnaires et des utilisateurs dans la gestion logicielle des espaces intelligents. Le projet de recherche attaché à cette thèse s'est déroulé en quatre principales phases, qui ont répondu a des objectifs à atteindre bien définis.

Ainsi, la première phase de notre travail consistait à spécifier nos besoins en gestion logicielle dans le contexte d'une approche autonome et de l'informatique diffuse. Ainsi, nous

avons, dans un premier temps, travaillé à la construction de notre vision de la gestion logicielle autonome dans les espaces intelligents [30]. Cette vision basée sur l'approche de l'informatique autonome, utilise le concept des quatre *selves* : auto-configuration, auto-protection, auto-optimisation et auto-réparation, pour répondre aux besoins des espaces intelligents et plus particulièrement de la gestion logicielle. Dans un deuxième temps, nous nous sommes basés sur notre vision de l'informatique diffuse autonome pour proposer les fonctionnalités et l'architecture d'un intergiciel distribué permettant de gérer de façon autonome le déploiement, l'organisation et la configuration d'applications sur les appareils des espaces intelligents [29] [31].

Notre objectif pour la seconde phase de nos travaux avait pour but de dresser un état de l'art de la gestion logicielle dans les espaces intelligents et de l'informatique autonome, en passant en revue les principaux travaux dans le domaine [28] [32] [33] [34] [26] et plus spécifiquement ceux abordant l'approche de l'informatique autonome. L'objectif lors de cette phase était double, soit de connaître les principaux travaux dans le domaine afin de situer les besoins du DOMUS et d'Handicom, et par la suite d'entrevoir l'innovation et la contribution que l'on pouvait apporter à cet état de l'art.

La troisième phase de notre travail consistait à implémenter l'intergiciel de gestion autonome pour les logiciels des espaces intelligents. Cette phase fut divisée en plusieurs tâches correspondant à des objectifs bien précis. Tout d'abord, puisque l'intergiciel allait exploiter l'information contextuelle de chaque espace, nous avons travaillé, en collaboration avec d'autres chercheurs du laboratoire DOMUS, à la modélisation et la conception d'une description sémantique des espaces intelligents [35] [1]. Cette description développée à l'aide du langage OWL, a permis de décrire dans un langage standard les diverses informations que l'intergiciel allait exploiter dans ses tâches de gestion. Dans la même foulée, puisque nous abordions les notions de contexte et de sensibilité à celui-ci, nous avons travaillé à la spécification d'une approche divisant le contexte en deux types : le contexte microscopique i.e. les informations propres à une composante et à son entourage proche et le contexte macroscopique i.e. les informations et méta-informations relatives à un espace, extraite ou non des composantes du milieu [12]. Cette approche est mise en oeuvre dans le cadre de l'intergiciel de gestion.

Ensuite, nous avons travaillé à l'implémentation de l'intergiciel proprement dit [36] [37]. Cette implémentation s'est déroulée en plusieurs sous-étapes correspondant aux fonctionnalités à développer. Ainsi, nous avons travaillé successivement sur :

- la structure de l'intergiciel basée sur une approche orientée-service (SOA) ;
- le mécanisme de découverte des composantes participantes ;
- les services de gestion de l'information contextuelle ;

- les services de gestion des logiciels ;
- les engins de raisonnement portant sur l'auto-organisation/configuration logicielle ;
- l'outil graphique de gestion ;
- le mécanisme d'auto-optimisation de l'organisation logicielle.

De toutes ces étapes de développement, le travail portant sur le développement de l'engin de raisonnement est probablement celui qui a monopolisé le plus de temps et de ressources.

Enfin, la dernière phase de notre travail avait pour but d'évaluer et de valider l'ensemble des solutions proposées. Nous avons testé les performances de l'intergiciel de gestion dans le cadre de plusieurs scénarios d'utilisation réelle. Nous avons également évalué les résultats proposés par l'intergiciel face à des balises issues de nos hypothèses et des scénarios. Durant cette phase, les infrastructures du laboratoire DOMUS ont servi de support matériel et logiciel pour le déploiement de l'intergiciel de gestion autonome, ainsi que pour la phase de mise en oeuvre des scénarios d'évaluation.

Contribution et originalité de cette thèse

Afin de répondre à la problématique de la gestion logicielle dans les espaces intelligents, nous proposons dans cette thèse des solutions théoriques et pratiques orientées vers l'autonomie des tâches et actions de gestion. Ainsi, notre objectif est donc de proposer des solutions afin de réduire la complexité et la difficulté de la gestion logicielle dans plusieurs espaces intelligents en appliquant les principes de l'informatique autonome et de l'informatique diffuse.

Ces solutions s'inscrivent donc dans la lignée des travaux de Kephart et Chess [28] et dans une certaine mesure, les travaux de Weiser [2]. Comme nous serons à même de le constater dans les prochains chapitres, ce travail de thèse se base également sur un ensemble d'autres travaux tels que ceux de Ranganathan [33], Trumler [32] ou Ghorbel [38].

Quelques travaux se sont donc penchés sur la problématique de la gestion logicielle dans les espaces intelligents. Toutefois, ceux-ci sont en général trop axés sur des scénarios précis et n'ont pas été conçus pour être appliqués à d'autres cas ou bien être étendu à des cas plus génériques. L'arrivée en force de nouveaux supports technologiques tels que le langage sémantique *Ontology Web Language* (OWL) [39] ou bien les *Web Services*, permettent d'offrir des solutions standardisées et facilement extensibles, que les précédents travaux n'ont pu exploiter. Notre travail repose sur l'utilisation de ces technologies afin de permettre une évolution aisée de notre solution vers d'autres contextes d'utilisation.

L'approche de l'informatique autonome répondait originalement à la problématique de la

gestion de flottes de serveurs et de systèmes informatisés complexes tels que ceux des fournisseurs de télécommunications ou ceux des banques. Le contexte des applications dans les espaces intelligents et de leur gestion est radicalement différent : hétérogénéité élevée, dynamisme des liens entre composantes, utilisateurs omniprésents, etc. Nous proposons donc dans cette thèse une vision de l'informatique autonome appliquée aux espaces intelligents, l'informatique diffuse autonome, et à la gestion des logiciels dans les espaces intelligents [30].

La principale contribution de notre travail de thèse consiste en la conception, l'implémentation et l'évaluation d'un intergiciel autonome permettant de réduire la complexité, la difficulté et le nombre d'actions reliées à la gestion des applications dans des espaces intelligents [31] [36]. Afin de réduire l'impact de la complexité du système et du besoin pour l'utilisateur d'avoir une connaissance pointue du contexte, de la topologie et de l'architecture de chaque espace, nous utilisons des bases de connaissances, des ontologies sémantiques, et des engins de raisonnement. Ainsi, lors d'un déploiement ou de la maintenance d'applications, la majorité du travail d'analyse sur le contexte est évacué vers le système lui-même. S'ajoute à cette fonctionnalité, la possibilité pour les applications de se réorganiser de façon autonome suite à des modifications dans le contexte des espaces, que l'on désignera plus loin comme étant l'auto-optimisation de l'organisation logicielle. Cet intergiciel, dénommé projet Tyche[2], représente un travail important. La dernière version de l'intergiciel comprend dix modules (bundle) pour la plateforme *Open Services Gateway initiative* ou OSGi, en plus de quatre modules optionnels, comptant au total plus de 35 000 lignes de code dans le langage Java et 10 000 lignes de code Java additionnelles, générées à l'aide d'outils de développement. Ceci ne comprend pas les diverses bibliothèques de code et les interfaces de programmation (API) utilisées. S'ajoutent à ces lignes de code les descriptions sémantiques OWL implémentées et les règles de raisonnement en logique floue développées dans le langage FCL (*Fuzzy Control Language*)).

Le développement de la base de connaissances et de l'engin de raisonnement ont représenté une partie importante de ce travail de thèse. Dans un premier temps, nous avons contribué à proposer un modèle ontologique standard décrivant les espaces intelligents [1]. Fait innovant dans la gestion des espaces intelligents, afin d'exploiter pleinement les informations contenues dans les ontologies, nous avons développé un engin de raisonnement utilisant à la fois la description logique et la logique floue, afin de déterminer l'organisation optimale des logiciels parmi les composantes des espaces [36]. Entre autres, la description logique [40] permet de travailler efficacement avec la description sémantique développée et la logique floue [41] per-

[2]déesse grecque de la chance et de ce qui advient sans le concours de l'homme

met de raisonner rapidement et facilement sur un ensemble de données à la fois quantitatives, par exemple la puissance d'un processeur, et qualitatives, par exemple le degré de saturation d'un processeur (élevé, moyen, minime, etc.). Fusionné, ces deux modes de raisonnement sont particulièrement efficaces dans le cas d'ontologie sémantique, puisqu'ils se complètent l'un et l'autre : le formalisme de la description logique, la versatilité de la logique floue, la capacité d'inférence de la description logique, la possibilité de gérer facilement l'incertitude dans la logique floue, etc.

De plus, une autre originalité de notre engin de raisonnement versus les autres travaux de l'état de l'art, consiste en la répartition du raisonnement entre des noeuds orchestrateurs et des noeuds participants actifs. Notre mécanique utilise donc une stratégie à la fois d'orchestration et de chorégraphie, puisqu'une partie du raisonnement est laissée aux composants de l'environnement et une partie est gérée par les orchestrateurs. Essentiellement, nous avons voulu intégrer une vision développée en partenariat avec d'autres chercheurs du Laboratoire DOMUS, qui consiste à définir une notion de hiérarchie dans l'information contextuelle des espaces intelligents [12] : le contexte microscopique i.e. les informations reliées à une composante et à son entourage proche ; le contexte macroscopique i.e. information générale sur le milieu et/ou abstrait des différents contextes microscopiques. Ainsi, une partie du traitement relié à l'information microscopique est faite dans les composantes actives de l'environnement et l'autre partie dans les orchestrateurs pour ce qui est des informations macroscopiques.

Enfin, la dernière innovation majeure de ce travail de thèse consiste en l'implémentation d'une mécanique de livraison d'applications sensible aux utilisateurs et à leur contexte [37]. Ce modèle utilise des modalités d'interaction : les sens, la perception, les sens moteurs (mobilité, motricité fine, force musculaire, etc.) et les capacités cognitives, afin de déterminer selon le profil de l'utilisateur, quel appareil ou composante dans l'environnement est le meilleur support pour l'utilisateur en question. Ce modèle utilise également les préférences et la localisation de l'utilisateur dans ses mécanismes de raisonnement. Ainsi, cette mécanique permet à des applications tierces d'utiliser l'intergiciel Tyche afin de délivrer une assistance vers un utilisateur sous la forme d'un logiciel.

Ce travail de thèse s'inscrit dans le cadre d'un projet de recherche de plus grande envergure au sein du laboratoire DOMUS de l'Université de Sherbrooke et du laboratoire Handicom de Telecom SudParis. Ce projet porte sur la création d'une architecture logicielle et matérielle permettant l'implémentation d'habitats intelligents pour personnes dépendantes, atteintes, entre autres, de déficiences cognitives telle que la maladie d'Alzeihmer ou de déficits moteurs. L'un des objectifs du projet porte sur la création de résidences alternatives qui permettront d'amé-

liorer l'autonomie des personnes dépendantes à l'aide d'applications d'assistance développées selon les concepts de l'informatique diffuse et de l'intelligence ambiante. Ainsi, ce travail de thèse s'imbrique dans ce projet en proposant un intergiciel et des stratégies afin de faciliter la gestion de ces habitats. De plus, comme nous le mentionnons plus haut, les mécanismes de déploiement dynamique d'applications offrent aux applications d'assistance tierces, la possibilité de déployer des modules d'assistances vers les utilisateurs.

Organisation de la thèse

Cette thèse est organisée en quatre principaux chapitres, correspondant aux phases et objectifs de notre travail de recherche présentés dans la section précédente.

Le premier chapitre présente les concepts de base de la gestion logicielle dans le contexte d'espaces intelligents. Suit une discussion sur les problèmes rencontrés dans le cadre des tâches et activités de gestion logicielles dans les espaces intelligents. Afin d'avoir un aperçu des solutions existantes permettant d'aider à la gestion des logiciels, un état de l'art des travaux et mécanismes est présenté.

Dans un second temps, le Chapitre 2 présente l'approche de l'informatique autonome et ses concepts, les quatre *selves*. Nous y présentons également notre vision de l'informatique autonome appliquée aux espaces intelligents et à la gestion logicielle, l'informatique diffuse autonome. Nous terminons ce chapitre par une revue des travaux portant sur l'utilisation de l'approche autonome dans la recherche de solutions à la gestion logicielle.

Le troisième chapitre aborde la conception et l'implémentation d'un intergiciel distribué permettant une gestion autonome des logiciels d'espaces intelligents. Celui-ci présente tout d'abord la modélisation de la description sémantique de l'espace intelligent. Ensuite, la spécification du contexte suivant une approche basée sur le contexte microscopique et macroscopique est présentée. Enfin, la dernière section de chapitre décrit les fonctionnalités de l'intergiciel de gestion autonome et plus particulièrement ses engins de raisonnement. Cette partie de la thèse correspond à notre contribution théorique et technique au domaine de la gestion logicielle dans les espaces intelligents.

Le quatrième et dernier chapitre fait une présentation détaillée de l'évaluation de notre intergiciel de gestion logicielle autonome. Ces résultats ont, entre autres, démontré la performance en temps de calcul de la solution d'auto-organisation proposée, ainsi que la précision, la stabilité et la qualité des organisations logicielles retournées. Tout d'abord, une présentation des scénarios d'évaluation et des hypothèses de validation est présentée. Ensuite, nous pré-

sentons une série de données récoltées lors de la réalisation de ces scénarios avec l'intergiciel de gestion logiciel autonome. Enfin, nous initions une discussion sur ces résultats, mettant en lumière les forces et faiblesses de notre solution.

Enfin, la conclusion de cette thèse dresse un bilan de notre travail de recherche doctorale et de nos apports originaux à l'avancement des travaux en gestion logicielle dans le contexte des espaces intelligents. Nous y présentons également des perspectives de travaux futurs permettant d'améliorer les solutions proposées ou d'étendre les solutions à d'autres problématiques ou cas.

Chapitre 1

La gestion logicielle dans les espaces intelligents

Ce chapitre présente l'état de l'art en ce qui concerne la gestion logicielle en général et dans le cas plus spécifique des espaces intelligents. Dans un premier temps, la définition de la gestion logicielle est donnée. Par la suite, la problématique de la gestion logicielle dans les espaces intelligents est présentée. Dans un troisième temps, les principaux mécanismes et stratégies de gestion logicielles sont introduits. Ensuite, une revue des solutions de gestion logicielle existantes est faite, tant du côté des systèmes d'exploitation que des passerelles d'accès, des plateformes logicielles ou des grappes de calcul. Ce chapitre se termine par une discussion sur la gestion logicielle dans les espaces intelligents.

1.1 Définition de la gestion logicielle dans les espaces intelligents

Le besoin de gérer les états dans lesquels sont les applications : active, en arrêt, dépendances résolues et en attente de démarrage, en installation ou en désinstallation, est apparu avec les premiers systèmes d'exploitation multitâche. Ces quatre états et les transitions entre ces états peuvent être représentés sous la forme d'un automate. On appelle de façon plus générale cet automate, le cycle de vie d'une application. Il est toutefois important de différencier ce cycle de vie de celui de la discipline du génie logiciel tel que le modèle en cascade : spécification, conception générale, conception détaillée, développement, intégration, mise en production et maintenance. Dans notre contexte, on peut relier notre définition du cycle de vie aux étapes

mise en production et *maintenance* du modèle en cascade.

Le cycle de vie d'une application représente donc les différents états dans lequel peut être une application à partir de son installation jusqu'à sa désinstallation sur un appareil ou un système. Voici donc les différents états du cycle de vie et leurs transitions (Figure 1.1) :

- en installation : l'application s'installe sur l'appareil et vérifie si les ressources nécessaires sont présentes ;
- résolue : dans cet état l'application est installée, les dépendances matérielles et logicielles sont résolues, l'application est prête à être utilisée. À partir de cet état, il est possible de mettre à jour l'application, i.e. retourner dans l'état en installation, de démarrer l'application ou bien désinstaller celle-ci ;
- en démarrage : l'application est en cours de démarrage, elle fait l'allocation de ses besoins en ressources et instancie ses processus ;
- active : l'application est démarrée, fonctionnelle et peut être utilisée. Dans cet état, l'application peut être arrêtée ;
- en arrêt : l'application est en cours d'arrêt, elle fait la désallocation des ressources utilisées et dé-instancie ses processus ;
- en désinstallation : l'application est en cours de désinstallation. À la sortie de cet état, l'application n'est plus disponible sur le système.

Cette représentation, somme toute simple du cycle de vie d'une application, défini par l'*OSGi Alliance* [42], ne présente pas les états et transitions inhérents à l'exécution de l'application : synchronisation de fils d'exécution, en attente d'entrées de données, etc. Une définition du cycle de vie d'un service est donnée par le *World Wide Web Consortium*(W3C) [43] et complète le cas plus général des applications logicielles.

Il est donc important de tenir compte d'autres transitions possibles. Par exemple, une application peut retourner dans l'état *en installation* dans le cas d'une mise à jour de ses composantes. Dans le cas de fautes logicielles ou matérielles, l'arrivée de tels événements peut provoquer un changement d'état, tout dépendant de la façon dont le système traite la faute. Par exemple, dans le cas de la spécification OSGi, toutes les fautes non traitées à l'aide d'un *try-catch*, entraînent l'arrêt du module et son retour à l'état *résolu*. Dans le cas d'OSGi, peu de fautes logicielles peuvent entraîner l'arrêt de la plateforme et de tous les modules, mais certaines peuvent altérer les performances telles que la saturation de la pile mémoire (*Stack Overflow*).

Afin de bien cerner le contexte du travail, il est important de définir précisément ce qu'est la gestion logicielle. On peut définir celle-ci au sens large comme le processus menant à la

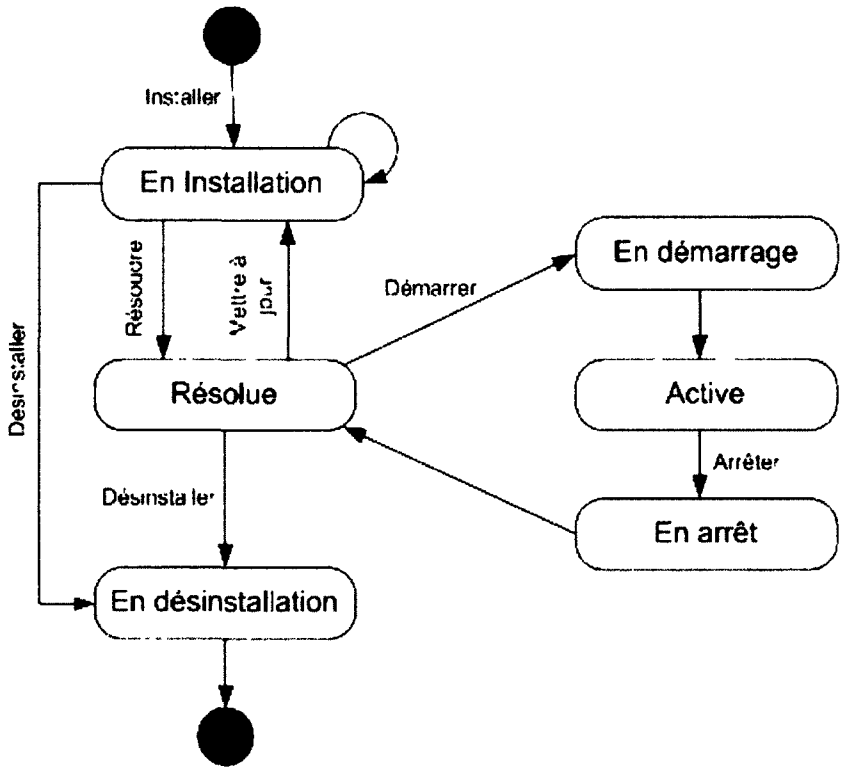

FIGURE 1.1 – Graphe présentant les états du cycle de vie d'une application et ses transitions

prévision, la manipulation (commandement), l'organisation, la coordination et le contrôle des logiciels dans un milieu en rapport aux besoins du gestionnaire et/ou des besoins des individus représentés par le gestionnaire. Cette définition, appliquée aux logiciels, est inspirée de celle de Henri Fayol sur la gestion en général [44].

La notion de manipulation ou de commandement des logiciels amène une manipulation de l'état du cycle de vie des applications à un instant donné. La prévision, l'organisation et la coordination peuvent quant à elles être résumées au concept de configuration des logiciels, i.e. la disposition stratégique des applications sur les entités d'un milieu face aux besoins des applications, les caractéristiques des composantes et utilisateurs des milieux. La configuration logicielle traite également de l'adaptation des applications aux contextes des entités et de l'adaptation des modes d'interaction homme-machine face aux profils des utilisateurs. En regard du cycle de vie des applications, la configuration logicielle concerne essentiellement le déploiement des applications dans un milieu, i.e. les actions reliées à l'installation d'un logiciel sur une entité : téléchargement du code (binaire ou non selon le type d'application), installation de celui-ci sur l'entité, vérification des dépendances de l'application sur l'entité et, pour terminer, passage à l'état résolu. Le déploiement d'un logiciel peut concerner une ou plusieurs

applications à déployer à la fois et pouvant avoir entre elles des dépendances logicielles, avec un couplage fort (dépendance de code) ou un couplage faible (dépendance de services).

Enfin, le contrôle des logiciels peut être résumé à la maintenance de ses logiciels. La maintenance vise à maintenir, à rétablir ou même améliorer les logiciels déployés dans un milieu. Elle concerne donc la résolution des pannes ou des fautes logicielles, la mise à jour des logiciels suite à la résolution des fautes et finalement les manipulations de l'état du logiciel selon les besoins donnés.

En conclusion, la gestion des logiciels dans les espaces intelligents comprend à la fois la manipulation des états des logiciels, la configuration des logiciels dans le milieu ainsi que la maintenance de ceux-ci au cours de leur utilisation. Dans tous les cas, une manipulation de l'état de vie des logiciels est nécessaire.

1.2 Problématique de la gestion logicielle dans les espaces intelligents

Comme nous le mentionnions dans l'introduction de cette thèse, le potentiel de l'informatique diffuse et de l'intelligence ambiante est immense et pourrait révolutionner, dans un avenir à plus ou moins court terme, notre façon de travailler, de vivre et d'interagir avec nos milieux de vie. Toutefois, plusieurs obstacles se dressent devant une utilisation plus large de l'informatique diffuse. Dans un premier temps, la complexité de la mise en place d'espaces intelligents et de leur maintenance amène des coûts élevés, dû principalement aux ressources humaines nécessaires, à l'expertise de ces ressources et du temps relié au déploiement et à la maintenance des espaces.

Dans un deuxième temps, la place prépondérante des utilisateurs, comme entité centrale des espaces intelligents, amène également un lot de besoins et de contraintes à respecter. Ces besoins et contraintes s'ajoutent à la problématique, en amenant, par exemple, la nécessité de gérer la sécurité des informations personnelles des utilisateurs des espaces intelligents.

Dans un troisième temps, les nouvelles méthodes de conception et de développements logiciels ont accéléré la mise en production des applications. Ces applications sont souvent mises en production alors que leurs fonctionnalités ne sont pas entièrement complétées, faisant participer les utilisateurs à leur amélioration. Ainsi, plus de mises à jour sont nécessaires, augmentant par le fait même le nombre de tâches de gestion logicielle pour les utilisateurs/gestionnaires et contribuent par effet de bord, à l'augmentation des coûts de maintenance des espaces

intelligents.

Enfin, comme tous les systèmes informatiques complexes, les espaces intelligents peuvent être le théâtre de fautes logicielles et de bris matériel. Ces fautes et bris, en plus de nécessiter l'attention des utilisateurs/gestionnaires, ont comme impact d'entraver les services d'assistance auprès des utilisateurs.

Ces quatre aspects additionnés amènent la nécessité de trouver des solutions permettant de simplifier la gestion logicielle des espaces intelligents, de réduire les coûts de déploiement et de maintenance et de diminuer l'impact des fautes et bris sur les utilisateurs. Ces quatre aspects de la problématique de la gestion logicielle dans les espaces intelligents.

1.2.1 L'hétérogénéité et la complexité des espaces intelligents

Les espaces intelligents sont composés de plusieurs appareils, sur lesquels sont déployées plusieurs applications. Dans la plupart des cas, ces appareils possèdent des différences marquées entre eux, certains ayant une puissance de calcul importante alors que d'autres ont des capacités réduites. Les différences entre les appareils ne sont pas que physiques, elles sont aussi techniques. Ainsi, les systèmes d'exploitation utilisés par ces appareils peuvent varier grandement, souvent en fonction des capacités de l'appareil, mais aussi en fonction du contexte d'utilisation. Bref, les appareils dans les environnements intelligents sont hétérogènes. À l'image des appareils, les applications présentes dans les espaces sont hétérogènes, elle utilise des ressources différentes ayant des durées de vie différentes, des modes de traitement de l'information différents et des architectures différentes.

Enfin, il peut résider de nombreuses dépendances fonctionnelles entre les appareils et les applications des milieux de sorte qu'il puisse devenir difficile de s'y retrouver. Par exemple, une application peut être dépendante à la fois d'une bibliothèque de code et d'un service distant sur un autre appareil, alors que le service sur l'appareil en question possède également des dépendances internes ou externes et ainsi de suite.

La gestion des applications dans ces milieux peut donc s'avérer relativement complexe. Si les gestions logicielles sont faites localement, machine par machine, on réalise rapidement qu'une telle méthode sera coûteuse en temps et en ressources humaines, d'autant plus si les lieux sont dispersés géographiquement. La gestion est encore plus complexe s'il s'agit de gérer des appareils mobiles qui sont en déplacement constant et qu'il faut récupérer physiquement afin de les administrer.

Par exemple, imaginons qu'une entreprise louant des appartements intelligents ait à super-

viser une centaine d'environnements dotés d'au moins une dizaine d'appareils informatiques dans chaque environnement. Le groupe posséderait donc une flotte d'environ mille appareils. Si on suppose qu'une mise à jour logicielle donnée soit nécessaire sur l'ensemble des appareils, ceci représente une tâche colossale pour les travailleurs responsables de la gestion logicielle de ces milieux. De plus, cet exemple ne tient pas compte de l'hétérogénéité des appareils et des applications, qui peuvent être radicalement différents dans leur façon d'être gérés. De tels problèmes de gestion sont également vécus dans d'autres domaines de l'informatique tels que les télécommunications ou les services Web [27].

Il est donc nécessaire d'alléger la tâche des utilisateurs experts en leur permettant de gérer à distance les applications dans les espaces intelligents. De plus, des mécanismes faisant la résolution des dépendances permettent de réduire la complexité des liens entre les applications et les appareils et ainsi alléger la tâche des gestionnaires. Des solutions facilitant plus ou moins la gestion des applications existent et quelques unes sont présentées un peu plus loin dans ce chapitre (1.4).

D'un autre côté, d'autres types d'utilisateurs peuvent avoir à intervenir dans la gestion des logiciels des espaces intelligents. Par exemple, un aidant professionnel tel qu'un(e) infirmier(ère) ou un(e) professionnel(le) de la réadaptation peut avoir à déployer des applications dans les milieux de vie de leurs patients afin de les assister dans leurs activités quotidiennes. Ces utilisateurs n'ont souvent que des connaissances fonctionnelles en informatique et n'ont pas de connaissances approfondies sur la configuration même des milieux et de leurs systèmes. Face à la complexité des milieux et aux vastes données sur ceux-ci, il est alors nécessaire de compenser les lacunes des utilisateurs non-experts en leur proposant des mécanismes de déploiement, d'organisation et de maintenance des logiciels autonomes. En déplaçant une large partie des tâches de gestion logicielle, ainsi que le raisonnement sur la configuration des milieux vers les systèmes informatiques des espaces intelligents, il est possible de réduire la complexité des tâches de gestion et d'abstraire l'hétérogénéité des milieux. Il est également utile de synthétiser l'information sur les systèmes afin de la vulgariser à un groupe d'utilisateurs plus large. Outre les utilisateurs non experts des espaces, de tels mécanismes autonomes peuvent aider les gestionnaires à gérer un plus grand nombre d'espaces, réduire le temps imparti aux tâches de gestion et diminuer les besoins de connaissances spécifiques à chaque espace intelligent. La conception et le développement d'un système d'organisation logicielle autonome pour espaces intelligents représentent la principale contribution de notre travail de thèse. Plus de détails sur les stratégies, l'architecture et les fonctionnalités de cette solution sont présentés dans cette thèse au Chapitre 3.

1.2.2 Le profil des utilisateurs/gestionnaires et la sécurité

Les espaces intelligents offrent des services d'assistance aux utilisateurs des milieux. Afin de proposer des assistances adaptées aux utilisateurs, les applications utilisent des informations décrivant l'utilisateur, i.e. le profil d'utilisateur, afin d'adapter l'assistance aux préférences de celui-ci, à ses méthodes d'interactions, à sa personnalité, etc. Dans le cas d'utilisateurs avec des déficiences cognitives ou physiques, l'assistance peut être adaptée à ses déficiences afin de les éviter ou de réduire leurs impacts.

Pour deux espaces intelligents semblables, mais ayant des utilisateurs différents, la composition logicielle des espaces peut donc être fort différente. Cela ajoute de l'hétérogénéité entre les espaces intelligents, ce qui a un effet direct sur l'augmentation de la complexité dans la gestion de ceux-ci.

Puisque le profil de l'utilisateur est partagé entre plusieurs applications, et que celui-ci peut contenir des informations sur la vie privée des utilisateurs (préférences, déficiences, informations médicales, etc.), la notion de confidentialité de ces informations entre alors en jeu. La sécurité et ses quatre concepts généraux : la confidentialité des données, la confiance envers les données/services, la sécurité des données/services et la sûreté [45] doivent également être gérées par les gestionnaires des espaces intelligents. On peut donc ajouter la gestion de la sécurité à la définition de la gestion logicielle dans les espaces intelligents.

Or, la gestion de la sécurité dans les espaces intelligents peut devenir un réel casse-tête quand il y a plusieurs utilisateurs, quand les espaces intelligents ont une nature dynamique, i.e. que des appareils et des utilisateurs peuvent entrer et sortir rapidement du milieu, lorsque les appareils et leurs moyens de communication sont hétérogènes, etc. Il est alors nécessaire de posséder des outils permettant de simplifier la gestion de la confidentialité de l'information, la gestion des méthodes d'authentification des utilisateurs/appareils et la gestion des moyens de chiffrement des données.

La gestion des espaces intelligents peut impliquer plusieurs types d'acteurs qui n'ont pas le même profil ni la même expérience. Par exemple, un gestionnaire de systèmes informatiques, qui a des connaissances accrues dans ce domaine, peut avoir à intervenir dans la gestion des espaces intelligents. On peut considérer ce gestionnaire comme étant un expert. D'un autre côté, d'autres types d'utilisateur non expert tel que des aidants professionnels : infirmières, médecins, ergothérapeutes, etc. ; peuvent également avoir à intervenir dans certaines situations (voir les scénarios, Chapitre 2, Section 2.2). La solution de gestion logicielle pour espaces intelligents doit donc offrir des fonctionnalités avancées pour les utilisateurs experts et permettre également une utilisation facile par les gestionnaires non experts.

Enfin, il est nécessaire de gérer les droits d'accès et d'utilisation des applications et services par les utilisateurs et les systèmes informatisés des environnements. Dans un espace intelligent, plusieurs utilisateurs peuvent y cohabiter et se partager des appareils et applications. Tout dépendant du type d'application et des données utilisées par celles-ci, chaque utilisateur et service peut avoir des droits d'utilisation différents. Par exemple, un habitant d'un espace intelligent aura plus de droits qu'un invité. Dans notre cas, un service de gestion logicielle des espaces aura des droits d'accès aux données et services plus élevés qu'une application d'assistance auprès des utilisateurs. Il est également nécessaire de gérer la concurrence entre les applications et/ ou les utilisateurs d'un environnement. Ainsi, alors qu'un utilisateur interagit avec une application d'assistance contenant des informations privées et qu'un autre utilisateur s'approche de celui-ci, il peut être nécessaire, selon les liens de confiance entre les utilisateurs, de protéger le contenu de l'application de l'utilisateur externe en cachant une partie de l'interface à l'approche de celui-ci. Encore là, afin de diminuer la charge de travail des gestionnaires quant à la gestion des droits d'accès et d'améliorer la sécurité des informations des espaces intelligents, l'automatisation de la gestion apparaît comme une solution prometteuse.

1.2.3 Les méthodes de développement et de conception

Les méthodes de travail en informatique ont changé considérablement depuis ses tout débuts. L'amélioration des langages de programmation, l'utilisation de l'approche orientée objet, le perfectionnement des méthodes de conception des logiciels, les frameworks et les outils de développement tel que les IDE permettent maintenant aux développeurs de concevoir plus rapidement des applications plus importantes. Notamment, des approches comme la programmation agile permettent aux développeurs d'obtenir rapidement des prototypes qui peuvent être présentés ou testés auprès des clients. Plus précisément, la programmation agile propose des méthodes de travail qui visent, entre autres, la satisfaction des utilisateurs par la mise à l'essai rapide et régulière des logiciels. Grâce à de telles méthodes de développement, le cycle de conception des applications a été considérablement réduit, permettant de faire plus de mise en production pour une même période de temps [46].

Dans un même courant de pensée, plusieurs entreprises ont profité des nouvelles technologies et des modèles de conception rapide, pour déployer des applications n'ayant pas été complètement testées. Ainsi, elles rendent disponibles des versions du logiciel plus rapidement et font participer les utilisateurs à l'amélioration des applications par l'entremise de fonctionnalités d'envois d'erreurs. Une conséquence importante d'un tel mode de développement et de

conception est la nécessité d'appliquer régulièrement des mises à jour sur les applications afin de corriger les erreurs trouvées par les utilisateurs. Dans le contexte de plusieurs espaces intelligents, l'utilisation de logiciels nécessitant des mises à jour régulières implique une charge de travail importante.

Ces observations s'appliquent également aux logiciels des espaces intelligents. Dans un tel contexte, il devient nécessaire d'avoir des mécanismes permettant de faciliter la mise à jour des applications soit en permettant aux gestionnaires de déclencher les mises à jour à distance et de façon simple, méthode *Push*, ou en permettant aux logiciels de vérifier régulièrement la présence de mises à jour, méthode *Pull*. Ces deux méthodes sont présentées dans la section suivante (1.3).

1.2.4 Les fautes logicielles et les bris matériels

Comme tous les types d'environnement informatique, les espaces intelligents ne sont pas à l'abri des bris de matériels aléatoires, par exemple un pic de tension électrique mettant hors d'usage une composante électrique, ou provoqués, par exemple, par un utilisateur échappant un appareil électronique au sol. Des fautes et erreurs logicielles peuvent également survenir au cours de leur utilisation. Ces fautes peuvent dans certains cas arrêter l'exécution de l'application ou bien empêcher son utilisation. De plus, puisque les espaces intelligents utilisent l'informatique distribuée dans leur traitement de l'information, des fautes inhérentes aux communications peuvent également survenir. Il faut donc considérer l'impact des "huit illusions de l'informatique distribuée" [47] sur les espaces intelligents :

1. Le réseau est fiable ;
2. Le temps de latence est nul ;
3. La bande passante est infinie ;
4. Le réseau est sûr ;
5. La topologie du réseau ne change pas ;
6. Il y a un et un seul administrateur réseaux ;
7. Le coût de transport est nul ;
8. Le réseau est homogène.

La gestion des fautes et des bris se fait généralement en aval de la configuration logicielle et concerne donc la maintenance. Les fautes et les bris peuvent donc représenter pour les gestionnaires un travail de maintenance considérable dans : la gestion du remplacement ou de la

réparation des appareils brisés, dans l'analyse des fautes logicielles, dans le développement de solutions et finalement dans la mise en place de ces solutions sous la forme de mises à jour ou de modules complémentaires.

De plus, les fautes et bris peuvent avoir un impact important sur l'utilisation des applications et appareils par les utilisateurs. Par exemple, l'incapacité d'interagir avec un appareil ou une application apportant une assistance peut avoir des impacts considérables sur les utilisateurs dépendant de cette assistance, pouvant même aller jusqu'à les mettre en danger. Le laboratoire DOMUS effectue des recherches portant sur la création d'application d'assistance aux personnes atteintes de troubles cognitifs, avec entre autres, des agendas électroniques sur téléphones intelligents. La perte d'un téléphone intelligent par un utilisateur, alors que celui-ci l'utilise à tous les jours comme outil cognitif, peut avoir des effets néfastes, passant par l'oubli de rendez-vous à la prise multiple (ou non) de médicaments. Dans ce dernier cas, on parle alors de dépendance de l'utilisateur envers des entités de l'environnement [48], ce qui concerne la notion de sureté dans la sécurité des espaces intelligents.

Dans tous les cas, les fautes logicielles et les bris mécaniques complexifient le travail des gestionnaires en leur apportant des tâches de gestion supplémentaires, ainsi que de l'information à analyser. Elles demandent également, dans certains cas, une rapidité d'intervention et une juste analyse de la situation, afin d'assurer une continuité de service aux utilisateurs. Dans ce contexte, la réduction et la synthèse des informations relatives aux fautes et bris, permet d'accélérer l'analyse de la situation par les gestionnaires et la prise rapide de décision. De plus, l'utilisation de mécanismes offrant une meilleure tolérance aux fautes [49] permet de diminuer l'impact des erreurs sur les entités des espaces intelligents. Toutefois, la mise en place de tels mécanismes n'est pas simple, entre autres, par la difficulté de découvrir et de confirmer les fautes ou bris, tel que dans le problème bien connu des deux généraux [50].

En conlusion, la problématique de la gestion logicielle dans les espaces intelligents est large et comporte plusieurs thèmes de recherche : l'organisation autonome des logiciels, la résistance aux fautes, l'auto-protection, etc. Il est impossible dans un seul travail de thèse de couvrir tous les aspects de la problématique de la gestion des espaces intelligents. Nous verrons dans la suite de cette thèse, comment il est possible de répondre à cette problématique à l'aide, entre autres, de l'informatique diffuse autonome [30] (Chapitre 2). Un peu plus loin, dans le Chapitre 3, nous abordons le coeur de notre travail de thèse avec l'organisation logicielle autonome et nous proposons une solution d'organisation autonome sous la forme d'un intergiciel d'organisation logicielle autonome pour espaces intelligents.

1.3 Méthodes et stratégies de gestion logicielle

La gestion des architectures logicielles dans les espaces intelligents implique donc de manipuler les états des logiciels selon les besoins des utilisateurs. Comme nous l'avons présenté à la Section 1.2.1, il existe un besoin réel de manipuler les états des applications à distance afin de réduire la complexité et les coûts de leur configuration et de leur maintenance.

Pour reprendre la définition de la gestion de Fayol [44], il est donc nécessaire de commander le changement de ces états à distance. Ceci peut être fait en utilisant deux méthodes de base le *Pull* et le *Push*, impliquant dans les deux cas des requêtes de gestion. Selon le choix fait, deux stratégies d'utilisation et de synchronisation de la gestion peuvent être utilisées : l'orchestration et la chorégraphie. Cette section présente donc dans un premier temps les méthodes *Pull* et *Push* (1.3.1) puis les stratégies d'orchestration et de chorégraphie (1.3.2).

1.3.1 Les méthodes *Pull* et *Push*

La méthode du *Pull* concerne principalement les entités sur lesquelles doivent être appliqué les requêtes de gestion. Dans cette méthode, ce sont ces entités qui ont la responsabilité d'aller vérifier l'existence des requêtes de gestion sur des ressources, par exemple un serveur d'applications, de télécharger les requêtes, puis de les appliquer. La Figure 1.2 illustre la méthode du *Pull*. Cette méthode est abondamment utilisée par les systèmes d'exploitation et les applications commerciales dans leur processus de mise à jour logicielle ou d'installation de modules complémentaires. Dans ces deux contextes d'utilisation, les entités vont interroger un serveur afin de vérifier l'existence de mise à jour. Si tel est le cas, les mises à jour sont téléchargées et installées sur l'entité. Le *Pull* peut être généralisé à toutes les manipulations de l'état d'un logiciel et non pas seulement aux mises à jour.

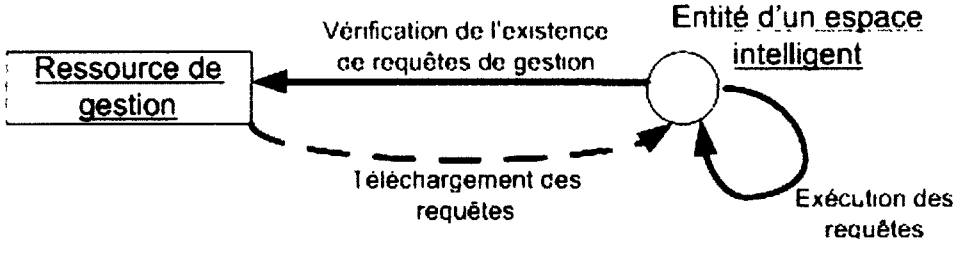

FIGURE 1.2 – Diagramme présentant la méthode *Pull* en gestion logicielle

L'un des principaux avantages de cette méthode est la facilité de l'implémenter dans toutes sortes de situations. Elle ne nécessite que des ressources externes contenant les requêtes de

gestion et un protocole permettant d'interroger les ressources, tels que le *Simple Object Access Protocol*(SOAP) ou le *Remote Method Invocation*(RMI). De par cette architecture, le *Pull* peut être étendu à un nombre très important d'entités à gérer. Puisque ce sont les entités elles-mêmes qui déclenchent la méthode, par des vérifications périodiques ou à l'intervention des utilisateurs, le *Pull* réduit l'implication des gestionnaires. Toutefois, le *Pull* à le désavantage d'offrir un contrôle moindre sur la configuration/organisation des milieux par les gestionnaires. Ainsi, le *Pull* revient en quelque sorte à mettre une directive de gestion sur un babillard et d'attendre que les entités consultent le babillard rapidement. Pour des environnements hétérogènes à organiser, la mise en place de système par *Pull* est plus complexe que dans le cas général, par exemple, de la mise à jour d'un logiciel. Plus de directives sont nécessaires avec un ensemble d'options à respecter. Ainsi, dans le cas où N espaces intelligents hétérogènes doivent être configurés/organisés, N directives devront être publiées sur un babillard.

Contrairement au *Pull*, le *Push* revient à l'image d'un patron d'entreprise passant parmi ses employés et distribuant directement les directives aux employés. Le contrôle par le patron sur la conduite des directives par les employés est plus important et les rétroactions sur les effets des directives sont directement observables. Ainsi, dans le cadre du *Push*, ce sont plutôt les ressources de gestion, comme les gestionnaires des espaces intelligents, qui envoient les requêtes de gestion vers les appareils/applications. Ceux-ci appliquent ensuite les requêtes et retournent, si nécessaire, des rétroactions aux ressources de gestion. La Figure 1.3 illustre cette méthode.

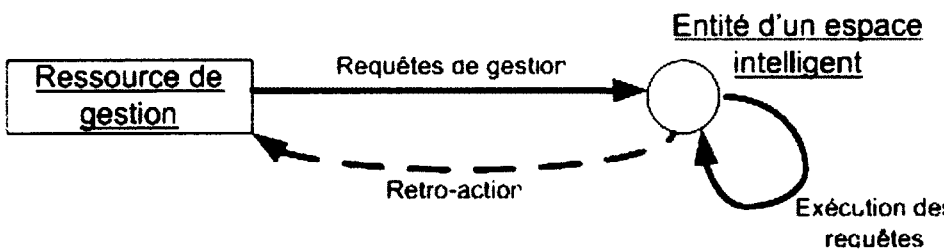

FIGURE 1.3 – Diagramme présentant la méthode *Push* en gestion logicielle

Les avantages et les inconvénients du *Push* sont à l'inverse de ceux du *Pull*. Dans la méthode du *Push*, l'hétérogénéité des milieux n'est pas en soi un obstacle à la gestion puisque cette méthode permet facilement de gérer une entité d'un milieu indépendamment des autres. Le *Push* permet également de réduire le délai entre le besoin exprimé de gestion et la mise en place de la mesure. Toutefois, elle nécessite que les ressources impliquées soient disponibles lors de l'envoi des requêtes de gestion. D'un autre côté, comme nous le mentionnons précé-

demment, le *Pull* est particulièrement utile lors de la gestion d'un groupe important d'entités. Toutefois, l'envoi des requêtes, l'envoi des rétroactions et leur analyse peuvent rapidement devenir complexes et ardus pour des entités et espaces intelligents hétérogènes. En résumé, le *Push* offre un contrôle élevé et une rétroaction rapide, mais est difficile à mettre en place un grand nombre d'entités, alors que le *Pull* est efficace pour un grand nombre d'entités, mais offre moins de contrôles et de rétroactions.

1.3.2 Les stratégies d'Orchestration et de Chorégraphie

Acheminer les requêtes de gestion vers les entités peut être fait en utilisant les méthodes *Pull* et *Push*. Toutefois, afin d'appliquer ces méthodes à un nombre important d'entités, tel que dans les espaces intelligents, il existe principalement deux stratégies de gestion, l'orchestration et la chorégraphie. Ces deux stratégies permettent principalement de gérer la propagation des requêtes, la synchronisation de celles-ci entre les entités et dans certains cas, la prise de décision. Ces stratégies permettent donc de configurer en partie les architectures logicielles des espaces intelligents.

On peut comparer la stratégie d'orchestration logicielle à l'image d'un maestro, la ressource de gestion, et de ses musiciens, les entités à gérer. Celui-ci dirige la gestion en distribuant les requêtes parmi les entités selon les besoins et coordonne les efforts de celles-ci (Figure 1.4). L'orchestration repose essentiellement sur l'utilisation du *Push* comme moyen de distribuer les requêtes de gestion, quoique le *Pull* soit possible à mettre en oeuvre. Cette stratégie très hiérarchisée, même autocratique, ne laisse pas beaucoup de pouvoir aux entités. L'orchestration a le grand avantage de permettre à plusieurs entités d'être dirigées facilement. Elle permet également l'ajout rapide de nouvelles entités à l'ensemble, pourvu que la ressource de gestion ait une certaine connaissance de la nouvelle entité et de ses capacités. De plus, elle permet l'utilisation à la fois du *Pull*, les entités écoutant la ressource de gestion, et du *Push*, les entités recevant les ordres de la ressource.

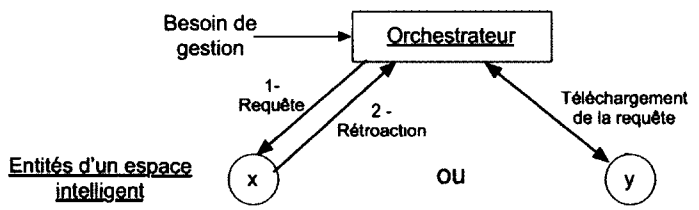

FIGURE 1.4 – Diagramme présentant la stratégie d'orchestration avec la méthode Push et Pull

Toutefois, l'orchestration possède ses défauts. La prise de décision et la coordination étant

centralisées aux mains de l'orchestrateur, sa perte signifie la fin de l'orchestration. Il faut donc inclure des mécanismes permettant d'éviter de telles situations en utilisant par exemple plusieurs ressources de gestion redondantes. Toutefois, ceci amène un autre problème, la synchronisation des ressources de gestion par rapport aux besoins de l'espace, des gestionnaires et des utilisateurs et le reflet de ces besoins dans les entités des espaces intelligents.

La stratégie de la chorégraphie est quant à elle beaucoup moins hiérarchisée que l'orchestration. Dans la chorégraphie, la distribution, la direction et la coordination de la gestion sont faites par les entités elles-mêmes. Cette chorégraphie peut être réalisée à l'aide de comportements prédéterminés dans chaque entité, i.e. chaque entité sait à l'avance comment réagir à un besoin, ou bien improvise en utilisant des comportements prédéterminés et en les adaptant au contexte. Dans tous les cas, une synchronisation dans les mesures de gestion à appliquer doit être établie entre les entités participantes. La Figure 1.5 illustre la chorégraphie, avec ses entités qui communiquent entre elles afin de répondre à un besoin capté par une des entités. Dans les chorégraphies, les méthodes du *Push* et *Pull* sont tout autant utilisées dans la propagation et la synchronisation de la gestion. Il est important de noter qu'en chorégraphie, les entités n'ont pas besoin d'être toutes en communication directe, pourvu qu'il ait un lien de communication indirecte. Les liens peuvent également être dynamiques, ce qui se prête bien aux réseaux ad hoc.

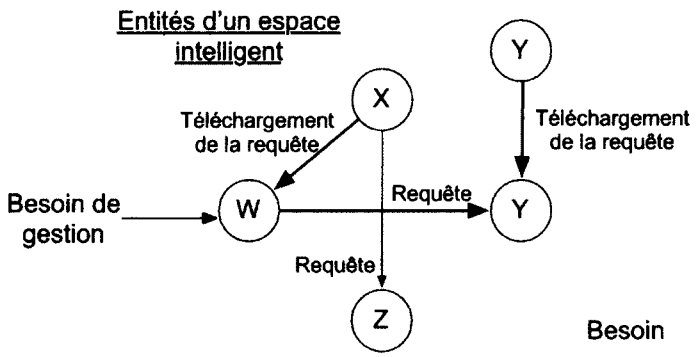

FIGURE 1.5 – Diagramme présentant la stratégie de la chorégraphie

Puisqu'il n'y a pas d'autorité dominante dans la chorégraphie, la perte d'une entité a donc peu d'effets sur le reste de l'ensemble. Cette stratégie est donc plus tolérante aux fautes. De plus, puisque les entités ont plus de pouvoir dans la gestion, cette stratégie est moins autocratique et plus polyvalente. Toutefois, elle est difficile à implémenter, notamment pour un grand nombre d'appareils. La nécessité de synchroniser les entités entre elles augmente de beaucoup le volume de communications et complexifie la mise en place de la stratégie. La chorégraphie

est utilisée dans des approches informatiques telles que les systèmes multi-agents [51], les intelligences collectives et les grilles informatiques (*grid computing*). Elle est également utilisée dans certains protocoles tels que les réseaux avec topologie en treillis et les réseaux ad hoc.

1.4 Revue des solutions de gestion logicielle existantes

Maintenant que les principes généraux et la problématique de la gestion logicielle ont été introduits et que les mécanismes et stratégies de base en gestion ont été expliqués, une revue des solutions de gestion logicielle existantes peut être présentée. Dans un premier temps, un aperçu de quelques travaux de recherche académique et industriel pertinents en gestion logicielle est présenté. Ces travaux proposent des pistes de solutions pouvant être appliquées à la problématique de la gestion logicielle dans les espaces intelligents. Dans un deuxième temps, une revue des solutions commerciales offrant des fonctionnalités de gestion logicielle est présentée. À l'heure actuelle, les principales solutions de gestion logicielle proviennent :

- des systèmes d'exploitation, tels que les *Debian Packages* sous Linux/Debian ;
- des grappes de calcul, où plusieurs appareils doivent être gérés lors de la mise en place d'applications de calculs scientifiques ;
- des plateformes applicatives, permettant le déploiement d'applications sur des systèmes embarqués ;
- les outils de gestion de flottes d'appareils, permettant la gestion logicielle dans des parcs d'appareils ou des flottes d'appareils mobiles.

Plusieurs de ces solutions ont influé sur les choix technologiques et architecturaux qui ont menés à la solution d'organisation logicielle autonome des espaces intelligents présentée au Chapitre 3. Par exemple, la plateforme OSGi [42] (présenté un peu plus loin) a été choisie comme plateforme logicielle permettant, entre autres, de supporter aisément la gestion du cycle de vie des applications, le déploiement de celles-ci et la gestion des dépendances logicielles. Une discussion suit la présentation de cette revue des solutions, présentant plus en détails l'analyse de ces solutions et les aspects de celles-ci qui ont influé sur la conception et l'implémentation de notre solution.

1.4.1 Travaux de recherche en gestion logicielle pour espaces intelligents et autres

Plusieurs travaux de recherche ont été publiés, ces dernières années, portant sur la gestion logicielle dans le contexte de l'informatique diffuse. Ces travaux ont attaqué la problématique présentée précédemment en proposant des mécanismes de description des besoins de gestion (déploiement, mise à jour, etc.), utilisation d'architectures à composantes, système d'organisation des composantes et le déploiement automatisé d'applications.

Dans [52], Becker *et al.* présente PCOM, un système à composantes pour informatique diffuse. Ce système, développé conjointement avec le projet BASE [53] (une plateforme de développement pour petit appareil), offre une plateforme permettant :

- le déploiement d'applications/composantes sur des appareils ayant peu de ressources (plateforme basée sur la machine virtuelle Java J2ME) ;
- la gestion du cycle de vie des composantes ;
- l'adaptation dynamique du contenu applicatif des appareils selon la disponibilité des services d'un milieu.

L'originalité de ce travail repose sur la partie adaptation des composantes et l'utilisation de contrats afin de décrire les besoins en composantes des systèmes. Ainsi, chaque composante (module logiciel regroupant des services et processus de traitement de l'information) définit à l'aide d'un contrat, les services qu'elle offre et ceux qu'elle requiert, par exemple, un dispositif d'entrée tel un clavier ou une composante de messagerie instantanée [52]. Selon la disponibilité des services dans les milieux visités par les systèmes implémentant PCOM, les systèmes téléchargent, si nécessaire, les services/composantes. Si des composantes sont indisponibles, le système peut se mettre en attente, espérant l'arrivée des composantes nécessaires ou bien tenter une autre stratégie de résolution de problèmes. Les stratégies d'adaptation sont issues de l'évaluation des contrats ou peuvent être pré-définies par des développeurs pour répondre à des situations spécifiques. Par exemple, l'utilisation de composantes logicielles liées à l'utilisation d'une souris et d'un clavier virtuel, dans le cas où une un clavier physique est absente du milieu ou d'un appareil. La fonctionnalité d'adaptation est intéressante afin de gérer les dépendances des applications envers des composantes/services. Toutefois, elle n'offre pas de solution permettant de gérer la répartition des composantes parmis plusieurs appareils et n'intègre qu'une sensibilité au contexte des environnements réduite à la découverte des services des milieux.

Le système O2S [54] propose un système similaire à PCOM, basé sur une architecture à composante. L'innovation de ce travail repose sur l'utilisation d'un arbre décrivant les buts de-

vant être atteints par un système afin de déclencher le déploiement de composantes. À travers cet arbre de buts, il est possible de définir des dépendances logicielles envers d'autres composantes, y attacher des actions telles que le déploiement de composantes dans le cas où celles-ci ne seraient pas présentes, vérifier et valider certaines informations contextuelles nécessaires au déploiement de la composante, etc. Pour chaque but défini dans l'arbre, un plan d'action (*Planlet*) y est relié. Lors de l'évaluation du but, O2S vérifie la viabilité du plan et si les contraintes sont respectées, le plan est mis en place. Pour la résolution des contraintes, O2S se base sur un engin de raisonnement et de planification développé en PROLOG. Contrairement à PCOM, O2S permet d'intégrer davantage les informations contextuelles d'espaces intelligents en ajoutant des contraintes de déploiement à l'arbre des buts.

Le projet *EasyLiving* de Microsoft [55], autrefois très médiatisé, propose une architecture permettant de gérer le déploiement d'applications d'assistance dans une maison intelligente. Entre autres, le *EasyLiving Geometric Model (EZLGM)* [20], une fonctionnalité de surveillance des utilisateurs, offre des mécanismes permettant de déterminer quels appareils, pour un espace intelligent donné, sont les mieux disposés pour offrir une interaction homme-machine de qualité avec un utilisateur. Cette fonctionnalité se base principalement sur la position des utilisateurs dans le milieu afin de déterminer les appareils les mieux disposés à recevoir des applications. Le système EasyLiving gère alors le déploiement et le démarrage de l'application sur les appareils visés. Cette fonctionnalité a d'abord été conçue pour permettre la lecture de médias numériques sur des appareils selon la pièce visitée par un utilisateur, mais permet le déploiement d'autres types de contenu. Le projet *EasyLiving* a été l'un des premiers, sinon le premier, projet en informatique diffuse à intégrer, de façon fonctionnelle, une sensibilité au contexte à la gestion des espaces intelligents. Toutefois, sa notion de contexte reste limitée, entre autres, dans sa définition du profil de l'utilisateur.

Plusieurs projets de recherche dans le cadre des grilles informatiques travaillent à la recherche de solution permettant de simplifier la gestion des ressources des grilles et le déploiement des applications sur les noeuds participants à la grille, problème similaire à celui de la gestion logicielle dans les espaces intelligents. Dans [56], Cho propose une solution intelligente de gestion des ressources dans les grilles informatiques basée sur l'utilisation d'ontologie et d'un raisonnement par description logique. Cette solution repose avant tout sur un agent intelligent effectuant, à la demande des clients, des raisonnements sur la disponibilité de la grille informatique. Cette disponibilité est évaluée en fonction d'un ensemble de données, dont les ressources disponibles (processeurs, mémoires), les priorités des clients, les règles de fonctionnement de la grille, etc. L'utilisation d'une ontologie pour représenter les informations

contextuelles offre des possibilités intéressantes : description standardisée des informations, possibilité de raisonner sur le contenu en inférant concepts, propriétés sémantiques et instances via la description logique, engin de recherche de données, etc. Les ontologies dans les espaces intelligents ont été également abordées dans bon nombre de travaux sur l'informatique diffuse, dont certains [57] [38] [58] sont présentés un peu plus loin dans cette thèse .

Les quatre derniers travaux, décrit précédemment, présentent cinq concepts ou fonctionnalités qui ont particulièrement influencé nos travaux. Premièrement, PCOM gère les dépendances fonctionnelles des applications par son système de contrats. Un système permettant de gérer les dépendances de façon automatique permet de réduire de façon considérable la complexité de la mise en place des applications. Deuxièmement, O2S avec sa description de buts et du contexte nécessaire au déploiement de composantes permet d'une part d'automatiser une grande partie des tâches de déploiement et d'une autre part de réduire le besoin des utilisateurs de connaître spécifiquement le système visé par le déploiement. Dans un troisième temps, les architectures à composantes utilisées par PCOM et O2S permettent également une division modulaire des applications, réduisant le couplage fort des applications à l'aide des services, améliorant la réutilisation de code et permettant dans certains cas, le déploiement dynamique, en cours d'utilisation, de nouvelles composantes. Nous verrons un peu loin que d'autres systèmes à composantes/services, tels qu'OSGi et OpenCable, offre des performances et fonctionnalités plus étendues que PCOM et O2S.

Dans un quatrième temps, l'utilisation des informations contextuelles des espaces intelligents et des informations relatives aux utilisateurs par le projet *EasyLiving* permet l'intégration des utilisateurs des milieux dans le processus de déploiement des applications. Elle permet également l'amélioration de la qualité de la livraison d'application (*software provision*) en tenant compte du contexte des milieux visités par les utilisateurs.

Enfin, l'utilisation d'ontologies pour décrire les informations propres à un système tel que dans les travaux de Cho [56], offre un support à la description des concepts et instances de concepts des espaces intelligents. Toutefois, elles offrent surtout la possibilité de décrire les liens sémantiques entre les concepts et instances. Cette sémantique peut être utilisée par des engins de raisonnement afin d'inférer de nouveaux concepts et instances et d'agréger ou d'abstraire des données. Dans le cadre de nos travaux, les ontologies en conjonction avec un engin de raisonnement permettront de trouver les meilleures organisations logicielles pour des groupes d'applications à déployer dans des espaces intelligents donnés. Nous discuterons davantage de ces cinq concepts et fonctionnalités dans les prochains chapitres.

1.4.2 Outils d'installation et de mise à jour

Les systèmes d'opération Linux sont de plus en plus populaires, notamment auprès des professionnels des technologies de l'information ou des utilisateurs experts. Basées sur le noyau Linux, à l'origine créé par Linus Torvalds, et sur les utilitaires du projet GNU, une panoplie de distributions de ce système « à la UNIX » ont été développées : Debian/Ubuntu, Gentoo, Red Hat/Fedora, etc. Entre autres, la distribution Debian met l'emphase sur la distribution des logiciels. Pour ce faire, cette distribution utilise un système de gestion logiciel appelé *Debian Package*(dpkg) [59].

Les *Debian Package* offrent donc une suite d'outils de bas niveaux permettant de télécharger sur des serveurs, des logiciels ou des bibliothèques de programmes. De plus, ce système intègre certains mécanismes permettant de gérer les dépendances existantes entre les composantes, par exemple en téléchargeant et installant dans l'ordre nécessaire les diverses dépendances, puis l'application demandée. De plus, les *Debian Package* gèrent la mise à jour des applications en intégrant une mécanique de vérification des numéros de versions des logiciels, le téléchargement des mises à jour et l'application de celles-ci. D'autres mécanismes semblables existent dans plusieurs autres distributions de Linux comme le *Red Hat Package Manager*(RPM) pour la distribution Red-Hat/Fedora.

La mécanique des *Debian Package* a été initialement créée pour le déploiement d'applications natives. La mécanique des dpkg permet donc d'installer, de désinstaller ou de démarrer des applications localement sur l'appareil, selon les options des paquets téléchargés. Il est toutefois impossible d'utiliser cette mécanique pour connaître les états des applications ou bien les arrêter.

La mécanique des *Debian Package* applique la méthode *Pull*, car ce sont les appareils qui vérifient la présence de mises à jour ou télécharge de nouvelles applications à partir de serveurs d'applications. Puisqu'il n'y a pas de coordination dans la gestion entre les appareils et qu'il n'y a pas de décision prise par une ressource de gestion externe, les Debian Package n'appliquent donc pas de stratégie de gestion.

Enfin, pour les versions du système d'exploitation Microsoft Windows, la mécanique des Windows Update a été développée. Elle fonctionne également sur une mécanique de type *Pull* et permet la mise à jour du système d'exploitation et l'ajout de composantes logicielles additionnelles. Cette mécanique peut être utilisée en mode manuel, i.e. ce sont les utilisateurs qui déclenchent le processus de vérification des mises à jour, ou bien automatique, i.e. le système d'exploitation vérifie périodiquement la présence de contenus sur le serveur de mises à jour.

Les outils d'installation tel que les *Debian Packages* permet d'installer facilement des ap-

plications sur des appareils à partir de serveurs d'applications et offre des fonctionnalités intéressantes de gestion des dépendances. Toutefois, elle ne permet pas d'être directement géré à distance, hormis en les utilisant via une connexion SSH ou Telnet, et ne permet pas de gérer complètement le cycle de vie des applications tels que le démarrage ou l'arrêt des applications. De plus, elle n'offre pas de solution au problème de gestion de plusieurs appareils par espace intelligent et n'est disponible que pour Debian/Ubuntu donc peu utile dans sa version intégrale afin de contrer le problème de l'hétérogénéité des systèmes.

1.4.3 Grappes de calcul

Les grappes de calcul, ensemble d'ordinateurs destiné à l'exécution d'applications distribuées et/ou parallèles, ont des caractéristiques semblables à celles des espaces intelligents. Elles possèdent beaucoup d'appareils et peuvent avoir des liens complexes entre appareils au cours de l'exécution des applications de calculs distribuées. Les gestionnaires des grappes de calcul ont donc des besoins de gestion similaires à ceux des espaces intelligents dans la capacité à gérer facilement et rapidement plusieurs appareils.

Afin de proposer des outils communs, la communauté des développeurs et utilisateurs de grappes de calcul a entre autres développé une plateforme libre, l'*Open Source Cluster Application Resources* (OSCAR) [60], permettant de gérer à distance le déploiement et le cycle de vie des applications de calcul sur les grappes d'ordinateurs. Utilisant le système d'exploitation Linux et supportant plusieurs distributions telles que Debian et Fedora, OSCAR permet l'installation et la configuration de ces systèmes d'exploitation, des applications de calculs et des outils d'administration sur les noeuds de la grappe de calcul. Pour ce faire, il créé des images de disques sur mesures, selon les besoins en calcul des gestionnaires, et gère la distribution de ces images sur les ordinateurs de la grappe de calcul. Ensuite, il offre aux gestionnaires les outils pour administrer le démarrage des applications de calcul, le monitoring du calcul et l'arrêt de celui-ci.

Dans la gestion de la grappe de calcul, OSCAR utilise strictement la méthode *Push*, où les images de disque et les requêtes de gestion sont envoyées par l'entité gestionnaire vers les ordinateurs de la grappe. C'est aussi l'entité gestionnaire qui gère la coordination et commande les ordinateurs, OSCAR applique donc la stratégie d'orchestration.

La suite d'outils OSCAR complète les lacunes des Debian Packages en permettant de distribuer à distance les installations Linux sur plusieurs appareils et de gérer par la suite l'exécution des applications de calcul. Toutefois, OSCAR a été conçu pour le contexte des grappes

de calcul où les appareils et les applications déployés sont homogènes. Son utilisation dans des milieux hétérogènes, comme les espaces intelligents, est donc difficilement réalisable.

1.4.4 Plateformes applicatives

L'Open Cable Application Platform

L'*Open Cable Application Platform* (OCAP) est une plateforme logicielle, basée sur le standard *Globally Executable Multimedia Home Platform* (GEM-MHP) [61], permettant de gérer et de livrer des services télévisuels chez des clients de télévision câblée. Conçu par le *Cable Television Laboratories*, un consortium de recherche en câblodistribution, OCAP est à la fois un système d'exploitation pour passerelle d'accès à la télévision câblée et un intergiciel pour la gestion des services chez les clients.

Développé à partir du langage orienté objet Java version Micro Édition(J2ME), OCAP est un intergiciel pouvant être déployé sur plusieurs types de passerelle, pourvu qu'il y ait une machine virtuelle J2ME fonctionnelle correspondante au matériel. À l'aide du câble télévisuel, il est possible d'envoyer des requêtes de gestion ou de mise à jour de composantes logicielles vers les passerelles OCAP. Cet intergiciel permet un contrôle total du cycle de vie des applications par l'installation, le démarrage, l'arrêt, la désinstallation et la mise à jour des applications. OCAP utilise majoritairement la méthode *Push* pour acheminer les requêtes de gestion, i.e. que les requêtes sont envoyées par les gestionnaires vers les passerelles. Il est d'ailleurs important de noter qu'il est possible de gérer indépendamment chaque passerelle ou bien de les gérer globalement. De plus, la stratégie de gestion adoptée est l'orchestration, i.e. que ce sont les gestionnaires qui gèrent le contenu et la synchronisation des passerelles. Il n'y a d'ailleurs aucune communication directe entre les passerelles, ce qui empêche l'utilisation d'une stratégie de chorégraphie. Enfin, en plus de la gestion des logiciels, la plateforme OCAP intègre des mécanismes de facturation de services et offre des fonctionnalités permettant de récolter des statistiques d'utilisation.

L'intergiciel OCAP permet de gérer à distance le cycle de vie complet des applications déployées sur des passerelles d'accès à la télévision câblée. Il offre également des outils intéressants de journalisation et de surveillance des applications déployées. Toutefois, à l'image d'OSCAR, cette solution a été conçue pour un type de système précis, les passerelles télévisuelles, et convient plus ou moins bien au contexte hétérogène des espaces intelligents. Toutefois, son architecture et ses fonctionnalités permettant de gérer aisément plusieurs passerelles et leurs applications à la fois doivent être retenue.

OSGi

L'OSGi Alliance[1], anciennement connu sous le nom de l'Open Services Gateway initiative, est un regroupement d'entreprises qui se sont unies en 1999 pour créer des standards ouverts de plateformes logicielles. On compte parmi ses membres les compagnies Ericsson, Nokia, Siemens, IBM et biens d'autres. Ainsi, l'OSGi Alliance propose des spécifications [42] de plateformes Java appliquant l'approche orientée service. Ces spécifications ont adopté dès le départ des mécanismes permettant de gérer tous les aspects du cycle de vie des modules, incluant un mécanisme de mise à jour, téléchargeant les nouvelles versions des modules via des serveurs d'applications.

Les spécifications OSGi reflètent les besoins des entreprises membres du consortium de pouvoir déployer et gérer facilement des logiciels dans leurs produits commerciaux. Ainsi, des entreprises telles que Volvo, Ericsson et Philips utilisent OSGi comme plateforme de déploiement de services dans leurs produits (voitures, téléphone portable, système multimédia), permettant ainsi une gestion logicielle plus efficace. Fait important à noter, l'IDE Eclipse est également basé sur la plateforme OSGi et son concept de plug-in est une extension de la mécanique modulaire d'OSGi.

Tout comme OCAP, la spécification OSGi repose sur la technologie Java, une machine virtuelle Java doit donc être présente sur l'appareil accueillant la plateforme. Dans la spécification OSGi, les différentes applications déployées sont divisées en modules, contenus dans des fichiers de type jar, nommés *bundles*. Ces modules s'échangent entre eux des services et du code afin de former les fonctionnalités de l'application : interface homme-machine, gestion de la persistance des données, communication externe, etc. Un service, dans la plateforme OSGi, est un objet offrant des méthodes pouvant être exécutées par les autres modules. L'intérêt de l'approche orientée service permet un dynamisme plus important entre les modules, par exemple en intégrant dynamiquement de nouveaux modules aux applications existantes, et une gestion indépendante des modules, permettant la mise à jour de parties précises des applications, tout en réduisant les effets de bord des actions de gestion sur la continuité de service des applications. L'exécution et l'échange des données entre modules sont régis par plusieurs couches définies dans la spécification. Ainsi, OSGi divise son architecture en quatre couches :

- la couche Sécurité : gérant la validité du code des modules ;
- la couche Module : gérant le chargement du code et l'exécution de celui-ci ;
- la couche Cycle de vie : permettant de gérer l'état des divers modules ;

[1] Site web de l'OSGi Alliance : http ://www.osgi.org/

– la couche Service : permettant l'échange de services entre les divers modules.

Dans la version quatre de la spécification OSGi, aucun mécanisme de gestion à distance des modules n'est intégré. Toutefois, il est relativement aisé d'en développer et quelques modules permettant la gestion via une interface web, par *Telnet* et par SOAP ont été développés par la communauté de développeur OSGi. En considérant la spécification de base et sa fonctionnalité de mise à jour, on peut affirmer qu'OSGi utilise la méthode *Pull*. Toutefois, l'utilisation des modules de gestion à distance permet le mode *Push*. Enfin, puisque les mécanismes de gestion d'OSGi concernent les applications déployées localement sur la plateforme et non entre plateformes, il n'y a donc pas de stratégie de gestion. En utilisant les modules de gestion à distance cités plus haut, on tend alors vers une stratégie d'orchestration, où le gestionnaire coordonne et commande la gestion des plateformes OSGi.

Plusieurs spécifications additionnelles à OSGi existent. Deux de ces spécifications sont particulièrement intéressantes face à notre problématique : *l'OSGi Bundle Repository* (OBR) et *distributed OSGI* (dOSGi). La première spécification définit une mécanique permettant de définir les dépendances fonctionnelles de modules OSGi envers d'autres modules. Un mécanisme de résolution des dépendances est également spécifié permettant de gérer automatiquement l'installation et le démarrage, si nécessaire, des dépendances d'une application. Sans être aussi complet que les Debian Package, OBR offre donc des fonctionnalités permettant de réduire de façon significative la difficulté de la gestion des dépendances fonctionnelles. La spécification dOSGi quant à elle définie la mécanique permettant à des modules OSGi d'offrir des services aux plateformes OSGi déployées sur d'autres matériels via des web services. De plus, la création des web services et des *stubs*[2] chez les plateformes clientes est automatisée par dOSGi. Les fonctionnalités de dOSGi permettent, entre autres, le développement de services offrant les fonctionnalités de gestion du cycle de vie des modules déployés sur la plateforme OSGi

La plateforme à service OSGi offre des fonctionnalités permettant de contrôler entièrement le cycle de vie des applications qui y sont déployées. Ses capacités orientées services et la possibilité de modulariser les applications offrent des possibilités très intéressantes pour la continuité de service. OSGi permet également un dynamisme dans l'intégration des nouvelles applications qui est inégalé par les solutions précédentes, en permettant l'ajout de composantes logicielles à la plateforme en cours d'exécution du système. À la fois un avantage et un inconvénient, l'utilisation du langage Java permet à OSGi d'être utilisé sur un nombre important d'appareils, ce qui est important dans le contexte hétérogène des espaces intelligents. Par

[2]Un stub est un objet permettant de représenter les fonctionnalités d'un service distant chez un système client et d'abstraire la mécanique de communication avec le système distant

contre, quoi qu'il soit possible d'utiliser des bibliothèques natives via le *Java Native Interface* (JNI), OSGi est plutôt cantonné aux applications utilisant la technologie Java. Pour les systèmes embarqués de faible puissance, la disponibilité de machine virtuelle Java est faible (via les JVM Squawk ou Dalvik d'Android si le matériel est supporté).

1.4.5 Outils de gestion de flottes d'appareils

Il existe sur le marché quelques solutions pour la gestion logicielle de flottes d'appareil. Comparativement aux grappes de calcul ou aux passerelles d'accès télévisuelles, les flottes d'appareils sont en général composées d'appareils hétérogènes qui peuvent être à la fois fixe ou mobile et avoir à la fois beaucoup de capacité de traitement ou peu.

Premièrement, il existe les solutions tel que *Symantec PcAnywere*[3], permettant aux gestionnaires d'accéder à distance à des appareils comme s'ils y étaient localement. Ces logiciels permettent également d'accéder à des appareils ayant plusieurs types de système d'exploitation : Windows, Mac OS et Linux. Ces solutions implémentent la méthode *Push* et puisque ces solutions n'offrent pas de synchronisation entre les appareils de la flotte, elles n'appliquent pas de stratégie de gestion. Les solutions du type *PcAnywere* sont largement utilisées dans les entreprises pour la gestion des serveurs ou l'assistance aux utilisateurs. Elles ont comme principal avantage de permettre un accès visuel à un appareil distant et d'y avoir un contrôle total sur ses fonctionnalités et ses applications. Toutefois, elles ne proposent pas de mécanismes permettant de gérer facilement plusieurs appareils ou applications à la fois.

Afin de faciliter la gestion de plusieurs appareils à la fois, des solutions telles que le *Prosyst mPower Remote Manager*[4] et *Makewave Ubicore*[5] ont été développées. Toutes deux basées sur la technologie OSGi, qu'elles complètent en offrant une mécanique de gestion à distance, ces solutions offrent des fonctionnalités permettant de gérer les modules OSGi à la fois sur plusieurs appareils ou indépendamment sur chaque appareil. Ces deux solutions sont comparables à OCAP, mais permettent plus de versatilité en terme de type d'appareil. De plus, elles offrent des services de surveillance de l'activité des appareils, de journalisation, d'aide à la livraison de contenu et de gestion de la facturation. Ces deux solutions offrent donc des capacités de gestion des plateformes OSGi beaucoup plus étendue que la spécification de base d'OSGi. Ces deux solutions utilisent à la fois les méthodes *Pull* et *Push*. Elles implémentent également la stratégie d'orchestration puisque le processus décisionnel est géré par l'entité de gestion.

[3]Site web de PcAnywere : http ://www.symantec.com/business/pcanywhere

[4]Site web de Prosyst : http ://www.prosyst.com/products/back_end_mgmt.html

[5]Site web de Makewave : http ://www.makewave.com/site.en/products/ubicore$_o sgi.shtml$

Les solutions *Prosyst mPower Remote Manager* et *Makewave Ubicore* permettent de gérer à la fois plusieurs plateformes OSGi sur plusieurs appareils hétérogènes. Toutefois, le contexte d'utilisation de ces deux produits ne répond pas en totalité aux besoins des espaces intelligents tels que la réduction de l'impact des fautes et bris sur les utilisateurs et la réduction de la complexité de la gestion de la sécurité dans les espaces intelligents. De plus, quoiqu'elles réduisent la complexité de la gestion de plusieurs applications sur plusieurs appareils, ils restent que l'utilisation de ces deux solutions nécessite des connaissances poussées en informatique, leur utilisation est donc plus ou moins adaptée aux utilisateurs non experts. Enfin, le *mPower Remote Manager* et *Ubicore* simplifient la gestion des plateformes OSGi en permettant de gérer le cycle de vie des applications à distance, mais leur utilisation nécessite un grand nombre d'opérations et des connaissances approfondies sur les systèmes gérés et leur contexte d'utilisation.

1.5 Discussion

Comme la section précédente le démontre, il existe sur le marché plusieurs solutions offrant des fonctionnalités de gestion logicielle. Toutefois, ces solutions sont plus ou moins adaptées aux caractéristiques des espaces intelligents et ne répondent pas toutes à la problématique de la gestion logicielle dans ces espaces (1.2).

Pour réduire de façon importante la difficulté et la complexité de la gestion logicielle dans les espaces intelligents, il est nécessaire de réduire de façon notable le nombre d'actions de gestion faites par les gestionnaires et utilisateurs, i.e. moins de manipulation du cycle de vie des applications, moins d'opérations de maintenance, moins de gestion de la sécurité, etc.

Pour ce faire, nous croyons qu'il est nécessaire de laisser une partie du processus de gestion logicielle aux espaces eux-mêmes, i.e. laisser plus d'autonomie aux espaces dans la gestion de leurs logiciels. Il est donc nécessaire d'automatiser le plus d'actions de configuration, de maintenance et de manipulation du cycle de vie des applications possible. Pour y arriver, l'implémentation de mécanismes intelligents qui s'adaptent au contexte des espaces versus les besoins de gestion et qui tiennent compte du profil des utilisateurs est nécessaire. Les travaux académiques présentés précédemment (PCOM, O2S, *Easyliving*, travaux de Choi) offre des pistes de solutions à l'autonomie recherchée avec :

- la description des dépendances fonctionnelles des applications (PCOM) ;
- la description des besoins matériels et/ou contextuels des applications (02S) ;
- l'utilisation d'un système à composantes/services (PCOM et O2S) ;

- l'utilisation de l'information contextuelle et du profil de l'utilisateur (*Easyliving*) ;
- l'utilisation d'ontologies décrivant les informations contextuelles et d'agents intelligents exploitant les données des ontologies (Cho).

La plateforme OSGi répond à deux de ces pistes de solution avec son architecture et ses fonctionnalités orientées services/composants, et la gestion des dépendances fonctionnelles des applications avec l'API OBR. La création de service de gestion à distance (dOSGi), d'outils de journalisation et de surveillance inspiré d'OCAP, et d'applications de configuration assistée telle que dans OSCAR, peaufineraient les fonctionnalités de OSGi et permettraient à cette solution de mieux répondre à la problématique de la gestion logicielle des espaces intelligents.

Il est également important de synthétiser et de réduire l'information retournée aux gestionnaires et aux utilisateurs, ainsi que d'adapter la présentation de cette information selon le profil du lecteur. Ceci permet d'améliorer la compréhension de l'information par les acteurs et d'accélérer la prise de décision par rapport à ces informations. Les solutions *Prosyst mPower Remote Manager* et *Makewave Ubicore* synthétisent jusqu'à un certain point les informations des systèmes gérés, mais s'adressent à une clientèle d'experts. Dans le cadre d'espaces intelligents dédiés à l'amélioration de la qualité de vie et de l'autonomie de personnes dépendantes, d'autres types d'utilisateurs/gestionnaires auront à gérer sporadiquement les logiciels de ces espaces. Par exemple, un ergothérapeute pourrait avoir à déployer une nouvelle application d'assistance dans le milieu lorsque son client/patient éprouve des difficultés à la réalisation d'une tâche particulière. Ultimement, il serait intéressant que les aidants professionnels et même les utilisateurs des milieux puissent s'affranchir du personnel technique.

Toutes les solutions présentées dans la revue précédente se basent sur l'orchestration. Ainsi, OCAP, OSCAR, *Prosyst mPower Remote Manager* et *Makewave Ubicore* se base sur l'existence d'entités orchestrant les tâches de gestion par l'envoi de requête vers les noeuds participants. Or, l'utilisation de la chorégraphie dans la gestion logicielle peut apporter plus d'autonomie aux espaces intelligents en implémentant une intelligence répartie parmi les noeuds participants (système multi-agent), améliorant la tolérance aux fautes puisqu'il y a peu ou pas d'entités centralisées vulnérables aux fautes. Toutefois, comme nous l'avons écrit précédemment dans ce chapitre, cette stratégie de gestion est difficile à implémenter, notamment quand il y a beaucoup d'entités participantes. Pour un système sensible au contexte, l'utilisation de la chorégraphie nécessite un ensemble de mécanismes gérant la synchronisation des données entre les noeuds, la gestion des fautes, la sécurité, etc. De plus, les avantages de la chorégraphie n'ont pas, à ce jour, répondu à des besoins tangibles dans le marché de la gestion logicielle amenant des entreprises à envisager cette stratégie. Toutefois, dans le cadre de nos travaux de

thèse et des espaces intelligents, une utilisation mixte de l'orchestration et de la chorégraphie peut amener à une augmentation de l'autonomie des entités dans leurs actions de gestion, sans augmenter de façon significative la difficulté d'implémentation de la stratégie.

En résumé, afin de répondre à la problématique de la gestion logicielle dans les espaces intelligents, il est nécessaire de réduire la complexité et le nombre de tâches de gestion, réduire le besoin pour les utilisateurs de connaître les informations détaillées propres aux systèmes des milieux et offrir des outils de gestion efficients pouvant être utilisés par des utilisateurs experts et non experts. La plateforme OSGi offre les solutions permettant de manipuler aisément le cycle de vie des applications et gère les dépendances fonctionnelles des applications automatiquement. Une solution inspirée de l'approche d'orchestration de *Prosyst mPower Remote Manager* et *Makewave Ubicore*, en y ajoutant une part de chorégraphie, permettrait de gérer aisément plusieurs appareils situés dans plusieurs espaces intelligents.

La réduction significative de la complexité des tâches de gestion et du volume de données devant être analysée par les utilisateurs/gestionnaires passe par la création d'une intelligence sensible au contexte des milieux, des besoins des applications, des ressources des appareils et du profil des utilisateurs. Cette intelligence permettra de transférer la majorité du travail de gestion vers les systèmes des espaces intelligents, rendant les espaces intelligents plus autonomes dans leur gestion logicielle. Pour nous guider dans la conception et l'implémentation d'une telle solution, l'approche de l'informatique autonome ou *autonomic computing* [28] apporte des concepts qui répondent justement à nos besoins de donner plus d'autonomie aux espaces intelligents dans leur configuration, leur maintenance et leur sécurité. Cette approche et la façon dont elle peut répondre à notre problématique, sont présentées dans le chapitre suivant (Chapitre 2).

Chapitre 2

L'informatique autonome

Comme nous l'avons abordé dans le chapitre précédent, la complexité des systèmes d'informations actuels rend de plus en plus difficiles et coûteux leur gestion et leur déploiement. Face à ce problème, IBM a lancé en 2001 une initiative nommée informatique autonome ou *autonomic computing* [28] [62]. Cette approche vise à réduire la complexité de la gestion des systèmes d'informations en rendant ceux-ci plus autonomes dans leur gestion. Les systèmes appliquant l'informatique autonome s'adaptent eux-mêmes aux changements de leur contexte et cherchent constamment à améliorer leur performance. La complexité des mécanismes implémentant ces fonctionnalités demeure cachée aux utilisateurs et aux gestionnaires.

Pour arriver à ses fins, l'informatique autonome repose sur quatre concepts clés, chacun s'adressant à une des problématiques de la gestion des systèmes informatiques :

- l'auto-configuration ou *self-configuration* : la configuration et la gestion automatique des applications et appareils ;
- l'auto-réparation ou *self-healing* : la détection et la correction automatique des fautes ou bris ;
- l'auto-optimisation ou *self-optimization* : mécanisme proactif cherchant constamment à améliorer les performances des systèmes ;
- l'auto-protection ou *self-protection* : l'identification et la résolution autonome des dangers dans la sécurité des systèmes.

Ainsi, l'auto-configuration permet aux gestionnaires de déployer des applications complexes en seulement quelques actions simples. Cette facilité de configuration est assurée par des mécanismes intelligents s'occupant de la complexité des déploiements. L'auto-réparation et l'auto-protection rendent les systèmes informatiques plus sécuritaires et moins fragiles face aux fautes ou bris pouvant se produire à tout moment. Finalement, l'auto-optimisation permet

aux systèmes d'être plus proactif, en cherchant constamment des opportunités de se rendre plus performant, par exemple, en équilibrant la charge de calcul entre les entités d'un système informatique.

Afin d'implémenter ces quatre concepts, l'informatique autonome utilise une stratégie de chorégraphie, où chaque entité possède ses propres mécanismes d'autonomie. Ces mécanismes sont représentés par une boucle de contrôle (Figure 2.1[1]), la MAPE-K (*Monitoring, Analyze, Planning, Execution - Knowledge*), divisé en quatre étapes et centré autour d'une base de connaissances :

1. la surveillance de nouveaux événements ;
2. l'analyse des évènements ;
3. planification d'une réponse aux événements ;
4. exécution de la réponse.

Pour chaque événement capté par une entité, celui-ci est analysé et si nécessaire, une action est entreprise en réponse à cet événement. De plus, certains types d'événement, concernant l'ensemble du système, sont transmis aux autres entités. Une décision commune est alors prise face à ces événements à l'aide de mécanismes de négociation inter-entités. Par exemple, la détection d'une faute logicielle lors de l'exécution d'une application déployée sur un appareil, déclencherait l'analyse de la faute logicielle, la planification de mesures permettant de diminuer ou sinon d'éliminer l'impact de la faute avec, par exemple, le redémarrage de l'application ou le déploiement d'une version antérieure plus stable. L'analyse et la planification des mesures reposent sur les informations contenues par la base de connaissances.

2.1 L'informatique diffuse autonome

L'approche de l'informatique autonome est principalement centrée sur la résolution de la complexité des systèmes d'information dans un contexte d'entreprise. Elle vise donc des systèmes informatiques ayant d'importantes capacités de traitement de l'information, qui sont relativement homogènes, qui sont fixes, qui ne tiennent pas vraiment compte du contexte où ils se trouvent et qui n'interagissent pas directement avec les utilisateurs. Les espaces intelligents représentent un contexte d'application totalement différent. Pour cette raison, l'informatique autonome doit être adaptée aux espaces intelligents et à l'informatique diffuse : l'informatique diffuse autonome [30].

[1]Cette figure est extraite de [28]

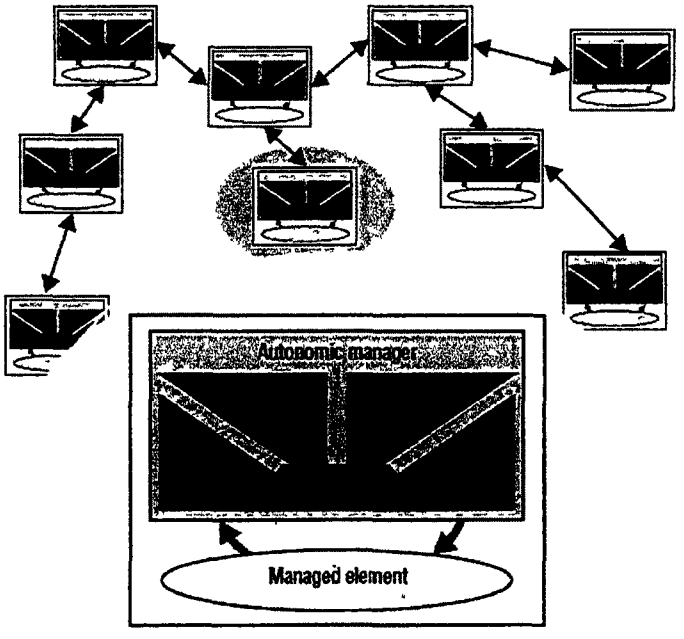

FIGURE 2.1 – La boucle de contrôle de l'informatique autonome (Kephart 2003)

2.1.1 Travaux en informatique diffuse autonome

Un nombre important de travaux de recherche existent sur l'informatique autonome dans les systèmes de technologies de l'information. À l'heure actuelle, plusieurs recherches utilisent l'approche autonome dans la gestion des web services [63], des réseaux de communication [64], des bases de données [65] ou de la sécurité des systèmes [66]. De plus, l'équipe d'IBM sous la direction de Jeffrey Kephart propose également des architectures et outils d'implémentation de l'informatique autonome [67] [68]. Or, la plupart de ces travaux sont axés sur leur domaine d'application respectif et leurs résultats sont difficilement réutilisables dans le contexte des espaces intelligents.

Toutefois, plusieurs travaux apportent des pistes de solutions intéressantes pouvant être appliquées et adaptées aux espaces intelligents. Par exemple, l'utilisation de l'informatique réflective dans la recherche de solutions de configuration, d'optimisation, de réparation ou de protection, permet une dynamique et une adaptation dans la réponse aux événements se produisant dans les espaces intelligents [69]. Tel que nous l'avions abordé dans le chapitre précédent, l'utilisation d'ontologies et de représentations sémantiques du contexte des espaces intelligents permet de définir la base de connaissances et d'implémenter une partie de l'intelligence néces-

saire aux fonctionnalités autonomes. Quelques travaux portant sur l'informatique autonome ont basé leur solution sur les ontologies [70] [71].

Enfin, quelques travaux phares ont porté directement sur l'application de l'informatique autonome à l'informatique diffuse. Ces travaux ont été des sources d'inspirations importantes pour notre travail de thèse. Ils ont également agi comme point de comparaison, permettant de dégager les innovations de notre solution par rapport aux solutions proposées par ces travaux. Les prochains paragraphes introduisent ces travaux.

Projet AMUN

L'un des travaux les plus importants sur l'informatique diffuse autonome a été entrepris par Wolfgang Trumler *et al.* de l'Université d'Augsburg en Allemagne. Dans le cadre du projet *Autonomic Middleware for Ubiquitous eNvironments* (AMUN) [32], les auteurs proposent un intergiciel permettant de faciliter la gestion et l'utilisation des environnements intelligents en intégrant des fonctionnalités basées sur l'informatique autonome. Cet intergiciel adopte une architecture fortement basée sur l'approche initiale de Kephart et Chess [28]. AMUN intègre ainsi une boucle de contrôle plus ou moins basée sur la MAPE-K, avec :

- un module de surveillance des événements systèmes ;
- une base de connaissance divisée en trois catégories : les informations propres aux applications (applications déployées, ressources nécessaires), les informations propres aux événements provenant des éléments surveillés (événements passés) et des métriques sur le système même (utilisation du processeur, utilisation du réseau, etc.) ;
- des algorithmes de contrôle ;
- un module de configuration utilisant les informations systèmes et les algorithmes de contrôle pour mettre en place les mesures en réponse aux événements captés.

Les processus autonomes entre appareils d'AMUN sont basés sur une stratégie de chorégraphie, où chaque entité de l'environnement possède son propre gestionnaire d'autonomie (boucle de contrôle), auquel sont greffés des modules permettant de gérer les communications avec les partenaires. Les processus de déploiement des applications sur les noeuds participants à l'intergiciel sont gérés par l'ensemble des noeuds, à l'aide d'un protocole de négociation. Dans le cadre de leur travail, les applications à déployer sont tout simplement des collections de classes Java archivées dans des fichiers compressés de type *jar/zip*. AMUN a, entre autres, été utilisé pour déployer les applications du projet *Smart Doorplate* [32], projet proposant des informations et de l'assistance sur des écrans disposés sur les portes de bureaux d'entreprise.

Dans [72], les auteurs décrivent plus en détails ce protocole de négociation original basé sur les comportements sociaux lors de la distribution de tâches dans un groupe (organisation logicielle). Dans le cadre de ce protocole, une entité coordonnatrice (étant elle-même un des noeuds participants), distribue la liste des applications à déployer aux autres entités. Celles-ci évaluent alors leur capacité à exécuter chacune des applications et, à tour de rôle, communiquent ses capacités aux autres entités via l'entité coordonnatrice. La capacité ou qualité de service des entités est annotée à l'aide d'une valeur numérique $[0, 1]$ correspondant à la moyenne des qualités de service par type de ressource d'une entité. Cette qualité de service par ressource est calculé à l'aide de cette formule :

$$quos = 1 - \frac{rr - r}{R}$$

où r est la quantité de ressource nécessaire à un service, rr est la quantité de ressource restante et R la quantité de ressource totale. Si une autre entité croit pouvoir mieux faire que cette dernière, elle communique alors sa capacité et ainsi de suite. Tout au long des négociations, l'entité coordonnatrice note les entités ayant les meilleures capacités et dresse ainsi une configuration. Une fois le processus de négociation terminé, cette configuration est envoyée à tous les noeuds et appliquée par ceux-ci. Dans [73], Trumler *et al.* décrivent une mécanique permettant d'équilibrer la charge de calcul de composantes distribuées entre elles, et ce, de façon dynamique. Les fonctionnalités d'auto-réparation d'AMUN ne possèdent qu'un ensemble de solutions limitées : relancer dans un premier temps l'application et s'il y a toujours une faute, redéployer l'application sur une autre composante. Enfin, AMUN n'intègre aucun aspect de l'auto-protection.

Projet Gaia

Dans le cadre du projet Gaia, Anand Ranganathan et Roy Campbell de l'Université de l'Illinois à Urbana-Champaign proposent eux aussi des mécanismes basés sur l'informatique diffuse autonome [33]. Face à la complexité de la configuration, ces derniers proposent une solution basée sur l'algorithme de planification STRIPS. Cet algorithme est utilisé afin de rechercher une solution de déploiement basée sur les besoins des utilisateurs, de démarrer des multimédias sur les appareils (dispositifs d'affichage, haut-parleurs, lecteurs audios, etc.) présents dans une salle de conférence/présentation. Basée sur une stratégie d'orchestration, où une entité dirige la configuration, la méthode proposée dans GAIA pourrait être adapté, avec plusieurs modifications à la mécanique interne, à d'autres contenus que les multimédias telle que

des applications, services et modules.

Ranganathan et al. [57] propose également une autre solution d'auto-configuration, basée cette fois-ci sur les ontologies et la comparaison sémantique. Dans cette solution, les buts des utilisateurs sont tout d'abord adaptés au contexte de l'espace. Par exemple, si l'utilisateur exprime le besoin de contrôler un multimédia à partir d'un appareil mobile, le système traduit ce besoin par la liste des appareils mobiles susceptibles de répondre au besoin. Une fois la liste produite, une comparaison sémantique est faite entre les besoins précis de l'utilisateur et les descriptions des entités de la liste. De cette comparaison émerge la configuration que doit prendre l'espace intelligent pour répondre aux besoins des utilisateurs.

Enfin, les auteurs proposent également une solution d'auto-réparation, surveillant les appareils des espaces à l'aide d'une mécanique basée sur les battements du coeur. Les battements du coeur sont simulés par des baux envoyés par les appareils et renouvelés sans cesse avant leur échéance. L'arrêt involontaire d'un battement signifie alors une faute et déclenche alors une reconfiguration où les multimédias ou contrôleurs, qui étaient présents sur l'appareil en faute, sont redéployés sur d'autres composantes de l'environnement. Cette mécanique d'auto-réparation est issue des travaux de Chetan sur la tolérance aux fautes dans les espaces intelligents [49].

Autres travaux

Aly Syed *et al.* [74] proposent une architecture permettant d'organiser de façon autonome des processus parmi des appareils d'un système basé sur l'informatique diffuse. Pour ce faire, les auteurs proposent l'utilisation d'une intelligence basée sur une base de connaissances divisée en quatre types de données : des recettes, des capacités, des règles et des propriétés. Les recettes définissent des contextes sur lesquels le système doit réagir en appliquant des règles, réaction à un contexte particulier. Les capacités servent à définir les appareils participant au système et leurs fonctionnalités. Enfin, les propriétés sont liées aux capacités où elles définissent la présence ou non de périphériques, de fonctionnalités, etc. Par exemple, lors l'arrivée d'une requête demandant la lecture d'une chanson sur l'un des systèmes multimédias d'un espace intelligent, le système compare le contexte de la requête avec les contextes de la base de recettes. Dans le cas où, les conditions de la recette sont vérifiées, soit la présence d'un appareil muni d'un lecteur musical (propriété et capacité), une règle de déploiement est mise en place. Semblable à la solution de Ranganathan, l'approche de Syed repose sur l'orchestration. Le coordonnateur, l'intelligence et la base de connaissances sont centralisés sur un système présent dans chaque espace intelligent. La principale lacune des travaux de Syed *et al.* est le manque

de dynamisme dans l'acquisition et la construction de la base de connaissances. Celle-ci est définie au démarrage du système et n'est pas modifiée ou améliorée aux furs et à mesure des manipulations sur le système.

Martin Feeney et Richard Frisby du *Telecommunications Software and Systems Group*, en Irlande, propose, quant à eux, une architecture offrant des capacités d'auto-réparation [75] pour les espaces intelligents. Basé sur la plateforme OSGi et les outils autonomes d'IBM [76], tel que la boucle de contrôle, leur solution permet la détection des fautes sur un appareil et sa résolution à l'aide d'une base de connaissances et de règles de réparation. Toutefois, la solution de Feeney et Frisby ne tient pas compte du contexte distribué des espaces intelligents. Le système proposé par Feeney et Frisby a été conçu pour être exécuté sur un seul appareil, un serveur, disposé dans une maison intelligente. Leur solution ne gère donc que les fautes logicielles ou les bris physiques mineurs se produisant localement sur l'appareil où est situé le processus d'auto-réparation et ne comprend pas de mécanisme de détection de fautes entre appareils.

Complémentaire aux travaux sur l'informatique autonome, les travaux de Sumi Helal, de l'Université de Floride, entre autres, porté sur l'auto-intégration des appareils aux espaces intelligents [17] [77], offrant également des solutions intéressantes à la problématique de la gestion des espaces. L'auto-intégration est fortement liée à l'auto-configuration : une configuration autonome des espaces intelligents passe inévitablement par une intégration facile, rapide et, si possible, automatique des appareils et matériels aux espaces intelligents. Pour ce faire, Helal et son groupe de recherche, le *Mobile and Pervasive Computing Laboratory*, ont travaillé à la conception de mécanismes permettant d'intégrer le paradigme *Plug and Play* (PnP) aux espaces intelligents. Par exemple, l'auteur utilise des étiquettes identifiables par fréquence radio (RFID) sur les prises des appareils électriques, conjointement avec des lecteurs RFID sur les prises d'alimentations des espaces intelligents, afin de connaitre la disposition des appareils électriques dans les milieux. Helal propose également l'utilisation d'outils de développement graphique [34] afin de programmer graphiquement, à l'aide de composantes faisant l'abstraction des fonctionnalités internes des modules applicatifs. Bref, il s'agit de relier par des connecteurs graphiques des modules et des appareils afin de former des applications pouvant, par exemple, allumer les lumières d'une pièce lors de la détection d'un utilisateur dans cette dernière. Cette méthode de conception jumelée au paradigme PnP, permet la conception et l'intégration aisée des applications dans des espaces intelligents. Elle permet également à des développeurs de gérer facilement les dépendances d'une application envers d'autres modules ou bien du matériel des espaces intelligents.

C'est donc autour de ces travaux que nos réflexions ont été orientées. Notre travail de recherche vise donc à regrouper, compléter et étendre cet état de l'art en proposant une vision différente et de nouvelles approches dans la recherche d'autonomie dans la gestion autonome des espaces intelligents. Fait marquant, aucune de ces solutions n'inclut l'utilisateur avec ces capacités et préférences dans leurs fonctionnalités autonomes. Comme nous l'avons mentionné auparavant dans l'introduction de la problématique de la gestion logicielle des espaces intelligents, les utilisateurs des espaces ont un rôle et une place prépondérante dans le milieu et son système. La solution proposée dans le cadre de notre travail de thèse vise, entre autres, à intégrer le profil de l'utilisateur dans le processus d'organisation logicielle via les préférences et modalités d'interaction des ceux-ci.

De plus, les raisonnements autonomes dans les travaux présentés précédemment sont stricts et ne laisse pas de place à la demi-mesure. Ainsi, dans les travaux de Ranganathan, Trumler et Syed portant sur l'organisation logicielle, la disponibilité d'un appareil en rapport aux besoins d'une application correspond à une valeur binaire : les appareils peuvent ou non accueillir une application. Dans la réalité, la disponibilité d'un appareil face à des applications est plutôt reliée à la performance que l'appareil peut donner à une application. Ainsi, une application peut être déployée de façon sous-optimale sur un appareil si aucune autre solution n'existe. Dans le cadre de notre travail, nous avons utilisé une métrique de viabilité des appareils, afin de quantifier la viabilité d'un déploiement d'application sur un appareil. Cette métrique, issue d'un calcul basé sur la logique floue, est utilisée par la solution proposée pour déterminer les appareils ayant la meilleure viabilité, bref la métrique la plus élevée.

Certains des travaux présentés précédemment sont basés sur l'orchestration (Ranganathan, Syed) alors qu'un autre est basé sur la chorégraphie (Trumler). La solution d'auto-organisation proposée dans le cadre de nos travaux, se base sur une approche mixte, la facilité de mise en place de l'orchestration avec l'efficacité de la chorégraphie. Ainsi, notre solution répartira l'intelligence autonome entre un noeud coordonnateur, gérant le raisonnement et traitant des données propres à l'environnement même et aux utilisateurs, et des noeuds appareils traitant les données contextuelles propres aux appareils. Cette approche entre dans le cadre de notre vision microscopique et macroscopique de la sensibilité au contexte, présenté à la Section 3.2. Dans la même veine, les travaux de Ranganathan et Syed utilisent des bases de connaissances ou ontologies centralisées afin de conserver les données propres à l'environnement général, que nous décrirons plus loins comme le macro-contexte. Dans le cadre de notre solution, nous étendrons l'utilisation des ontologies en proposant une définition plus étendue de l'information contextuelle des espaces intelligents. Le contenu de l'ontologie sera dynamique, il sera modifié

selon les appareils découverts dans les milieux, l'utilisation des ressources et les processus de gestion logicielle effectuée sur le système.

En plus de proposer ces innovations par rapport à l'état de l'art, nous avons travaillé à la spécification de l'informatique diffuse autonome, en proposant notre vision de celle-ci, appliquée au contexte de la gestion logicielle dans les espaces intelligents. Cette vision a servi à guider nos efforts, en définissant les divers critères qu'un système autonome doit remplir afin de répondre à la problématique de la gestion logicielle dans les espaces intelligents. Cette vision va au-delà de la portée de notre travail, y répondre de façon complète est impossible dans le cadre d'un seul travail de thèse. C'est pour cette raison que nous avons volontairement mis l'emphase sur l'organisation/configuration logicielle, élément central dans notre vision de l'informatique diffuse autonome.

2.1.2 Notre vision de l'informatique diffuse autonome

La vision originale de l'informatique autonome ne convient pas, au départ, au contexte des espaces intelligents distribués. Contrairement aux vastes systèmes informatiques tels que les serveurs d'entreprises, les espaces intelligents sont composés d'une multitude d'appareils hétérogènes ayant parfois peu de puissance de calcul. Il peut alors être impossible pour ces petits appareils d'avoir leur boucle de contrôle avec leur propre base de connaissances et de mesures. De plus, la quantité et la nature des données traitées dans les espaces intelligents dans le cadre de la gestion logiciel est tout autre que pour les systèmes d'entreprises. Certaines informations sont propres aux environnements eux-mêmes (topologie des milieux) ou aux utilisateurs et doivent être traitées par des entités dédiées, par exemple spécialisées dans le traitement des informations d'interaction homme-machine. L'utilisation d'une approche mixte de chorégraphie/orchestration, où les fonctions autonomes les plus complexes sont encadrées par certaines entités de l'espace, est, selon nos observations et notre analyse, plus adaptée aux espaces intelligents. On peut alors parler d'entités coordonnatrices, qui ont comme rôle de coordonner les efforts d'autonomie entre les divers participants et de s'occuper des tâches les plus complexes.

Le fonctionnement d'une entreprise est une bonne analogie à notre vision des espaces intelligents et de leurs fonctions autonomes. Les employés d'une entreprise ont à la fois un certain degré d'autonomie dans leur tâche et coopèrent régulièrement entre eux pour la réalisation d'actions. Certains de ces employés peuvent réaliser des tâches en réaction à des événements, par exemple, un commis de bureau qui distribue le courrier lors de l'arrivée de celui-ci, ou bien des actions de leur propre chef, définis dans leur description de tâches. Certains employés

peuvent avoir également des fonctions prioritaires par rapport à celles d'autres employés ou des poids décisionnels plus lourds (patrons). Quoiqu'une entreprise soit constituée de plusieurs employés, elle peut être considérée comme un tout, pouvant être manoeuvrée à l'aide de l'organisation hiérarchique par les gestionnaires. Nous voyons les espaces intelligents de la même manière, comme une entité composée de composantes : appareils, applications, périphériques d'interaction, capteurs, effecteurs et données ; coopérant entre eux à la réalisation de tâches ou d'assistances vers les utilisateurs.

Ainsi, les mécanismes autonomes sont distribués parmi les entités de l'espace, où certaines entités ont des rôles plus spécialisés tels les employés d'une entreprise. Par exemple, les capteurs ont comme rôle de capter l'information et de rendre celle-ci disponible aux autres entités. D'autres entités ont comme rôle de conserver les connaissances de l'environnement et de tenir à jour ces connaissances à l'aide des informations captées. Enfin, certaines entités ont comme rôle de coordonner les fonctions autonomes, de réfléchir sur les mesures à prendre ou de mettre celles-ci en place. Pour résumer, les mécanismes autonomes ne sont donc pas présents dans toutes les entités de l'espace. En fonction de leur complexité et des ressources nécessaires, elles sont distribuées dans certaines entités de l'espace et coordonnent les efforts d'autonomies.

Outre la structure et la disposition des mécanismes autonomes, l'informatique diffuse autonome diverge de l'approche originale par rapport à la place de l'utilisateur dans le système. Comme notre travail au laboratoire DOMUS se situe dans un contexte d'espaces intelligents dédiés à l'assistance d'utilisateurs dépendants, notre vision inclue davantage l'utilisateur dans les fonctionnalités autonomes. Ainsi, les mécanismes autonomes doivent tenir compte des utilisateurs évoluant dans les espaces, en utilisant, par exemple, les préférences des utilisateurs lors de l'organisation autonome des logiciels ou de l'optimisation autonome de la performance des appareils.

L'auto-configuration

Dans notre vision de l'informatique diffuse autonome, l'auto-configuration est le concept clé dont dépendent plusieurs mécanismes des trois autres concepts. Premièrement, l'auto-configuration vise à répondre aux besoins des gestionnaires et des utilisateurs de simplifier la gestion matérielle et logicielle des espaces intelligents. Elle doit donc permettre aux gestionnaires de déployer et de gérer les applications dans les milieux en seulement quelques actions simples et rapides. Deuxièmement, l'auto-configuration doit également offrir les fonctionnalités permettant, par exemple, à l'auto-réparation de redéployer une application dans l'environ-

nement suite à une panne (voir Figure 2.2). Certains besoins de gestion peuvent être découverts ou reconnus par le système autonome, par exemple, en reconnaissant les activités des utilisateurs et certains contextes. Toutefois, dans plusieurs situations, le déclenchement d'un processus d'auto-configuration peut être initié par des utilisateurs/gestionnaires à l'aide d'outils de gestion, par exemple, par un aidant professionnel déployant de nouvelles fonctionnalités dans un environnement, suite à un diagnostic de l'autonomie d'un utilisateur. L'auto-configuration doit donc être accessible à la fois aux gestionnaires et aux autres mécanismes autonomes ou intelligences de l'espace intelligent.

Il est donc nécessaire de simplifier la gestion des dépendances entre les applications, de réduire le nombre de manipulations nécessaires par les gestionnaires et rendre plus compréhensible les rétroactions retournées aux gestionnaires lors des étapes de gestion. Pour y arriver, nous croyons que les espaces doivent accomplir la majorité des tâches reliées à la sélection des appareils face aux besoins des applications [29]. Pour ce faire, les espaces intelligents doivent avoir une connaissance minimale d'eux-mêmes, par exemple à l'aide d'ontologies et de descriptions sémantiques [1]. La connaissance du contexte et le raisonnement doivent être flexibles et dynamiques pour s'adapter à la nature des milieux : hétérogène, flexible, non fiable, etc.

L'auto-configuration/organisation des espaces intelligents est un problème complexe par la quantité de données contextuelles gérées, par la nature de ces données quantitative et/ou qualitative, et par les conflits pouvant résider entre l'analyse des besoins et le contexte des espaces. Par exemple, l'organisation logicielle avec contrainte de ressources et de contexte est un problème de type NP-Complet (voir plus de détails à la Section 3.5.2). Des stratégies de raisonnement basé sur le *soft computing*, approche proposant des solutions plus ou moins inexactes à des problèmes de calculs difficiles, permettent de modéliser des problèmes complexes avec des solutions relativement simples, en introduisant des marges d'erreurs volontaires (logique floue, modèle bayésien). Elles permettent, entre autres, d'améliorer les temps de traitement en réduisant le nombre de règles en substituant des valeurs qualitatives aux valeurs quantitatives, telles que dans la logique floue (Section 3.5.1). D'autres solutions en *soft computing* se basent sur des mécanismes biologiques pour répondre à des problèmes complexes tels que les réseaux de neurones et algorithmes génétiques L'utilisation d'une solution tirée des travaux sur le *soft computing* est inévitable dans le cadre de l'informatique diffuse autonome. D'ailleurs, la solution proposée d'auto-organisation logicielle (Chapitre 3) se base son raisonnement sur la logique floue.

L'auto-configuration comprend également les aspects reliés à la manipulation autonome de l'état de vie des applications. Entre autres, les applications doivent se mettre à jour elles-

mêmes en tenant compte de leur contexte d'utilisation et des utilisateurs de leur espace. Les applications doivent également pouvoir être démarrée ou arrêtée facilement, automatiquement, à la demande d'un utilisateur ou d'une composante logicielle des milieux, et ce, en faisant abstraction des complexités inhérentes suivantes : gestion des dépendances, vérification des ressources disponibles, gestion de conflit lorsqu'une utilisation est en cours d'interaction avec un utilisateur, etc. La facilité de gestion du cycle de vie des applications n'est pas une fonction proprement dites autonome, mais elle est nécessaire à la présence de fonctionnalités autonomes complexes, tel que l'auto-organisation des logiciels.

Afin de réduire de façon significative la complexité de la gestion pour les utilisateurs, ceux-ci doivent avoir accès à des outils à ce point simple, qu'il leur soit possible de déployer une application ou d'en démarrer une, sans formation ou connaissance préalable du système. Cette simplicité doit également permettre à des applications externes d'utiliser les fonctionnalités d'auto-configuration de façon transparente et abstraite. Par exemple, lorsqu'un appel conférence est inscrit à l'emploi du temps d'un utilisateur, un agenda électronique déployé sur un appareil d'un espace intelligent pourrait déclencher le déploiement et/ou le démarrage d'une application de visioconférences sur un appareil du même espace, en ne spécifiant que les besoins propres à l'application et à l'appel conférence, sans se soucier des détails techniques. Les fonctions autonomes de l'espace prennent alors en charge l'analyse des ressources disponibles sur les appareils, le choix du bon appareil et le déploiement/démarrage de l'application.

L'auto-configuration doit également permettre une intégration facile des nouveaux appareils et applications dans l'environnement, i.e. leur auto-intégration. Ainsi, la découverte automatique des appareils, applications et services présents dans l'environnement, couplée à des outils de recherche complets et des mécanismes de notification, permet de trouver les nouvelles composantes, d'informer le système lors de leur arrivée et de les intégrer aux systèmes existants. Entre autres, les architectures orientées services, comme celle d'OSGi, offre de tels mécanismes et permettent l'intégration dynamique de nouvelles composantes. L'auto-intégration du matériel peut être accomplie à l'aide de standard permettant de décrire la taxonomie des appareils, de reconnaître de façon distincte les appareils et ainsi savoir leurs fonctionnalités et leurs capacités [77].

Globalement, nous croyons que la configuration autonome doit être divisée en trois grandes étapes :

- le déploiement ou la manipulation des applications : regroupe la convivialité des outils et services de gestion, le raisonnement face aux requêtes de gestion et la mise en place des requêtes ;

- la configuration des applications à l'appareil visé : comprend la vérification des dépendances internes et externes, tests de fonctionnement de base et allocation des ressources matérielles ;
- l'intégration de l'application à l'espace : concerne l'offre des services de l'application aux autres entités, la découverte de l'application par les autres entités, etc.

En résumé, les innovations de notre vision de l'auto-configuration reposent sur l'intégration à l'approche originale d'aspects propres aux espaces intelligents et à l'informatique diffuse. Ainsi, nous proposons une utilisation étendue de la sensibilité au contexte avec l'intégration du profil des utilisateurs. Nous croyons que la réduction qu'une partie de l'automatisation et autonomisation passe par des interfaces homme-machine plus conviviales et productives, ainsi que des services faisant une abstraction plus importante des mécanismes internes des applications. Nous croyons également que l'auto-configuration peut se doter de techniques de raisonnement avancées foncièrement différentes du raisonnement par cas traditionnel de l'informatique autonome, afin de répondre à la complexité de la configuration et de l'organisation des systèmes. Enfin, notre vision diverge de la version originale, par l'utilisation d'une stratégie mixte d'orchestration et de chorégraphie, prenant en compte la nature distribuée, hétérogène et dynamique des espaces intelligents.

L'auto-protection

La sécurité et la protection dans les espaces intelligents sont primordiales. Puisque les espaces sont hautement dynamiques et que plusieurs utilisateurs, avec leurs appareils mobiles, peuvent entrer et sortir rapidement, il est important de protéger les données, les appareils, les applications et les utilisateurs. La sécurité dans les espaces intelligents s'articule donc autour de quatre concepts :

- la confidentialité : propriété empêchant les accès aux informations aux personnes non accréditées ;
- l'authentification : propriété permettant d'identifier les composantes dans le but de contrôler l'accès aux services et aux informations ;
- la confiance : propriété mesurant la confiance qu'une entité a envers une autre dans son comportement et les données qui en sont issues ;
- la sureté : propriété concernant le contrôle de l'intégrité, des utilisateurs, de l'environnement et des données.

Les espaces intelligents utilisent donc des méthodes d'authentification et des contrôles d'accès électronique, tels que des noms d'utilisateurs, des mots de passe, des certificats électroniques

ou physiques, tels que des clés, les empreintes digitales, des tags RFID, etc. À ces méthodes de contrôle sont reliées des règles afin d'attribuer à chaque entité des droits d'accès. La confiance peut être mesurée par ces droits d'accès, mais également par des métriques mesurant la qualité des informations ou des services rendus par les entités des espaces [78]. Enfin, les communications entre les entités doivent être chiffrées adéquatement afin d'éviter l'interception des données par des tierces parties.

La gestion de la sécurité dans les espaces intelligents est longue et complexe. Comme les utilisateurs et les applications évoluent, il est nécessaire de modifier, plus ou moins régulièrement, les polices d'accès, les moyens d'authentification, les méthodes de chiffrement, etc. En rendant la sécurité et la protection plus autonome, i.e. l'auto-protection, il est possible à la fois de réduire les tâches de gestion et d'améliorer le niveau de sécurité des espaces intelligents.

Par exemple, des mécanismes intelligents gérant automatiquement les règles d'accès selon le profil de l'utilisateur et la confiance de l'espace envers les entités, permettraient de réduire les tâches de gestion et de rendre les espaces plus dynamiques. Des appareils personnalisant leurs moyens d'authentification selon le profil des utilisateurs [79] amélioreraient les interactions des utilisateurs avec l'environnement. Des mécanismes proactifs détectant les appareils compromis ou les entités malfaisantes et les isolants automatiquement du reste de l'espace, permettraient de rendre les environnements plus sûrs. Enfin, l'intégration assistée lors de la conception, ou automatique lors de la compilation, de patron de conception en sécurité [80] permettraient d'améliorer le niveau de sécurité des applications.

L'auto-réparation

Une part importante des ressources utilisées lors de la gestion et de l'implémentation des espaces intelligents est directement reliée à la résolution des fautes et bris se produisant dans les environnements [28]. De plus, un environnement se configurant et s'optimisant de façon autonome est inutile si les applications qui y sont déployées sont incapables de donner les bons services aux bons moments. L'utilisation de mécanismes détectant automatiquement les bris et fautes, envoyant des rapports compréhensibles aux gestionnaires et, si possible, réparant automatiquement les fautes, est un atout qui permet à la fois de réduire le temps imparti à la gestion des fautes et bris et d'améliorer la continuité de service via la tolérance aux fautes.

Plusieurs actions peuvent être entreprises afin de résoudre un bris ou une faute, mais celles-ci dépendent directement du type des fautes, de la capacité du système à les détecter et de donner un bon diagnostic [49]. Les fautes logicielles se produisant sur un système peuvent être détectées en majeure partie par la lecture des événements systèmes, journalisés par la plupart

des systèmes d'exploitation [81]. Par exemple, tous les journaux d'événements de Mac OS X sont accessibles via l'application Console, comprenant entre autres les fautes logicielles. Une autre stratégie de détection des fautes et erreurs est l'utilisation d'un système de type *heartbeat* ou à baux [57] [49], où les appareils et applications d'un environnement envois régulièrement des messages contenant un bail à une entité gestionnaire. Un bail non renouvelé sous-tend un problème matériel ou la fin abrupte d'une application. Cette méthode n'est pas entièrement fiable, car les baux peuvent être renouvelés alors qu'une application est verrouillée dans un état ou qu'un bris sur une composante matérielle d'un système n'entraîne pas l'arrêt du renouvellement des baux. De plus, le système de baux ne donne pas d'indices sur l'origine de l'erreur ou du bris. Suite à la détection d'une erreur, le système d'auto-réparation doit tenter de répondre à ces événements par la mise en place de mesures. Par exemple, dans le cas d'un bris matériel, les applications présentent sur l'appareil peuvent être relancées sur d'autres composantes de l'espace, si d'autres composantes répondent à leurs besoins. Toutefois, dans le cas d'une faute logicielle, l'application peut être relancée sur la même composante ou bien sur une autre, mais dans les deux cas, il y a de fortes probabilités que l'erreur se reproduise. Une solution basée sur le déploiement d'une version antérieure, plus stable, de l'application permettrait une continuité de service, même si l'application redéployée ne correspond pas aux besoins exacts des utilisateurs ou des gestionnaires. Enfin, il y a le cas des données corrompues provoquant des erreurs systèmes. Ce type de données peuvent être ponctuel et non-déterministe, et forcer la mise en place de mesures de réparation. De plus, ce type d'erreur est difficile à détecter et peut amener vers des états où les systèmes fonctionnent mais ne donnent pas les résultats attendus. Ce type d'erreur peuvent être détecté et évité lors de l'analyse des entrées d'un logiciel, au prix d'une conception et d'une programmation plus étendue des logiciels.

Afin de mettre en place des mesures de réparations des fautes, une intelligence doit être développée ayant comme tâche d'analyser les événements détectés, planifier une réponse à ces événements et mettre en place la réponse. La mise en place de la réponse passe par l'utilisation des autres fonctionnalités autonomes telles que l'auto-configuration. Le raisonnement par cas, où une base de connaissances des fautes possibles et des mesures de réparation est utilisée par la mécanique autonome pour réparer les erreurs détectés [82], est l'une des solutions les plus utilisées dans les travaux sur l'auto-réparation. Une extension du raisonnement par cas, avec l'ajout de cas adaptatif et de mécanismes d'introspection des cas et mesures, permettrait une certaine forme d'apprentissage, en modifiant les cas et mesures selon le contexte de l'environnement et des fautes [83]. Ces extensions permettraient une amélioration constante des fonctionnalités d'auto-réparation, par l'amélioration de la base de cas et le peaufinement des

mesures de réparation.

Dans certains cas, les espaces intelligents doivent assurer une continuité de service, notamment pour les personnes dépendantes qui se fient, jusqu'à un certain point, sur l'assistance apportée par les applications des espaces. Lorsque les applications sont développées selon une approche par composantes ou services, le redémarrage/redéploiement de la composante en faute et non de l'application, est évidemment la solution la plus envisageable, n'altérant pas la disponibilité ou la qualité des applications.

L'auto-optimisation

La recherche d'un équilibre dans la charge de calcul des appareils, telle que dans la vision initiale de l'informatique autonome, est une solution vers des environnements plus performants. Toutefois, l'auto-optimisation dans les espaces intelligents est également reliée à l'adaptation des applications et services face au contexte et aux profils des utilisateurs. Par exemple, afin d'améliorer l'assistance donnée à un usager, certaines applications peuvent adapter leurs interfaces graphiques en fonction des préférences et du profil des utilisateurs. Ces adaptations contribuent à améliorer la performance du point de vue de l'utilisation des applications, elle améliore donc la convivialité de ces dernières.

De plus, l'auto-optimisation dans l'informatique diffuse autonome est fortement reliée à l'auto-configuration. Durant le déploiement des applications, la configuration optimale est choisie face aux besoins des applications et à la taxonomie des appareils présents dans l'espace. Puisque les espaces intelligents sont dynamiques, le contexte des environnements peut changer rapidement. Pour la configuration logicielle des milieux, cette dynamique représente tout un défi puisqu'une configuration optimale à un temps donné, peut rapidement devenir sous-optimale par les changements dans le contexte de l'espace, par exemple lors de l'arrivée d'un nouvel appareil.

L'utilisation de mécanismes permettant de reconfigurer de façon optimale les espaces lors de modifications au contexte permet donc d'avoir des espaces plus performants et plus adaptés aux besoins des utilisateurs et des gestionnaires. Ainsi, l'une des principales tâches de l'auto-optimisation est de veiller à ce que la qualité des services proposée par les appareils des milieux reste à un certain niveau de qualité ou qu'elle soit améliorée de façon proactive. Pour ce faire, l'auto-optimisation peut utiliser des mécanismes d'équilibre des charges [73] afin de répartir l'utilisation des ressources parmi les appareils des milieux. L'auto-optimisation doit également gérer l'arrivée et le départ d'appareils dans les milieux et vérifier si les nouveaux appareils peuvent apporter une contribution significative à l'organisation de l'espace intelligent.

Pour conclure, la Figure 2.2 résume notre vision de l'informatique diffuse autonome et des mécanismes qui la composent. Cette illustration divise les espaces intelligents en quatre couches logiques : les entités de l'espace, les moyens de communication, la couche de gestion logicielle autonome et enfin la couche applicative, i.e. les applications.

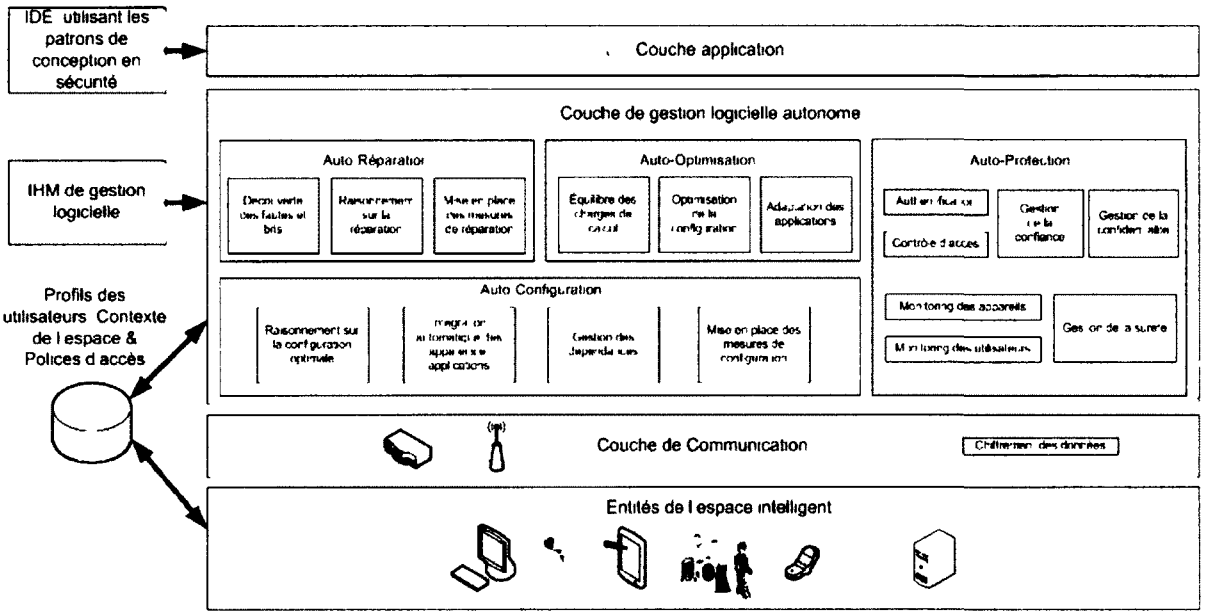

FIGURE 2.2 – Illustration résumant l'informatique diffuse autonome, ces mécanismes et sa disposition dans un espace intelligent

2.2 Scénarios de gestion logicielle autonome des espaces intelligents

Les scénarios suivants illustrent par des cas concrets, l'application de l'informatique diffuse autonome dans la gestion logicielle des espaces intelligents. Chacun des scénarios présente un aspect de l'informatique diffuse autonome :

1. configuration/organisation autonome de logiciels dans un espace intelligent, où les besoins de configuration sont exprimés et dirigés par un gestionnaire non expert ;
2. auto-configuration de logiciels dans un espace intelligent, où le processus autonome est déclenché par une composante logicielle du milieu ;
3. détection et mise en place autonome de mesures de réparation suite à un bris matériel ;
4. amélioration proactive de la performance des logiciels d'un espace intelligent ;

5. mise en place autonome de mesures de protection suite à la détection d'une vulnérabilité dans la sécurité des systèmes informatiques d'un milieu.

Les noms et lieux fictifs utilisés servent à mieux illustrer le rôle des utilisateurs et gestionnaires dans les processus de gestion.

2.2.1 Scénario #1 : Auto-configuration logicielle dans un espace intelligent par un gestionnaire non expert

Une nouvelle application de communication famille/utilisateur a été développée afin de permettre aux personnes dépendantes de communiquer avec leur famille. Cette application a été conçue pour être exécutée sur un appareil possédant une caméra numérique, un micro, des haut-parleurs et un écran. Cette application est offerte aux ergonomes, aux aidants professionnels, à la famille et aux résidents d'appartement intelligent.

Monsieur Latour, un nouveau résidant d'un appartement intelligent, discute avec sa gériatre, Madame Leroy, de son adaptation à son nouvel environnement. Monsieur Latour, qui était habitué de rencontrer amis et famille, a maintenant de la difficulté à sortir de chez lui. Madame Leroy lui propose alors l'application de communication famille/utilisateur, comme l'une des solutions lui permettant de mieux s'adapter à sa nouvelle situation. Après avoir reçu l'autorisation de Monsieur Latour pour que Madame Leroy puisse déployer l'application dans son appartement, Madame Leroy se connecte de façon sécurisée à la passerelle de l'appartement de Monsieur Latour via l'application de gestion logicielle des espaces intelligents. Madame Leroy ne connaissant pas les détails et spécificités de l'appartement de Monsieur Latour, elle choisit alors l'application de communication famille/utilisateur dans la liste d'applications proposées et choisit de déployer l'application dans l'appartement de Monsieur Leroy. L'outil de gestion produit alors une requête de déploiement décrivant les besoins contextuels et les ressources nécessaires à l'installation et à l'utilisation de l'application.

À la réception de la requête de déploiement, le service d'auto-configuration de l'appartement de Monsieur Latour, analyse la requête et recherche dans les informations contextuelles de l'appartement, l'appareil électronique répondant aux besoins de l'application de communication. Puisque Monsieur Latour possède un tel appareil dans son bureau, le service d'auto-configuration achemine alors à cet ordinateur la requête de déploiement de l'application de communication. L'ordinateur ciblé télécharge alors l'application à partir du lien contenu dans la requête, l'installe sur l'appareil et démarre l'application afin que Monsieur Latour puisse l'utiliser immédiatement. Un message de notification apparait à l'écran de l'ordinateur de Mon-

sieur Latour, annonçant l'arrivée d'une nouvelle application et proposant un guide d'utilisation. Une rétroaction est également retournée à l'application de gestion de Madame Leroy, spécifiant quel appareil a été choisi par l'environnement. Si nécessaire, Madame Leroy peut alors guider Monsieur Latour vers l'ordinateur du bureau et l'initier à l'application.

Au final, Monsieur Latour et Madame Leroy n'auront pas eu à s'empêtrer dans les détails techniques du déploiement de l'application dans l'appartement de Monsieur Latour. Le raisonnement sur l'appareil le mieux disposé à recevoir l'application, la gestion des dépendances de l'application et les opérations de déploiement ont été gérés par le service d'auto-configuration du milieu. De plus, Madame Latour, qui n'est pas une gestionnaire experte, a pu malgré son manque de connaissances techniques sur l'appartement de Monsieur Latour, déployer une application complexe.

2.2.2 Scénario #2 : Auto-configuration logicielle dans un espace intelligent

Monsieur Richebourg est atteint de paraplégie et se déplace en fauteuil roulant. Pour l'aider à être autonome dans la vie de tous les jours, Monsieur Richebourg habite une maison intelligente qui est adaptée à son handicap physique et où il peut recevoir de l'assistance via les appareils et applications présents dans les pièces de sa maison.

Suite à une rencontre avec son ergothérapeute, Monsieur Richebourg signifie son besoin de rencontrer un aidant quotidiennement afin de faire le point sur son état de santé. Afin d'accommoder Monsieur Richebourg et d'améliorer l'efficacité des aidants, il est convenu qu'ils communiqueront par visioconférence. Monsieur Richebourg et son aidant, planifient alors les rencontres quotidiennes grâce è l'agenda électronique de Monsieur Richebourg, en décrivant brièvement l'activité comme étant une visioconférence.

L'agenda électronique ayant été intégré à la maison intelligente, dispose d'un accès aux fonctionnalités d'auto-configuration. Ainsi, l'agenda utilise ces fonctionnalités pour démarrer l'application de visioconférence avec l'aidant professionnel à chaque rendez-vous prévu, et ce, sur l'appareil disposant des ressources nécessaires le plus près de Monsieur Richebourg. Par exemple, si Monsieur Richebourg est dans son salon, l'auto-configuration démarrera le service de visioconférence sur l'ordinateur multimédia connecté au téléviseur. Si celui-ci était dans la salle de bain, la visioconférence serait démarrée sur l'ordinateur de la pièce voisine la plus près. De plus, si l'application de visioconférence n'est pas installée sur l'appareil visé, celle-ci sera téléchargée, installée, configurée, puis démarrée. Le démarrage des services de visioconférence

agit en accord avec les rappels de l'agenda électronique de Monsieur Richebourg, qui l'informe en plus de l'emplacement où l'application a été démarrée.

Puisque Monsieur Richebourg a des difficultés à se déplacer, la possibilité de démarrer des applications automatiquement selon sa localisation, lui permet d'interagir plus rapidement avec celles-ci. Évidemment, pour respecter sa vie privée, la visioconférence n'est pas établie automatiquement, il ne reste à Monsieur Richebourg qu'à démarrer la conversation avec son aidant professionnel à l'aide d'une action simple, par exemple en cliquant sur un bouton à l'aide d'une souris ou d'un écran tactile, en appuyant sur une touche du clavier ou bien utilisant des commandes vocales.

Ce scénario illustre les possibilités de l'auto-configuration dans une vision d'intégration des technologies informatiques aux espaces intelligents et de l'amélioration de la qualité des interactions entre les utilisateurs et l'environnement. Ainsi, en plus de rappeler automatiquement la visioconférence à Monsieur Richebourg, le système informatisé du milieu réduit le nombre d'interactions nécessaires à l'utilisateur en démarrant automatiquement l'application de visioconférence sur l'appareil le mieux disposé au contexte présent.

2.2.3 Scenario #3 : Auto-réparation par les composantes d'un espace intelligent

Madame Chambertin habite un appartement pour personne ayant des troubles cognitifs, composé de plusieurs appareils lui apportant une assistance dans ses activités de la vie quotidienne. Entre autres, Madame Chambertin peut compter sur une application d'assistance à la cuisine, l'aidant lors de la préparation de ses repas.

Or, à un moment donné, l'appareil informatique présent dans la cuisine et offrant l'assistance à la préparation des repas subit un bris mécanique le rendant hors d'usage. Immédiatement, un service présent dans l'environnement, surveillant la disponibilité des appareils, détecte le bris mécanique et demande à l'environnement de déployer à nouveau l'application d'assistance sur un autre appareil de l'environnement.

N'ayant pas d'autres appareils dans la cuisine ayant les ressources et les périphériques d'interaction nécessaires à l'exécution de l'application d'assistance, le système d'auto-réparation redéploie temporairement l'application d'assistance sur l'ordinateur portable de Madame Chambertin, qui dispose alors des ressources nécessaires. Madame Chambertin peut alors amener l'ordinateur portable dans la cuisine afin de l'assister pendant la préparation des repas.

Afin d'informer Madame Chambertin du bris survenu et de la décision prise par l'environ-

nement de redéployer l'application d'assistance sur son ordinateur portable, un message clair et compréhensible est envoyé sur son téléphone mobile. Un message est également envoyé aux techniciens gérant l'infrastructure informatique des résidences, pour les informer du bris mécanique et de la réparation temporaire effectuée par l'espace intelligent.

Dans ce scénario, le système d'auto-réparation détecte automatiquement le bris mécanique d'un appareil de l'appartement et répond automatiquement à ce bris par la mise en place d'une mesure de réparation proposant une utilisation alternative de l'application d'assistance aux repas. L'impact du bris sur l'autonomie de Madame Chambertin est diminuée et à l'instar de l'indisponibilité de l'application d'assistance.

2.2.4 Scénario #4 : Auto-optimisation dans un espace intelligent

Madame Montrachet est atteinte d'une déficience cognitive à la suite d'un accident de voiture. Pour l'aider à être plus autonome, elle peut compter sur son téléphone intelligent qui lui propose plusieurs applications d'assistance : agenda, cartes interactives, etc. Ces applications ont été proposées à Madame Montrachet par son aidant professionnel et c'est ce dernier qui les a déployées sur son téléphone.

Or, Madame Montrachet une certaine soirée, a fait une gaffe et elle a échappé son téléphone dans la cuvette des toilettes[2]. Suite à cette gaffe, les fonctionnalités d'auto-réparation ont détecté le bris et redémarré temporairement les applications sur les appareils des espaces intelligents. Informé de la perte du téléphone, l'ergothérapeute de Madame Montrachet, lui apporte un téléphone intelligent en remplacement. À l'arrivée du téléphone dans l'environnement, la mécanique d'auto-optimisation détecte le nouvel appareil et vérifie si ce dernier peut accueillir des applications qui ont été déployées dans un contexte sous-optimal. Or, les applications de l'ancien téléphone, redéployées par l'auto-réparation, ont été principalement conçue pour être utilisées sur un téléphone intelligent et ont été déployées de façon sous-optimale sur d'autres appareils du milieu comme alternative à la perte du téléphone. Le système d'auto-optimisation détermine alors que le nouvel appareil offre des performances plus adaptées aux applications mobiles. Ces applications sont alors déployées sur le nouvel appareil et désinstaller des appareils hôtes temporaires. Les gestionnaires des appartements, l'aidant professionnel et Madame Montrachet sont informés du processus d'optimisation autonome par un message sur le nouveau téléphone intelligent, par des notifications sur les appareils hôtes temporaires et par un message envoyé aux gestionnaires des milieux.

[2]remarque : l'une des raisons les plus évoquées de bris par les propriétaires de téléphones cellulaires

L'auto-optimisation permet l'amélioration de la configuration logicielle d'un espace intelligent en redéployant, si nécessaire, des applications vers des appareils mieux disposés, donnant de meilleures performances ou une meilleure qualité d'interaction. D'ailleurs, ce scénario illustre la coopération entre plusieurs fonctionnalités autonomes, avec l'auto-réparation qui redéploie temporairement les applications et l'auto-optimisation qui améliore par la suite la configuration mise en place.

2.2.5 Scénario #5 : Auto-protection dans un espace intelligent

Madame Paulliac est atteinte de troubles cognitifs et de handicaps physiques dus à une maladie provoquant des dégénérescences. Afin de l'aider à rester autonome, elle habite dans une résidence utilisant l'informatique diffuse comme technologie pour apporter une assistance adaptée à son profil. Entre autre, la résidence utilise des mécanismes de sécurité qui sont couplés aux profils des utilisateurs et au contexte de l'environnement, afin d'améliorer la sécurité des lieux et des utilisateurs. Ainsi, les règles d'accès aux espaces et aux appareils sont modifiées en temps réel suite aux modifications dans le profil des utilisateurs. Par exemple, suite à une dégénérescence accrue de sa mobilité inscrite à son dossier, le système informatique a réduit l'accès aux cages d'escalier à Madame Paulliac, l'obligeant à être accompagné par un aidant professionnel dans leur utilisation. Évidemment, en cas d'alarme d'incendie, le contrôle d'accès aux escaliers de Madame Paulliac est résilié pour la durée de l'alarme.

Dernièrement, Madame Paulliac a vécu des pertes de fonctionnalités cognitives et physiques qui l'ont forcé à modifier ses habitudes de vie. Heureusement, les espaces intelligents de la résidence lui ont apporté une aide précieuse. Par exemple, suite aux modifications dans son profil par les aidants professionnels, Madame Paulliac ne peut emprunter les escaliers, afin d'éviter tout risque de chute. Suite aux mêmes modifications, puisque Madame Paulliac a des difficultés à retenir des mots de passe, les appareils électroniques lui proposent par défaut d'autres moyens d'authentification tels que les empreintes digitales, un tag RFID ou des mots-clés reliés aux faits marquants de sa vie.

Enfin, le système de sécurité des résidences assure également la sécurité des informations électroniques des utilisateurs, et ce, de façon proactive. Par exemple, le petit-fils de Madame Paulliac, lors d'une visite, tente à plusieurs reprises de se connecter sans autorisation au réseau internet des résidences à l'aide d'un appareil mobile. Ayant essayé un nombre anormalement élevé de fois, le système de sécurité bloque de façon temporaire l'accès à son appareil et envoie un rapport aux gestionnaires des résidences. De plus, la mécanique d'auto-protection, couplée

au système de localisation des occupants, surveille de près les agissements du petit-fils de Madame Paulliac et coupe de façon préventive l'accès aux appareils informatisés situés près de celui-ci. Des messages sur les ordinateurs contrôlés, informe les occupants des espaces intelligents de la situation et demande au petit-fils de Madame Paulliac de régulariser sa situation auprès des aidants professionnels ou des gestionnaires des milieux.

Ce scénario illustre le couplage possible au niveau du système d'auto-protection entre les règles et contrôles d'accès et le profil des utilisateurs. Ce couplage diminue, entre autres, les modifications aux données électroniques et réduit le temps de mise en place des nouvelles mesures de contrôle. De plus, des mécanismes proactifs de sécurité permettent une prévention des risques et la mise en place rapide de mesures de protection.

Chapitre 3

Projet Tyche, un intergiciel de gestion logicielle autonome pour espaces intelligents

Ce chapitre présente le coeur de notre travail de recherche, soit la conception et l'implémentation d'une solution permettant la gestion logicielle autonome des espaces intelligents, plus particulièrement de l'auto-organisation des logiciels. Tout d'abord, une introduction du projet est faite et une description de l'architecture de notre solution est présentée. Dans un deuxième temps, une description des concepts et instances utilisés est présentée ainsi qu'une description sémantique du contexte de la gestion logicielle des espaces intelligents.

Dans un troisième temps, nous introduisons nos travaux sur une approche contextuelle basée sur une vue microscopique et macroscopique du contexte. Ces deux approches sont intégrées dans notre solution.

Par la suite, nous présentons notre mécanique d'auto-optimisation de l'organisation logicielle des espaces intelligents et nos outils graphiques de gestion. Enfin, nous terminons ce chapitre sur une discussion à propos de la conception et de l'implémentation de notre solution de gestion, sur son avenir et ses possibles évolutions.

Chapitre 3. Projet Tyche, un intergiciel de gestion logicielle autonome pour espaces intelligents

3.1 Introduction

3.1.1 Objectifs du projet Tyche

Comme nous l'avons spécifié précédemment dans l'introduction, l'objectif général de ce travail de thèse est la recherche de solutions à la problématique de la gestion logicielle dans les espaces intelligents. Toutefois, cette problématique est très large et comprend une multitude d'aspects dont le déploiement d'application, la configuration des logiciels face aux caractéristiques des appareils, la maintenance des applications, la gestion des erreurs, la gestion de la qualité de service, l'optimisation des performances logicielles, la gestion de la sécurité des logiciels, le contrôle des accès, etc. Dans une optique de rendre la gestion plus autonome et de donner aux espaces intelligents une plus grande part de responsabilité dans les tâches de gestion, les divers aspects énumérés précédemment peuvent être reliés de près ou de loin, aux quatre *selves* de l'informatique autonome.

À la lumière de nos travaux initiaux sur la gestion logicielle à distance [84], nous avons concentré nos efforts sur un aspect de l'auto-configuration des logiciels : l'auto-organisation des logiciels, i.e. la répartition stratégique des logiciels parmi les appareils d'un environnement donné selon le contexte de celui-ci. Le contexte correspond aux concepts suivants : les utilisateurs, les applications et les appareils, leurs besoins, leurs dispositions et leurs caractéristiques, et enfin la topologie de l'environnement. Cette auto-organisation se fait donc selon le contexte présent à un moment donné. Afin de permettre une amélioration proactive de l'organisation logicielle, nous nous sommes penchés sur la problématique reliée aux modifications dynamiques dans le contexte et à l'impact de ces modifications sur l'organisation logicielle, par exemple lors de l'arrivée de nouveaux appareils dans un environnement donné. Nous associons la mécanique de réorganisation autonome des logiciels des milieux au concept d'auto-optimisation de l'informatique diffuse autonome (Chapitre 2).

Le choix de l'auto-organisation logicielle comme sujet principal de notre travail s'est affirmé de soi. Comme nous l'avons présenté au cours du Chapitre 2, nous voyons la configuration autonome et l'organisation autonome comme les concepts clés de l'informatique diffuse autonome. Afin d'optimiser les performances des systèmes, relancer des applications suite à des bris matériels, relocaliser des applications en cas de faille de sécurité, etc., il est nécessaire d'avoir les mécanismes permettant le déploiement et la configuration autonome d'applications. Une fois l'auto-configuration/organisation implémentée, les autres mécanismes pourront se reposer sur celles-ci. Nous verrons d'ailleurs dans ce chapitre qu'une métrique d'évaluation de la viabilité d'appareil face aux déploiements d'applications peut agir comme clés d'intégration

des quatre *selves*.

Nous avons donc visé un ensemble d'objectifs à atteindre quant à la conception et l'implémentation d'une solution de gestion logicielle autonome pour espaces intelligents. Une partie de ces objectifs sont d'ailleurs repris au cours des tests et de l'évaluation du travail.

Le premier objectif consiste à offrir à des utilisateurs experts, tels des gestionnaires d'espaces intelligents, un outil permettant de gérer à distance le cycle de vie des applications déployées sur les appareils d'espaces intelligents. Cet objectif est en fait la suite logique de nos travaux de maîtrise où nous avions abordé la gestion à distance des appareils des espaces intelligents. L'outil de gestion doit être efficace et permettre un gain de productivité dans la gestion des logiciels. Cette efficacité et ce gain de productivité sont évalués dans le cadre de nos tests et évaluations et sont présentés au Chapitre 4. L'outil doit également résumer et synthétiser les informations envoyées aux utilisateurs experts, à l'aide d'une représentation visuelle plutôt que textuelle, par exemple via l'utilisation d'une carte de l'espace intelligent pour représenter la position des appareils dans un environnement. L'utilisation d'une métrique d'évaluation de la viabilité des appareils est également l'une de ces stratégies permettant de résumer les performances des appareils.

Le second objectif consiste à offrir aux utilisateurs des fonctionnalités d'auto-organisation et de déploiement autonome des logiciels pour les divers espaces intelligents gérés par les utilisateurs/gestionnaires. Ces fonctionnalités d'auto-organisation devront abstraire aux utilisateurs le contexte des espaces intelligents, résumer les possibilités d'organisation et déplacer le raisonnement quant à la disposition des logiciels dans les espaces intelligents des utilisateurs vers les espaces intelligents eux-mêmes. L'auto-organisation utilisera l'information contextuelle des espaces intelligents afin de trouver les solutions optimales pour un groupe de logiciels à déployer. Évidemment, cette fonctionnalité d'organisation devra également offrir des gains de productivité et une aisance d'utilisation. Nous croyons également que la fonctionnalité d'auto-organisation peut permettre à des utilisateurs non experts tels que des aidants professionnels, de gérer une partie des logiciels des espaces intelligents. Le déploiement autonome pourrait également être utilisé par des applications tierces telles que des agendas électroniques ou des agents d'assistance, afin de déclencher le déploiement d'applications d'assistance auprès des utilisateurs d'environnements.

Afin d'assurer une cohérence dans l'organisation logicielle dans le temps, notre troisième objectif consiste à implémenter des mécanismes d'auto-optimisation de l'organisation logicielle. Ces mécanismes devront tenir compte de plusieurs critères dans leurs décisions d'optimisation, comme entre autres la notion de gains de performance.

Notre quatrième objectif est d'offrir un intergiciel distribué modulaire qui sera facilement extensible et dont les composantes internes seront dynamiques. Dans une optique où nous visons à ajouter les autres fonctionnalités de l'informatique diffuse autonome à notre travail, ce dynamisme et cette modularité permettront une intégration aisée. Nous verrons un peu plus loin en quoi l'utilisation de la plateforme OSGi et des standards WebService-* permettent cette extensibilité.

Notre cinquième objectif vise la création d'une métrique qui permettra de quantifier aisément la viabilité du déploiement d'une application auprès d'un appareil de l'environnement. Cette métrique est utilisée afin d'abstraire la notion de viabilité et de performance des appareils auprès des utilisateurs. Elle pourra entre autres être utilisée dans les rapports envoyés aux utilisateurs afin de quantifier et résumer les décisions d'auto-organisation et d'auto-optimisation. Enfin, cette métrique pourra être utilisée par les différents *selves* afin de quantifier leurs décisions et actions envers un appareil.

Puisque nous visions à utiliser intensivement la notion de contexte, il était nécessaire de représenter celle-ci avec ses concepts, leurs instances et les liens sémantiques entre ces informations. Notre sixième objectif vise donc la modélisation des concepts et instances des espaces intelligents propre à l'organisation des logiciels dans les environnements. Bref, modéliser et concevoir une description sémantique des espaces intelligents. Ces données sont utilisés par les fonctionnalités d'auto-organisation et d'auto-optimisation dans leur processus de raisonnement. Comme les espaces intelligents sont dynamiques et que de nouvelles entités peuvent s'y greffer à tout moment, le support à l'implémentation de cette description doit donc être extensible et dynamique. Le langage *Ontology Web Language* (OWL) répond à ces besoins. L'utilisation de ce langage, conjointement avec une plateforme sémantique permet l'utilisation d'outils de raisonnement, de recherche de données ou bien d'inférence de concepts et d'instances, qui sont particulièrement utile aux fonctionnalités autonomes.

Enfin, notre septième et dernier objectif vise l'inclusion du profil de l'utilisateur des espaces intelligents dans le processus d'organisation des logiciels. Contrairement aux systèmes informatiques classiques, les utilisateurs dans le cadre de l'informatique diffuse évoluent directement à l'intérieur du système. Dans les environnements intelligents pour personnes dépendantes, la notion d'utilisateurs est même centrale au système, puisque dans ceux-ci la majorité des services applicatifs visent l'assistance des utilisateurs dans la réalisation de leurs différentes tâches. L'utilisation du profil des utilisateurs dans le processus de raisonnement sur l'organisation logicielle permettra d'améliorer la disposition des logiciels selon les caractéristiques et les préférences des utilisateurs. Elle permettrait même, jusqu'à un certain point, d'assister

les utilisateurs en déployant des applications d'assistances sur les appareils les mieux adaptés à leur profil. De plus, dans les travaux connexes de la communauté scientifique sur l'auto-organisation, la plupart des solutions [74] [72] [57] n'incluent pas le profil des utilisateurs dans le raisonnement d'auto-organisation/configuration (à l'exception des travaux de Ghorbel [38]). Une solution donnant une place centrale aux utilisateurs dans le processus d'organisation logicielle est donc une contribution importante à l'état de l'art de l'auto-organisation. Le Tableau 3.1 résume les objectifs du projet Tyche.

TABLEAU 3.1 – Résumé des objectifs du projet Tyche

Objectifs	Description
1	Permettre une gestion des logiciels à distance efficace
2	Offrir des fonctionnalités d'auto-organisation des logiciels
3	Permettre l'auto-optimisation de l'organisation logicielle
4	Intergiciel distribué modulaire, extensible et dynamique
5	Création d'une métrique de comparaison des appareils
6	Modélisation et conception d'une description du contexte des espaces intelligents
7	Intégrer le profil des utilisateurs au processus d'auto-organisation

3.1.2 Architecture et fonctionnalités

Tout d'abord, nous avons intitulé notre travail de recherche sous le nom de projet Tyche. Tyche ou Tykhe étant la déesse grecque de la chance, protectrice des cités, mais surtout, croyait-on à l'époque, instigatrice de ce qui arrivait sans le concours de l'être humain. Puisque l'un de nos objectifs était de rendre autonome la majorité des tâches de gestion, ce nom était tout désigné à notre travail.

Le projet Tyche consiste donc en un intergiciel distribué sur un ou des noeuds *coordonnateurs* et des noeuds *appareils* à l'intérieur d'espaces intelligents, tels que des appartements/maisons résidentiels. Ces noeuds en communicant entre eux, permettent la gestion à distance des logiciels déployés sur ces noeuds et le déploiement de nouvelles applications. Cette architecture est basée sur une approche d'orchestration où l'orchestrateur, le coordonnateur, envoie les requêtes de gestion aux noeuds appareils. Cependant, on verra un peu plus loin qu'une partie du processus décisionnel quant à l'organisation logicielle est effectué par les noeuds appareils. La Figure 3.1 présente l'architecture générale des composantes du projet Tyche.

L'intergiciel (Figure 3.1) comprend deux composantes centrales, une ontologie et un engin de raisonnement, qui travaillent en tandem afin de trouver la meilleure organisation logicielle

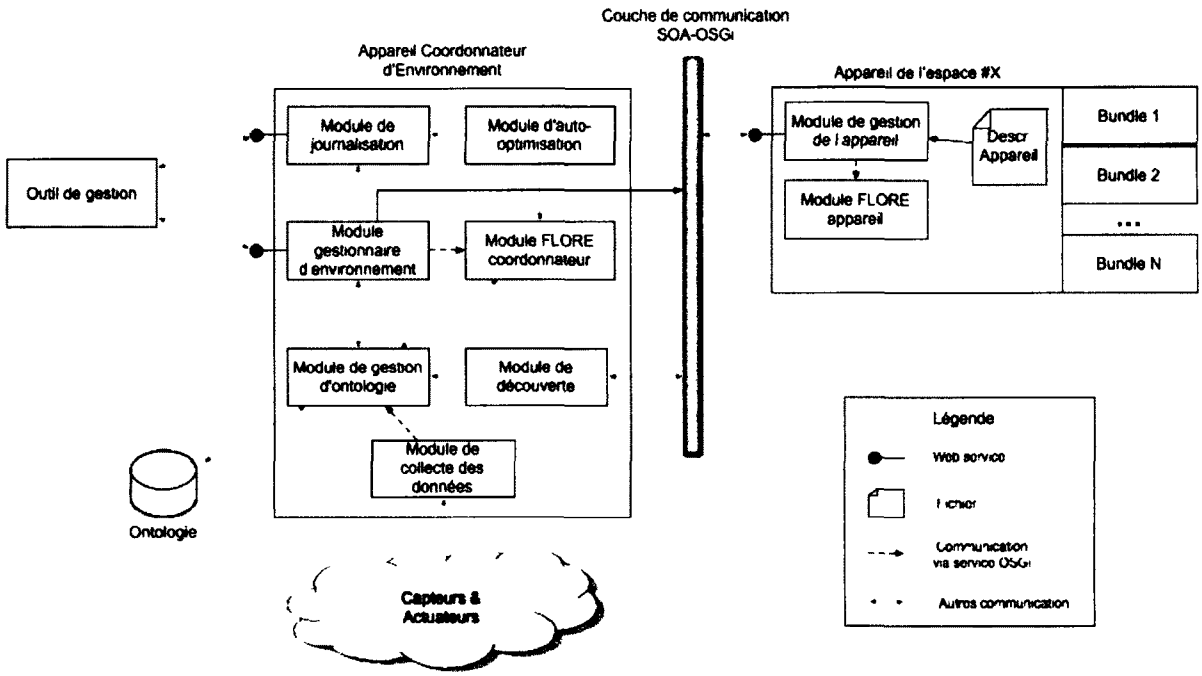

FIGURE 3.1 – Architecture générale des composantes de l'intergiciel du projet Tyche

pour un groupe d'applications à déployer, et ce, à partir du contexte des espaces intelligents. L'engin de raisonnement intitulé *Fuzzy Logic Organization Reasoning Engine* (FLORE) utilise la logique de description et la logique floue afin de donner une métrique d'évaluation aux appareils face aux besoins d'une application à déployer, le *Device Capability Quotient* (DCQ). Le FLORE est divisé en deux parties : celle propre aux raisonnements sur le contexte global de l'espace (topologie, utilisateur), le FLORE-Coordonnateur (FLORE-D), et celle propre aux raisonnements sur le contexte des appareils (caractéristique matérielle, emplacement, périphérique, etc.), le FLORE-Appareil (FLORE-A). Ces deux parties du FLORE traitent respectivement le macro-contexte et le micro-contexte. Plus de détails sur cette approche contextuelle sont présentés à la Section 3.2 de ce chapitre.

L'ontologie est instanciée et gérée par le module de gestion de l'ontologie. Celle-ci est utilisée afin de définir les concepts des espaces intelligents (T-Box) tels que les appareils, utilisateurs, zones, ainsi que leurs propriétés et les relations entre ces concepts. Elle sert également à stocker les instances de ces concepts (A-Box) tels que le profil d'un utilisateur ou bien le profil d'un appareil en particulier. L'ontologie a été implémentée à l'aide du langage *Ontology Web Language* (OWL) et ses concepts et instances ont été instanciés à l'aide de la plateforme sémantique Jena [85].

Afin d'améliorer l'organisation logicielle dans le temps, nous avons conçu et implémenté

un service d'auto-optimisation de l'organisation, instancié à même le module d'auto-optimisation. Ce service découvre les nouveaux appareils entrant dans l'espace intelligent en question et détermine si ceux-ci peuvent apporter une contribution à l'organisation logicielle de l'espace. L'auto-organisation se base entre autres sur les DCQ antérieurs des applications déployées afin de déterminer si un nouvel appareil peut apporter une contribution significative à l'organisation logicielle. Cette fonctionnalité a été implémentée dans le cadre du module d'auto-optimisation.

La découverte des appareils de l'espace est accomplie grâce au module de découverte et aux modules de gestion des appareils. À l'aide du protocole de découverte WS-Discovery [1], le module en question découvre les *web services* offert par les modules de gestion des appareils. Les descriptions sémantiques des appareils des espaces, décrites dans le langage OWL/RDF, sont alors récupérées par le module de découverte, via le service de gestion des noeuds appareils (module de gestion de l'appareil) et sont ajoutées à l'ontologie via le module de gestion de l'ontologie.

Les requêtes d'auto-organisation et de maintenance des logiciels sont gérées par le module gestionnaire d'environnement. Celui-ci reçoit les requêtes de l'outil de gestion via son *web-service*, les analyse et met en oeuvre l'auto-organisation ou la maintenance auprès des applications et appareils visés. Le module de journalisation permet à l'outil de gestion de récupérer les messages provenant des divers modules du projet Tyche et de les présenter aux utilisateurs/-gestionnaires. Ces messages peuvent être par exemple les résultats d'une auto-organisation ou des messages d'erreur lors d'un déploiement d'application.

Le module de collecte des données permet de récupérer les données provenant des capteurs, effecteurs ou systèmes de localisation des espaces intelligents, et de mettre à jour les valeurs associées dans l'ontologie. Ce module s'interface entre autres avec le projet Demeter, un système pair-à-pair de gestion des évènements et des commandes d'effecteur, développé au laboratoire DOMUS et basé sur la couche de protocole *Device Profile for Web Service* (DPWS).

Enfin, des outils de gestion ont été développés afin d'interagir avec l'intergiciel de gestion. Ces outils permettent une consultation à distance de l'organisation logicielle de plusieurs espaces intelligents, la gestion des applications sur les appareils des espaces, l'envoi de requêtes d'auto-organisation, la lecture des journaux systèmes, etc. À ce jour, une version experte a été développée pour les ordinateurs de travail (*Desktop*) et une version simplifiée pour utilisateur non expert a été développée pour les téléphones intelligents munis du système d'exploitation *Android*. Les modules et fonctionnalités présentés précédemment sont présentés dans les sections suivantes de ce chapitre.

[1] Standard WS-Discovery : http ://docs.oasis-open.org/ws-dd/discovery/1.1/wsdd-discovery-1.1-spec.html

Le diagramme de séquence à la Figure 3.2 présente le fonctionnement général de l'intergiciel lors d'une requête d'auto-organisation. La requête d'auto-organisation de l'exemple est initiée par l'outil de gestion, mais peut être initiée par d'autres composantes logicielles via le *web service* du noeud coordonnateur tel que pour le module d'auto-organisation ou un éventuel module d'auto-réparation. Afin de simplifier le diagramme, un seul noeud appareil est présenté.

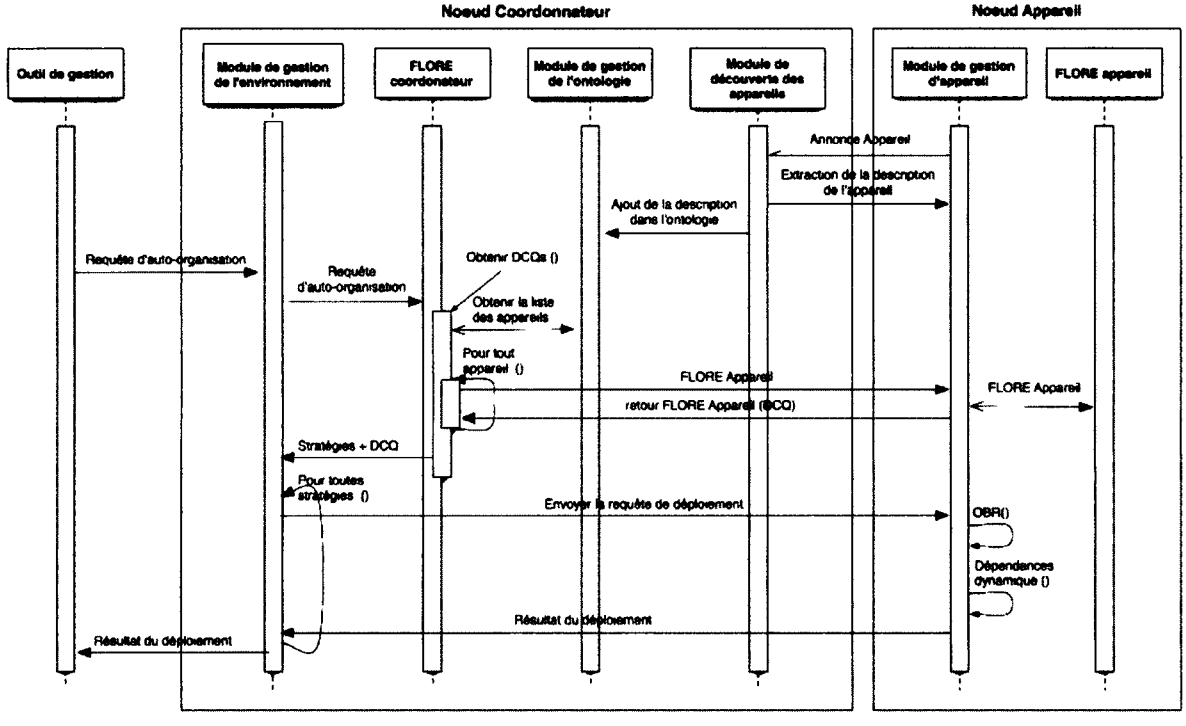

FIGURE 3.2 – Exemple sous la forme d'un diagramme de séquence, d'une requête d'auto-organisation sur l'intergiciel du projet Tyche

Dans l'exemple de la Figure 3.2, les appareils sont découverts par le noeud coordonnateur avant que la requête d'auto-organisation soit initiée. Une fois la requête d'auto-organisation reçue par le module de gestion de l'environnement, celui-ci demande à l'engin de raisonnement d'auto-organisation en place de trouver la meilleure organisation logicielle pour les applications à déployer mentionnées dans la requête. En effet, grâce au dynamisme et au couplage faible d'OSGi, il est possible d'interchanger plusieurs types d'engin de raisonnement différents, pourvu que ceux-ci implémentent l'interface (Java) générique d'un service d'auto-organisation. La possibilité d'interchanger des engins de raisonnement permet une évolution aisée des algorithmes ou la création d'autres mécaniques de raisonnement pouvant être mieux adaptés au contexte de certains espaces intelligents. Dans le cadre du projet Tyche, nous n'avons implémenté qu'un engin de raisonnement d'auto-organisation, le FLORE.

La requête d'auto-organisation est donc envoyée via le service d'auto-organisation à un engin de raisonnement d'organisation logicielle. Sans entrer dans les détails du FLORE, car ceux-ci sont présentés à la Section 3.5, celui-ci communique avec les appareils, récupère des DCQ initiaux auprès des appareils, calcule les DCQ finaux en tenant compte du profil des utilisateurs, puis choisi des stratégies de déploiement selon les DCQ évalués. Ces stratégies, basées sur le *design pattern Strategy*, encapsulent les requêtes de déploiement vers les appareils les mieux adaptés aux besoins des applications à déployer. Les résultats de ces déploiements sont retournés à l'outil de gestion afin d'être présentés aux utilisateurs.

3.1.3 Implementation

La dernière version du projet Tyche s'articule autour de trois grandes technologies : le standard de plateforme orientée service OSGi [42], les standards WebService-*(discovery, service, eventing, etc) du *World Wide Web Consortium* (W3C) et le langage Java.

La première technologie, OSGi, permet une gestion complète du cycle de vie des applications, une modularisation poussée permettant de réduire le couplage logiciel et un dynamisme élevé dans les interactions entre les modules grâce à l'échange de services. Tous les modules de l'architecture du projet Tyche reposent sur le concept de module ou *bundle* d'OSGi. Le concept de généricité des services permet d'avoir plusieurs implémentations différentes d'un même service (tel que pour les engins de raisonnement) sur une même plateforme. Il est également possible d'avoir plusieurs types d'ontologie ou de mécanisme de découverte. Enfin, les applications qui sont déployées dans le cadre de l'intergiciel proposé sont des *bundles* OSGi. La gestion des dépendances entre ces modules est faite par l'*OSGi Bundle Repository* (OBR), un API gérant dynamiquement les dépendances à l'aide d'un fichier de description des dépendances déployé sur les répertoires d'applications. La version courante du projet Tyche est divisée en treize modules OSGi s'échangeant des services et des bases de code. Elle inclue également plusieurs bibliothèques de codes externes dont Jena et JFuzzyLogic, et repose sur d'autres modules OSGi, développées par différentes organisations tel que l'*EventAdmin*, le *BundleRepository* ou bien les modules de Apache CXF. La Figure 3.3 présente ces modules et leurs dépendances fonctionnelles.

Les standards WebServices en conjonction avec le langage Java ont permis d'assurer une utilisation multiplateforme du projet Tyche, mais également une extensibilité quant aux cadres d'utilisation : les webservices permettent, en toute fin pratique, à un noeud appareil développé dans un autre langage que le Java (C++ par exemple) de se joindre à l'intergiciel et d'offrir un

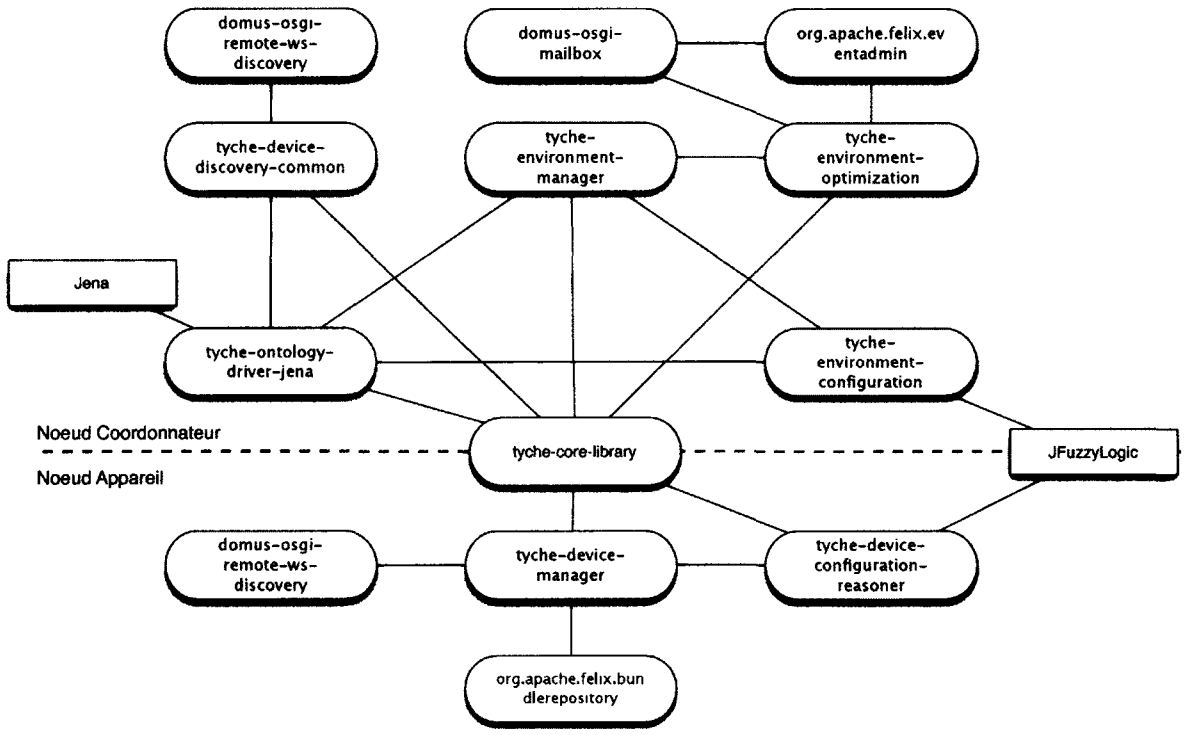

FIGURE 3.3 – Les modules OSGi du projet Tyche et leurs dépendances fonctionnelles

support de déploiement. Toutefois, dans la version courante, puisque les applications à déployer sont des *bundles* (modules) OSGi développés en Java cette possibilité se voit restreinte à la source. D'un autre côté, les webservices permettent d'aller rejoindre d'autres implémentations du langage Java tel que les machines virtuelles J2ME ou la Dalvik d'Android OS. Ainsi, les webservices ont permis de développer un outil de gestion sur un système mobile, dans notre cas avec Android, et de s'interconnecter sur les noeuds coordonnateurs sans éprouver de problèmes de compatibilité. Plus de détails sur l'implémentation exacte du projet Tyche sont présentés tout au cours de ce chapitre.

Les WebServices ont été instanciés grâce à l'API Apache CXF dOSGi[2], qui génère dynamiquement à la volée, les services *Simple Object Access Protocol* (SOAP) (avec WSDL et déploiement sur un serveur Web) à partir des services OSGi. Cet API recrée tout aussi dynamiquement les services OSGi à partir des services Web du côté des plateformes clientes. Enfin, la mécanique de découverte des services, telle que les services de gestion des appareils (Module de gestion des appareils), est basée sur le protocole *WS-Discovery* (implémentation Java de

[2]Site web de dOSGi : http ://cxf.apache.org/distributed-osgi.html

Magnus Skjegstad[3]). Ce protocole permet une découverte de services Web de type pair à pair (P2P) sur un réseau à l'aide des fonctionnalités *Multicast* ou *Broadcast* du protocole internet (IP).

Comme nous l'avons présenté plus haut (Section 3.1.1), l'un de nos objectifs était d'utiliser le langage OWL afin d'implémenter les concepts et instances propres à l'information contextuelle des espaces intelligents. L'ontologie ainsi créée a été instanciée à l'aide de la plateforme Jena [85], elle-même intégrée dans le module de gestion de l'ontologie du noeud coordonnateur. La description des appareils est également décrite à l'aide du langage OWL et contenue dans les noeuds appareil. Ces descriptions sont extraites des appareils via le service du module de gestion des appareils et ajoutées à l'ontologie du noeud coordonnateur. Plus de détails quant à l'implémentation de chacune des fonctionnalités de notre intergiciel sont présentés dans les prochaines sections de ce chapitre.

3.2 Approche contextuelle macroscopique et microscopique

À l'image de la gestion logicielle dans les espaces intelligents, la sensibilité au contexte peut être difficile à mettre en place due à la complexité et à la richesse en information des milieux. La création d'un système sensible au contexte nécessite la manipulation d'un grand nombre de données complexes, ayant souvent un nombre élevé de contraintes telles que des contraintes temporelles. La création d'une représentation du contexte nécessite souvent l'utilisation de plusieurs techniques tels que l'inférence d'états ou de données à partir de données statistiques ou brutes [86], le filtrage ou l'agrégation de données afin de réduire la surcharge d'information [87] ou bien la gestion de conflit entre données [88].

De plus, la définition de sensibilité amène plusieurs questions : qui ou quoi est sensible au contexte ? Qui est sensible à quoi ? Quelles actions peuvent être entreprises face aux contextes ? La sensibilité et le contexte forme un tout, qui peut être perçu par des processus intelligents comme des données émergentes des espaces intelligents. Ces processus peuvent être à la fois des consommateurs de contexte, tel que le FLORE, ou des fournisseurs de contexte tel que le système de découverte des appareils, le module de collecte des données et les modules de gestion des appareils.

Dans le cadre de nos recherches, nous avons initié des travaux consistants à spécifier un classement et une utilisation des contextes en deux types : la sensibilité aux contextes microscopiques et macroscopiques [12]. Ces deux types de contexte aident à simplifier la modéli-

[3]Page web de Java-WS-Discovery : http ://code.google.com/p/java-ws-discovery/

sation et l'utilisation du contexte. Ainsi, la sensibilité aux contextes microscopiques englobe la perception et la compréhension d'un processus ou d'une entité de son environnement, alors que la sensibilité aux contextes macroscopique est l'information globale d'un environnement, que les micro-contexte aident ou non à construire.

L'idée générale derrière le principe de contexte et de sensibilité macroscopique correspond à l'utilisation que font plusieurs chercheurs des ontologies. Une base de données centralisée de concepts et d'information issus des capteurs d'un environnement et utilisés par les systèmes dans leurs assistances auprès des utilisateurs. De plus, ces données sont organisées et cohérentes. Le raisonnement sur le contexte se fait de façon globale sur l'ensemble des données d'un environnement.

D'autre part, le contexte et la sensibilité microscopique correspondent plutôt à une vision du contexte reliée directement à une composante ou entité d'un milieu. L'utilisation de ce type de contexte serait alors plutôt faite sur ses composantes internes, alors qu'une représentation abstraite ou agrégée de ces informations est partagée avec les autres composantes des milieux. En ce sens, cette utilisation du contexte correspond à l'idée que se faisait Mark Weiser dans ses premiers travaux sur l'informatique diffuse [2], où les traitements mêmes des données sont répartis dans les objets des milieux le plus près des sources d'informations. Un système purement basé sur une approche microscopique du contexte correspondrait à un ensemble de composants ou agents, ayant leur propre micro-contexte et le partageant, jusqu'à un certain point, entre eux. Dans un tel système, l'information entre micro-contexte peut être incohérente et conflictuelle. Il peut être également difficile ou même impossible d'obtenir une vue globale sur le système, dû à des pertes momentanées d'une composante. De tels systèmes doivent donc être opérationnels dans des conditions difficiles, avec plusieurs contraintes. Le concept de micro-contexte est entre autres utilisé par certains chercheurs, tel que Biswas [89], afin de partitionner logiquement et physiquement le traitement de l'information provenant de capteurs et de renvoyer aux couches applicatives supérieures une représentation abstraite ou préanalysée des données. Cette utilisation abonde à notre vision de la sensibilité et à l'utilisation du contexte microscopique. Enfin, les travaux de Patrice Roy [90], candidat au doctorat au sein du laboratoire DOMUS, portent sur la modélisation de la sensibilité microscopique dans les espaces intelligents ouverts et a contribué de façon plus que significative, aux travaux du DOMUS sur le contexte micro et macroscopique.

Le projet Tyche est un système mixte basé sur une sensibilité au contexte à la fois macroscopique et microscopique. Si la répartition de l'utilisation du contexte pouvait être précisément quantifiée, la part du macroscopique correspondrait aux deux tiers de l'utilisation du contexte

alors que la part du microscopique serait d'un tier. L'intergiciel de gestion logicielle autonome utilise une ontologie afin de représenter l'état global des espaces intelligents. Une grande part du raisonnement d'auto-organisation est effectuée sur la représentation globale du contexte des environnements, par le FLORE partie coordonnateur, incluant le profil des utilisateurs, la topologie des espaces, la représentation générale des appareils, etc.

Toutefois, une partie du raisonnement est également fait dans une perspective microscopique du contexte. Ainsi, les FLORE-A traitent des données propres aux entités des appareils : l'utilisation des ressources (CPU, RAM, espace disque), les périphériques attachés, les zones dans lesquelles sont présents les appareils ainsi que les dépendances dynamiques (modules, services, bibliothèques de code). Dans son évaluation d'un DCQ, les FLORE-A utilisent également des informations relatives aux macro-contextes tel que le profil des applications, envoyé vers les noeuds appareils par le noeud coordonnateur. Toutefois, il n'existe pas à ce jour de communications et de partages des micro-contextes entre les différents noeuds appareils des espaces intelligents. Dans le cas du projet Tyche, un parallèle entre l'utilisation du contexte macroscopique et microscopique et les stratégies de gestion d'orchestration et de chorégraphie peut être fait. L'orchestration correspondrait plutôt à un système sensible au contexte macroscopique et la chorégraphie à un système sensible au contexte microscopique. La Figure 3.4 présente un résumé de l'utilisation des différentes informations contextuelles par le projet Tyche. On peut y constater que l'information dynamique propre aux appareils, appartenant au micro-contexte, est traitée par les noeuds appareils et les informations moins tangibles telles que le profil de l'utilisateur sont traitées par le noeud coordonnateur.

L'utilisation d'une vision mixte macro et microscopique du contexte dans le projet Tyche a amené plusieurs avantages. Ainsi, une grande partie de l'évaluation des capacités d'un appareil à recevoir et exécuter une application est effectuée par les appareils eux-mêmes. Elle concorde avec une vision subjective du système et d'un certain mode de pensée affirmant qu'une entité se connait toujours mieux elle-même ("connais-toi toi-même", Socrate).

Dans une perspective d'hétérogénéité du matériel des appareils, une évaluation personnalisée des performances du système est donc possible. Par exemple, un appareil critique à l'environnement (système de sécurité, de détection de chute, etc.) pourrait fort bien surestimer l'utilisation de ses ressources afin de privilégier d'autres appareils. Dans notre implémentation actuelle de Tyche, le traitement par les appareils est homogène, mais rien n'empêche en quelques minutes (grâce aux fonctions d'appartenance floue du FLORE (Voir Annexe B), détails à la Section 3.5) de configurer un traitement du contexte particulier pour un appareil ciblé. L'utilisation du micro-contexte permet également l'ajout dynamique de nouveaux appareils à

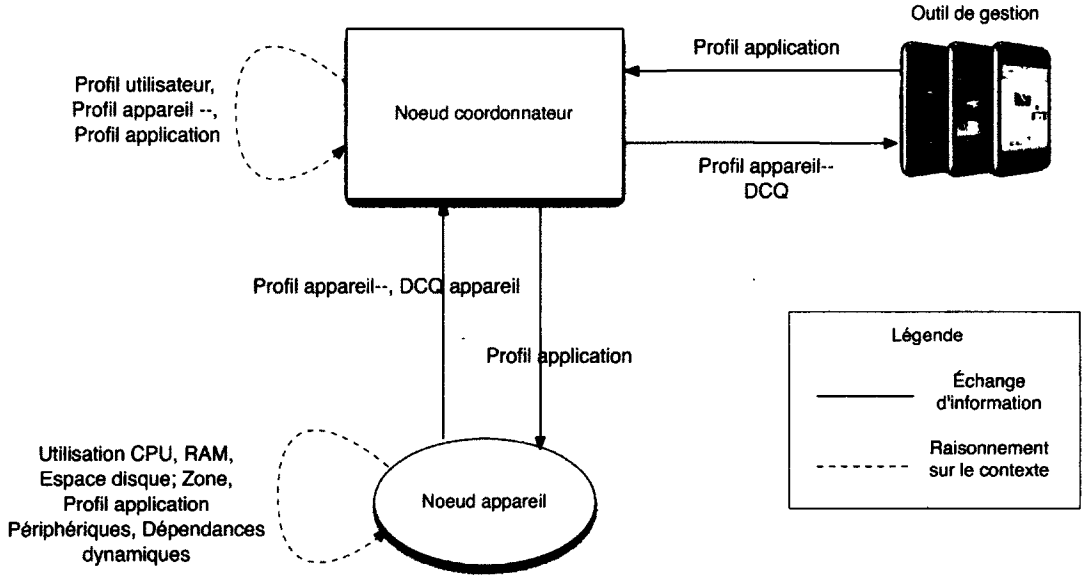

FIGURE 3.4 – Utilisation des informations contextuelles par les noeuds orchestrateurs et appareils

l'environnement et à son intégration aux raisonnements d'auto-organisation. Enfin, elle permet également de répartir une certaine partie de la charge de calcul entre plusieurs appareils, améliorant les performances globales du FLORE. Dans les futures évolutions du projet Tyche, nous tenterons entre autres d'étendre l'utilisation du micro-contexte en impliquant d'autres composantes reliées aux quatre *selves* de l'informatique autonome. Les raisonnements propres à l'auto-répartion ou à l'auto-protection pourraient alors être effectués dans des agents dédiés ayant une vue subjective de ces contextes.

3.3 Description sémantique des espaces intelligents

Dans le cadre des recherches du laboratoire DOMUS, une initiative portant sur la conception, l'implémentation et l'utilisation d'une description sémantique des espaces intelligents a été entreprise. Cette initiative, à laquelle nous avons participé, a proposé les concepts et propriétés de base des entités et composantes des espaces intelligents afin de constituer des méta-ontologies des milieux. Le résultat de cette initiative consiste en une description sémantique implémentée dans le langage OWL, proposant entre autres le concept/propriété de références entre les lieux et le temps pour les multiples instances des environnements intelligents

décrite dans la méta-ontologie. Cette description d'ontologie a été présentée à la conférence *International Conference on Information Integration and Web-based Applications & Services* (iiWAS2009) [35] et a été étendue dans le cadre du journal *International Journal of Web and Grid Services* (IJWGS) [1].

3.3.1 État de l'art des descriptions sémantiques et ontologies

Les descriptions sémantiques et ontologies existantes sont principalement utilisées dans des domaines très spécifiques de l'informatique et encore peu connu du grand public. Dans le domaine des environnements intelligents et de l'informatique diffuse, ils existent des ontologies traitant de la modélisation du contexte [91] [92] [93], de la qualité de service [94] ou des interactions homme-machine [95]. Toutefois, plusieurs problèmes restent irrésolus et n'ont pas été abordés par les travaux cités précédemment. Parmi ces problèmes, la notion de référence en terme de position/emplacement et de temps, la notion de dynamisme par rapport aux changements dans les espaces (ajout et retrait de composantes) et la notion de structure ou de topologie des espaces.

L'ontologie SOUPA [96] est un bon exemple des travaux effectués en descriptions sémantiques pour espaces intelligents. Cette ontologie a été conçue pour le support d'applications mobiles et ses concepts sont dérivés d'autres ontologies existantes tel que *Friend of Friend* (FOAF) [4] ou DAML-Time [5]. FOAF est une description sémantique permettant de décrire les personnes, leurs caractéristiques de base et les relations d'amitié entre ces personnes, alors que DAML-Time est une ontologie en rapport aux connaissances générales sur le temps et les évènements temporels. Il y a également DogOnt [97], une ontologie conçue pour l'interopérabilité entre systèmes domotiques.

Comme nous l'avions écrit précédemment (Chapitre 1), Ranganathan [57] et Roman [98], dans leurs travaux sur le projet GAIA, ont également proposé l'utilisation d'ontologies pour décrire les informations contextuelles des milieux. Dans GAIA, l'ontologie propre aux milieux est utilisée dans ces opérations d'adaptation des applications aux profils des appareils. Ranganathan utilise entre autre le couplage sémantique (*semantic matching*) comme outil de raisonnement sur l'ontologie de GAIA.

Les ontologies précédemment citées possèdent plusieurs faiblesses auxquelles nous voulons remédier. Dans SOUPA, la description des appareils est plutôt faible et n'intègre pas des propriétés dynamiques telles que les ressources, périphériques, etc. Dans l'ontologie du projet

[4]FOAF Ontology : http ://www.foaf-pro ject.org/

[5]http ://www.daml.org/services/owl-s/1.0/owl-s.html

GAIA, l'absence des concepts d'utilisateur et de position des objets est un manque quant aux besoins de l'auto-organisation de raisonner sur ces concepts. De plus, la description de l'ontologie de GAIA est à ce point axée sur son contexte d'utilisation (la diffusion et le contrôle de multimédias dans des salles de conférence), qu'il serait difficile de l'étendre à d'autre type d'espace.

3.3.2 Description sémantique utilisée

Les trois concepts de base de notre description sémantique sont les concepts relatifs à l'Environnement, les Êtres (*Being*) et les concepts dynamiques (Figure 3.5). Tous les autres concepts de cette ontologie héritent directement de ces trois items. Le concept Environnement inclut les objets tangibles tels que les appareils ou les meubles, mais également les concepts intangibles comme les zones d'un espace intelligent. Le concept Être inclut les concepts relatifs aux entités vivantes (organiques) des espaces intelligents : les utilisateurs, aidants, animaux, etc. Les profils de ces entités sont également définis à partir de ce "super" concept. Enfin, le concept Dynamique comprend les sous-concepts qui sont à la fois intangibles et dynamiques. Par exemple, un logiciel, un service et un processus informatique sont des sous-concepts de Dynamique, ainsi que les activités des utilisateurs.

Notre travail de recherche contribue donc à la construction de cette méta-ontologie pour la description des informations contextuelles des espaces intelligents. Elle la complète avec de nouveaux concepts tel que les *NonDedicatedDevices* (appareils), les *DedicatedDevices* (périphériques ou systèmes embarqués), les zones, les applications, etc. Plusieurs propriétés ont également été ajoutées aux concepts existant à l'instar du profil de l'utilisateur avec l'acuité visuelle, la vitesse de déplacement moyenne, etc. La Figure 3.6 présente ces concepts, leurs propriétés et les liens qui existent entre ceux-ci.

Les appareils sont décrits à l'aide d'une description OWL stockée dans les noeuds appareil des espaces intelligents. Comme nous l'écrivions précédemment, cette description est extraite des noeuds via le système de découverte des appareils. Les propriétés des appareils sont utilisées par le FLORE dans son évaluation de l'organisation logicielle et dans son calcul du DCQ. Le FLORE partie appareil (FLORE-A) traite les propriétés des appareils relatives au micro-contexte des appareils, i.e. les ressources consommées et les périphériques connectés. D'autres informations sur l'appareil sont utilisées lors du raisonnement sur le macro contexte dans le FLORE partie coordonnateur (FLORE-C) tel que l'emplacement des appareils, périphériques, orientation des dispositifs d'affichage, etc. Comme la description OWL des appareils

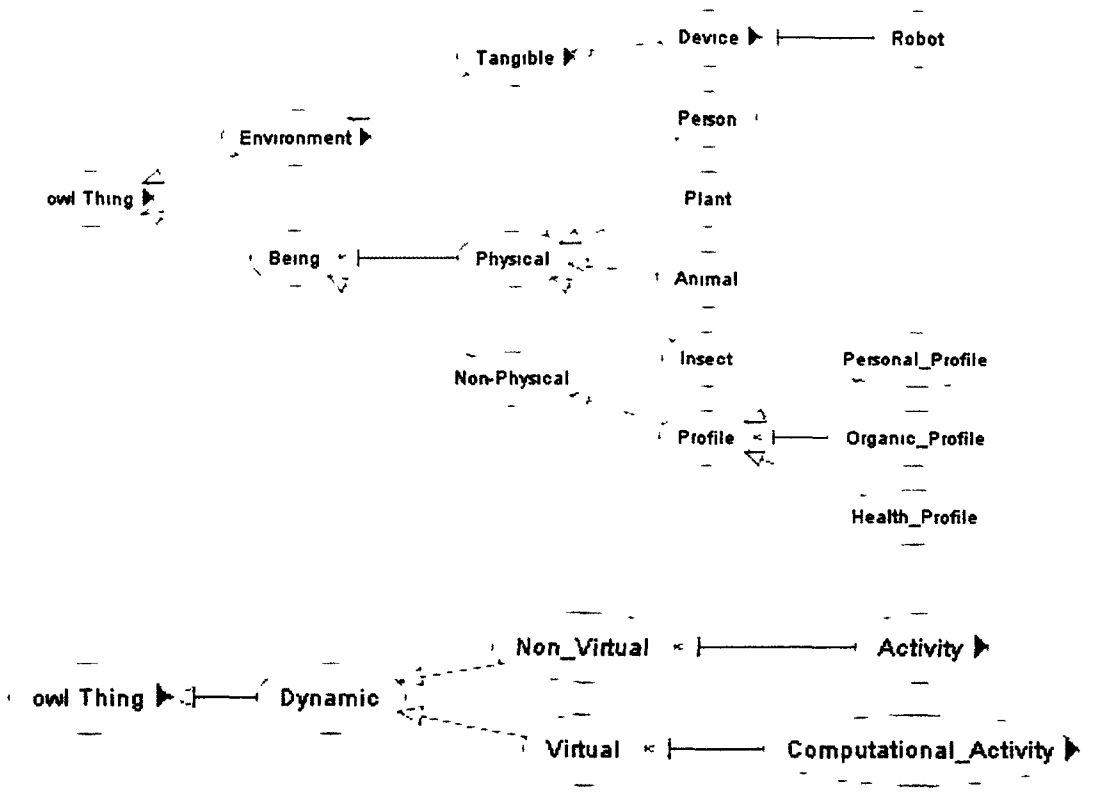

FIGURE 3.5 – Concept de base de la description sémantique (tiré de [1])

ne concerne qu'un nombre limité de concepts et de propriétés, celle-ci n'est pas instanciée dans une plateforme sémantique Jena sur les noeuds appareils des environnements. Un analyseur syntaxique XML (*parser*) est plutôt utilisé pour accéder aux propriétés des appareils. De plus, le déploiement d'une plateforme sémantique Jena sur certains appareils ayant peu de puissance de calcul ou de mémoire vive entraînerait également des problèmes de performance, dû principalement à la taille de la plateforme Jena.

Plusieurs travaux ont été effectués en informatique et en électronique sur la taxonomie et la description d'appareils électroniques. Dans le cadre des grilles informatiques (*grid computing*), il a été nécessaire de décrire les ressources des systèmes participants afin d'organiser la distribution des processus [99]. À travers ces travaux, l'idée d'utiliser les résultats de bancs d'essai de performance (*benchmark*), tels que ceux de la *Standard Performance Evaluation Corporation* (spec)[6], afin de décrire la capacité de traitement d'un système (par la bande le processeur), revient régulièrement. Nous avons donc adopté cette approche dans notre description des ap-

[6]Site web du spec . http ://www.spec.org/

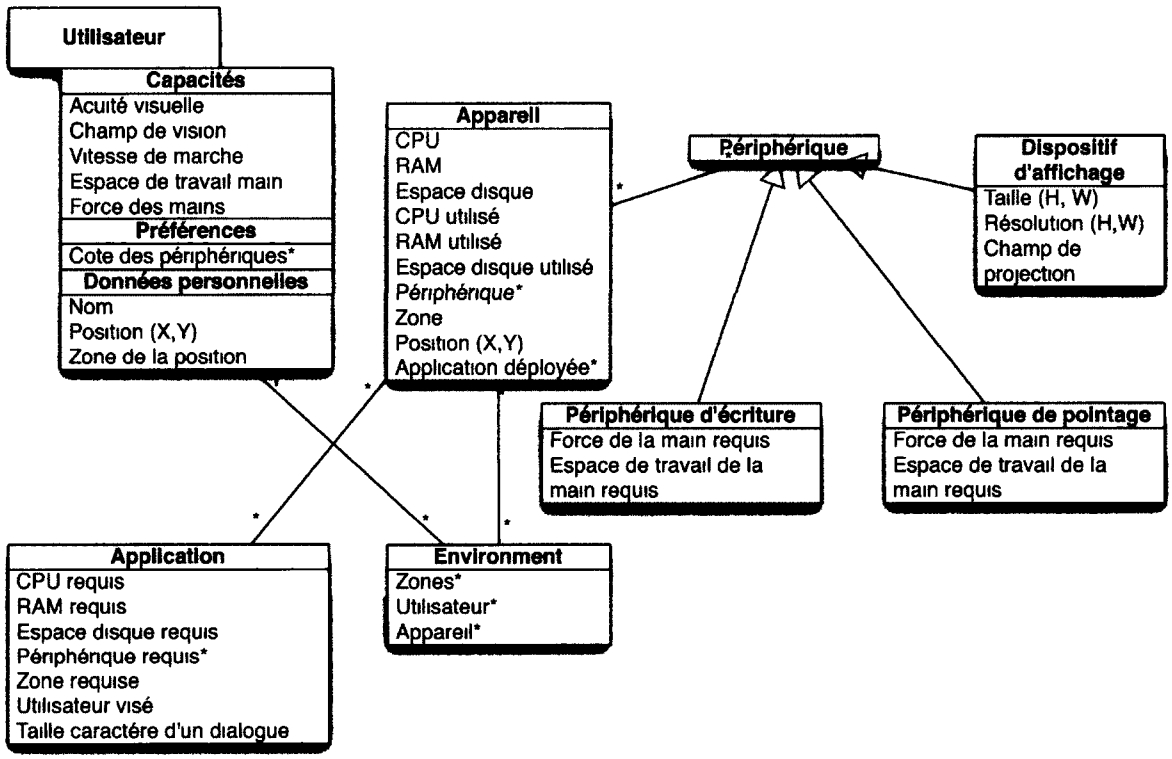

FIGURE 3.6 – Concept utilisé dans le cadre du projet Tyche)

pareils afin de décrire les capacités de traitement des processeurs (CPU). Quant à la mémoire vive (RAM) et l'espace disque des appareils, seule leur capacité en giga-octets est utilisée à ce stade-ci du projet Tyche. Toutefois, les vitesses de lecture de ces mémoires devront être prises en compte puisqu'elles ont un impact sur la performance des applications.

Les utilisateurs sont décrits à l'aide de trois groupes de propriétés : les capacités des utilisateurs (force de la main, acuité visuelle, etc.), leurs préférences et les données personnelles (nom, position dans l'espace et zone[7] reliée à la position). Pour des raisons pratiques et logiques, ces données ont été liées au macro contexte, contenue dans l'ontologie de l'environnement (gérée par le noeud coordonnateur) et sont donc traitées par le FLORE-C. Les informations relatives aux utilisateurs sont gardées localement dans l'ontologie du noeud coordonnateur et ne sont pas partagées avec les noeuds appareils, ayant entre autres en arrière pensée, l'idée de diminuer la publication de ces informations personnelles parmi les composantes des milieux.

Enfin, l'élément central dans l'organisation logicielle, la description des applications et de

[7]la zone est une partie logique ou physique délimitée d'un espace intelligent telle que la cuisine ou bien le bain. Une zone peut correspondre à un objet physique tel qu'un placard

leurs besoins, sont également définis dans l'ontologie. Leurs propriétés décrivent les ressources nécessaires à la bonne exécution des applications sur les appareils hôtes. Dans certains cas, une description d'application peut intégrer une propriété décrivant une zone dans laquelle l'application souhaiterait être déployée, par exemple, la zone cuisine pour une application d'assistance à la préparation de repas telle que Archipel[8] [100]. La description peut également intégrer l'identifiant d'un utilisateur des milieux visés par le déploiement de l'application. Dans le cas où une application nécessite un dispositif d'affichage et dispose d'une interface graphique, la description inclut la taille moyenne des caractères d'un dialogue. Cette dernière propriété est utilisée lors de l'évaluation de l'organisation face à l'acuité visuelle des utilisateurs. Enfin, les besoins d'une application en ressources matérielles sont également définis. Pour ce qui est de la puissance de calcul, à l'instar de la définition des appareils, celle-ci est exprimée en résultat de bancs d'essai. Ainsi, la valeur liée au CPU correspond à la performance moyenne nécessaire pour que l'application soit fonctionnelle sur un matériel donné. Des périphériques nécessaires à l'utilisation d'une application peuvent également être définis dans cette description des applications. À fin de consultation, un extrait de la description contextuelle des concepts des espaces intelligents, implémenté dans le langage OWL, est présenté à l'Annexe A.

Dans le cadre du projet Tyche, la description contextuelle des espaces intelligents a été implémentée dans le langage OWL, à l'aide de l'outil *Protege*, mais a été instanciée et supportée par la plateforme sémantique Jena. Cette plateforme, développée dans le langage Java, permet l'instanciation des concepts et instances définis dans la description sémantique et la recherche d'instances et de concepts via des requêtes dans le langage SPARQL, un langage basé sur la syntaxe du *Simple Query Language* (SQL). Cette plaforme permet également l'ajout d'engins de raisonnement et d'inférence comme *Pellet*[9], et de règles de description logique. Ces règles permettent d'inférer de nouveaux concepts et instances à partir de l'ontologie existante. Dans le FLORE, nous utilisons des requêtes SPARQL afin de récupérer un sous-ensemble des appareils satisfaisant les besoins en ressources des applications à déployer/organiser. Un exemple de requête SPARQL par le FLORE est présenté à la Section 3.5.

La future implémentation du module de collecte des données récupérera les données contextuelles provenant du projet Demeter (le framework évènementiel du DOMUS en cours d'implémentation) et ajoutera ou mettra à jour les instances de l'ontologie via la plateforme Jena. Ces données pourraient venir de sources telles qu'un système de localisation des utilisateurs,

[8]Archipel est une application développée au sein du Laboratoire DOMUS, donnant une assistance pas à pas à des utilisateurs dans la réalisation de tâches

[9]http ://clarkparsia.com/pellet/

des capteurs de température ou un système de surveillance/reconnaissance des activités. Des règles d'inférence seront ajoutées à la plateforme Jena afin d'inférer les zones dans lesquelles sont présents les utilisateurs ou les appareils mobiles après leur déplacement dans l'espace intelligent, etc.

La création de l'ontologie et son extension pour notre travail de thèse ont représenté un travail considérable. Plusieurs versions ont été créées avant de parvenir à la version actuelle. Toutefois, le choix technologique vis-à-vis le langage OWL et la plateforme Jena a été judicieux. Le langage OWL, basé sur le standard XML, est aisé à apprendre, lire et comprendre, par l'utilisation de terme usuel et de sa structure bien définie. La facilité d'ajouter de nouvelles instances à l'ontologie via la plateforme Jena a été également un avantage quant à l'utilisation de ces technologies. Petit bémol, si l'utilisation d'OWL et de l'outil Protege est relativement facile, l'utilisation de Jena et de ses API est beaucoup moins aisée. Une grande part du temps consacré à l'implémentation de l'ontologie et du module de gestion de l'ontologie a consisté à la création de classes et de méthodes permettant l'accès aux instances de l'ontologie via Jena. Afin de simplifier l'utilisation de l'ontologie dans l'intergiciel de gestion autonome, nous avons abstrait l'implémentation même de celle-ci par la création d'objets de type *JavaBean*. Ces objets simples en surface permettent l'accès aux données de l'ontologie via des méthodes de type *get* et *set*. Tout le travail d'accès et de requête vers Jena est encapsulé et abstrait par ces classes, simplifiant l'utilisation de l'ontologie pour les autres modules du projet Tyche. L'Extrait 3.1 présente un extrait de code d'un JavaBean permettant l'accès aux propriétés des zones de l'espace intelligent.

Extrait 3.1. Exemple d'un JavaBean simplifiant l'accès à l'ontologie

```
1  SmartSpaceZoneImpl implements SmartSpaceZone {
2
3      private static final String ADJACENT_PROPERTY = "adjacent"; private static
4      final String PARENT_PROPERTY = "parent"; private static final String
5      ROOM_LIMIT = "hasRoomLimitPosition"; private static final String
6      ROOM_LIMIT_X = "hasCoordinateX"; private static final String ROOM_LIMIT_Z =
7      "hasCoordinateZ"; private static final String ROOM_LIMIT_POSITION_ORDER =
8      "hasRoomLimitPositionOrder"; private static final String NAME = "hasName";
9
10     private Resource zoneIndividual = null; private String namespace = null;
11
12     public SmartSpaceZoneImpl(Resource zoneIndividual) {
13         this.zoneIndividual = zoneIndividual; this.namespace =
14         zoneIndividual.getNameSpace();
15     }// SmartSpaceZoneImpl(...)
16
17     @Override public String getZoneName() {
```

```
18          return zoneIndividual getLocalName(),
19      }// getZoneName()
20
21      public Resource getRessources() {
22          return this zoneIndividual,
23      } // getRessources()
24
25      @Override public Vector<SmartSpaceZone> getAdjacentZone() {
26          Vector<SmartSpaceZone> adjacentZones = new Vector<SmartSpaceZone>(),
27          Property property =
28          this zoneIndividual getModel() getProperty(this namespace +
29          ADJACENT_PROPERTY), Iterator it =
30          this zoneIndividual listProperties(property),
31
32          while (it hasNext()) {
33              Statement statement = (Statement) it next(), Resource zone =
34              (Resource) statement getObject(),
35
36              if (zone != null) {
37                  adjacentZones add(new SmartSpaceZoneImpl(zone)),
38              }// if (   )
39          }// while (   )
40
41          return adjacentZones,
42      }// getAdjacentZone()
43
44      @Override public SmartSpaceZone getParentZone() {
45          Property property =
46          this zoneIndividual getModel() getProperty(this namespace +
47          PARENT_PROPERTY), Statement statement =
48          this zoneIndividual getProperty(property),
49
50          if (statement != null) {
51              Resource zoneRessources = (Resource) statement getObject()  if
52              (zoneRessources != null) {
53                  return new SmartSpaceZoneImpl(zoneRessources),
54              }// if (   )
55          }// if (   )
56
57          return null,
58      }// getParentZone()
59      [   ]
60  }//SmartSpacezoneImpl
```

3.4 Auto-organisation logicielle des espaces intelligents

Les fonctionnalités d'auto-organisation du projet Tyche ont pour but de simplifier le déploiement d'applications dans des espaces intelligents en éliminant la majorité du raisonnement

sur les composantes de l'espace par les utilisateurs. Cette auto-organisation se doit donc d'être rapide, facile à utiliser et qu'il soit possible de manipuler celle-ci à distance afin de réduire les coûts en déplacement des utilisateurs/gestionnaires vers ces milieux [84].

Plusieurs stratégies quant à l'implémentation du raisonnement d'auto-organisation sur l'information contextuelle des espaces sont possibles :

1. extraction des données vers un outil de gestion chez les gestionnaires et raisonnement sur ces données à même l'outil de gestion (Utilisation de l'orchestration) ;
2. raisonnement basé sur une approche micro-contextuelle et la chorégraphie. L'outil de gestion n'est alors qu'une entité calculatoire[10] parmi les autres des espaces intelligents (entités appareils, entités utilisateurs, entités de localisation, etc.) ;
3. raisonnement basé sur une approche macro-contextuelle où le raisonnement est effectué par le noeud coordonnateur. Utilisation de l'orchestration. L'outil de gestion envoie les requêtes aux coordonnateurs des espaces intelligents ;
4. raisonnement basé sur une approche mixte où un agent coordonnateur gère le raisonnement d'organisation logicielle dans chaque espace intelligent, où l'outil de gestion envoie les requêtes vers les noeuds coordonnateurs des espaces intelligents et où une part du raisonnement est également laissé aux appareils des espaces.

La première stratégie a plusieurs désavantages : la nécessité d'accumuler ou du moins d'accéder à toute l'information contextuelle de chaque espace intelligent à gérer à partir de l'outil de gestion, un niveau élevé de communications entre l'outil de gestion et les espaces, la sécurité des informations, etc. La seconde stratégie présente des avantages certains : dynamismes, résistances aux fautes et bris, et autonomie des composantes. Par contre, elle est difficile à implémenter, nécessite un nombre important de messages échangés entre les agents et enfin une performance à vérifier. La troisième stratégie est à l'inverse de la deuxième stratégie : facile à implémenter, peu de messages échangés entre les composantes des milieux et une encapsulation du contexte dans les espaces intelligents vis-à-vis l'outil de gestion. Toutefois, elle est moins dynamique et résistante aux fautes. Enfin, la quatrième stratégie, qui a été évidemment choisie (tel que décrit dans la Section 3.2), allie les avantages de l'orchestration/macro-contexte et de la chorégraphie/micro-contexte.

En encapsulant et en plaçant le raisonnement sur le contexte à même les espaces intelligents, l'échange d'information entre l'outil de gestion et les milieux est réduit, la sécurité des informations est améliorée et les milieux reçoivent plus d'autonomie notamment en ce qui

[10] le terme d'agent peut également être utilisé

a trait à l'auto-optimisation des organisations, à l'organisation des logiciels provenant de requêtes internes aux milieux et aux autres futures fonctionnalités autonomes (auto-réparation, auto-protection).

3.4.1 Le Device Capability Quotient (DCQ)

Indépendamment de la stratégie utilisée, puisque la mécanique d'auto-organisation doit comparer les performances des appareils versus les besoins des applications et le contexte des milieux (utilisateurs, topologie, etc.), il était nécessaire de trouver une métrique d'évaluation suffisamment universelle pour être facilement assimilée par les utilisateurs et versatile pour être utilisée par les autres fonctionnalités du projet Tyche.

Depuis toujours les êtres humains ont donné des métriques d'évaluation qualitatives ou quantitatives aux performances de leurs activités, d'eux-mêmes et des objets qui les entourent. Les notes données aux élèves, les cotes en finance et la puissance des moteurs en chevaux vapeurs ne sont que quelques exemples de ce type de métriques d'évaluation. L'utilisation d'un quotient variant entre des valeurs bornées telles que les pourcentages ($[0, 1]$), est particulièrement intéressante puisque ce quotient permet de bien saisir les limites d'un système et est largement utilisé par la population, donc aisément assimilable.

Le projet Tyche utilise donc un quotient, le Device Capability Quotient (DCQ), comme métrique d'évaluation de la viabilité d'un déploiement d'une application sur un appareil en particulier. Le DCQ quantifie la performance d'un appareil, de ses ressources et de son micro-contexte, face aux besoins des applications et du macro-contexte des espaces intelligents (profil des utilisateurs, topologie). Afin d'intégrer la charge d'une application sur un appareil, le DCQ est calculé de façon à intégrer les ressources consommées par les applications sur les appareils en question, une fois le déploiement effectué. Le DCQ représente donc les performances globales d'un appareil après un hypothétique déploiement d'application sur un appareil. Afin d'améliorer la compréhension du DCQ auprès des utilisateurs/gestionnaires, nous avons fait varier les valeurs du DCQ entre $[0, 100[$ où 0 point représente l'incapacité totale d'un appareil à recevoir et exécuter une application et 100 points représentent un appareil dont les propriétés sont optimales aux besoins d'une application. Cette dernière valeur est théoriquement impossible à atteindre puisque le DCQ intègre dans son évaluation les ressources consommées par l'application, l'utilisation des ressources d'un appareil ne sera donc jamais nulle et se reflètera par une baisse dans la valeur du DCQ, parfois minime.

Lors du processus d'auto-organisation, selon les options utilisées par les utilisateurs (option

avec déploiement autonome désactivé, voir Section 3.7), les résultats retournés à l'outil de gestion correspondent à une liste d'appareils et de DCQ associés. Cette liste présentée en ordre décroissant de DCQ, offre aux utilisateurs une vue rapide de la situation, du classement des appareils et des options de déploiement/organisation (Figure 3.7).

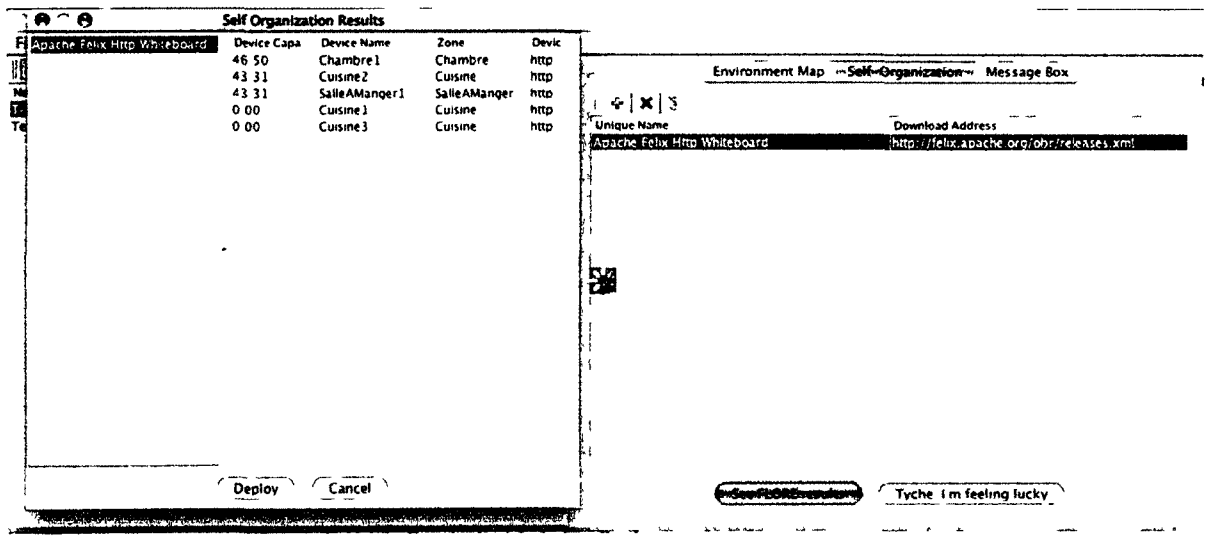

FIGURE 3.7 – Aperçu des résultats d'auto-organisation en mode manuel de l'outil de gestion du projet Tyche

Le DCQ agit également comme métrique de ralliement entre les fonctionnalités autonomes de l'intergiciel de gestion logicielle autonome. Dans l'approche de l'informatique diffuse autonome (Chapitre 2), l'auto-configuration/organisation est la fonctionnalité autonome sur laquelle les autres fonctionnalités se basent dans la réalisation de leurs tâches. Par exemple, dans le cas de l'auto-réparation et d'un bris matériel, cette fonctionnalité autonome utiliserait l'auto-organisation afin de trouver des appareils dans le milieu, pouvant héberger les applications de l'appareil en panne. Le DCQ dans ce cas serait utilisé par la fonctionnalité d'auto-réparation afin de comparer les solutions possibles et les mettre en pratique via l'auto-organisation. Exemple plus concret, la fonctionnalité d'auto-optimisation de l'organisation logicielle de Tyche utilise le DCQ afin d'évaluer les gains de performance. Le calcul du *Device Capability Quotient* est présenté dans les prochaines lignes présentant le *Fuzzy Logic Organization Reasoning Engine* (FLORE).

3.5 Le Fuzzy Logic Organization Reasoning Engine

L'engin de raisonnement d'auto-organisation, le FLORE, a comme tâche de calculer le DCQ pour chaque appareil présent dans les milieux, et ce, pour chaque application à déployer. Pour ce faire, le FLORE implémente un algorithme utilisant des règles de raisonnement en description logique et en logique floue.

Le FLORE utilise donc la logique de description, à même l'ontologie des espaces intelligents, et la logique floue afin d'évaluer la viabilité des appareils des espaces intelligents. Chacune de ses approches de raisonnement a ses avantages et inconvénients. La logique de description [40] est particulièrement puissante sur des valeurs quantitatives, la sémantique des concepts et l'inférence de concepts à partir des sémantiques. Par contre, elle a comme désavantage d'être relativement lourde en mémoire et en temps de calcul (tout dépendant de la taille des ontologies). La logique floue [101] quant à elle a été conçu pour raisonner sur des valeurs qualitatives versus des valeurs quantitatives. Elle est également rapide (relatif au nombre de règles et de fonctions d'appartenance) et gère bien l'imprécision des données. La conception du FLORE repose particulièrement sur l'utilisation de la logique floue dans le calcul du DCQ.

Un dernier avantage de la logique floue est sa facilité à modéliser un raisonnement sur des concepts et problèmes complexes, sans le besoin de développer des modèles mathématiques, parfois difficile à définir (dans une perspective de *soft computing*). C'est cet avantage qui a mené la logique floue à être utilisé dans les freins ABS (gestion de la température versus le freinage) et la gestion de la température dans les centrales nucléaires, où la modélisation mathématique de la gestion de la température en temps réel est un problème complexe. Dans le cas de Tyche, la logique floue a permis de raisonner sur l'organisation logicielle en tenant compte de plusieurs contraintes tout en gardant un niveau de complexité relativement bas. Le raisonnement sur des valeurs qualitatives a simplifié la construction des règles de raisonnement en évitant une modélisation mathématique poussée du problème. Évidemment, la simplification du problème a un prix, avec les marges d'erreur induites par les fonctions d'appartenance (introduit un peu plus bas), mais ces marges d'erreur dans le contexte de la gestion autonome sont minimes et ont au final peu d'impact sur les organisations proposées.

D'autres approches de raisonnement auraient pu être utilisées, telles qu'une approche probabiliste basée sur un modèle bayésien. Par exemple, les modèles markoviens cachés [102] permettent de modéliser un processus de résolution de problèmes à partir de faits, désigné comme des états dans un réseau bayésien [103], partiellement observables à partir d'observations indirectes et des probabilités de transition entre les divers états du réseau. Les modèles

markoviens cachés sont particulièrement adaptés à l'organisation logicielle, puisque l'état futur de l'organisation (partiellement observable) peut être déduit des informations contextuelles contenues dans l'ontologie et les noeuds appareils de Tyche. Toutefois, l'utilisation d'un modèle bayésien nécessite des connaissances approfondies sur les probabilités de transitions entre les divers états possibles du réseau bayésien. Dans le cas de l'organisation logicielle, la quantité importante d'information, le nombre important de possibilités d'états nécessiterait un modèle bayésien complexe, avec plusieurs états, transitions et évidemment de probabilités rattachées à ces transitions. Ces modèles doivent également être dynamiques par la nature mouvante des espaces intelligents, où des appareils, des applications et des utilisateurs peuvent entrer et sortir à tout moment des milieux. Toutefois, la principale difficulté dans la création d'un tel modèle vient de l'assignation des probabilités de transition aux états du réseau bayésien. Comme aucune donnée recueillie n'existe sur les organisations logicielles dans les espaces intelligents, il est impossible, dans notre contexte, d'utiliser une stratégie d'apprentissage afin d'en tirer les probabilités du modèle. Une assignation manuelle des probabilités basées sur une perception humaine du modèle serait alors nécessaire, une tâche longue et complexe, qui amènerait un nombre important d'erreurs de modélisation. Une approche basée sur la planification de l'organisation pourrait également être utilisée, mais la nature dynamique et non-déterministe des espaces intelligents amène un lot de contraintes et sont discutées dans la Section 3.5.2.

Enfin, il existe dans le domaine de l'intelligence artificielle une quantité incroyable d'approches et d'algorithmes pouvant être appliqués de près ou de loin à la problématique de l'organisation logicielle. Par exemple, les réseaux de neurones sont particulièrement utile dans les cas où un apprentissage est nécessaire, ce qui n'est pas à priori un besoin primaire pour l'organisation autonome. Dans l'état de l'art des travaux sur l'informatique autonome (Chapitre 2), nous avons présenté quelques travaux [32] [74] utilisant un raisonnement par cas. Une telle approche permet de pré-programmer des réponses à certains contextes pouvant être rencontrés lors d'une organisation logicielle, mais elle est plus lourde que la logique floue, moins flexible en terme de résultats et nécessite, encore là, la construction d'une base de cas, tâche qui peut s'avérer longue et ardue.

Avec la logique floue, une assignation manuelle a également été nécessaire telle que pour une approche bayésienne sans apprentissage, mais à moindre échelle. Les ensembles flous, avec leurs fonctions d'appartenance, ont été créés à partir de données bibliographiques (Section 3.5.5) et des comportements reliés à la logique même de l'organisation logicielle. En bref, l'avantage fondamental de la logique floue est sa facilité de modélisation de problèmes complexes avec des informations alliant valeurs quantitatives et qualitatives. Les prochains pa-

ragraphes présentent une courte introduction sur la logique floue alors qu'un peu plus loin son utilisation dans le cadre du FLORE est détaillée.

3.5.1 Introduction à la logique floue

La logique floue a été créée en 1969 par Lotfi Zadeh [104]. Basée sur la théorie des ensembles floues formalisée par Zadeh, la logique floue étend la logique des ensembles classiques pour prendre en compte des ensembles plutôt imprécis. La force de la logique floue vient donc de sa capacité à traiter des données imprécises, mouvantes et des concepts imprécis, contrairement à la logique traditionnelle, la logique *crispée*, qui nécessite des données exactes et fixes. L'être humain, sans le savoir, utilise dans la majorité de ses actions des raisonnements comparables à ceux de la logique floue. Par exemple lors d'une partie de volley-ball, la vitesse et l'effet sur le ballon sont constamment attribués à des fonctions d'appartenance et des ensembles flous : rapide, lent, rotation faible, rotation rapide, etc. Intrinsèquement, l'être humain traduit ces observations en des valeurs qualitatives, fait des raisonnements sur ces valeurs et puis prend des décisions encore là en rapport à des valeurs qualitatives : se déplacer lentement, rapidement, frapper le ballon doucement, fort, etc. Pour ce qui est de l'utilisation de la logique floue en sciences et en ingénierie, celle-ci est utilisée entre autres en automatisme : les systèmes à freins ABS et les systèmes d'air climatisé, en météorologie lors des prévisions météorologiques, en intelligence artificielle notamment dans les systèmes de diagnostic.

L'exemple traditionnel en logique floue demeure le contrôle de la température dans des fluides [101]. Supposons une cuve d'eau qui doit être maintenue à une température tiède. Le concept tiède est déjà flou puisqu'il ne correspond pas à une température précise, mais à un ensemble de valeur avec des degrés d'appartenance variable à cet ensemble. Dans l'hypothétique système, si la température de 27 degrés Celsius est membre à part entière de cet ensemble, donc ayant un degré d'appartenance de 100%, la température de 24 degrés Celsius peut être considérée à la fois comme tiède et froide, donc avoir, par exemple, un degré d'appartenance de 70% à l'ensemble tiède et 30% à l'ensemble froid. La Figure 3.8 présente ces ensembles et leurs hypothétiques fonctions d'appartenance. L'action d'associer des valeurs d'entrées à des ensembles floues et à leurs fonctions d'appartenances se nomme dans le jargon du domaine la *fuzzification*, le passage d'une valeur *crispée* à l'univers flou.

Un système de contrôle de la température utiliserait alors ces ensembles et des règles de contrôle pour s'assurer que la température de la cuve reste dans des valeurs relatives à la fonction d'appartenance tiède, selon une certaine marge correspondant à une limite dans les degrés

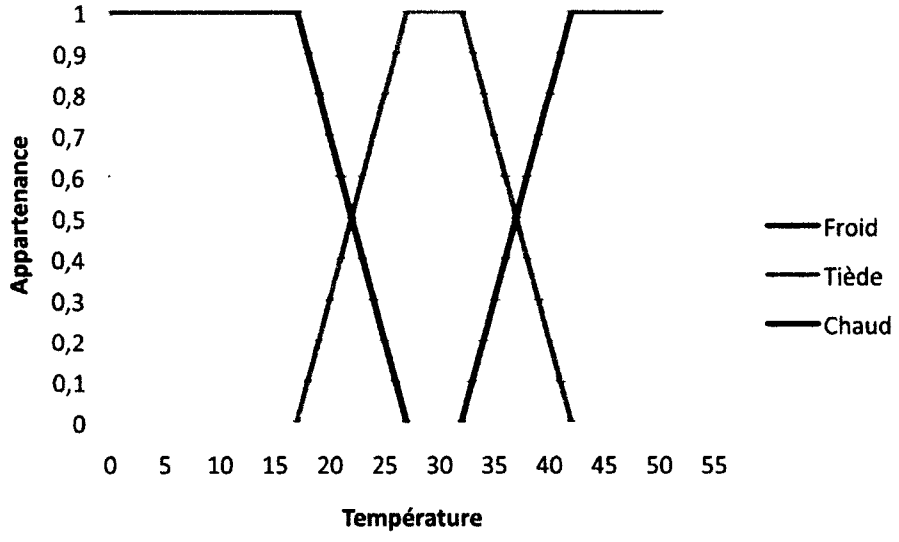

FIGURE 3.8 – Fonction d'appartenance des ensembles flous Froid, Tiède et Chaud

d'appartenance, en y ajoutant de l'eau chaude ou de l'eau froide. L'extrait 3.2 présente un exemple de règle de raisonnement dans la notation utilisée dans la logique floue.

Extrait 3.2. Exemple de règles de logique floue pour le contrôle de la température

```
SI température EST froide ALORS ajouter_eau_chaude
SI température EST tiède ALORS ajouter_eau_chaude_et_froide
SI température EST chaude ALORS ajouter_eau_froide

//Il est également possible de faire des règles plus complexes,
//Exemple :
SI température EST tiède ET température EST froide ALORS
ajouter_eau_chaude
```

L'évaluation des règles respecte les lois de Morgan quant à logique traditionnelle. Chacun des opérateurs à une opération mathématique "floue" associée [41]. Par exemple, l'opérateur "ET" est traditionnellement associé à la fonction minimum (min(A,B)) où la valeur minimale entre le degré d'appartenance de A et de B est conservée. Par exemple, dans la quatrième règle de l'extrait 3.2, si la température d'entrée de la cuve est de 24 degrés Celsius alors le degré d'appartenance sélectionné sera de 30%. Chaque résultat d'une règle est alors associé à un ensemble flou et des fonctions d'appartenances que l'on nomment ensemble de *defuzzification*, permettant le passage de valeurs floues (quantitatives) vers des valeurs *crispées* (qualitatives). Ces ensembles cumulent alors les résultats de toutes les règles, puis effectuent une opération

mathématique afin d'évaluer la valeur moyenne de l'ensemble flou. L'opération mathématique la plus souvent utilisée est le centroïde. La Figure 3.9 présente un exemple d'ensemble de *defuzzification*, le cumul des résultats des règles de l'extrait 3.2 et son centroïde. Selon les fonctions d'appartenance de la Figure 3.9, le résultat du raisonnement par logique floue pour une température de 24 degrés Celsius correspondrait alors à l'ouverture de la valve d'eau chaude à 56 % (centroïde de l'ensemble de defuzzification) et à l'ouverture de la valve d'eau froide à 44 % (1 - ouverture de la valve d'eau chaude). Cet exemple de raisonnement par logique floue reste très simple, n'ayant que trois règles et n'incluant qu'un ensemble de *fuzzification*. Dans le cadre du projet Tyche, la somme des deux parties du FLORE (partie appareil et coordonnateur) compte cinquante règles, deux ensembles de *defuzzification* et neuf ensembles de *fuzzification*.

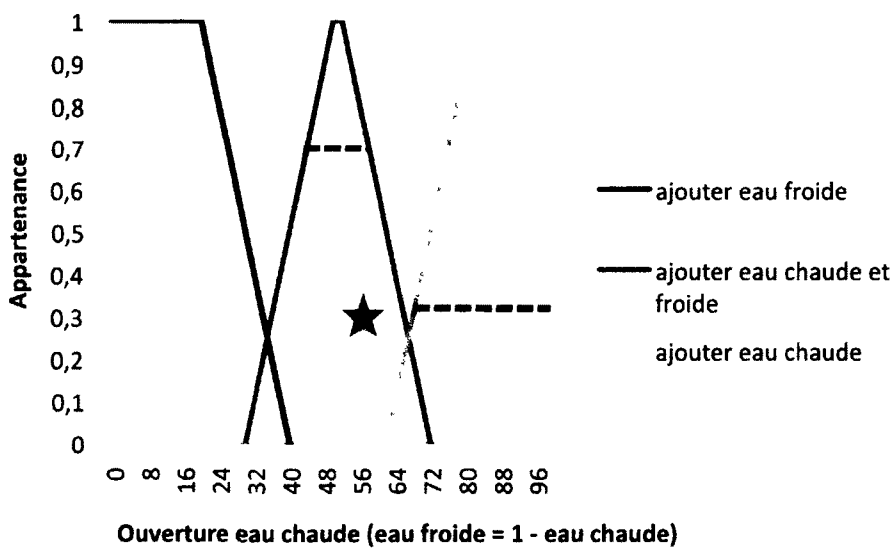

FIGURE 3.9 – Fonctions de *defuzzification*, le résultats des règles et le centroïde

L'implémentation du FLORE n'utilise que les concepts de base de la logique floue. Plusieurs extensions à la théorie ont été ajoutées au cours des dernières décennies. Par exemple, il est possible de définir des fonctions d'appartenance à trois dimensions (logique floue Type-2 [105]), les deux premières étant les variables d'entrée et le degré d'appartenance aux fonctions, tel que dans le modèle de base présenté plus haut, et la troisième variable correspondant à une marge d'erreur relative quant au degré d'appartenance. D'autres extensions à la logique floue permettent également un apprentissage et une modification dynamique des ensembles flous et des fonctions d'appartenance en utilisant de l'auto-apprentissage, par exemple, à l'aide d'algorithme génétique [106]. Ces extensions pourraient être utilisées dans les prochaines ver-

sions du FLORE où une marge d'erreur serait associée aux fonctions d'appartenance correspondant à la vitesse de marche des utilisateurs, leur position, etc. L'auto-apprentissage permettrait également de modifier les fonctions d'appartenance dynamiquement en regard de la fréquence des évènements et du contexte des espaces intelligents.

3.5.2 L'algorithme du FLORE

L'implémentation du FLORE est divisée en deux parties déployées dans le noeud coordonnateur des espaces intelligents et les noeuds appareils. Le FLORE partie appareil (FLORE-A) évalue les performances des appareils face aux besoins d'une application en ce qui concerne l'utilisation des ressources, la zone où l'appareil est situé et la présence de périphériques. Le FLORE partie coordonnateur (FLORE-C) évalue dans un premier temps les besoins de base d'une application en ressources versus les ressources des appareils. Si nécessaire, il évalue dans un deuxième temps le profil des utilisateurs et le contexte autour de l'utilisateur versus les appareils et leurs périphériques. Enfin, il intègre le DCQ de la partie application et de l'appareil pour former le DCQ final.

L'algorithme du FLORE comporte trois étapes selon les besoins et caractéristiques des applications à déployer et organiser : l'étape générale de raisonnement sur le micro-contexte, l'étape de traitement des zones et l'étape de traitement du profil des utilisateurs. La première étape concerne le déploiement d'une application qui a des besoins en ressources et en périphériques, mais ne vise pas un déploiement dans un appareil d'une zone particulière de l'espace intelligent, ni un déploiement pour un utilisateur de l'environnement. Dans cette étape, la majeure partie du raisonnement se fait sur le micro-contexte des appareils et est donc fait par ces derniers.

La deuxième étape concerne le déploiement d'une application visant une zone particulière des espaces. Dans ce cas-ci, une évaluation entre la zone des appareils et la zone visée par les applications est effectuée dans le noeud coordonnateur. Cette évaluation utilise les liens sémantiques entre les zones pour définir la proximité d'une zone avec les autres zones des milieux et les relations d'inclusion entre des zones. Cette évaluation est ensuite retournée aux noeuds appareils qui l'incluent dans l'évaluation du premier DCQ.

La troisième étape concerne le déploiement d'une application visant une utilisation particulière par un utilisateur des espaces intelligents. Cette évaluation intègre alors le profil de l'utilisateur : ses caractéristiques, ses préférences et ses capacités et les données des profils versus le contexte des milieux dans le calcul du DCQ.

Présenter l'ensemble de l'algorithme du FLORE dans cette thèse seraient long et sa lecture complexe pour les lecteurs. Une version abrégée et simplifiée est donc présentée sous forme de pseudo-code dans l'Extrait 3.3.

Extrait 3.3. Algorithme de l'évaluation de l'organisation par le FLORE partie coordonnateur (FLORE-C)

```
1
2     Liste<Liste<Pair<Double, Appareil>>> resultats,
3
4     POUR TOUTE application DE applications {
5         //Filtre les appareils repondant aux besoins en ressources de l'application
6         Liste<Appareil> appareils = Ontologie avoirAppareil(application),
7
8         Liste<Pair<Double, Appareil>> resultatApplication,
9
10        POUR TOUS appareil DE appareils {
11            Double DCQ_Final,
12            Double DCQ_appareil,
13            //Étape 1   general être
14            SI application zone = NULLE {
15                //Appel via le webservice, du FLORE-A
16                DCQ_appareil = appareil calculDCQ(application), }
17            //Étape 2   zone
18            SINON {
19                //Evaluation des liens entre les zones
20                Lien_Zone lien_Zone = evaluerZones(appareil zone,application zone),
21                DCQ_appareil = appareil calculDCQ(application, lien_Zone),
22            }
23            //Étape3   profil utilisateur
24            SI application utilisateur != NULLE {
25                Double DCQ_utilisateur = evaluerProfilUtilisateur(appareil, application,
26                application utilisateur),
27
28                SI DCQ_utilisateur = 0  OU DCQ_Appareil = 0 {
29                    DCQ_Final = 0,
30                } SINON {
31                    //Moyenne des deux DCQs
32                    DCQ_Final = (DCQ_utilisateur + DCQ_appareil) / 2,
33                }
34            } SINON {
35                DCQ_Final = DCQ_appareil,
36            }
37            resultatApplication ajouter({DCQ_Final, appareil}),
38        }
39        Pair<DCQ, appareil> maxDCQ = maximum_DCQ(resultatApplication),
40        maxDCQ appareil deployer(application),
41        resultats ajouter(resultatApplication),
42    }
43    retourner resultats,
```

44 }

En résumé, cet algorithme itère donc sur les applications à déployer définies par les requêtes d'organisation. Pour chaque application, l'algorithme récupère le DCQ final des appareils, puis déploie l'application en question sur l'appareil ayant le DCQ le plus élevé. Cette implémentation d'auto-organisation est simple et rapide avec une complexité algorithmique générale de O(kn) où k est le nombre d'applications à déployer et n est le nombre d'appareils d'un milieu.

Toutefois, cet algorithme ne correspond pas à la meilleure solution d'auto-organisation, car celle-ci ne trouve pas l'organisation optimale pour un groupe d'applications à déployer. Puisque le FLORE tient compte des ressources et de leurs utilisations dans le calcul des DCQ, le déploiement d'une première application sur un appareil d'un espace intelligent influence directement le calcul des DCQ suivants. Prenons par exemple trois applications à déployer et trois appareils dont les résultats des DCQ seraient calculés indépendamment sans répercussion des consommations de ressources d'un calcul à l'autre (Tableau 3.2).

TABLEAU 3.2 – Exemple de DCQ pour des applications à déployer et organiser

	Appl.1	Appl.2	Appl.3
Appareil 1	80	79	0
Appareil 2	79	48	80
Appareil 3	0	78	82

Dans cet exemple les appareils choisis comme cible de déploiement seraient alors l'appareil 1 pour l'application 1, l'appareil 1 pour l'application 2 et l'appareil 3 pour l'application 3. En poussant le problème un peu plus loin, supposons que chaque déploiement d'application abaisse le DCQ des évaluations suivantes pour un appareil choisi de cinq points, dû aux consommations des ressources. Suivant l'algorithme du FLORE présenté à l'Extrait 3.3, la valeur du DCQ de l'appareil 1 pour l'évaluation de l'application 2 diminuerait donc pour se situer à 74 points. Le meilleur appareil serait non plus l'appareil 1, mais l'appareil 3 avec 78 points. Par effet de cascade, le DCQ pour l'évaluation de l'appareil 3 pour l'application 3 diminuerait également pour se situer à 78 points et l'appareil ayant le plus grand DCQ deviendrait alors l'appareil 2 avec 80 points. La moyenne des DCQ des appareils choisis serait alors de 79,33 points.

Or, une solution plus optimale existe entrainant pour chaque application à déployer des DCQ résultants plus élevés. Par exemple, en sachant à priori que l'utilisation de l'appareil 1 pour l'application 1 va entraîner une baisse de son DCQ pour l'application 2, il serait possible d'utiliser plutôt, l'appareil 2. La solution finale serait alors l'appareil 2 pour l'application

1, l'appareil 1 pour l'application 2 et l'appareil 3 pour l'application 3. Le DCQ moyen de cette solution serait alors de 80 points, légèrement supérieur à la solution implémentée par le FLORE.

La problématique du FLORE et de la consommation de ressources dans l'évaluation du DCQ est un cas classique de problèmes de planification avec une contrainte de ressources (RCPSP) [107] et se classe comme étant de type NP complet [108]. À priori, la solution optimale dans le cas du projet Tyche peut être trouvée en évaluant l'ensemble des possibilités d'organisation logicielle et en prenant celle donnant le meilleur résultat moyen. La complexité exacte de cet algorithme serait alors de $O(k^2n)$. Par contre, l'implémentation d'un tel algorithme est impossible ou du moins très complexe à mettre en oeuvre. Un tel algorithme nécessiterait de connaître à priori l'utilisation des ressources futures, tel que dans le Tableau 3.2, alors que celles-ci peuvent varier de façon imprévisible selon le type de système d'exploitation, le type de matériel, l'utilisation des appareils par les utilisateurs, le traitement d'information relative à un service, etc. De plus, dans le cas d'application associée à des utilisateurs, la position et l'orientation de ceux-ci ont un impact direct sur le calcul du DCQ, ce qui est impossible à prévoir. Certains travaux en planification utilisent des modèles probabilistes et statistiques [109] pour tenter d'inférer des états futurs, mais ils s'appliquent difficilement à notre contexte. Toutefois, dans les prochaines évolutions du FLORE, une approche basée sur les RCPSP sera étudiée afin de modéliser jusqu'à un certain point les états futurs et de s'approcher d'une solution optimale.

Quoique la solution du FLORE présentée dans l'Extrait 3.3 ne trouve pas l'organisation optimale, elle offre des avantages non négligeables. L'algorithme d'organisation du FLORE nécessite moins de mémoire et s'effectue avec un temps de traitement plus rapide. Le troisième avantage de notre approche consiste en la notion de priorité qui est créée entre les applications. Ainsi, la première application à déployer contenue dans une requête d'organisation a un droit de veto sur l'appareil à choisir et utilisera, dans tous les cas, l'appareil donnant le meilleur DCQ. Les applications suivantes auront alors une priorité inférieure quant aux ressources disponibles que les applications précédentes, mais une priorité supérieure aux applications subséquentes. Cette relation de priorité peut être utilisée avantageusement pour déployer des applications où l'utilisation des ressources a un impact important sur leur qualité de service/interaction.

Les trois prochaines sections présenteront en détail les trois étapes du raisonnement du FLORE introduit précédemment et inclus dans l'algorithme d'organisation présenté dans l'Extrait 3.3.

3.5.3 Raisonnement sur le micro-contexte des appareils

Le raisonnement sur le micro-contexte est effectué en trois étapes, la première s'effectuant sur le noeud coordonnateur avec la présélection d'appareils à partir de l'ontologie et les deux dernières étapes sur le noeud appareil avec la vérification des périphériques attachés à l'appareil et l'évaluation du DCQ liée à l'utilisation des ressources de l'appareil.

La présélection des appareils est faite via le module de gestion de l'ontologie. À cette étape, une requête formulée dans le langage SPARQL est envoyée à la plateforme Jena, puis transformée par Jena sous forme de règles de description logique. Cette requête est utilisée afin de préfiltrer les appareils des environnements qui répondent aux besoins de base en ressources des applications. Contrairement à la description de l'algorithme présentée dans l'Extrait 3.3, l'implémentation réelle du FLORE itère parmi les appareils potentiels à même la requête SPARQL. Cette requête SPARQL est présentée à l'Extrait 3.4. Ainsi, pour chaque appareil répondant aux besoins minimaux en ressources, la méthode *FLOREDeviceEvaluationARQFunction* est appelée, méthode encapsulant les autres étapes de l'algorithme du FLORE.

Extrait 3.4. Extrait de la classe OntologyQueryServiceImpl où la requête SPARQL est formulée

```
61 OntologyQueryServiceImpl implements OntologyQueryService { [ ] public ArrayList<
       FloreDeviceResultBean> getSpaceDevicesMatchingApplication(
62             SmartSpaceApplication application, ConfigurationReasoner configurationService) {
63
64         ArrayList<FloreDeviceResultBean> evaluationResults = new ArrayList<
               FloreDeviceResultBean>(),
65         FunctionRegistry get() put(
66                 "http //domus usherbrooke ca/tyche#FLOREDeviceEvaluationARQFunction",
67                 new TycheARQFunctionFactory(application, this, configurationService)),
68
69         // Formatting SPARQL query
70         String query = "PREFIX␣pre ␣<" + driver getOntologyPath() + "#>'
71                 + "PREFIX␣f ␣<http //domus usherbrooke ca/tyche#>"
72                 + "SELECT␣?" + DEVICE_TYPE + "␣?dcq␣"
73                 + "WHERE␣{"
74                 + "␣␣␣␣␣?" + DEVICE_TYPE + "␣pre hasCPU␣?cpu␣ "
75                 + "␣␣␣␣␣FILTER␣(?cpu␣>=␣" + application getNeededCPU() + ")"
76                 + "␣␣␣␣␣?" + DEVICE_TYPE + "␣pre hasRAM␣?ram␣ ␣"
77                 + "␣␣␣␣␣FILTER␣(?ram␣>=␣" + application getNeededRAM() + ")"
78                 + "␣␣␣␣␣?" + DEVICE_TYPE + "␣pre hasPermanentMemory␣?pm␣ ␣"
79                 + "␣␣␣␣␣FILTER␣(?pm␣>=␣" + application getNeededPermanentMemory() + ")"
80                 + "␣␣␣␣␣?" + DEVICE_TYPE + "␣pre hasStatus␣\"Started\"␣ ␣"
81                 + "␣␣␣␣␣LET␣(?dcq␣ =␣f FLOREDeviceEvaluationARQFunction(?" + DEVICE_TYPE + ")
                       )␣ ␣"
82                 + "␣␣␣␣␣FILTER␣(?dcq␣>=␣0)"
83                 + "␣}␣ORDER_BY_DESC(?dcq)",
```

```
84
85          // Executing the query on the smart space ontology
86          List<QuerySolution> results = driver executeSparQLQuery(query),
87
88          Iterator<QuerySolution> itSolutions = results iterator(),
89          // Extracting the results under the form of SmartSpaceDevice Beans
90          while (itSolutions hasNext()) {
91              QuerySolution solution = itSolutions next(),
92              SmartSpaceDevice device = new SmartSpaceDeviceImpl(solution getResource(
                    DEVICE_TYPE)),
93              float dcq = solution getLiteral("dcq") getFloat(),
94              FloreDeviceResultBean resultBean = new FloreDeviceResultBean(),
95              resultBean setDevice(BeanFactory createDeviceBeanFromSmartSpaceDevice(device)),
96              resultBean setDcq(dcq),
97              evaluationResults add(resultBean),
98          }
99          return evaluationResults,
100     }
101     [   ]
102 }
```

Dans la seconde étape du raisonnement sur le micro-contexte des appareils, la présence des périphériques nécessaires à l'utilisation de l'application est vérifiée. Dans la description contextuelle, les périphériques sont classés en trois types : les périphériques d'écriture (*keying*), les périphériques de pointage (*pointing*) et les périphériques d'affichage. Ces types de périphériques peuvent alors être dérivés en plusieurs sous-types : souris, trackball, clavier virtuel, etc. La vérification de la présence d'un périphérique chez un appareil n'est pas faite sur l'instance même d'un périphérique, par exemple sur la souris de l'ordinateur du salon, mais sur le type ou le sous-type de périphérique. Ainsi, une application peut nécessiter l'utilisation d'un périphérique générique de pointage ou bien raffiner ses besoins et nécessiter la présence chez un appareil d'une souris par exemple. Cette vérification ne nécessite pas de donnée provenant du macro-contexte (hormis le profil de l'application) et est effectuée à l'aide d'expressions régulières. Par exemple, si un appareil possède un périphérique de type *pointage* et de sous type *trackball*, la correspondance ou *matching* avec une application nécessitant un périphérique de type pointage serait positif, alors qu'une application nécessitant une souris serait négatif. Si la vérification de la présence de périphériques échoue, le DCQ de l'appareil reçoit automatiquement une valeur de 0 point.

Enfin, la troisième étape consiste en l'évaluation des ressources disponibles sur les appareils face aux besoins des applications. Cette évaluation utilise les fonctionnalités de la logique floue afin de qualifier l'utilisation des ressources des appareils vers des fonctions d'appartenance. Des règles sont ensuite utilisées afin d'évaluer ces utilisations et d'y attribuer ensuite

une valeur quantitative propre à l'aide d'un ensemble de *defuzification*, i.e. le DCQ appareil, à différencier avec le DCQ utilisateur et le DCQ final tel que dans l'Extrait 3.3. L'utilisation des trois types de ressources est associée à trois ensembles flous ayant chacun, quatre fonctions d'appartenance : ressource libre, ressource utilisée, ressource très utilisée et ressource saturée. Ces fonctions d'appartenance peuvent être de type linéaire, logarithmique ou gaussienne, et peuvent être personnalisée pour le matériel d'un appareil. Dans le cadre de nos tests et évaluations (Chapitre 4), nous avons utilisé des fonctions d'appartenance gaussienne, afin de représenter une distribution normalisée de l'utilisation des ressources, ce qui est également le cas pour les fonctions d'appartenance à l'ensemble de *defuzzification*. La Figure 3.10 et 3.11 illustre ces ensembles flous avec leurs fonctions d'appartenance.

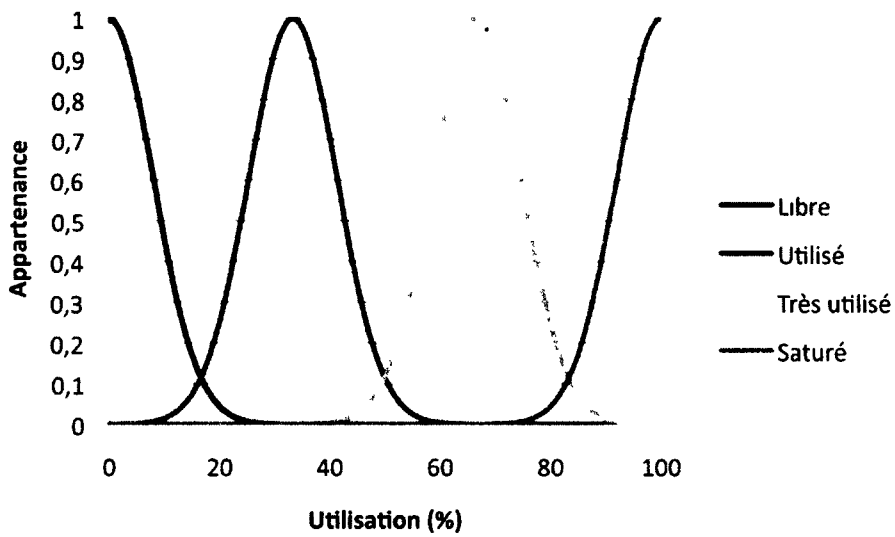

FIGURE 3.10 – Fonctions de *fuzzification* de l'utilisation des ressources du FLORE partie appareil (FLORE-A)

Les règles de logique floue du FLORE-A utilisent donc ces ensembles flous, ainsi que la relation entre les zones, introduite à la prochaine section, dans l'évaluation du DCQ. Pour cette partie du FLORE, le nombre de règles de raisonnement est réduit à neuf règles. Ces règles implémentées à l'aide de la spécification *Fuzzy Control Language* (FCL IEC 61131 partie 7), sont présentées en entrée, avec la description des fonctions de *fuzzification* et de *defuzzification*, à l'outil *JFuzzyLogic*[11], une implémentation Java d'un contrôleur de logique floue. Un extrait des données envoyées vers le contrôleur de logique floue est présenté dans l'Extrait 3.5. Le

[11] JFuzzyLogic : http ://jfuzzylogic.sourceforge.net

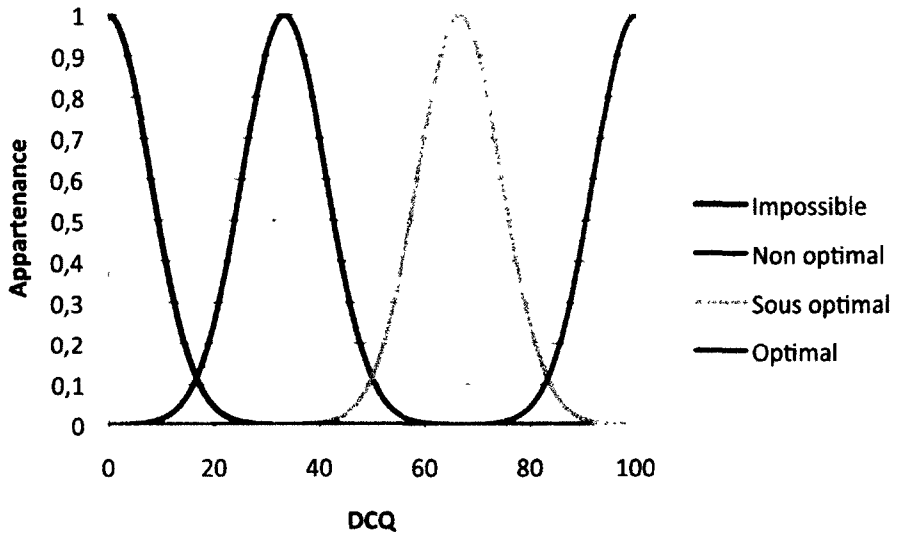

FIGURE 3.11 – Fonctions de *defuzzification* du FLORE partie appareil (FLORE-A) vers le DCQ

résultat du raisonnement correspond au DCQ appareil et est retourné au FLORE-C via le retour de l'appel du *webservice* de gestion et de raisonnement des noeuds appareils.

Extrait 3.5. Extrait du fichier de description du système de logique floue pour le FLORE partie appareil

```
1  FUNCTION_BLOCK deviceDCQEvaluation
2  [...]
3  FUZZIFY deviceCPUSaturation
4          TERM free := gauss 0 8;
5          TERM used := gauss 33.66 8;
6          TERM veryUsed := gauss 66.66 8;
7          TERM saturated := gauss 100 8;
8  END_FUZZIFY
9  [...]
10 FUZZIFY zoneSimilarity
11         TERM same := 4;
12         TERM directlyAdjacent := 3;
13         TERM sameParent := 2;
14         TERM dissociated := 1;
15 END_FUZZIFY
16 DEFUZZIFY ressourcesQualification
17         TERM impossible := gauss 0 8;
18         TERM subOptimal := gauss 66.66 8;
19         TERM notOptimal := gauss 33.33 8;
20         TERM optimal := gauss 100 8;
21         METHOD : COG;          // Use 'Center Of Gravity' (Centroid)
```

```
22          DEFAULT = 0,
23          RANGE = (0     100),
24  END_DEFUZZIFY
25
26  RULEBLOCK No1
27          AND   MIN,                          // Use 'min' for  and' (also implicit use 'max' for '
                or' to fulfill DeMorgan's Law)
28          ACT   MIN,                          // Use 'min' activation method
29          ACCU   MAX,                         // Use 'max' accumulation method
30
31          RULE 1   IF deviceCPUSaturation IS free AND deviceRAMSaturation is free AND
                devicePMSaturation is free AND zoneSimilarity is same THEN
                ressourcesQualification IS optimal WITH 1,
32          RULE 2   IF (deviceCPUSaturation IS free OR deviceRAMSaturation is free OR
                devicePMSaturation is free) AND
33                                  (deviceCPUSaturation IS used OR deviceRAMSaturation is used OR
                                        devicePMSaturation is used) AND
34                                  zoneSimilarity is same THEN ressourcesQualification IS optimal
                                    ,
35          RULE 3   IF (deviceCPUSaturation IS free OR deviceRAMSaturation is free OR
                devicePMSaturation is free) AND
36                                  (deviceCPUSaturation IS used OR deviceRAMSaturation is used OR
                                        devicePMSaturation is used) AND
37                                  zoneSimilarity is directlyAdjacent THEN
                                        ressourcesQualification IS subOptimal,
38          RULE 4   IF (deviceCPUSaturation IS used OR deviceRAMSaturation is used OR
                devicePMSaturation is used) AND
39                                  (deviceCPUSaturation IS veryUsed OR deviceRAMSaturation is
                                        veryUsed OR devicePMSaturation is veryUsed) AND
40                                  zoneSimilarity is same THEN ressourcesQualification IS
                                        subOptimal,
41          RULE 5   IF deviceCPUSaturation IS veryUsed AND deviceRAMSaturation is veryUsed AND
                devicePMSaturation is veryUsed AND
42                                  zoneSimilarity is same THEN ressourcesQualification IS
                                        subOptimal,
43          RULE 6   IF deviceCPUSaturation IS veryUsed OR deviceRAMSaturation is veryUsed OR
                devicePMSaturation is veryUsed THEN ressourcesQualification IS notOptimal,
44          RULE 7   IF deviceCPUSaturation IS saturated OR deviceRAMSaturation is saturated OR
                devicePMSaturation is saturated THEN ressourcesQualification IS impossible,
45          RULE 8   IF zoneSimilarity is sameParent THEN ressourcesQualification is notOptimal,
46          RULE 9   IF zoneSimilarity is dissociated THEN ressourcesQualification is impossible
                WITH 1,
47  END RULEBLOCK
```

3.5.4 Raisonnement sur les zones des espaces intelligents

Les espaces intelligents dans la description contextuelle macroscopique sont divisés en zones correspondant aux divisions physiques des milieux, tel que des pièces, des chambres,

ou à des divisions logiques telles que des espaces de travail. Ces divisions de l'espace peuvent être également appliquées à des meubles ou autres objets pouvant contenir d'autres objets tels qu'un réfrigérateur ou un placard.

La description contextuelle définit également trois types de relations entre ces zones. La première est une hiérarchie de type mère-fille pouvant lier des zones entre elles, créant une relation d'inclusion. Ainsi, une sous-zone telle qu'un réfrigérateur peut être incluse dans la zone cuisine en ayant comme zone mère cette dernière. Dans le cas de l'appartement du DOMUS, la zone "Appartement Domus" est la zone mère des zones cuisine, salon, chambre de bain, etc. La deuxième relation possible est celle d'adjacence où deux zones mitoyennes sont directement et physiquement accessible de l'une à l'autre. Le troisième type de relation concerne la relation de zone "soeur", dans le cas où des zones ont une zone mère commune. Ces relations sont utilisées par le FLORE dans son évaluation de l'organisation logicielle.

Avec les zones et leurs relations, il est possible de faire des organisations et des déploiements visant les appareils d'une zone particulière. Par exemple, une application d'assistance à la préparation de repas (Archipel) pourrait être liée à la zone cuisine et ainsi privilégier les appareils de cette zone dans l'évaluation de l'organisation logicielle. Dans les règles de raisonnement du FLORE, les appareils dont les emplacements sont dans les zones adjacentes à la zone visée par une application sont privilégiés par rapport aux appareils des zones non adjacentes. Dans le cas où la zone d'un appareil a la même zone mère que la zone visée par une application, l'appareil en question est également privilégié par rapport aux autres appareils des zones sans relations. Enfin, dans le cas où un appareil est positionné dans une sous-zone de la zone visée par une application, cet appareil est considéré comme membre à part entière de la zone mère (inclusion).

Comme il est difficile de représenter les relations entre les zones avec des fonctions d'appartenance tel que pour l'utilisation des ressources, nous avons utilisé le concept de fonction d'appartenance *singleton*. Un singleton consiste en une fonction d'appartenance ayant qu'une valeur possible avec un degré d'appartenance de 100%. De cette façon, les relations entre les zones ont pu être intégrées dans le raisonnement flou du FLORE-A. La définition de l'ensemble flou pour les relations de zone et son utilisation dans le cadre des règles de raisonnement du FLORE est présentée dans l'Extrait 3.5.

3.5.5 Raisonnement sur le profil des utilisateurs

Les sections du FLORE les plus complexes sont celles traitant des données du profil de l'utilisateur et calculant le DCQ utilisateur. Ce DCQ est calculé dans le cas où une requête de déploiement d'application vise un utilisateur particulier d'un espace intelligent. Le DCQ utilisateur est ajouté au DCQ appareil, calculé par les étapes de raisonnement présentées précédemment, pour former le DCQ final, celui utilisé par le FLORE dans l'organisation des logiciels. Le raisonnement sur le profil de l'utilisateur versus le contexte des espaces intelligents et les besoins des applications est basé sur quatre types de données : ses capacités d'interaction, sa position dans l'espace, son orientation et ses préférences en terme d'utilisation de périphériques.

Le FLORE base son évaluation des capacités d'interaction de l'utilisateur avec l'environnement sur les modalités d'interaction de ceux-ci : les sens, les perceptions, les sens moteurs et les capacités cognitives [110]. Plus particulièrement, la version courante de Tyche et de son FLORE utilise les modalités d'interaction suivante :

- Sens - Vision : le champ de vision des utilisateurs versus le champ de projection des dispositifs d'affichage ;
- Perception - Vision : l'acuité visuelle des utilisateurs versus les données présentées sur les dispositifs d'affichage ;
- Sens moteur - Mobilité : la distance des utilisateurs avec les appareils et sa vitesse de déplacement ;
- Sens moteur - Interaction physique : les capacités physiques des utilisateurs versus les caractéristiques matérielles des périphériques : force de la main requise et ouverture de la main.

Ses modalités représentent les façons traditionnelles d'interagir avec les appareils informatiques/électroniques : la vue et le toucher. Bien sûr beaucoup d'autres modalités existent et pourraient être évaluées telle l'ouïe des utilisateurs versus le volume de haut-parleurs, la force d'élocution versus la sensibilité d'un micro ou bien des aspects cognitifs tels que la langue utilisé par les interfaces homme-machine des applications et des appareils. L'intégration des modalités d'interaction dans le processus d'auto-organisation représente une innovation dans le domaine de l'auto-organisation et de l'informatique autonome en général. Les travaux de thèse de Ghorbel [38] et de Kadouche [58] ont utilisé les capacités physiques des utilisateurs dans une perspective de livraison de services vers des appareils mobiles. Toutefois, le contexte d'utilisation est différent puisqu'il ne s'agit pas d'organisation logicielle, mais du déploiement d'application vers un appareil dédié à un utilisateur. Dans ces travaux, l'évaluation des mo-

dalités d'interaction des utilisateurs est réduite à deux seuls capacités physiques : la force des mains et l'ouverture de celles-ci. Le type d'évaluation est catégorique, les utilisateurs ont ou n'ont pas les capacités d'interaction pour interagir avec un appareil.

Capacités physiques des utilisateurs

Dans un premier temps, les capacités des utilisateurs à interagir avec les périphériques des appareils sont vérifiées. Cette vérification est faite en comparant les capacités physiques des utilisateurs et les caractéristiques matérielles nécessaires afin d'utiliser les périphériques en question. Le FLORE se base donc sur les travaux de Kadouche [58] en la matière, en vérifiant la force de la main et l'espace de travail des mains des utilisateurs i.e. le degré d'ouverture des mains. Dans le cas où cette vérification échoue, les appareils reçoivent directement un DCQ d'une valeur de 0 point.

Vision et acuité visuelle

Le FLORE utilise dans un deuxième temps, le champ de vision de l'utilisateur et le champ de projection afin de déterminer si l'utilisateur visé par le déploiement d'une application a des dispositifs d'affichage dans son champ de vision. Ces éventuels appareils seraient alors considérés prioritairement par le FLORE dans son calcul des DCQ. Alors que le champ de projection des dispositifs d'affichage est caractérisé dans la description contextuelle des espaces par une orientation en degré et un angle de vision en degré également, le champ de vision des utilisateurs est défini par l'orientation de l'utilisateur en degré et de deux angles de vision correspondant à chaque oeil de l'utilisateur. Ainsi, un utilisateur peut avoir un champ de vision normal pour l'oeil droit et un champ de vision réduit pour l'oeil gauche ou vice versa. La vérification des champs de vision est relativement simple et utilise des changements de plan orthonormé vers la position de l'utilisateur et son orientation, puis vers les dispositifs d'affichage des appareils. Le pseudo-code 3.1 présente l'algorithme de vérification des champs de vision.

Dans un troisième temps, le FLORE évalue la capacité des utilisateurs d'interagir avec les applications, en calculant des ratios d'acuité visuelle. Ce ratio d'une valeur variant entre $[0, 1]$ est calculé à partir de l'acuité visuelle des utilisateurs, leur position, la taille moyenne des caractères des dialogues des applications à déployer, la position des dispositifs d'affichage, la taille des dispositifs et leur résolution. L'objectif de ce ratio est de quantifier la capacité d'un utilisateur à lire l'information textuelle d'une application à partir de sa position dans l'envi-

Pseudo-code 3.1 Algorithme de vérification du champ de vision des utilisateurs

1 - Calculer la matrice de déplacement du plan orthonormé où u = Utilisateur

$$M^t = \begin{bmatrix} 1 & 0 & u.position.x \\ 0 & 1 & u.position.y \\ 0 & 0 & 1 \end{bmatrix} \quad M^r = \begin{bmatrix} cos(u.orientation) & -sin(u.orientation) & 0 \\ sin(u.orientation) & cos(u.orientation) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$M = M^t * M^r$

2 -Pour chaque dispositif d'affichage recalculer sa position et son orientation et vérifier si l'utilisateur a ces dispositifs dans son champ de vision

$\forall a \in Affichage$

$a.position = a.position * M$

SI $a.position.y \geq 0$ ALORS

$pente_{u,a} = a.position.y / a.position.x$

$\Theta_{u.champ.droit} = (90\,\text{deg} + u.champ.droit) * (\pi / 180\,\text{deg})$

$\Theta_{u.champ.gauche} = (90\,\text{deg} - u.champ.droit) * (\pi / 180\,\text{deg})$

SI $a.position.x < 0$ OU $a.position.y < 0$ ALORS

$\Theta_{u,a} = \pi - |\arctan(pente_{u,a})|$

SINON $\Theta_{u,a} = \arctan(pente_{u,a})$

SI $\Theta_{u,a}\Theta_{u.champ.droit}$ ET $\Theta_{u,a} \leq \Theta_{u.champ.gauche}$ ALORS

le dispositif d'affichage est dans le champ de vision de l'utilisateur, refaire les étapes 1 et 2 mais avec comme centre du plan orthonormé le dispositif d'affichage. Si l'utilisateur est également dans le champ de projection du dispositif alors les deux entités ont leur champ en commun.

ronnement. Évidemment, les utilisateurs auront probablement à se déplacer vers les appareils où les applications seront déployées pour interagir avec celles-ci, mais elle permet de vérifier si dans un premier temps, l'utilisateur est capable de reconnaître les applications et d'avoir un minimum d'information sur celles-ci.

Le calcul du ratio utilise donc l'acuité visuelle moyenne des utilisateurs exprimée avec l'échelle de Snellen [111]. Cette échelle est largement utilisée dans l'optométrie pour quantifier l'acuité visuelle en vérifiant la capacité d'une personne à lire un caractère de cinq arcminutes (hauteur et largeur) à une distance traditionnelle de vingt pieds, la fameuse vue 20/20 ou 6/6 (en mètres). Dans le cas d'une personne avec une acuité d'une échelle de Snellen de 10/20, celle-ci doit être à une distance de dix pieds pour lire les cinq arcs minutes de vingt pieds, alors qu'une personne avec une acuité de 20/20 les reconnait de la distance *normale* de vingt pieds. Les cinq arcs minutes de l'échelle correspondent à la taille d'un arc de cercle d'un angle de cinq minutes pour un cercle d'un rayon 20 pieds, ce qui équivaut à un caractère d'une hauteur et

d'une largeur de 8,9 millimètres. L'optotype des utilisateurs adaptés à la distance des dispositifs d'affichage est comparé avec la taille des caractères des applications sur ces mêmes dispositifs. Plus le ratio est élevé plus près sont les caractères de l'optotype des utilisateurs et plus le ratio tend vers 0 et plus les utilisateurs ont de la difficulté à lire les caractères sur les dispositifs de leur position dans l'espace. Les valeurs du ratio sont bornées de $[0, 1]$ pour éviter que les grands dispositifs d'affichage ayant une petite résolution aient des ratios très élevés, les favorisant peu importe la distance entre ceux-ci et les utilisateurs et que les petits dispositifs ayant une grande résolution reçoivent des ratios négatifs les défavorisant grandement. Ainsi, soit les utilisateurs sont capables de lire les caractères (ratio de 1) et soit les utilisateurs en sont plus ou moins incapables (ratio de 0 et plus). Le Pseudo-code 3.2 présente l'algorithme du calcul du ratio d'acuité visuelle.

Pseudo-code 3.2 Algorithme de calcul du ratio d'acuité visuelle

$\forall d \in Appareils | a \in Applications, u \in Utilisateurs, a.utilisateur = u :$
$distance = \sqrt{(u.position.x - d.position.x)^2 + (u.position.y - d.position.y)^2}$
$primtre = 2 * distance * \pi$
$caractreMoyen = d.taille.hauteur * \frac{a.tailleCaract}{d.resolution.hauteur}$
$optotype = 5 * \frac{primtre}{arcminutes} * \frac{acuitvisuellemoyenne}{u.acuitVisuelle}$
$ratio = 1 - \frac{optotype}{caractremoyen} | ratio = [0, 1]$
$returnratio$

Mobilité des utilisateurs

La quatrième modalité d'interaction évaluée est la mobilité de l'utilisateur. Cette évaluation est faite en grande partie à l'aide de la logique floue et comporte en réalité deux sous-évaluations interreliées. La première sous-évaluation consiste à qualifier la vitesse de déplacement de l'utilisateur. Pour ce faire, nous avons fait une revue de littérature des recherches sur les mesures de la vitesse de déplacement d'utilisateur. Parmi les différents documents revisés, nous avons conservé une enquête générale sur la vitesse de marche des piétons en milieu urbain [112] et un article sur l'évaluation de la mobilité des personnes âgées [113], afin de catégoriser et qualifier les vitesses de déplacement des utilisateurs. Nous avons donc divisé les vitesses de déplacement en quatre types : déplacement impossible, déplacement lent, déplacement normal et enfin déplacement rapide. Chacun de ses types de déplacement a été associé à une fonction d'appartenance de type gaussienne normale avec une vitesse moyenne et un

écart-type. Ces moyennes et écarts-types ont été tirés de notre revue de littérature. L'association des vitesses de déplacement des utilisateurs à ces fonctions d'appartenance sera utilisée par le contrôleur de logique floue du FLORE-C pour son calcul du DCQ utilisateur. Ces fonctions d'appartenance sont présentées à la Figure 3.12.

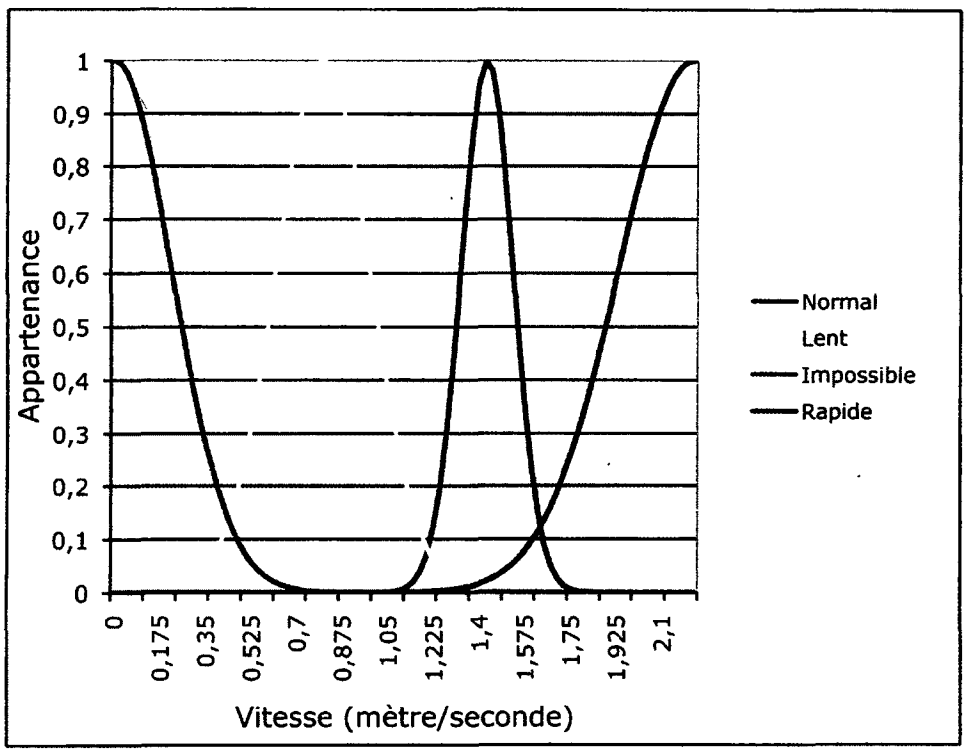

FIGURE 3.12 – Fonctions d'appartenance de la vitesse de déplacement des utilisateurs du FLORE

La seconde sous-évaluation concerne le temps imparti à un utilisateur pour se déplacer jusqu'à un appareil. Cette sous-évaluation calcule donc le temps en secondes pour atteindre un appareil à l'aide de sa vitesse moyenne de déplacement et la distance entre l'utilisateur et l'appareil. Le temps de déplacement est ensuite qualifié dans un ensemble flou où quatre fonctions d'appartenance sont présentes : temps instantané, temps rapide, temps lent et temps très lent. Un temps instantané correspond à un temps de déplacement moyen de 0 secondes et 2 secondes d'écart-type, un temps rapide de 10 secondes et avec 2 secondes d'écart-type, un temps lent de 20 secondes et 2 secondes d'écart-type et finalement un temps très lent de 30 secondes et plus avec un écart-type de 2 secondes vers les valeurs de moins de 30 secondes (les valeurs supérieures à 30 secondes ont un degré d'appartenance de 1 à la fonction temps très lent). À l'instar de la première sous-évaluation de la mobilité des utilisateurs, le degré

d'appartenance à ces fonctions est évalué par le contrôleur de logique floue du FLORE-C.

Préférences envers les périphériques d'interaction

Enfin, la dernière étape d'évaluation du profil des utilisateurs versus le contexte des espaces intelligents et des besoins des applications à déployer, traite des préférences des utilisateurs envers les périphériques d'interaction. Dans la description du profil des utilisateurs, il est possible de définir un ensemble de préférence des utilisateurs envers des types de périphérique d'interaction. Ces préférences ont comme rôle d'avantager dans l'évaluation de l'organisation logicielle, les appareils ayant des périphériques appréciés par certains utilisateurs et inversement de désavantager les appareils munis de périphériques moins appréciés. Ces préférences peuvent être utilisées comme un outil complémentaire aux capacités physiques afin de déterminer les types de périphériques que des utilisateurs potentiels seront à même d'employer. Dans la version actuelle du projet Tyche et lors de l'évaluation de celui-ci (Chapitre 4), cinq types de périphérique ont été utilisés : clavier, souris, trackball, clavier virtuel et curseur sur écran tactile. D'autres types de périphériques peuvent aisément être ajoutés à l'ontologie. Dans la définition des profils d'utilisateur, les préférences envers ces types de périphérique ont été définies à l'aide d'une simple échelle de Likert [114] : l'utilisateur "aime" ce type de périphérique (valeur de 1), l'utilisateur est "neutre" face à ce type (valeur de 0) et l'utilisateur "n'aime pas" ce type (valeur de -1). Les échelles de Likert sont utilisées dans un vaste ensemble de domaines afin de déterminer le degré d'appréciation envers des concepts. Elles sont particulièrement utilisées en interaction homme-machine dans l'évaluation de la convivialité [115]. La moyenne des préférences d'un utilisateur face aux types de périphérique d'un appareil, correspond donc à son appréciation générale quant à l'interaction avec les périphériques de cet appareil. Encore là, l'appréciation générale est ajoutée en entrée au contrôleur de logique floue du FLORE-C et est utilisée dans les règles de raisonnement pour le calcul du DCQ.

Les modalités d'interaction et les préférences des utilisateurs sont donc injectées dans le contrôleur de logique floue qui calcule alors un DCQ utilisateur pour chaque appareil de l'environnement ayant les ressources minimales à l'exécution des applications à déployer. Ces données sont associées aux ensembles de *fuzzification*, traitées par les règles de raisonnement vers un ensemble de *defuzzification*, puis le DCQ utilisateur est calculé à l'aide du centroïde des résultats de l'ensemble de *defuzzification*. Le contrôleur de logique floue du FLORE-C comprend donc cinq ensembles flous, dix-sept fonctions d'appartenance et quarante et une fonctions de raisonnement. Puisque le fichier de description du contrôleur de logique floue contient trop de données, un extrait de celui-ci est présenté dans l'Annexe B.

Ajouter d'autres modalités à l'évaluation de l'organisation logicielle fait partie des futurs objectifs du projet Tyche, notamment en ce qui a trait à l'assistance ponctuelle d'utilisateurs dans les milieux i.e. le déploiement à un temps donné d'une application d'assistance auprès des utilisateurs. L'ajout d'autres modalités représente cependant un travail considérable puisqu'il est nécessaire, comme nous l'avons fait pour les modalités utilisées dans la version courante, de modéliser le mode d'interaction avec l'information contextuelle des milieux et puis d'intégrer le modèle de raisonnement au reste du FLORE. La création d'un modèle complet des modalités d'interaction des utilisateurs dans un cadre d'organisation logicielle ou d'assistance ponctuelle dans les espaces intelligents, représente un travail de recherche complet, qui pourrait s'avérer une suite logique à notre travail de recherche. Toutefois, les objectifs de ce travail de thèse sont plus larges et à un autre niveau et il était impossible d'approfondir l'utilisation des modalités plus loin.

3.6 Auto-optimisation de l'organisation logicielle

Le projet Tyche intègre un module permettant d'améliorer de façon autonome l'organisation logicielle lors de l'arrivée de nouveaux appareils dans les espaces intelligents. Pour ce faire, elle se base sur le DCQ comme métrique de comparaison entre les performances des systèmes. Ainsi, si à un temps donné T une application est déployée sur un appareil avec un DCQ de valeurs N, l'arrivée d'un nouvel appareil à un temps donné T+1 déclenchera un processus de réorganisation des logiciels. Si le nouvel appareil en question offre un DCQ de valeur N+1 à l'application en question, l'appareil sera jugé plus performant et sera candidat au redéploiement de l'application.

Toutefois, une optimisation de l'organisation logicielle à chaque arrivée de nouveaux matériels est inconcevable, car elle pourrait mener à des mouvements plus ou moins inutile d'applications alors que les performances offertes par les nouveaux appareils n'apporteraient pas de gains notables de performance. Le module utilise donc trois règles afin de limiter l'optimisation. Dans un premier temps, afin d'optimiser l'organisation, les applications en question doivent se déclarer candidates à un tel type d'action. Certaines applications peuvent ainsi refuser d'être déplacées vers d'autres appareils pour des raisons telles que l'occupation d'une position stratégique ou pour des raisons de convivialités avec les utilisateurs. Dans un deuxième temps, les performances données par les nouveaux systèmes doivent proposer une augmentation significative de performance, soit une amélioration du DCQ. Par exemple, dans notre évaluation du projet Tyche (Chapitre 4), nous avons fixé l'écart à 20 points avec le DCQ origi-

nal (précédent). Enfin, dans un troisième temps, nous avons fixé des valeurs planchers afin de s'assurer que le nouvel appareil donne des performances décentes à l'environnement. Encore là, nous avons fixé la valeur plancher du nouveau DCQ à 60 points afin de déclencher le processus d'optimisation. Ainsi, une application ne peut être optimisé que si elle a été déployé sur un appareil ayant un donné un DCQ inférieur à 80 points (dû à l'écart de 20 points nécessaire) et vers un nouvel appareil donnant un DCQ supérieur à 60 points. Dans notre implémentation, ces valeurs peuvent toutefois être revisées pour des environnements particuliers. Dans une prochaine version, ces mesures restrictives pourraient être directement déclarées dans le profil des applications et donc propres à chaque application pouvant être optimisée.

Chaque description d'application déployée lors d'organisation logicielle est conservée dans l'ontologie du noeud coordonnateur avec le DCQ de l'appareil hôte. Ainsi, lors de l'arrivée de nouveaux appareils dans les espaces intelligents, le module d'optimisation de l'organisation logicielle vérifie alors si le nouvel appareil apporte une contribution notable aux applications pouvant être optimisées, par le calcul du DCQ de l'appareil en question à partir du FLORE. Dans l'affirmative, l'application est redéployée vers le nouvel appareil et deux notifications sont envoyées aux utilisateurs et aux gestionnaires des espaces intelligents. La première via un message à même le dispositif d'affichage des appareils (si présent, sinon l'information propre à l'optimisation est journalisée sur l'appareil), la seconde via un système de messagerie, consultable à partir des outils de gestion des espaces intelligents. Dans le cas des applications déclarant qu'elles ne sont pas candidates à l'optimisation, une évaluation d'optimisation est quand même faite par le module d'optimisation. Dans le cas où un appareil offrirait un gain de performance correspondant à l'écart et au seuil d'optimisation, une notification aux gestionnaires est envoyée via le système de messagerie, afin de les informer des possibilités de réorganisation.

Évidemment, cette fonctionnalité d'auto-optimisation est assez restreinte et ne concerne que les cas où de nouveaux matériels sont découverts dans les espaces intelligents. Dans l'approche de l'informatique diffuse autonome, l'auto-optimisation comprend également l'amélioration de l'organisation logicielle selon l'utilisation des ressources des appareils (*load balancing*), les périphériques reliés aux appareils, les moyens de communication utilisée, etc. Plusieurs recherches sur la recherche d'équilibre dans l'utilisation des ressources entre appareils ont été faites dans les domaines tels que les systèmes distribués [116] et les système orienté service [117], et pourraient être intégré au projet Tyche. Il serait alors nécessaire d'implémenter des mécanismes fiables et indépendant du matériel pour surveiller l'utilisation des ressources et un mécanisme événementiel permettant d'informer le ou les modules d'optimisation des changements dans l'utilisation des ressources.

3.7 Outils de gestion graphique

Les outils de gestion logicielle du projet Tyche sont les applications graphiques permettant de visualiser les informations propres aux applications et appareils des espaces intelligents et de manipuler le cycle de vie des applications. Ces outils ont pour objectifs de permettre une gestion à distance des logiciels des espaces intelligents d'une façon efficiente. Le projet Tyche possède actuellement deux versions de l'outil de gestion. La première, destinée aux gestionnaires des espaces intelligents et aux développeurs d'application, permet une gestion en détails des applications de plusieurs espaces intelligents à la fois. La seconde version, destinée à des utilisateurs non experts tels que des aidants professionnels, permet de gérer les applications dans les espaces intelligents où les gestionnaires sont physiquement présents à partir de téléphones intelligents, tel que des téléphones Android. Cette dernière version comporte moins d'options de gestion et présente moins d'information technique aux utilisateurs. Elle a été implémentée de façon à être utilisée de façon plus intuitive que la version pour expert, avec une courbe d'apprentissage réduite.

La première version de l'outil de gestion a été développée pour être utilisée à partir d'ordinateur muni d'une machine virtuelle Java SE 1.6 pouvant être démarré sur les systèmes d'exploitation Windows et la majorité des versions de Linux, Unix et Mac OS X. L'outil est implémenté en Java avec les bibliothèques de code Swing et SwingX (interface graphique), JAX-WS (web services) et XStream (sérialisation XML). Les principales fonctionnalités de l'outil permettent de :

- gérer de plusieurs espaces intelligents à distance ;
- consulter l'information concernant les appareils découverts des espaces intelligents : ressources, position, périphériques et zone ;
- parcourir la liste des applications déployées sur les appareils des espaces ainsi que leur état de cycle de vie ;
- déployer et organiser le déploiement d'applications de façon manuelle ou autonome ;
- gérer le cycle de vie des applications déployées ;
- parcourir les journaux d'évènements de gestion et d'organisation des espaces intelligents.

À partir d'une carte des espaces intelligents téléchargée chez les noeuds coordonnateurs des milieux, les gestionnaires peuvent aisément connaître la position approximative des appareils. À partir de cette interface, les utilisateurs peuvent également accéder à la liste des applications déployées et gérer leur cycle de vie. Un élément de l'interface, présentant les applications pouvant être déployées dans les milieux, est accessible via un formulaire où les utilisateurs peuvent

accéder aux descriptions des applications et les modifier aux besoins. À partir de cette interface, les utilisateurs peuvent également choisir des applications à déployer de façon autonome dans les espaces intelligents à l'aide du FLORE ou bien demander au FLORE de retourner une pré-évaluation des DCQ et de laisser le choix de l'organisation logicielle aux utilisateurs. Enfin, une autre interface permet d'accéder aux évènements propres à Tyche ayant eu cours dans l'espace intelligent à la façon d'une boîte de messagerie électronique. Ces messages traitent d'événements d'auto-optimisation ou bien d'erreurs logicielles ayant eu cours. Les Figures 3.13, 3.14 et 3.15 présentent des captures d'écran de cette version de l'outil de gestion logicielle pour espaces intelligents.

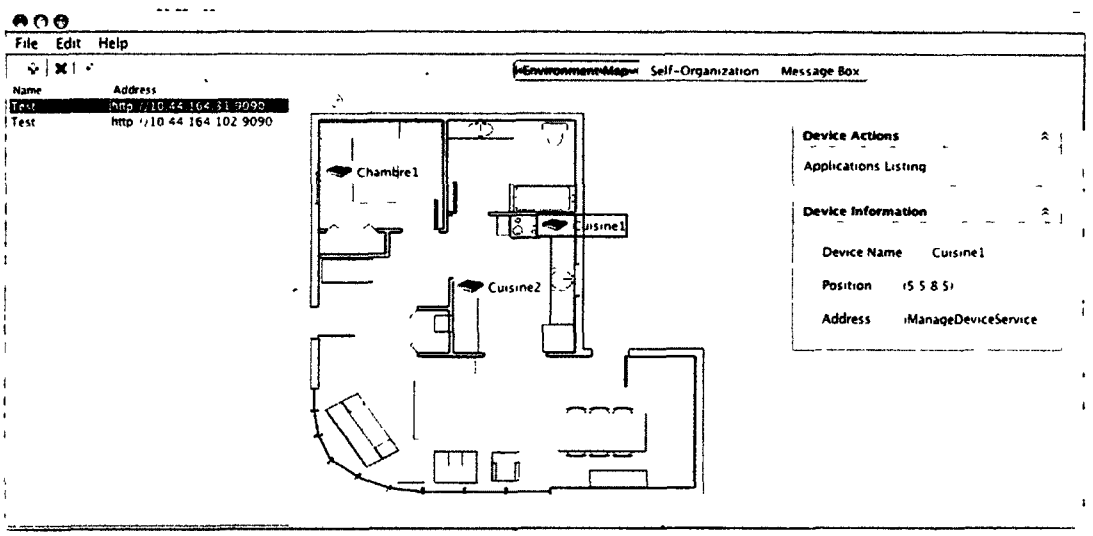

FIGURE 3.13 – Capture d'écran de l'outil de gestion du projet Tyche, présentant la carte des espaces intelligents avec les appareils découverts

Comme il a été mentionné précédemment, la seconde version de l'outil de gestion vise un groupe d'utilisateurs non expert aux domaines de l'informatique et de la gestion de systèmes informatisés. Le choix du support technologique, soit les téléphones intelligents, a été motivé par la série de travaux au sein du laboratoire DOMUS et Handicom proposant les appareils mobiles comme aide technologique auprès d'aidants professionnels et d'utilisateurs dépendants [25]. L'outil de gestion pour appareils mobiles est donc allégé de plusieurs fonctionnalités expertes. Par exemple, il est possible de consulter une carte des espaces avec la disposition des appareils et d'accéder à une description allégée des appareils, mais la fonctionnalité de gestion du cycle de vie est restreinte aux déploiements d'application via le FLORE. Il est donc impossible aux utilisateurs de modifier l'état d'un module OSGi système (serveur web par exemple) contrairement à la version expert. Par contre, les utilisateurs peuvent toujours uti-

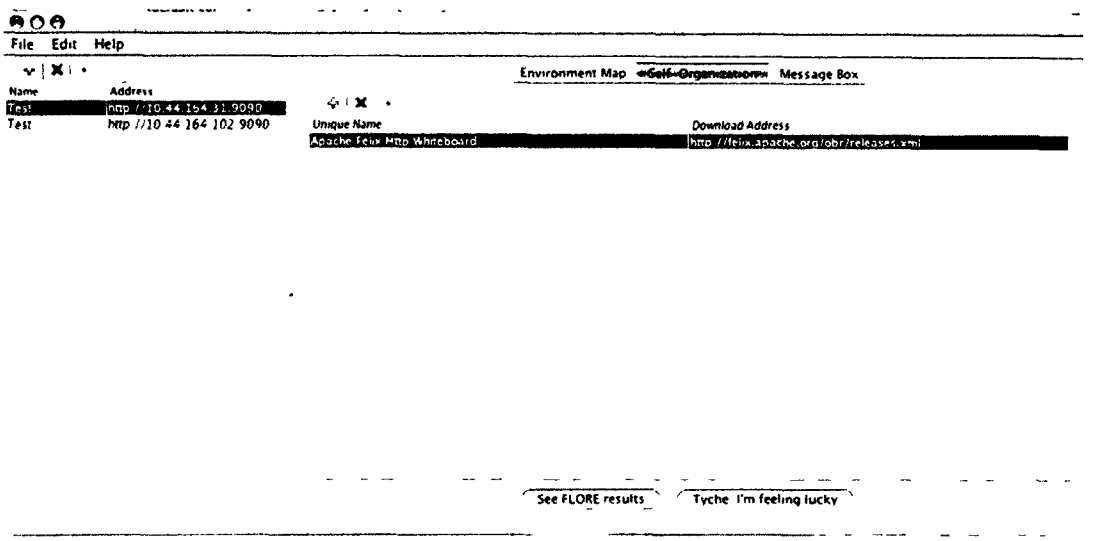

FIGURE 3.14 – Capture d'écran de l'outil de gestion du projet Tyche, présentant la liste des applications pouvant être déployée et les options d'organisation/déploiement pas à pas (*See FLORE results*) ou bien autonome (*Tyche, I'm feeling lucky*)

liser l'option d'organisation manuelle avec l'aide du FLORE ou bien l'organisation autonome à l'aide d'une requête du type "je me sens chanceux" de l'engin de recherche Google. Dans le cas de l'organisation autonome, les applications sont déployées selon l'algorithme présenté dans l'Extrait 3.3.

Toutefois, la fonctionnalité qui diffère le plus de la version experte est l'accès distant aux espaces intelligents. La version mobile a été développée pour être utilisée à même les espaces intelligents lors, par exemple, d'une visite d'un aidant professionnel chez un client. Il est donc impossible d'interagir via l'outil avec plusieurs espaces intelligents distants. L'implémentation même de la version mobile de l'outil de gestion (Figure 3.16) s'inscrit dans un projet de développement d'une application pour appareil mobile, agissant comme portail aux services des espaces intelligents (Projet DomusPortail). L'idée derrière ce projet est la découverte dynamique des divers services utilisateurs accessibles, lors de l'arrivée d'appareils mobiles dans le réseau des espaces intelligents, et de pouvoir interagir avec ces services à l'aide d'interfaces graphiques téléchargées dynamiquement à partir des web services. L'intergiciel du projet Tyche est donc l'un de ces services et l'application portail interagit avec celui-ci via le webservice du module de gestion des environnements du noeud coordonnateur. La découverte des services est faite à la fois à partir d'annuaires centralisés des web services de type "pages blanches" et de mécanismes de découverte de services de type pair-à-pair, tel que *WS-Discovery*.

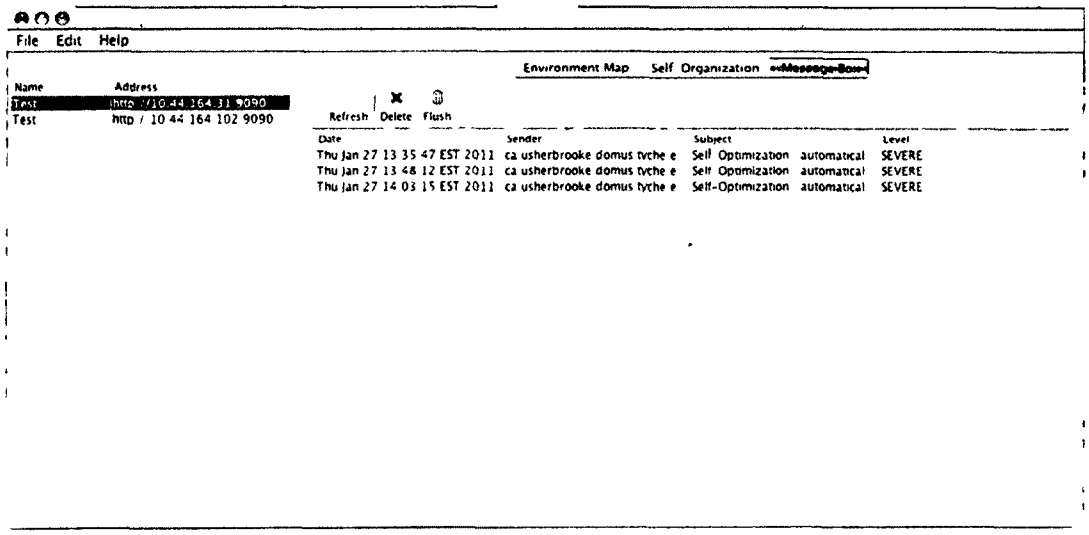

FIGURE 3.15 – Capture d'écran de l'outil de gestion du projet Tyche, présentant le service de messagerie

3.8 Discussion

Nos travaux de recherche sur la gestion autonome des logiciels dans les espaces intelligents ont démontré que l'approche de l'informatique diffuse autonome avec ses quatre fonctionnalités autonomes (configuration, protection, réparation, optimisation) propose plusieurs solutions quant à la simplification de la complexité de la gestion de ces milieux. Toutefois, force est de constater que la conception et l'implémentation d'une solution englobant les quatre fonctionnalités autonomes représentent un travail considérable qui ne pouvait être couvert à même un seul travail de thèse. Puisque les fonctionnalités d'auto-configuration/organisation des logiciels dans les espaces intelligents demeurent la fonctionnalité centrale au sein de l'approche autonome, notre travail de thèse s'est donc concentré sur la mise en place d'une solution d'auto-organisation.

Dès le départ, un ensemble d'objectifs à atteindre dans le cadre du projet Tyche (Tableau 3.1) avait été fixé, objectifs qui ont tous été atteints. Ainsi, dans le cadre de ce travail, des fonctionnalités de gestion des logiciels à distance ont été implémentées à l'aide des services offerts par les noeuds coordonnateurs des environnements et les outils de gestion. Ces outils sont efficaces et les résultats des évaluations de notre travail (Chapitre 4) démontrent les performances du projet Tyche quant aux temps de traitement et à la précision des organisations (Objectif 1). L'implémentation du FLORE et l'utilisation du DCQ comme métrique de performance des appareils ont répondu aux objectifs concernant l'implémentation de mécanismes

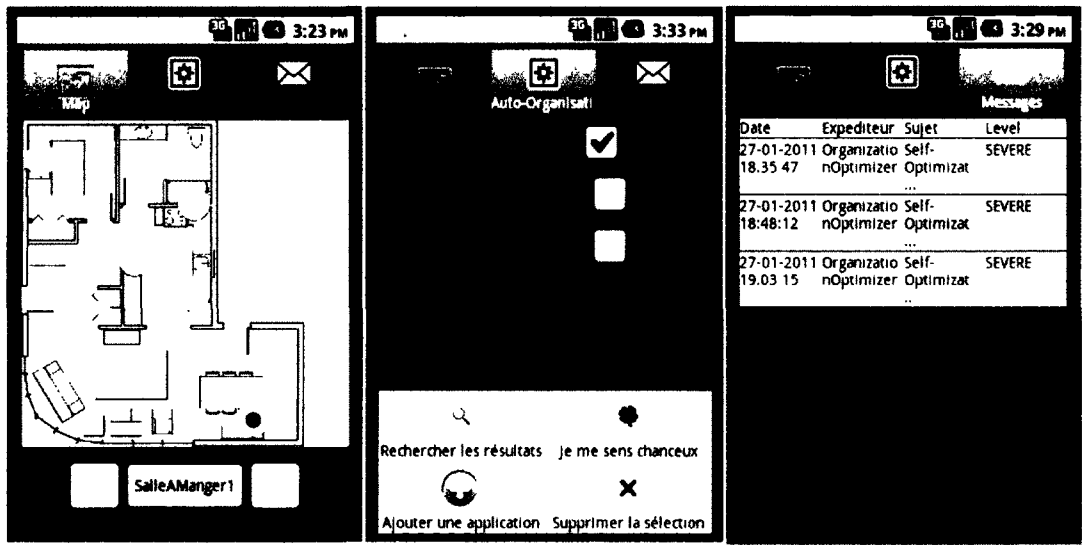

FIGURE 3.16 – Captures d'écran de l'outil de gestion pour appareil mobile Android

d'auto-organisation (Objectif 2). Le DCQ s'est révélé être une métrique polyvalente pouvant être utilisée par les autres fonctionnalités autonomes comme point de comparaison entre les performances des appareils et leurs capacités à recevoir des applications (Objectif 5). Le DCQ a même été utilisé afin de quantifier la qualité, ou du moins la capacité, d'interaction des utilisateurs avec ces mêmes appareils et les applications à déployer. À ce chapitre, la logique floue s'est avérée un outil de raisonnement puissant et flexible, et a compté pour beaucoup dans la création du concept de DCQ.

L'adoption de la plateforme OSGi comme support technologique à l'implémentation de Tyche a été un choix judicieux. Le dynamisme et la réduction du couplage qu'amène cette plateforme ont permis un développement rapide, un découpage modulaire aisé de l'intergiciel et l'implémentation de plusieurs solutions interchangeable (Objectif 4). La division modulaire des applications OSGi et le contrôle du cycle de vie de celles-ci ont d'ailleurs joué pour beaucoup dans le choix de l'architecture du projet Tyche. Également, l'utilisation des standards WebService-* (*eventing, discovery*, etc.) a été un élément majeur dans les choix architecturaux, puis dans l'implémentation de l'intergiciel et des outils de gestion. Les web services ont permis une utilisation multiplateforme de Tyche et une capacité d'extension que d'autres standards ou protocoles n'offrent pas (Juxtapose, RMI, Jini). Toutefois, dans la forme actuelle du projet Tyche, les implémentations utilisées des standards WS-*, soit le serveur web Jetty et l'API Apache CXF, représente une charge de calcul et de mémoire importante pour les appareils. En particulier, CXF qui nécessite au minimum une machine virtuelle Java *Standard*

Edition 1.6 (ou OpenJDK 1.6), restreint les possibilités d'utilisation sur certains matériels et système d'exploitation. Ainsi, il est impossible d'exécuter les deux types de noeud de Tyche sur des appareils munis du système d'exploitation Android. Toutefois, l'implémentation d'une version des noeuds compatible avec la Dalvik, la machine virtuelle d'Android ou avec des versions antérieures à une machine virtuelle Java 1.6, est évidemment possible en utilisant d'autres API tels que kSOAP2, à l'instar de l'outil de gestion logicielle pour appareil mobile.

L'engin de raisonnement sur l'organisation logicielle, le FLORE, du projet Tyche utilise l'information contextuelle selon une approche microscopique et macroscopique du contexte. Les diverses informations contextuelles propres aux utilisateurs, applications, appareils et autres concepts des milieux ont été décrites à l'aide du langage OWL et RDF (Objectif 6). Ces informations sont emmagasinées dans une ontologie propre à chaque espace intelligent, sur lesquels il est possible de faire des requêtes, de l'extraction de données, de l'inférence de concepts et d'instance, etc. L'utilisation du langage OWL/RDF permet une description des instances standardisées selon les concepts définis au sein des laboratoires DOMUS, une indépendance par rapport aux architectures matérielles et une capacité d'extension accrue. De plus, puisque le langage OWL/RDF est en fait une extension du langage XML, celui-ci peut être utilisé dans plusieurs cadres ne nécessitant pas une plateforme sémantique telle que Jena, comme cela a été fait pour les FLORE-A.

À travers les descriptions des utilisateurs et le FLORE, le projet Tyche a intégré le profil de l'utilisateur au processus d'auto-organisation en tenant compte des capacités, des préférences et des contextes des utilisateurs (Objectif 7). Ainsi, un modèle de raisonnement basé sur les modalités d'interaction, les préférences et la logique floue a été implémenté. Ce modèle calcule les DCQ propres aux capacités d'un appareil à héberger des applications face aux profils des utilisateurs et des besoins des applications. La façon dont les profils des utilisateurs ont été intégrés au raisonnement sur l'organisation, l'utilisation intensive de la logique floue pour l'auto-organisation et l'utilisation du DCQ représentent une innovation dans la communauté travaillant sur les notions d'auto-organisation et de gestion autonome des espaces intelligents. Quoique le modèle présenté soit restreint en terme de modalité couverte, les jalons quant à leur utilisation et leur intégration dans l'auto-organisation ont été posés et le modèle pourra être raffiné et étendu dans de futurs travaux.

Enfin, le projet Tyche comprend un module implémentant des fonctionnalités d'optimisation autonome de l'organisation logicielle (Objectif 3). Ces fonctionnalités permettent le redéploiement d'applications lors de l'arrivée d'appareils dans les environnements, qui proposent des performances plus importantes et/ou des interactions supérieures aux appareils hôtes. Ainsi,

l'auto-optimisation de l'organisation logicielle permet une amélioration progressive des performances des applications par leur redéploiement vers des appareils mieux adaptés à leurs besoins en terme de ressources, de localisation dans les milieux ou bien de qualité d'interaction avec les utilisateurs.

Le projet Tyche propose donc une base fonctionnelle à l'implémentation complète des fonctions autonomes de l'informatique diffuse autonome. Le FLORE pourra être utilisé par les mécanismes d'auto-réparation afin de relancer les applications d'appareils défectueux vers d'autres appareils de l'environnement ou bien de réinstaller des versions antérieures d'applications lors de faute logicielle grave. Une auto-optimisation avancée pourrait utiliser le FLORE pour équilibrer l'utilisation des ressources ou du réseau en réorganisant les logiciels parmi les appareils des milieux. Enfin, l'auto-protection pourrait utiliser le FLORE afin de relancer, encore une fois, des applications provenant d'appareils mis en quarantaine sur d'autres appareils des milieux.

Quoique nos objectifs de conception et d'implémentation ont tous été atteints, la version actuelle du projet Tyche est un prototype de recherche et certains aspects devront être corrigés dans les prochaines versions. Des fonctionnalités et mécanismes supplémentaires devront également être implémentés afin de répondre aux divers besoins que nous avons énumérés précédemment dans ce chapitre. Les sections suivantes présentent ces aspect et fonctionnalités additionnelles du projet Tyche, qui devront être implémentés. D'autres fonctionnalités sont présentées dans les perspectives de travaux futurs, à la conclusion de cette thèse.

3.8.1 Sécurité

Dès les débuts du travail de thèse, les dangers quant à la sécurité des systèmes avaient été identifiés comme une problématique. Le projet Tyche permet la gestion complète du cycle de vie des applications OSGi sur les appareils des espaces intelligents. Il est donc possible à un utilisateur malveillant d'utiliser ces fonctionnalités afin de démarrer des programmes illicites sur les appareils des milieux. Le projet Tyche avec ses noeuds coordonnateurs et appareils peut être comparé, et avec raison, à une mécanique de type chevaux de Troie. Le déploiement, par exemple, d'une application sur le noeud coordonnateur accédant aux services de l'ontologie, pourrait conduire à donner un accès à des utilisateurs malveillants aux données contextuelles des environnements, dont les profils utilisateurs, les positions des utilisateurs, les types d'appareil du milieu, les informations capteurs, etc. Les accès aux fonctionnalités de Tyche doivent donc être contrôlés.

L'implémentation de mécanismes sécuritaires a été délaissée au profit de l'amélioration du FLORE et à l'implémentation de l'auto-optimisation. Toutefois, des travaux initiaux ont été entrepris afin d'améliorer la sécurité de l'intergiciel. Ainsi, un système d'authentification par clés publiques a été implémenté afin de contrôler l'accès aux web services du noeud coordonnateur et chiffrer les données échangées entre ceux-ci et les outils de gestion. Toutefois, cette implémentation était couplée à l'utilisation de l'API Apache Axis, une API de gestion de web services, qui a été un peu plus tard remplacée par l'API Apache CXF. Le système mis en place avec une autre plateforme à web services, Apache Axis[12], n'était pas compatible avec CXF et n'a pas été ré-implémenté pour celui-ci. L'authentification et le chiffrement des données devraient également être étendus aux communications entre les noeuds coordonnateurs et appareils.

Dans le cadre des travaux initiaux de cette thèse, une mécanique de contrôle aux fonctionnalités de gestion logicielle avait été proposée (rapport d'examen pré-doctoral), s'intégrant à la description des utilisateurs des espaces intelligents. Cette mécanique prévoyait des rôles aux différents utilisateurs, restreignant par exemple, les opérations de gestion des utilisateurs non experts. Ces mêmes rôles pourraient être utilisés afin de contrôler l'accès à l'ontologie.

Pour ce qui est du contrôle des applications déployées sur les appareils, la plateforme OSGi intègre déjà une mécanique d'authentification des applications par certificats et clés de hachage [42]. Cette mécanique permet de vérifier l'origine et l'intégrité des modules OSGi, puis de contrôler le déploiement des applications selon les organisations ayant produit les modules OSGi. Par exemple, il serait alors possibles de restreindre le déploiement aux seuls modules signés par le laboratoire DOMUS, l'Université de Sherbrooke, le laboratoire Handicom ou tout autre partenaire. OSGi intègre également un contrôle d'accès aux services OSGi à l'intérieur même d'une plateforme. Fonctionnalité qui permettrait d'éviter qu'un module déployé sur le noeud coordonnateur n'utilise, par exemple, les services du module de gestion de l'ontologie sans permission d'accès.

Enfin, les fonctionnalités d'auto-protection de l'informatique diffuse autonome permettront également d'améliorer la sécurité du projet Tyche en identifiant les failles ou risques au niveau de la sécurité et par la mise en place autonome de mesures de contrôle de la sécurité. Une autre option à l'amélioration proactive de la sécurité des applications pour espaces intelligents passe par l'insertion dynamique de patrons de conception (*design pattern*) de sécurité. Des recherches sur le sujet sont d'ailleurs conduites au sein du laboratoire DOMUS par Pierre Busnel [80]. Ces patrons de conception permettraient la génération dynamique de code gérant

[12]http ://axis.apache.org/axis/

des aspects de la sécurité et l'insertion de celui-ci dans les portions critiques des applications diffuses. Par exemple, l'authentification à un web service pourrait être dynamiquement ajoutée à des applications à la création de ces services, sans que les développeurs et les gestionnaires n'aient à coder ou gérer cette partie. L'ajout automatique de contrôle de la sécurité dans les applications des espaces intelligents permettrait d'améliorer de façon significative la protection des données contextuelles et l'intégrité des systèmes.

3.8.2 Le temps et la sensibilité au contexte

Quoique des notions de propriétés temporelles soient définies dans la description contextuelle [1], celles-ci ne sont pas utilisées par le FLORE. À partir de ces propriétés, il serait possible de définir des ordres temporels de déploiement, par exemple de créer une liste temporel d'applications à déployer à la suite des autre. Il serait également possible à l'aide d'un temporiseur, intégré à même l'ontologie, de définir des requêtes d'organisation logicielle liées à des contextes temporels particuliers.

L'ajout de propriétés temporelles à l'organisation logicielle du projet Tyche représenterait un ajout majeur, mais également un outil intéressant pour les développeurs d'application pour espaces intelligents. Ainsi, une application d'assistance à la cuisine pourrait recevoir comme propriété de déploiement d'être déployée lors de la reconnaissance de cette activité. Une application d'assistance à l'hygiène pourrait être lancée sur un appareil de la salle de bain d'un appartement intelligent après le réveil des utilisateurs. Les propriétés temporelles définies dans la description sémantique peuvent donc être reliées à des relations entre des concepts, tel que avant, pendant ou après une activité. Évidemment, les propriétés temporelles peuvent également traiter des plages particulières de temps. Ainsi, une requête de mise à jour d'applications pourrait être temporisé pour être mise en place à un moment particulier, tel que la nuit. Des propriétés temporelles reliées à des plages de temps pourraient être utilisée en conjonction avec les agendas électroniques des utilisateurs [118], déployant des applications nécessaires à l'accomplissement de certaines activités. Par exemple, une application de vidéo-conférence pourrait être démarrée et/ou installée sur un appareil des milieux lorsqu'un utilisateur a une réunion téléphonique inscrite à son agenda.

Les propriétés et concepts temporels sont déjà définis dans la description sémantique des espaces intelligents, mais les règles permettant de raisonner sur ces propriétés restent à être implémentée. La première piste de solution semble être l'utilisation de règles de description logique à même la plateforme sémantique Jena et l'intégration d'un engin de raisonnement tel

que Pellet[13]. Certains travaux [119] [120] utilisent également la logique floue comme mode de raisonnement sur les propriétés temporelles, puisque celles-ci comporte des éléments qualitatifs flous (avant, pendant, après) pouvant être aisément gérés par des fonctions d'appartenance et des groupes de règles. Comme le projet Tyche intègre déjà la logique floue et une part de description logique dans son raisonnement sur le contexte, l'intégration mixte de ces deux modes de raisonnement semble une solution prometteuse.

3.8.3 Modalités d'interaction

Comme nous l'avons introduit précédemment dans ce chapitre, le raisonnement d'organisation logicielle entourant le contexte des utilisateurs est basé sur les modalités d'interaction. Le FLORE utilise à ce jour quatre modalités alors que les possibilités de modalités dans un contexte ubiquiste sont innombrables. Sans entrer dans des cas spécifiques de modalités, le modèle de raisonnement du FLORE sur le profil d'utilisateur devrait être étendu pour inclure des modalités reliées à l'ouïe, mais surtout aux aspects cognitifs des utilisateurs. Puisque les recherches au sein des laboratoires DOMUS et Handicom visent entre autres l'assistance auprès de personnes atteintes de déficience cognitive, la modélisation de modalités cognitives serait un atout considérable au projet Tyche. Une modalité simple à implémenter serait le traitement des langues utilisées entre les appareils (système d'exploitation), les utilisateurs et les applications.

D'un autre côté, une mesure de quantification ou de qualification des capacités ou fonctionnalités mnémoniques et/ou cognitives générales des utilisateurs versus les caractéristiques des appareils et applications permettrait également d'ajouter cet aspect à l'organisation logicielle. De telles métriques existent en neuropsychologie sous la forme de résultats à des tests de fonctionnalités cognitives tels que le *Mini-Mental State Examination* [121] (MMSE) ou bien le *Disability Assessment for Dementia* (DAD) [122]. Toutefois, il resterait à quantifier ou qualifier les capacités nécessaires à l'utilisation des appareils et/ou des applications. Un tel travail nécessite à priori des compétences et des connaissances approfondies en neuropsychologie et en convivialité/ergonomie.

Afin de permettre de nouvelles modifications telles que l'ajout de nouvelles modalités d'interaction, l'utilisation de la structure de Rete [123] avec le système de gestion de règles *Drools* [124] est envisagée. Cette structure faciliterait l'ajout de nouvelles étapes et règles de raisonnement à notre algorithme et améliorerait la structure logique de l'algorithme d'auto-organisation.

[13]Site web de Pellet : http ://clarkparsia.com/pellet/

Enfin, l'implémentation même du projet Tyche est basé sur l'hypothèse d'un déploiement dans des milieux intelligents fermés et contrôlés. Dans une perspective d'une utilisation plus large telle que dans des espaces intelligents ouverts et auprès d'utilisateurs en mobilité, d'autres modalités d'interaction devront être modélisées et intégrées. L'utilisation de la mécanique d'auto-organisation logicielle comme moyen d'assistance ponctuelle auprès des utilisateurs amène également un lot de contraintes quant aux modalités utilisées. La perspective de l'assistance ponctuelle est discutée dans la conclusion de cette thèse.

3.8.4 Taxonomie et ressources dynamiques

La description des appareils utilisée dans le cadre du projet Tyche est restreinte à un nombre limité d'attributs : capacités des processeurs, utilisation des processeurs, capacités des espaces disques, position des appareils, périphériques reliés, etc. Dans un contexte de déploiement réel de l'intergiciel de gestion et non celui de prototype, la quantité d'information traitée actuellement est faible en regard de la quantité d'informations possibles : nombre de coeur des processeurs, vitesse d'accès à la mémoire vive et à l'espace disque, caractéristiques des interfaces réseaux, etc. L'utilisation des résultats aux bancs d'essai pour quantifier les performances des processeurs est une bonne piste de solution, car elle abstrait les différences matérielles des processeurs (nombre de coeurs, nombre de fils d'exécution, quantité de mémoires caches, etc.). La taxonomie des appareils reste un sujet de recherche entier et il n'existe pas à ce jour de solution complète pour le contexte des espaces intelligents.

D'un autre côté, l'utilisation des ressources des appareils dans le projet Tyche n'est qu'une approximation de la consommation réelle des appareils. Ainsi, lors de chaque déploiement, les ressources nécessaires à l'exécution d'une application sont retirées des capacités totales des appareils. Une telle mécanique est bien loin de la consommation réelle des ressources qui fluctue constamment sur les appareils, et ce, de façon hétérogène selon le matériel et les systèmes d'exploitation en présence. Dans une perspective d'auto-optimisation et d'équilibre dans l'utilisation des ressources des appareils d'un espace intelligent (*load balancing*), une mécanique précise faisant la surveillance en temps réel de l'utilisation des ressources serait nécessaire. L'API *Hyperic System Information Gatherer* (SIGAR)[14] est probablement la solution la plus appropriée, car elle permet à partir du langage Java d'accéder aux informations systèmes d'appareils ayant les systèmes d'exploitation Windows, Linux, Unix, FreeBSD et Mac OS X. Son intégration aux fonctionnalités de Tyche permettrait une lecture en temps réel de l'utilisation

[14]Site web de Sigar : http ://www.hyperic.com/products/sigar

des processeurs, mémoires RAM, espaces disques, interfaces réseaux, etc. Cette mécanique sera ajoutée aux prochaines versions de l'intergiciel de gestion puisqu'elle est nécessaire à l'implémentation de fonctions avancées d'auto-optimisation et au déploiement du projet Tyche dans des environnements réels tels que les résidences alternatives du Centre de Réadaptation de l'Estrie (CRE). Les résidences alternatives du CRE est un projet d'habitats intelligents pour personnes atteintes de déficits cognitifs. Ces habitats permettront à ces personnes de vivre de façon semi-autonome en recevant des assistances numériques dans leurs activités de la vie quotidienne.

3.8.5 Sous-espace intelligent et topologie

L'architecture générale de l'intergiciel est basée sur une hiérarchie à deux niveaux entre les composantes : les noeuds coordonnateurs et les noeuds appareils. Il est prévu que chaque espace intelligent possède minimalement un noeud coordonnateur, avec la possibilité d'intégrer d'autres noeuds coordonnateurs secondaires afin de parer aux fautes logicielles et matérielles. Toutefois, cette hiérarchie pourrait être étendue afin d'inclure des sous-espaces ayant à leur tour des noeuds coordonnateurs, une ontologie et un macro-contexte propre à ces sous-espaces, et bien sûr des noeuds appareils (Figure 3.17). Cette extension va d'ailleurs dans le même sens que notre vision macroscopique et microscopique du contexte, où certaines zones ou divisions logiques des espaces intelligents méritent une description macro-contextuelle indépendante du macro-contexte global d'un environnement. Si cette division peut être difficile à faire dans un habitat tel qu'un appartement ou une maison, la division en sous-espaces doit être considéré dans des espaces intelligents plus large tels que des centres commerciaux, des hôpitaux, etc. Dans le cadre de ce travail, il reste toutefois à déterminer, concevoir et implémenter les mécanismes qui permettront l'échange d'information et la constitution des macro-contextes à partir de leurs "sous-macro-contextes" ainsi que des micro-contextes des entités des espaces. Il sera ainsi nécessaire d'abstraire et de synchroniser certaines données entre les ontologies des sous-espaces. Ces travaux s'inscrivent entre autres, dans nos travaux futurs quant à la modélisation de la sensibilité au contexte macroscopique et microscopique.

Enfin, la description des espaces intelligents possède les propriétés et concepts pour définir ces sous-espaces et leurs séparations logiques et physiques : murs, fenêtre, porte, etc. Toutefois, le raisonnement du FLORE ne tient pas compte de ces séparations physiques dans le cadre de ses calculs de distances et de champs de vision. Par exemple, lors du calcul de la distance entre un utilisateur et un appareil pour la modalité de la mobilité, la distance utilisée correspond à

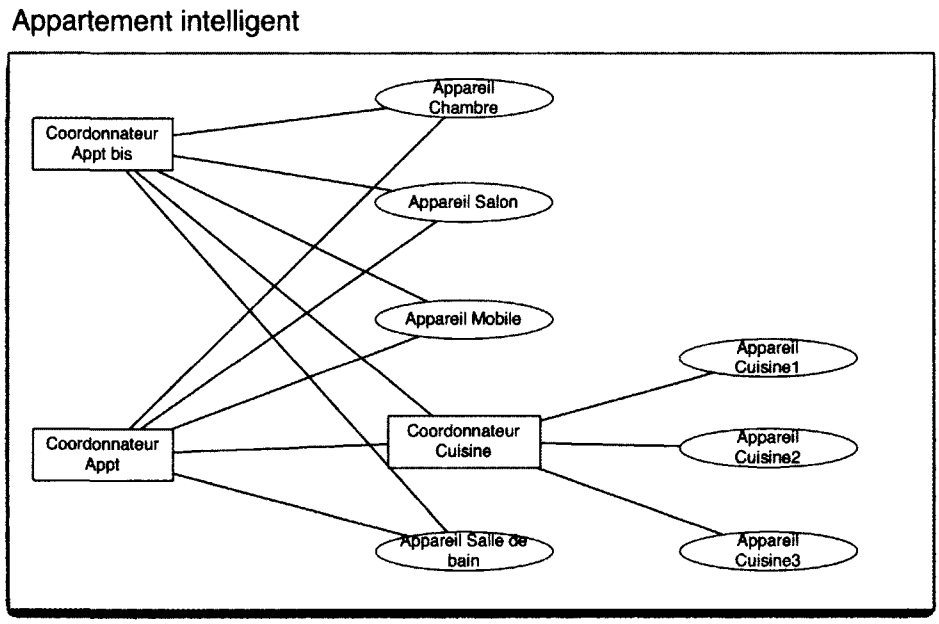

FIGURE 3.17 – Architecture multiniveaux du projet Tyche

la distance directe (ligne droite) entre les deux positions et ne tient pas compte des obstacles physiques possibles tels que des tables, chaises, murs, etc. Le calcul des champs de vision ne tient pas compte non plus des obstacles pouvant obstruer le champ de vision des utilisateurs et le champ de projection des dispositifs d'affichage. Pour que le projet Tyche passe du statut de prototype à une application déployable dans un contexte réel, une implémentation poussée de la topologie des espaces intelligents est nécessaire. Une fois le recensement des structures physiques des environnements et leur implémentation dans la description contextuelle terminée, plusieurs algorithmes de calcul de chemin et de champs de vision existant, notamment tirée de l'univers des jeux vidéo, pourront être intégrées au projet Tyche et amélioreront l'utilisation réelle de l'intergiciel de gestion autonome des espaces intelligents.

Chapitre 4

Évaluations, Tests et Résultats

Ce chapitre présente l'évaluation et les tests effectués sur le projet Tyche, l'intergiciel de gestion autonome des logiciels des espaces intelligents présenté au chapitre précédent. Ces évaluations et tests ont permis entre autres de vérifier la performance et la justesse des mécanismes proposés, d'identifier et de régler des problèmes et les erreurs d'implémentation, et d'identifier les forces et les lacunes de Tyche.

Le chapitre commence donc avec la description des objectifs visés par l'évaluation et les tests. Par la suite, les tests de performance effectués sur l'intergiciel de gestion autonome sont introduits et les résultats recueillis sont présentés. Ces tests ont principalement été effectués sur le *Fuzzy Logic Organization Reasoning Engine* (FLORE). Enfin, les six scénarios d'évaluation qui ont été mise en place pour évaluer le Projet Tyche sont présentés. Pour chaque scénario, les résultats recueillis sont présentés, accompagnés d'une discussion sur le déroulement du scénario et sur la signification des résultats.

4.1 Objectifs

L'objectif de la phase de tests et d'évaluation du travail de thèse consistait principalement à valider l'approche et la solution proposée. Ainsi, la phase de tests avait pour but de vérifier la performance de l'intergiciel en analysant les ressources utilisées, tel qu'en chronométrant des tâches à effectuer. Puisque l'un des objectifs du travail de recherche consistait à réduire le temps induit par les tâches de gestion logicielle, il est donc primordial que l'intergiciel de gestion accomplisse ces mêmes tâches dans un temps égal ou inférieur à celui d'un utilisateur/-gestionnaire.

Toutefois, il est difficile de quantifier facilement le temps nécessaire pris par un utilisateur

pour gérer les logiciels d'un espace intelligent en utilisant des moyens traditionnels comme le déploiement sur place (local) à l'aide des outils du système d'exploitation ou bien la gestion à distance en utilisant, par exemple, un *Secure Shell* (SSH). Une série d'expérimentation avec des gestionnaires potentiels aurait été nécessaire afin de calculer le temps moyen. Pour contourner cette problématique, l'objectif de performance a été bonifié, en visant comme **premier objectif** qu'une tâche de déploiement d'une application dans un espace intelligent nécessite un temps de traitement inférieur à la limite de convivialité proposé par Stuart K. Card [125] dans son évaluation du temps de réponse, soit entre une et deux secondes. Ainsi, la tâche apparaît alors à l'utilisateur comme instantanée, améliorant significativement le temps imparti à la gestion logicielle d'espaces intelligents. Évidemment, cet objectif de temps de traitement tient compte d'un environnement ayant un nombre moyen d'appareils (entre 5 et 15 appareils), pour une requête comprenant moins de dix applications à déployer à la fois et ne comprend pas le téléchargement en soi de l'application, seulement le temps intrinsèque à l'intergiciel (temps de raisonnement, requêtes sur l'ontologie, appels sur les web services, etc.) pour déterminer l'organisation logicielle, puis démarrer le déploiement des applications.

D'un autre côté, la phase de tests et d'évaluations a également été initié afin de démontrer que la solution proposée donne des résultats fiables répondant aux contextes des espaces intelligents, aux besoins d'une application donnée et aux capacités des utilisateurs. L'intergiciel de gestion autonome a donc été évalué en le mettant en scène dans six scénarios, mettant en lumière chacune des fonctionnalités du projet Tyche, présenté dans le chapitre précédent. Le **second objectif** d'évaluation visait à déterminer que les fonctionnalités d'auto-organisation proposent les solutions d'organisation logicielle optimale[1] pour chaque scénario de base, i.e. qu'il donne le *Device Capability Quotient* (DCQ) le plus élevé à l'appareil le mieux pourvu et situé dans le meilleur contexte et ce à tous les coups. Le **troisième objectif** visait à ce que le projet Tyche donne des DCQ correspondant globalement à l'évaluation mentale qu'un utilisateur se ferait du système. Toutefois, puisque l'évaluation de l'organisation logicielle par le FLORE reste complexe par la quantité de facteurs entrant en jeux, il était difficile de valider celui-ci. C'est donc dans de telles conditions que des scénarios d'évaluation prennent toute leur importance, puisque que pour chaque scénario, nous avons fait plusieurs itérations, modifiant à chaque fois quelques variables du scénario, puis en proposant des hypothèses quant aux résultats attendus. La différence entre nos hypothèses i.e. les résultats qu'obtiendraient un utilisateur expert après une analyse du contexte du milieu et les résultats de l'intergiciel correspond alors au degré de réussite de notre troisième objectif.

[1] en tenant compte de la discussion présentée à la Section 3.5.2

Le **quatrième objectif** visait à s'assurer que le projet Tyche offre une stabilité dans la qualité de ses résultats. Il doit donc rencontrer des temps de traitement stable, variant peu d'une fois à l'autre et donner les mêmes résultats, ou du moins similaire, pour des contextes semblables. Bref, que les résultats soient reproductibles aisément. Tyche doit également donner des résultats variant peu lorsqu'il y a des variations infimes entre les caractéristiques de deux contextes, par exemple que le DCQ varie que de quelques dixièmes alors que la position de l'utilisateur s'est déplacée de dix centimètres ou que la saturation du processeur d'un appareil a augmenté d'un demi-pourcent.

Lors de la création des règles de raisonnement flou et de la conception de l'algorithme d'organisation logicielle autonome, les appareils se situant dans des contextes particulièrement favorables ont été intentionnellement privilégiés, en leur donnant des DCQ élevés (plus de 80) alors que les appareils dans des situations moyennes reçoivent des valeurs entre 40 et 60 avec un écart-type faible. Enfin, les appareils dans des situations défavorables ont été volontairement défavorisés en leur donnant un DCQ faible, en deçà de 20. L'idée étant d'encourager des écarts importants entre les DCQ attribués entre deux appareils ayant une différence marquée de performance. Le **cinquième objectif** visait donc de s'assurer que des différences marquées existent lors de l'attribution de DCQ dans une situation de grande variation dans le contexte des appareils et du milieu. Ainsi, nous visions que les DCQ attribués par le FLORE soient inscrits dans trois classes d'appareils distinctes : les appareils en situation défavorable, les appareils moyens et les appareils en situation favorable. Le Tableau 4.1 résume les objectifs de nos tests et évaluations.

TABLEAU 4.1 – Résumé des objectifs des tests et évaluations

Objectifs	Description	Type	Notes
1	Performance	Test	Tâches d'organisation en moins de 2 secondes
2	Fiabilité	Évaluation	Donner la situation optimale pour un contexte donné
3	Représentation	Évaluation	Donner des résultats semblables à l'évaluation d'un utilisateur
4	Stabiltié	Test et Évaluation	Temps de traitement stable, résultats reproductibles et peu de variation quand il y a peu de différence dans le contexte
5	Répartition	Test et Évaluation	Grande variation dans les résultats lors de grande variation dans le contexte et répartition suivant trois classes

4.2 Tests de performance

L'un de nos objectifs d'évaluation était donc de vérifier la performance du Projet Tyche, en s'assurant que celui-ci offre un temps de traitement de moins de deux secondes pour le déploiement d'une application dans un espace intelligent donné. Toutefois, il était également intéressant de connaître le temps de traitement propre à notre système, dégagé des contraintes des mécanismes tiers tels que les appels vers les webservices de l'intergiciel, la latence du réseau, etc. L'engin de raisonnement d'auto-organisation, le FLORE, a donc été isolé du reste du travail et plusieurs appareils d'un espace intelligent ont été simulés sur le même appareil de test. Cette simulation a également permis de tester le FLORE dans des contextes particuliers, avec un nombre important d'appareils tel que 250 ou 500 appareils, qui auraient été impossibles, du moins très difficiles, à mettre en place dans des conditions réelles. Dans ce test, les application déployées étaient fictives et ne nécessitait pas le téléchargement de codes à partir d'un répertoire d'applications.

Ainsi, le FLORE a été utilisé pour évaluer l'organisation d'un groupe d'applications à déployer et le temps imparti pour évaluer l'organisation a été chronométré. Ces tests ont été effectués sur un ordinateur muni d'un processeur Intel Core 2 Duo E6600 cadencé à 2,4 GHz, d'un disque dur ayant une révolution de 5400 RPM, de 2 Gig de mémoire RAM cadencés à 667 MHz et d'un système d'exploitation Mac Os 10.5, et utilisant la machine virtuelle Java JDK 1.6. Le Tableau 4.2 et la Figure 4.1 présente les résultats de ce test.

TABLEAU 4.2 – Temps d'exécution du FLORE lors des tests de performance

Nombre d'appareils	Temps d'exécution moyens (sec.)		
	2 applications	5 applications	10 applications
2	0,033	0,072	0,131
10	0,139	0,262	0,476
25	0,287	0,592	0,977
50	0,404	0,940	1,829
100	0,780	1,762	3,523
250	1,770	4,343	8,470
500	3,426	8,134	16,476

À première vue, les résultats de ce test de performance montrent que le FLORE utilise un temps de traitement exponentiel, ce que l'analyse de l'algorithme de l'engin de raisonnement démontrait déjà. Les résultats du test démontrent également que le FLORE propose des temps de traitement inférieur à la limite visée, et ce, pour un groupe de 10 applications et moins et

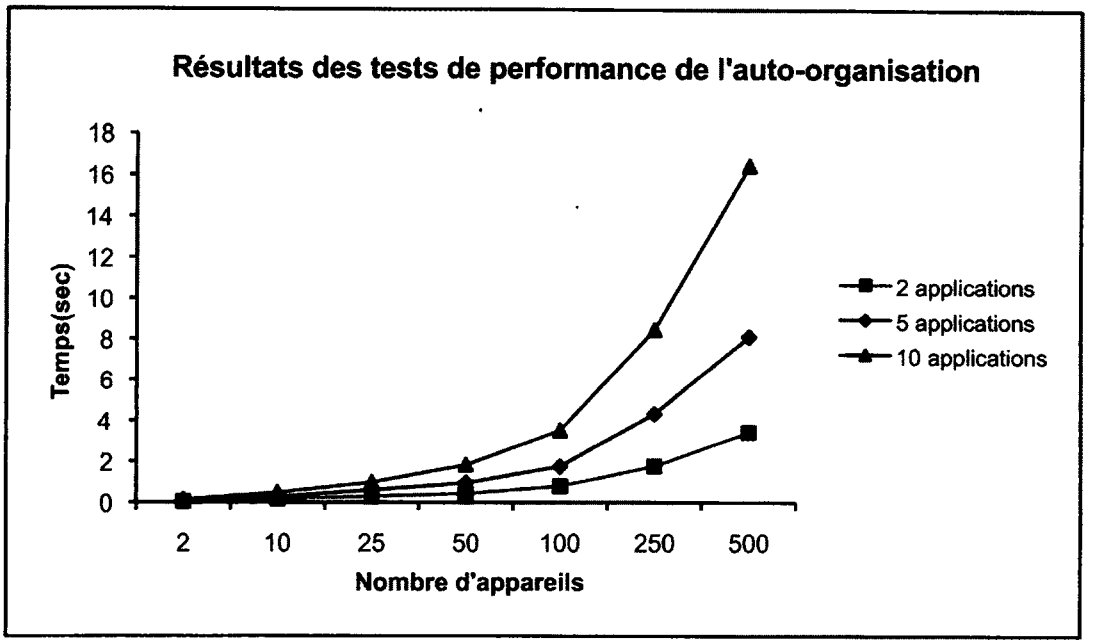

FIGURE 4.1 – Diagramme présentant les temps d'exécution de l'auto-organisation lors des tests de performance

pour un ensemble de 50 appareils et moins. En regard des projets de construction d'espaces intelligents actuels (résidences alternatives du Centre de réadaptation de l'Estrie) et ceux de la littérature [126] [127], rares sont les milieux ayant plus de 25 appareils susceptibles d'être sujets à un déploiement d'application. Ainsi, ces résultats sont satisfaisants et répondent aux attentes fixées. Pour ce qui est des temps de traitement dans des situations réelles, incluant les appels de services, la latence réseau, etc., les performances du Projet Tyche lors de la mise en place des scénarios d'évaluation ont été chronométrées. Ces résultats sont présentés un peu plus loin dans ce chapitre à la section traitant des scénarios.

Pour ce qui est de comparer les résultats avec les travaux de l'état de l'art, peu d'entre eux offrent des données permettant d'être comparées à ceux obtenus. Par exemple, Ranganathan dans [57] écrivait que sa solution lors de sa phase d'adaptation, une action comparable à notre organisation logicielle, prennait un temps moyen de traitement de 0,93 seconde, évaluation faite à l'époque sur un Pentium M 1.7 GHz. En ne tenant pas compte de la différence entre les systèmes de tests, ses temps de traitement sont plus long que ceux de notre système, puisque son test se fait avec un ensemble de données peu élevé comparativement aux données analysées par le FLORE. Son ensemble de données se comparant, dans notre contexte, à un déploiement de 2 applications parmi 10 appareils.

Les évaluations faites par Trumler [72] sur son système de configuration autonome ne

donne pas de temps de traitement pour son système, mais évalue plutôt les performances de ses solutions en terme de messages échangés entre les noeuds participants. Le nombre d'appels de web services lors d'un déploiement d'applications pour le projet Tyche correspond à la formule suivante :

$$Messages = (a \times d) + a + 1$$

où d est le nombre d'appareils et a le nombre d'applications. Cette formule inclue les messages échangés lors des calculs des DCQ, lors des requêtes finales de déploiement vers les appareils et lors de la requête initiale d'organisation logicielle. Pour cinq applications à déployer et dix appareils dans un milieu, le nombre d'appels vers les web services de Tyche est de 56 appels, alors que la solution de Trumler dans des conditions similaires est de 35 messages. L'écart dans le nombre de messages entre nos travaux est dû aux différences entre les architecture et les fonctionnalités de nos systèmes. Alors que le projet Tyche envoie un message pour chaque calcul de DCQ appareil ($a \times d$), Trumler envoi un message aux appareils participants encapsulant toutes les requêtes de déploiement d'applications envoyées vers le système. Dans notre cas, il est impossible d'utiliser la même stratégie que Trumler, puisque le calcul des DCQ nécessite l'utilisation subséquente des ressources des applications déployées aux préalables (Chapitre 3, section 3.5.2).

Dans un deuxième temps, le FLORE a été testé avec un ensemble d'appareils ayant des utilisations de leurs ressources variées. Le but de ce test était de vérifier la répartition des DCQ selon la saturation des ressources des appareils. Ce test permet d'évaluer en partie notre cinquième objectif d'évaluation, soit une répartition des DCQ selon trois groupements. Dix appareils ont été simulés avec des charges d'utilisations de leurs ressources variant de 0 à 100%, donc une variation de 10% d'un appareil à l'autre. La variation a été fixée au même taux pour les trois types de ressources (CPU, RAM et espace disque) afin de représenter des résultats homogènes et représentatifs. La Figure 4.2 présente les résultats de ce test.

Comme espéré, le FLORE retourne une distribution variable des DCQ où ceux-ci sont répartis en trois groupes distincts : les DCQ élevés pour les saturations faibles (0 à 25%), les DCQ moyens pour les saturations moyennes (30 à 80) et les DCQ faibles pour les saturations élevées (85 à 100%). En y regardant de plus près, il serait préférable que les DCQ du groupe moyen soient un peu plus élevés, i.e. qu'ils soient compris dans l'intervalle 40 à 60 plutôt que 40 à 50, tout en ayant une plus grande variabilité entre eux afin de présenter une meilleure distribution entre les quotients des appareils de ce groupe. Évidemment, ces résultats sont partiels puisque le FLORE, dans ce test, n'utilise qu'une partie restreinte de son algorithme, celle reliée à la saturation des appareils. De plus, rares seront les dispositions dans des contextes

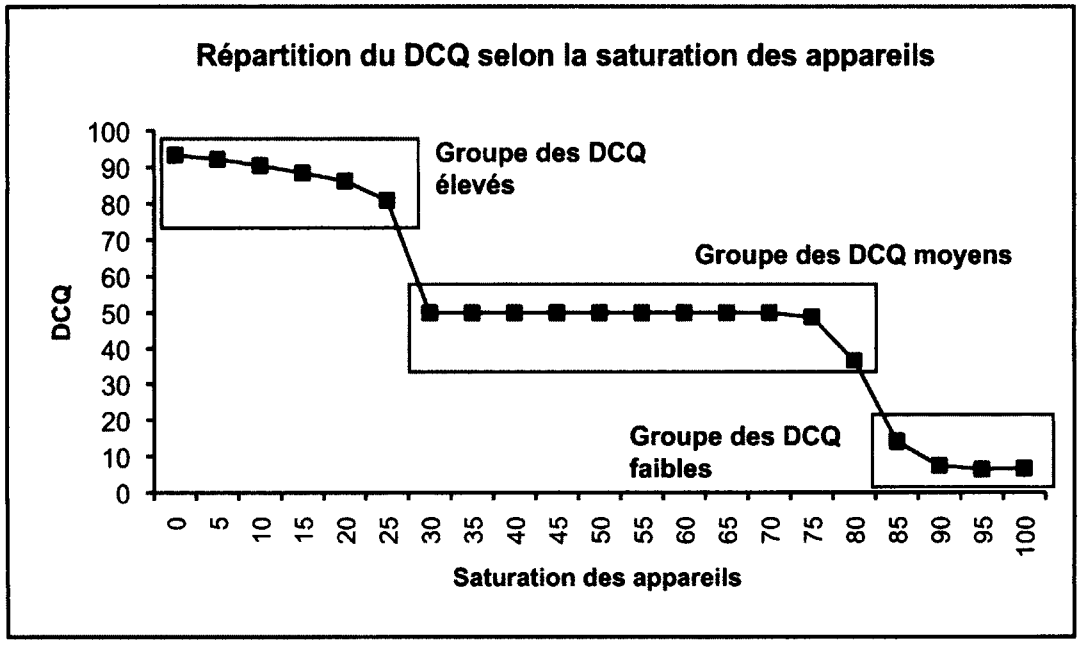

FIGURE 4.2 – Diagramme présentant la distribution du DCQ selon la saturation des ressources d'un appareil

réels, où la saturation sera la même pour les trois types de ressources (CPU, RAM et espace disque). Toutefois, ce test permet de vérifier que le FLORE réagit de la façon espérée dans un cas spécifique. Elle permet d'envisager des écarts marqués dans l'attribution des DCQ entre les appareils donnant des performances faibles, des performances moyennes et élevé. La répartition des DCQ dans des cas réels sera discutée lors de l'analyse des résultats des six scénarios d'évaluation.

4.3 Évaluations

Le but de l'évaluation du projet Tyche consistait à mettre celui-ci dans des situations réelles, mettant en lumière les principaux mécanismes et fonctionnalités du projet Tyche et de l'engin de raisonnement FLORE. Ainsi, six scénarios d'évaluation ont été définis comprenant pour chacun plusieurs itérations. Les scénarios visaient à mettre en lumière ces aspects :

- Scénario #1 : les besoins d'une application envers des périphériques ;
- Scénario #2 : le déploiement d'une application versus la saturation des appareils ;
- Scénario #3 : les relations entre les zones d'un espace intelligent ;
- Scénario #4 : la situation géographique de l'utilisateur et ses capacités ;

- Scénario #5 : les préférences de l'utilisateur ;
- Scénario #6 : l'auto-optimisation de l'organisation logicielle.

Pour mettre en place ces scénarios et leurs itérations, un banc d'essai a été mis en place permettant de gérer aisément les noeuds appareils et le noeud coordonnateur. Ainsi, deux ordinateurs munis chacun d'un processeur Intel Quad Core Q9550 cadencé à 2,83 GHz, d'un disque dur de 7200 RPM et de 4 Gigs de mémoire RAM à 800 MHz ont été utilisés. Chacun de ces ordinateurs ont été divisés en cinq machines virtuelles Linux Debian 5.0.5 à l'aide de l'outil de virtualisation VMWare. Les noeuds appareils utilisent la machine virtuelle Java 1.6 développée par Sun/Oracle pour Linux. Ces machines virtuelles ont été utilisées pour simuler les appareils de l'espace intelligent du Laboratoire DOMUS et ont permis d'installer et de démarrer facilement l'amorce (bootstrap) du noeud appareil via les outils *Secure File Transport Protocol* (SFTP) et SSH. Ainsi, huit noeuds appareils ont été créés, disposés dans l'environnement à des positions précises. La Figure 4.3 présente la disposition ainsi que l'orientation et le champ de vision des appareils et de leur dispositif d'affichage utilisé lors des scénarios d'évaluation. Quant au noeud coordonnateur, le même appareil que lors des tests de performance a été utilisé, soit un ordinateur utilisant la machine virtuelle Java 1.6 de Apple, configuré avec un processeur Intel Core 2 Duo E6600, un disque dur de 5400 RPM, 2 Gigs de RAM à 667 MHz et le système d'exploitation Mac OS 10.5. L'utilisation du même ordinateur pour le noeud coordonnateur que pour les tests de performance de la section précédente, a permis de comparer les temps de traitement utilisé par le Projet Tyche et de faire une approximation des temps moyens pour l'appel des web services.

4.3.1 Scénario #1 : Déploiement d'une application avec dépendances matérielles dans une zone de l'espace intelligent

Ce premier scénario vise à évaluer la fonctionnalité du FLORE alors qu'une application nécessitant des périphériques matériels particuliers doit être déployée dans l'espace intelligent. Pour ce faire, le banc d'essai a été utilisé avec les appareils disposés selon la Figure 4.3 et ayant les caractéristiques suivantes :

1. Appareil Cuisine #1 : Appareil de puissance moyenne (CPU 50/100, RAM 50/100, PM 5/10) avec écran tactile et clavier virtuel ;
2. Appareil Cuisine #2 : Appareil de faible puissance (CPU 30/100, RAM 30/100, PM 30/100) avec écran, clavier et souris ;

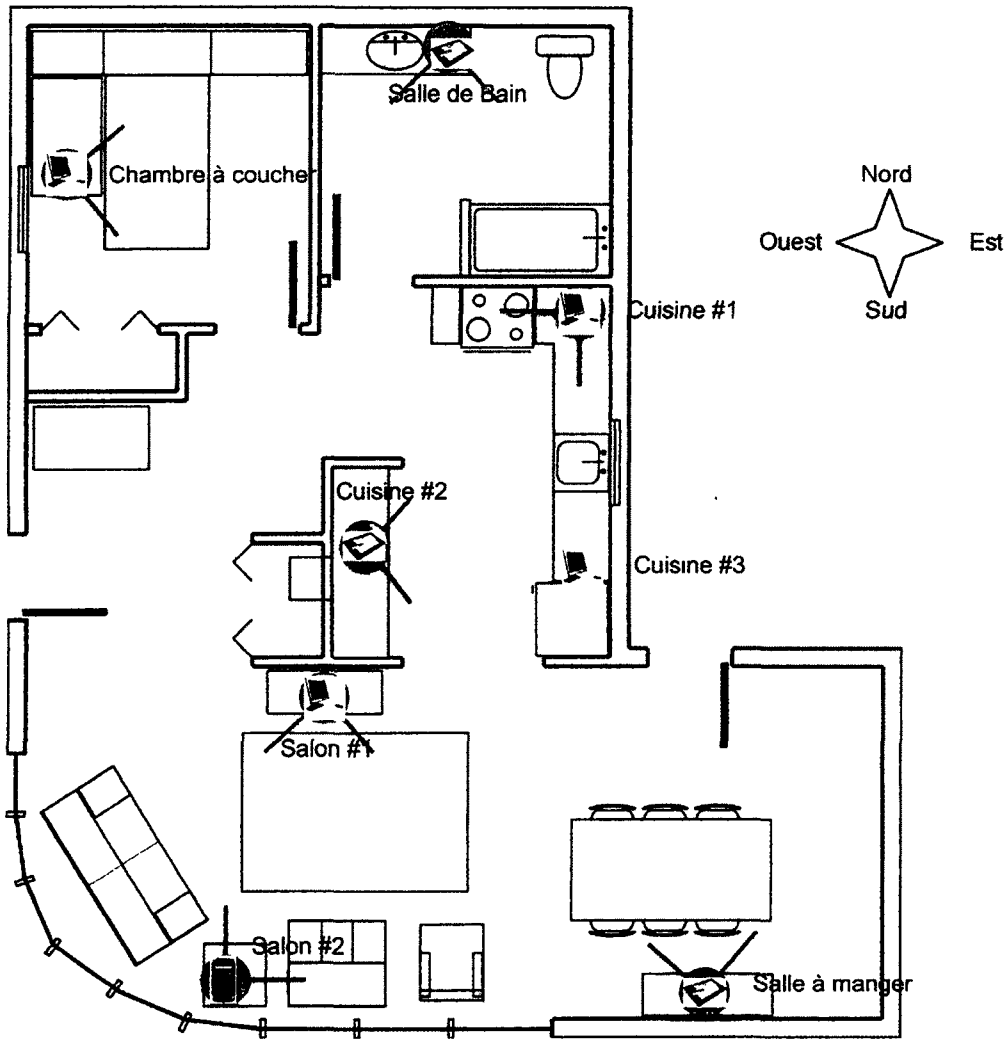

FIGURE 4.3 – Carte de l'appartement du Laboratoire DOMUS avec la disposition des appareils simulés lors des scénarios d'évaluation

3. Appareil Cuisine #3 : Appareil de puissance moyenne (CPU 70/100, RAM 70/100, PM 90/100) sans périphérique ;
4. Appareil Salon #1 : Appareil de puissance moyenne (CPU 70/100, RAM 70/100, PM 90/100) avec écran, clavier et souris ;
5. Appareil Salon #2 : Appareil faible puissance (CPU 20/100, RAM 20/100, PM 20/100) avec clavier tactile petit et clavier virtuel ;
6. Appareil Salle à manger #1 : Appareil de faible puissance (CPU 30/100, RAM 30/100, PM 30/100) avec écran, clavier et souris ;

7. Appareil Salle de bain #1 : Appareil de faible puissance (CPU 30/100, RAM 30/100, PM 30/100) avec écran, clavier et souris
8. Appareil Chambre à coucher #1 : Appareil de haute puissance (CPU 90/100, RAM 90/100, PM 100/100) avec écran, souris et clavier.

Malgré que ces appareils soient simulés sur deux ordinateurs divisés en machine virtuelle, leur emplacement et leurs propriétés ont été définis pour correspondre aux appareils de la Figure 4.3. L'utilisation des ressources pour tous les appareils a été fixée à 10% pour les trois types de ressources. La description des ressources des appareils a été également limitée à des valeurs entre 0 et 100 afin de faciliter l'homogénéité des ressources et faciliter l'analyse et la mise en place[2]. Enfin, l'application à déployer nécessitait pour les trois ressources une valeur de 5/100, soit 5% d'utilisation des ressources pour un appareil de haute performance (ressources à 100/100) et jusqu'à 17% pour les appareils de faible puissance comme l'appareil Salle à Manger #1.

Ensuite, les besoins d'une application à déployer ont été modifiés lors de plusieurs itérations, modifiant les besoins en périphérique et la zone dans laquelle l'application doit être déployée de préférence. Voici donc ces itérations/variations de ce scénario :

1. Déploiement de l'application nécessitant un clavier, une souris et un écran sur un appareil de la cuisine ;
2. Déploiement de l'application nécessitant un clavier, une souris, un écran sur un appareil du salon ;
3. Déploiement de l'application nécessitant un clavier, une souris et un écran sur un appareil de la salle à manger ;
4. Déploiement de l'application nécessitant un clavier, une souris et un écran sur un appareil de la salle de bain ;
5. Déploiement de l'application nécessitant un clavier, une souris et un écran sur un appareil de la chambre à coucher ;
6. Déploiement d'une application nécessitant un écran tactile dans la cuisine ;
7. Déploiement d'une application n'ayant aucun périphérique requis et zone visée.

Pour chacune de ces itérations, une hypothèse a été posée quant à l'appareil qui recevrait le meilleur DCQ. Ces hypothèses sont basées sur l'analyse du contexte, i.e. l'emplacement des appareils, leurs ressources, leurs périphériques, etc., et non sur la connaissance de l'algorithme

[2]ainsi il était facile de faire correspondre utilisation des ressources (saturations) et ressources totales

du FLORE. Il en va ainsi pour les hypothèses formulées lors des scénarios suivant. Voici les hypothèses ou résultats attendus pour le scénario #1 :

- Itération #1 : Appareil Cuisine 2 ;
- Itération #2 : Appareil Salon 1 ;
- Itération #3 : Appareil Salle à manger (SalleAManger1) ;
- Itération #4 : Appareil Salle de bain (SalleDeBain1) ;
- Itération #5 : Appareil Chambre à Coucher (Chambre1) ;
- Itération #6 : Appareil Cuisine 1 ;
- Itération #7 : Appareil Chambre à Coucher (Chambre1)

Le Tableau 4.3 présente les résultats de la mise en place du Scénario #1 et de ses sept itérations sur notre banc d'essai. Pour chaque itération du scénario, le temps de traitement moyen, l'hypothèse et les DCQ attribués aux appareils du banc d'essai sont présentés.

TABLEAU 4.3 – Résultats de la mise en place du Scénario #1

Itération	Temps moyen (sec.)	Hypothèse	Résultats	
			DCQ	Appareil
1	0,360	Cuisine2	85,34	Cuisine2
			66,66	Salon1
			6,35	SalleAManger1
			6,35	SalleDeBain1
			6,35	Chambre1
			0,0	Salon2
			0,0	Cuisine1
			0,0	Cuisine3
2	0,343	Salon1	89,86	Salon1
			6,35	Cuisine2
			6,35	SalleAManger1
			6,35	SalleDeBain1
			6,35	Chambre1
			0,00	Salon2
			0,00	Cuisine1
			0,00	Cuisine3
3	0,404	SalleAManger1	85,34	SalleAManger1

Suite à la page suivante

Itération	Temps moyen (sec.)	Hypothèse	Résultats	
			DCQ	Appareil
			6,35	Cuisine2
			6,35	SalleDeBain1
			6,35	Salon1
			6,35	Chambre1
			0,00	Salon2
			0,00	Cuisine1
			0,00	Cuisine3
4	0,321	SalleDeBain1	85,34	SalleDeBain1
			6,35	Cuisine2
			6,35	SalleAManger1
			6,35	Salon1
			6,35	Chambre1
			0,00	Salon2
			0,00	Cuisine1
			0,00	Cuisine3
5	0,324	Chambre1	90,24	Chambre1
			6,35	Cuisine2
			6,35	SalleAManger1
			6,35	SalleDeBain1
			6,35	Salon1
			0,00	Salon2
			0,00	Cuisine1
			0,00	Cuisine3
6	0,373	Cuisine1	88,41	Cuisine1
			37,49	Salon2
			0,00	Cuisine2
			0,00	SalleAManger1
			0,00	SalleDeBain1
			0,00	Cuisine3
			0,00	Salon1
			0,00	Chambre1

Suite à la page suivante

Itération	Temps moyen (sec.)	Hypothèse	Résultats	
			DCQ	Appareil
7	0,339	Chambre1	90,24	Chambre1
			89,61	Cuisine3
			88,41	Cuisine1
			85,34	Cuisine2
			66,66	Salon1
			37,49	Salon2
			6,35	SalleAManger1
			6,35	SalleDeBain1

Discussion

Les résultats récoltés lors de la mise en place du scénario #1 confirme les hypothèses que nous avions posées. Pour chacune des itérations, l'appareil que nous avions choisis comme étant le plus adapté à l'application et à ses besoins, a été également noté comme ayant le meilleur DCQ par l'engin de raisonnement. Comme ce scénario est assez simple et utilise les concepts de base du FLORE, il était primordial que nos hypothèses soient confirmées.

Sachant que le FLORE donne le DCQ minimal, soit 0, aux appareils n'ayant pas les périphériques requis, nous savions aux départs que les appareils n'ayant pas de clavier, souris et d'écran d'affichage dans les itérations 1 à 5 auraient alors un DCQ de 0. Toutefois, pour ce qui est de l'attribution des DCQ pour les autres appareils, nous ne savions pas à priori quelle serait leur valeur. Dans toutes les itérations, l'appareil ayant reçu le DCQ le plus élevé à reçu un quotient supérieur à 80, ce qui les classes nettement dans la catégorie des appareils à DCQ élevé, donc viable pour l'application à déployer. On voit également que les appareils situés dans des zones non contigües à la zone visée par l'application reçoivent des DCQ faibles en deçà de 20, comportement également visé, car ces appareils, quoiqu'étant membre de l'espace intelligent, n'appartiennent pas de près au contexte visé. Les liens existants entre les diverses zones d'un espace et leur importance dans le FLORE sont évalués plus en détail dans le scénario d'évaluation #2.

4.3.2 Scénario #2 : Déploiement d'une application sans dépendances logicielles et matérielles dans une zone de l'espace intelligent, avec variation de la saturation des performances des systèmes

Dans ce scénario, l'impact de l'utilisation des trois types de ressources lors d'un raisonnement d'auto-organisation a été évalué. Trois variations du scénario ont été implémentées :

1. Déploiement d'une application dans la cuisine alors que les ressources des appareils dans celle-ci sont saturées (utilisées) à 80 %. Les autres appareils dans l'espace intelligent sont à 20% de saturation ;
2. Déploiement d'une application dans la cuisine alors que les ressources des appareils dans celle-ci sont saturées (utilisées) à 80 %. Les autres appareils dans l'espace intelligent sont à 80 % de saturation ;
3. Déploiement d'une application dans la cuisine alors que les ressources des appareils dans celle-ci sont saturées (utilisées) à 20 %. Les autres appareils dans l'espace intelligent sont à 80 % de saturation.

Dans la première itération, nous voulions évaluer l'impact d'une saturation importante des ressources sur les appareils d'une zone alors qu'une application à déployer visait cette même zone. Dans la seconde variation, l'utilisation des ressources de tous les appareils a été augmentée à un seuil élevé afin de fournir des données comparatives à notre première itération. Enfin, la troisième itération servait à démontrer le clivage (Objectif #4) dans l'attribution d'un DCQ alors que les appareils de la zone visée sont peu utilisés et que le reste des appareils ont une saturation élevée. Comme l'application à déployer lors de ces trois itérations vise également une zone particulière, les liens résidant entre les zones sont également utilisés par le FLORE. Ainsi, le DCQ de certains appareils devrait être favorisé par leur proximité avec la zone visée.

À l'instar du premier scénario, des hypothèses ont également été posées en ce qui a trait aux résultats de la mise en place de ce scénario :

- Itération 1 : Les DCQ des appareils de la cuisine seront bas < 50 et le DCQ des appareils du Salon seront à plus de 50. L'appareil avec le meilleur DCQ sera l'appareil Salon1 ;
- Itération 2 : Les DCQ des appareils de la cuisine seront bas < 50 et le DCQ des appareils de l'espace seront très bas (< 25). L'appareil avec le meilleur DCQ sera Cuisine3 ;
- Itération 3 : Les DCQ des appareils de la cuisine seront élevé > 70 et le DCQ des appareils de l'espace seront très bas (< 25). L'appareil avec le meilleur DCQ sera Cuisine3.

Les résultats du Scénario #2 sont présentés dans le Tableau 4.4.

TABLEAU 4.4 – Résultats de la mise en place du Scénario #1

Itération	Temps moyen (sec.)	Hypothèse	Résultats	
			DCQ	Appareil
1	0,402	Salon1	66,61	Salon1
			33,34	Salon2
			25,55	Cuisine3
			14,10	Cuisine1
			6,79	Cuisine2
			6,48	SalleAManger1
			6,48	SalleDeBain1
			6,35	Chambre1
2	0,324	Cuisine3	25,55	Cuisine3
			17,85	Salon1
			14,10	Cuisine1
			12,46	Chambre1
			6,79	Cuisine2
			6,63	Salon2
			6,47	SalleAManger1
			6,47	SalleDeBain1
3	0,295	Cuisine3	85,06	Cuisine3
			80,90	Cuisine1
			49,99	Cuisine2
			17,85	Salon1
			12,46	Chambre1
			6,63	Salon2
			6,47	SalleAManger1
			6,47	SalleDeBain1

Discussion

Nos hypothèses pour l'organisation logicielle selon les zones des milieux ont été confirmées lors de la mise en place des trois itérations du scénario #2. Ainsi, dans la première évaluation, on peut voir que malgré le fait que l'appareil Salon1 ne soit pas dans la zone visée, celui-ci

reçoit un DCQ moyen, puisqu'il offre des performances supérieures aux appareils de la cuisine et qu'il est situé dans une zone adjacente à la cuisine. Toutefois, notre hypothèse d'un DCQ supérieur à 40 pour l'appareil Salon2 est infirmé, car la faible capacité de l'appareil a eu plus de poids dans l'attribution du DCQ que sa localisation dans une zone adjacente à la cuisine.

Pour ce qui est de l'itération 2, les résultats confirment nos hypothèses à savoir que des appareils avec une forte saturation recevront des DCQ faibles associé aux groupes des DCQ faible. Enfin, l'itération 3 démontre, comme nous le voulions, l'écart et la variabilité du DCQ entre des appareils bien situés, avec une saturation faible, vis-à-vis des appareils moins bien situés et saturés. Le résultat moyen de l'appareil Cuisine2 (DCQ de 49,99) est principalement dû à sa performance moindre (CPU, RAM et espace disque de 30/100). Ainsi, puisque l'appareil est saturé initialement à 20% et que l'application à déployer consommerait 5/100 de ressources, sa saturation future en cas de déploiement passerait à 37% versus les 27% pour l'appareil Cuisine3 et les 30% de Cuisine1.

4.3.3 Scénario #3 : Évaluation des relations entre les zones des espaces intelligents dans le contexte du déploiement d'une application visant une zone

Le but du scénario #3 visait à évaluer et tester l'intergiciel et son engin de raisonnement dans le cas où des applications ont à être déployées dans des zones précises d'un espace intelligent. Ce scénario met donc l'accent sur les liens existants entre les zones des espaces et l'influence de ces liens lors d'une requête d'auto-organisation. Le banc d'essai du scénario #1 a été réutilisé, en modifiant cependant la saturation des appareils afin de mettre l'emphase sur le traitement des zones. Quatre itérations ont été effectuées lors de ce scénario :

1. Déploiement d'une application liée à la cuisine. Tous les appareils sont saturés à 100 % excepté l'appareil du salon qui est à 0%. La relation de la zone Cuisine et de la zone Salon est qualifiée comme étant adjacente ;
2. Déploiement d'une application liée à la cuisine. Tous les appareils sont saturés à 100 % excepté l'appareil de la Salle à Manger qui est à 0%. Les zone Cuisine et Salle à manger ne sont pas qualifiées comme étant adjacente et ont un parent commun, l'appartement DOMUS ;
3. Déploiement d'une application liée à la zone « Appartement Domus ». Tous les appareils sont saturés à 100 % excepté l'appareil de la Cuisine 1 et 2 qui sont à 0%. La zone Appartement Domus est qualifié comme étant la zone mère de la zone Cuisine ;

4. Déploiement d'une application liée à la zone Voiture[3]. Tous les appareils sont saturés à 0%. La zone Voiture est disjointe des zones de l'appartement DOMUS.

Grâce à ces variations, les quatre types de relation de zone reconnus par le Tyche et le FLORE ont été testés : zones adjacentes, zones ayant un même parent, zones avec une relation de parenté et zones disjointes. Voici les hypothèses posées quant aux résultats attendus :

- Itération 1 : Les DCQ des appareils de la cuisine seront très bas, DCQ < 20, et les DCQ des appareils du Salon seront à plus de 40. L'appareil Salon2 aura le DCQ le plus élevé ;
- Itération 2 : Les DCQ des appareils de la cuisine seront très bas, DCQ < 20 et le DCQ de l'appareil de la salle à manger sera entre 30 et 40. L'appareil Salon2 aura le DCQ le plus élevé ;
- Itération 3 : Les DCQ des appareils de la cuisine seront élevé, DCQ > 70, et le DCQ des appareils des autres espaces seront très bas, DCQ < 25. L'appareil Cuisine3 aura le DCQ le plus élevé ;
- Itération 4 : Tous les appareils auront un DCQ de 0.

Le Tableau 4.5 présente les résultats du scénario #3 et ces quatre itérations.

TABLEAU 4.5 – Résultats de la mise en place du Scénario 3

Itération	Temps moyen (sec.)	Hypothèse	Résultats	
			DCQ	Appareil
1	0,358	Salon2	66,63	Salon2
			66,61	Salon1
			10,15	Cuisine2
			7,77	Cuisine1
			7,02	Cuisine3
			6,35	Chambre1
			6,35	SalleAManger1
			6,35	SalleDeBain1
2	0,558	Salon2	13,74	Salon2
			10,15	Cuisine2
			7,77	Cuisine1
			7,02	Cuisine3

Suite à la page suivante

[3]le nom ici n'a pas d'importance, nous avons choisi Voiture afin d'illustrer une zone disjoint n'appartenant pas à l'appartement DOMUS

Itération	Temps moyen (sec.)	Hypothèse	Résultats	
			DCQ	Appareil
			6,72	Salon1
			6,35	Chambre1
			6,35	SalleAManger1
			6,35	SalleDeBain1
3	0,383	Cuisine3	92,92	Cuisine3
			92,18	Cuisine1
			89,80	Cuisine2
			13,74	Salon2
			10,15	SalleAManger1
			10,15	SalleDeBain1
			6,72	Salon1
			6,64	Chambre1
4	0,286	-	6,35	Chambre1
			6,35	Salon1
			6,35	Cuisine2
			6,35	SalleAManger1
			6,35	SalleDeBain1
			6,35	Cuisine1
			6,35	Salon2
			6,35	Cuisine3

Discussion

Les résultats de la mise en place du scénario #3 ont démontré que le projet Tyche gère avec succès les relations entre les zones d'un espace intelligent. Par exemple, on peut voir lors de la première itération que les appareils de la zone Salon offrent des résultats supérieurs à ceux des autres zones, puisque ceux-ci sont entre autres situés dans une zone adjacente à la zone Cuisine. De plus, on peut voir dans la deuxième itération, que malgré sa saturation faible, l'appareil Salle à manger #1 ne se voit pas attribué un DCQ supérieur à ceux des appareils des autres zones, dû aux faibles relations entre la zone Cuisine et Salle à Manger (zone soeur via la zone mère "appartement DOMUS"). Dans la troisième itération, puisque la zone visée est l'appartement DOMUS, toutes les zones enfants de celle-ci sont aux mêmes pieds d'égalité

par rapport au contexte de déploiement. Toutefois, puisque les ressources des appareils de la cuisine sont moins saturées, ceux-ci reçoivent un DCQ plus fort que les autres appareils du milieu.

Enfin, la dernière itération de ce scénario n'a pas retourné les résultats prévus. Nous aurions aimé que les DCQ attribués dans le cas de zones disjointes soient de zéro. Nous comptions sur l'une de nos règles de logique floue pour abaisser le DCQ à 0, mais une autre règle (la règle 7 du raisonnement floue des noeuds appareils, voir Extrait 3.5) a été déclenchée et est venue élargir la fonction d'appartenance de defuzzification, déplaçant ainsi le centroïde de la fonction vers un DCQ de 6,35. Des modifications à l'engin de raisonnement ont permis de régler ce petit problème en attribuant un DCQ de 0 points lors que les zones sont disjointes.

4.3.4 Scénario #4 : Déploiement d'une application liée à un utilisateur en particulier

Le scénario #4 est l'un des scénarios les plus importants, puisqu'il évalue et teste dans un premier temps la majorité des fonctionnalités de l'algorithme du FLORE et, dans un deuxième temps, qu'il inclue, dans le raisonnement d'auto-organisation, les acteurs majeurs des espaces intelligents, les utilisateurs. C'est également le scénario qui utilise le plus de données contextuelles sont : les appareils, les applications, les utilisateurs, la topologie des espaces et les périphériques. Par l'importance des données et des fonctionnalités déclenchées par l'engin de raisonnement, il devrait s'agir également du scénario où les temps de traitement seront les plus longs.

Ainsi, ce scénario évalue le projet Tyche dans le contexte où une application à déployer dans un espace intelligent vise un utilisateur en particulier. Afin d'illustrer les divers cas de figure et de tester tous les cas de l'algorithme de raisonnement, sept itérations ont été construites avec des archétypes d'utilisateurs bien définis. Voici ces sept itérations avec pour chacune leur utilisateur relié :

1. Utilisateur 1 « générique »
   - Vue 20/20 ;
   - Champ de vision 170° ;
   - Force préhension 90 kg ;
   - Ouverture de la main : 110° ;
   - Vitesse de déplacement : 1,45 m/s ;
   - Préférences : Pour tous les périphériques 1 = aime.

2. Utilisateur 2 « myope »
   - Vue 10/20 ;
   - Champ de vision 170° ;
   - Force préhension 90 kg ;
   - Ouverture de la main : 110° ;
   - Vitesse de déplacement : 1,45 m/s ;
   - Préférences : Pour tous les périphériques 1 = aime.
3. Utilisateur 3 « vision restreinte »
   - Vue 20/20 ;
   - Champ de vision 60° ;
   - Force préhension 90 kg ;
   - Ouverture de la main : 110° ;
   - Vitesse de déplacement : 1,45 m/s ;
   - Préférences : Pour tous les périphériques 1 = aime.
4. Utilisateur 4 « difficulté de préhension »
   - Vue 20/20 ;
   - Champ de vision 170° ;
   - Force préhension 20 kg ;
   - Ouverture de la main : 30° ;
   - Vitesse de déplacement : 1,45 m/s ;
   - Préférences : Pour tous les périphériques 1 = aime.
5. Utilisateur 5 « mobilité lente »
   - Vue 20/20 ;
   - Champ de vision 170° ;
   - Force préhension 90 kg ;
   - Ouverture de la main : 110° ;
   - Vitesse de déplacement : 0,8 m/s ;
   - Préférences : Pour tous les périphériques 1 = aime.
6. Utilisateur 6 « immobile »
   - Vue 20/20 ;
   - Champ de vision 170° ;
   - Force préhension 20 kg ;
   - Ouverture de la main : 110° ;
   - Vitesse de déplacement : 0,2 m/s ;

- Préférences : Pour tous les périphériques 1 = aime.

7. Utilisateur 7 « personne avec handicaps multiples »
   - Vue 10/20 ;
   - Champ de vision 90° ;
   - Force préhension 20 kg ;
   - Ouverture de la main : 30° ;
   - Vitesse de déplacement : 0,8 m/s ;
   - Préférences : Pour tous les périphériques 1 = aime.

Pour éviter une interaction des préférences des utilisateurs (évalué au scénario #5), les préférences des utilisateurs ont été attribuées de façon à ce que les utilisateurs préfèrent tous les types de périphériques. Pour les sept itérations, l'utilisateur a été positionné dans la cuisine à 0,5 mètre à l'est de l'appareil Cuisine2 et 2 mètre au sud de l'appareil Cuisine1. Son orientation est centrée vers le nord, donc face à la porte de la salle de bain. L'utilisateur a été placé à cette position, dans la zone Cuisine, car celle-ci correspond, grosso-modo, au centre de l'appartement DOMUS, la distance moyenne entre tous les appareils sera donc moindre, donnant des probabilités de DCQ fort plus élevés. La Figure 4.4 illustre la position de l'utilisateur dans l'appartement lors de la mise en place du scénario #4. La zone orange correspond aux limites de la zone Cuisine, la zone rouge correspond à la zone couverte par l'utilisateur lors d'un déplacement considéré comme rapide et enfin la zone verte correspond au champ de vision de l'utilisateur et à son acuité visuelle optimale. Les données présentés dans la Figure 4.4 correspondent à l'itération de l'utilisateur #2.

La proximité des utilisateurs avec l'appareil Cuisine2 devrait favoriser ce dernier face aux autres appareils de l'espace. Toutefois, les appareils Cuisine1 et Salon1 offrent des performances plus importantes et possèdent des dispositifs d'affichage plus grand, favorisant la lecture et les interactions avec les applications. Pour chaque type d'utilisateur, des hypothèses ont, encore une fois, été posées quant aux résultats attendus :

1. Application déployé sur l'appareil Cuisine 2 avec un DCQ > 70 ;
2. Application déployé sur l'appareil Cuisine 2 avec un DCQ > 50 ;
3. Application déployé sur l'appareil Cuisine 2 avec un DCQ > 70 et Cuisine 2 avec un DCQ > 40 ;
4. Application déployé sur l'appareil Cuisine 1 avec un DCQ >70 et Cuisine 2 DCQ = 0 ;
5. Application déployé sur l'appareil Cuisine 2 avec un DCQ > 40 et Cuisine 1 avec un DCQ < 40 ;

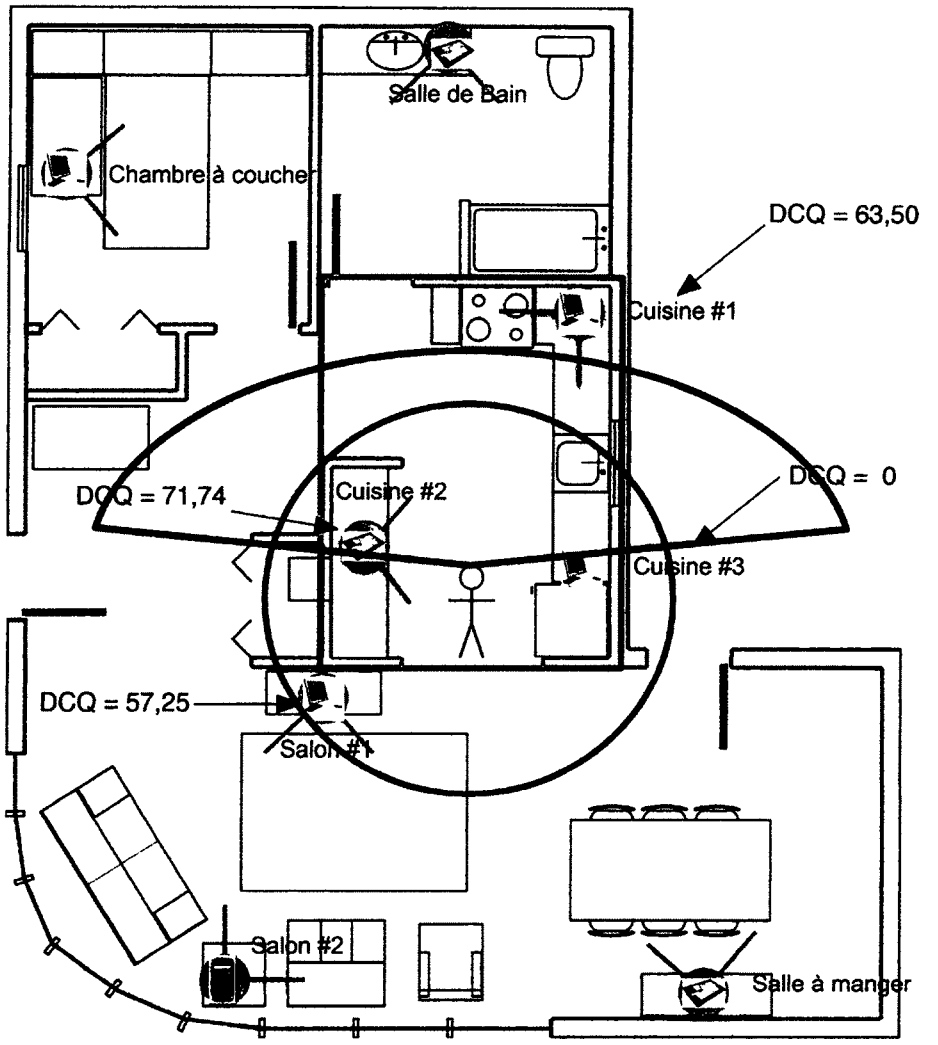

FIGURE 4.4 – Emplacement de l'utilisateur dans le Scénario #4 et les zones de modalités d'interaction correspondantes à l'itération #2

6. Application déployé sur l'appareil Cuisine 2 avec un DCQ entre 20 et 40. Cuisine 1 < 20 ;
7. Application déployé sur l'appareil Cuisine 1 avec un DCQ entre 20 et 40, Cuisine 2 < 20.

Le Tableau 4.6 présente les résultats du scénario #4.

TABLEAU 4.6 – Résultats de la mise en place du Scénario 4

Itération	Temps moyen (sec.)	Hypothèse	Résultats	
			DCQ	Appareil
1	0,549	Cuisine2	75,56	Cuisine2
			63,50	Cuisine1
			63,18	Salon1
			45,40	Salon2
			19,89	SalleDeBain1
			19,89	Chambre1
			15,30	SalleAManger1
			0,00	Cuisine3
2	0,534	Cuisine2	71,74	Cuisine2
			63,50	Cuisine1
			57,25	Salon1
			45,40	Salon2
			19,89	SalleDeBain1
			19,89	Chambre1
			15,30	SalleAManger1
			0,00	Cuisine3
3	0,540	Cuisine2	80,00	Cuisine2
			66,28	Salon1
			63,46	Cuisine1
			49,93	Salon2
			19,85	SalleDeBain1
			19,84	SalleAManger1
			19,84	Chambre1
			0,00	Cuisine3
4	0,285	Cuisine1	76,89	Cuisine1
			66,31	Salon1
			49,93	Salon2
			0,00	Cuisine2
			0,00	SalleAManger1

Suite à la page suivante

Itération	Temps moyen (sec.)	Hypothèse	Résultats	
			DCQ	Appareil
			0,00	SalleDeBain1
			0,00	Cuisine3
			0,00	Chambre1
5	0,510	Cuisine2	75,84	Cuisine2
			65,38	Salon1
			63,43	Cuisine1
			35,67	Salon2
			19,83	SalleDeBain1
			19,83	Chambre1
			5,57	SalleAManger1
			0,00	Cuisine3
6	0,620	Cuisine2	75,84	Cuisine2
			49,97	Salon1
			49,75	Cuisine1
			35,93	Salon2
			6,14	SalleDeBain1
			6,14	Chambre1
			5,84	SalleAManger1
			0,00	Cuisine3
7	0,429	Cuisine1	63,43	Cuisine1
			38,68	Salon1
			35,67	Salon2
			0,00	Cuisine2
			0,00	SalleAManger1
			0,00	SalleDeBain1
			0,00	Cuisine3
			0,00	Chambre1

Discussion

Les résultats de ce scénario sont d'une grande importance dans l'évaluation de notre travail de recherche. Dans un premier temps, nos hypothèses quant aux appareils ayant le meilleur

DCQ pour tous les archétypes d'utilisateur se sont avérées concluantes. Pour les sept scénarios, l'intergiciel Tyche a choisi les mêmes appareils que nous avions proposés, et ce, malgré le grand nombre de données qui dans plusieurs cas, amenait à des situations conflictuelles (distance versus performance, emplacement versus qualité de lecture, etc.).

Les temps de traitement pour les sept itérations sont en deçà des 1-2 secondes et varient selon le contexte ou le nombre de données à traiter. Au maximum, le temps pour retourner le DCQ a été de 0,6 seconde alors que 6 appareils sur 7 ont été analysés par le FLORE (performance, situation géographique et interaction homme-machine). Toutefois, une petite ombre au tableau s'est glissée quant à nos hypothèses sur la répartition des DCQ pour les itérations 5 à 7 , qui sont fausses. Les DCQ attribués ont été supérieurs à ceux attendus, par exemple, l'appareil Cuisine1 a reçu un DCQ de 63,43 dans l'itération #5, alors que nous nous attendions à moins de 40.

Nous avons donc analysé l'algorithme du FLORE et utilisé les fonctionnalités de mise au point de nos outils d'implémentation. Nous avons découvert que la taille des dispositifs d'affichage, la résolution de ceux-ci et la distance entre l'utilisateur et les appareils avaient plus de poids dans le raisonnement d'auto-organisation que la mobilité de l'utilisateur, son orientation et son champ de vision. Pour correspondre parfaitement à nos attentes, il faudrait, par exemple, augmenter l'importance de la mobilité de l'utilisateur ou bien augmenter le poids d'un appareil qui est situé dans le champ de vision de l'utilisateur.

Enfin, nos objectifs de répartition du DCQ et de variabilité sont atteints dans le cadre de scénario. Dans toutes les itérations, on peut voir qu'il y a une bonne répartition des DCQ avec des écarts importants entre les appareils de zones différentes, entre les appareils situés près de l'utilisateur et ceux distants, et évidemment ceux qui ont les périphériques dont les caractéristiques correspondent aux capacités des utilisateurs (force et ouverture de la main). Par exemple, dans l'itération #1, les DCQ attribués aux appareils Cuisine1, Cuisine2 et Salon1 sont particulièrement élevés et groupés grâce soit à leur emplacement, leur dispositif d'affichage ou bien leurs ressources élevés et libres. Les autres appareils ont des DCQ moins élevés selon leur distance avec l'utilisateur et leurs périphériques. Dans l'itération #7, un nombre important d'appareils sont exclues et des DCQ moyens sont attribués aux appareils à cause des capacités d'interaction réduites de l'utilisateur.

4.3.5 Scénario #5 : Évaluation des préférences des utilisateurs envers des périphériques d'interaction lors de l'organisation logicielle

Le scénario #5 visait à évaluer le FLORE lors de raisonnements utilisant les préférences des utilisateurs quant aux périphériques d'interaction. Dans le scénario précédent, les préférences des archétypes d'utilisateur était constantes et fixées à "aimer" pour tous les types de périphériques, afin que ceux-ci n'interviennent pas dans l'attribution des DCQ. Ce scénario-ci met donc en évidence le rôle des préférences dans l'auto-organisation.

À fin de rappel, chaque profil d'utilisateur peut définir des préférences quant aux périphériques d'interaction. Ces préférences consistent en trois valeurs quantitatives : aime ce type de périphérique, est neutre face à celui-ci ou bien n'aime pas interagir avec ce type. Ces préférences sont associées chacune à une valeur qualitative telle que dans une échelle de Likert (aime = 1, neutre = 0 et n'aime pas = -1). La moyenne de ces valeurs face aux périphériques d'un appareil correspond à l'idée générale qu'un utilisateur a de l'appareil et est ajoutée en entrée au FLORE, qui l'associe ensuite à des fonctions d'appartenances (aime, neutre et n'aime pas). Ces dernières valeurs quantitatives sont utilisées dans les règles de raisonnement flou.

Le profil de l'utilisateur du scénario #4 - itération 1 a donc été repris avec les appareils du banc d'essai des scénarios précédents, en attribuant cependant les mêmes ressources et saturation à tous les appareils, afin que les variations dans les DCQ ne soient exprimées que par les variations dans les préférences. Quelques variations sur les préférences de l'utilisateur ont alors été faites :

1. Utilisateur 1 « aime tout »
   - Écran tactile : préférence = 1 ;
   - Clavier : préférence = 1 ;
   - Souris : préférence = 1 ;
   - Trackball : préférence = 1 ;
   - Clavier virtuel + écran tactile = 1.
2. Utilisateur 2 « neutre à tout »
   - Écran tactile : préférence = 0 ;
   - Clavier : préférence = 0 ;
   - Souris : préférence = 0 ;
   - Trackball : préférence = 0 ;
   - Clavier virtuel + écran tactile = 0.
3. Utilisateur 3 « aime rien »

- Écran tactile : préférence = -1 ;
- Clavier : préférence = -1 ;
- Souris : préférence = -1 ;
- Trackball : préférence = -1 ;
- Clavier virtuel + écran tactile = -1.

4. Utilisateur 4 « n'aime pas écran tactile »
   - Écran tactile : préférence = -1 ;
   - Clavier : préférence = 1 ;
   - Souris : préférence = 1 ;
   - Trackball : préférence = 1 ;
   - Clavier virtuel + écran tactile = -1.
5. Utilisateur 5 « n'aime pas les interfaces physiques »
   - Écran tactile : préférence = 1 ;
   - Clavier : préférence = -1 ;
   - Souris : préférence = -1 ;
   - Trackball : préférence = -1 ;
   - Clavier virtuel + écran tactile = 1.
6. Utilisateur 7 « Charles »
   - Écran tactile : préférence = 0 ;
   - Clavier : préférence = 1 ;
   - Souris : préférence = 1 ;
   - Trackball : préférence = -1 ;
   - Clavier virtuel + écran tactile = 0.

Voici les hypothèses quant aux résultats attendus

- Utilisateur 1 : Toutes les applications auront le même DCQ > 70 ;
- Utilisateur 2 : Toutes les applications auront le même DCQ > 40 ;
- Utilisateur 3 : Toutes les applications auront le même DCQ < 40
- Utilisateur 4 : Appareil 2 et 3 avec un DCQ > 70, autres appareils avec DCQ < 40 ;
- Utilisateur 5 : Appareil 1 avec un DCQ > 70, Appareil 4,5 et 6 avec un DCQ > 40, autres appareils avec DCQ < 40 ;
- Utilisateur 6 : Appareil 2 avec un DCQ > 70, autres appareils avec DCQ > 40.

Le tableau 4.7 présente les résultats amassés lors de la mise en place de ce scénario et des ses itérations.

TABLEAU 4.7 – Résultats de la mise en place du Scénario 5

Itération	Temps moyen (sec.)	Hypothèse	Résultats	
			DCQ	Appareil
1	0,330	Tous > 70	79,74	Cuisine2
			79,57	Cuisine1
			63,47	Chambre1
			63,47	SalleDeBain1
			63,47	Salon1
			63,45	SalleAManger1
			63,45	Salon2
			0,00	Cuisine3
2	0,506	Tous > 40	79,84	Cuisine2
			79,68	Cuisine1
			63,47	Chambre1
			63,47	SalleDeBain1
			63,47	Salon1
			63,45	SalleAManger1
			63,45	Salon2
			0,00	Cuisine3
3	0,610	Tous < 40	77,16	Cuisine2
			76,10	Cuisine1
			63,47	Chambre1
			63,47	SalleDeBain1
			63,47	Salon1
			63,45	SalleAManger1
			63,45	Salon2
			0,00	Cuisine3
4	0,710	Cuisine2	79,74	Cuisine2
			76,10	Cuisine1
			63,47	Chambre1
			63,47	SalleDeBain1
			63,47	Salon1

Suite à la page suivante

Itération	Temps moyen (sec.)	Hypothèse	Résultats	
			DCQ	Appareil
			63,45	SalleAManger1
			63,45	Salon2
			0,00	Cuisine3
5	0,488	Cuisine1	79,57	Cuisine1
			77,16	Cuisine2
			63,47	Chambre1
			63,47	SalleDeBain1
			63,47	Salon1
			63,45	SalleAManger1
			63,45	Salon2
			0,00	Cuisine3
6	0,608	Cuisine2	79,74	Cuisine2
			79,68	Cuisine1
			63,47	Chambre1
			63,47	SalleDeBain1
			63,47	Salon1
			63,45	SalleAManger1
			63,45	Salon2
			0,00	Cuisine3

Discussion

De nos six scénarios d'évaluation, les résultats du scénario sur les préférences des utilisateurs sont probablement les plus décevants. Les préférences qu'un utilisateur a envers des types de périphériques n'ont pas eu l'impact auquel nous nous attendions. Quoique le mécanisme fonctionne et permet d'avantager ou de désavantager les appareils selon les préférences des utilisateurs, celui-ci ne permet pas de différencier de façon majeure le DCQ de deux appareils ayant des périphériques aimés et non aimés. Par exemple, dans l'itération #4 et #5, la différence entre le DCQ des appareils Cuisine1 et Cuisine2 n'est que 2 ou 3 points alors que dans ces deux cas les périphériques de Cuisine1 sont aimés et de Cuisine2 sont non aimés.

En analysant d'un peu plus près les résultats, on se rend compte que ce sont les préférences moyennes négatives (n'aime pas) qui n'exercent pas assez leur poids dans l'attribution

des DCQ. Dans les règles de raisonnement flous du noeud coordonnateur (voir Annexe B), nous avons principalement utilisé les préférences pour favoriser les appareils dans les règles de logique floue du FLORE. Ainsi, l'appartenance aux fonctions de préférence a été utilisée dans les 26 premières règles de logique floue de la partie noeud coordonnateur du FLORE et auraient dû être utilisée également dans les règles correspondantes aux fonctions de *defuzzification* non-optimal et impossible. Afin de couvrir toutes les combinaisons de règles possibles, l'ajout des fonctions d'appartenance des préférences dans les 15 dernières règles, triplerait le nombre de celles-ci (pour aime, neutre, n'aime pas), amenant probablement à une légère augmentation des temps de traitement. Ces ajouts de règles seront fait dans le cadre de nos travaux futurs, lors de l'ajout de nouvelles modalités d'interaction telles que la langue d'usage et les capacités cognitives.

4.3.6 Scénario #6 : Évaluation des fonctions d'auto-optimisation de l'organisation logicielle

Le dernier scénario visait à tester les fonctions d'auto-optimisation de l'organisation logicielle. Cette fonction permet à certaines applications d'être redéployé sur de nouveaux appareils, si ceux-ci offrent des performances supérieures à l'appareil précédent, i.e. qu'il reçoit un meilleur DCQ que l'appareil précédent. Tel que spécifié dans le Chapitre 3, une valeur plancher doit être franchie par le DCQ du nouvel appareil et un écart précis doit exister entre le DCQ des deux appareils pour que celui-ci soit considéré comme candidat à l'auto-optimisation.

Pour tester la fonctionnalité d'auto-optimisation, le banc d'essai du scénario #4 a été utilisé et une application a été déployée au préalable sur un appareil de l'appartement. Un nouvel appareil dans l'appartement a ensuite été ajouté à l'environnement, déclenchant ou non l'auto-optimisation de l'application sur le nouvel appareil. Les itérations ont consisté à faire varier le contexte du premier appareil sur lequel l'application était déployée en premier lieu et le contexte du nouvel appareil. Voici ces itérations :

1. Tous les appareils sont saturés à 80 %. Une seule application est déployée sur l'appareil Salon1. Un nouvel appareil pourvu d'une puissance moyenne (CPU 7/10, RAM 7/10, PM 9/10) est démarré dans la cuisine et possède une saturation de 10% ;
2. Tous les appareils sont saturés à 80 %. Une seule application est déployée sur l'appareil Salon1. Un nouvel appareil pourvu d'une puissance moyenne (CPU 7/10, RAM 7/10, PM 9/10) est démarré dans la cuisine et possède une saturation de 75% ;
3. Tous les appareils sont saturés à 80 %. Chaque application est déployée sur l'appareil

Cuisine1. Un nouvel appareil pourvu d'une puissance moyenne (CPU 7/10, RAM 7/10, PM 9/10) est démarré dans le salon et possède une saturation de 20%.

Nos hypothèses pour ce scénario sont relativement simples. Selon l'implémentation de l'algorithme d'auto-optimisation, la mécanique d'optimisation devrait être déclenchée pour la première itération, déplaçant l'application de l'appareil Salon1 vers l'appareil Cuisine3. Dans le cadre de la deuxième itération, aucune optimisation ne devrait être faite puisque le nouvel appareil ne propose pas des performances suffisantes pour justifier le redéploiement. Enfin, pour la dernière itération, l'auto-optimisation devrait être déclenchée et devrait redéployer l'application vers l'appareil Salon1.

Le Tableau 4.8 présente les résultats de la mise en place de ce scénario et de ses itérations.

TABLEAU 4.8 – Résultats de la mise en place du Scénario 6

Itération	Hypothèse	Appareil précédent	DCQ précédent	Nouvel appareil	DCQ
1	Cuisine3	Salon1	49,98	Cuisine3	91,73
2	pas d'optimisation	Salon1	49,98	-	-
3	Salon1	Cuisine1	46,49	Salon1	87,93

Discussion

Nos trois hypothèses se sont révélées positives. Le mécanisme de découverte des appareils a découvert le nouveau noeud rapidement, a vérifié si cet appareil pouvait donner un meilleur DCQ que les appareils précédents et a procédé au redéploiement, ou non, de l'application sur le nouvel appareil de l'espace. En ne tenant pas compte du téléchargement de l'application test et de son démarrage, l'auto-optimisation a été presque instantané, prenant moins d'une seconde. Les notifications d'auto-optimisation ont été ajoutées aux boîtes de messages du noeud coordonnateur et ont été accessibles via les outils de gestion.

Le seuil et l'écart de déclenchement que nous avions fixés par défaut à 60 points et 20 points, respectivement, ont bien fonctionné. Comme les DCQ attribués à des appareils offrant des performances moyennes se situent généralement entre 40 et 60, un seuil de 60 points de DCQ restreint l'optimisation à des appareils offrant dans un premier temps des performances de bonne à moyenne. Par contre, c'est surtout l'écart de déclenchement qui contrôle la mise en place de réorganisation logicielle. Avec un écart de 10 points, la fonctionnalité d'auto-optimisation a réduit les éventuels redéploiements d'application pour des gains mineurs de DCQ.

Toutefois, à la lumière de nos expérimentations, le seuil et l'écart ne devraient pas être unique à toutes les situations, mais devraient dépendre de la valeur du DCQ. Par exemple, pour une application déployée sur un appareil avec un DCQ de 10 points, donc considéré comme faible, un gain même de 5 points peut représenter pour l'application et les utilisateurs un gain important. De plus, avec un seuil de déclenchement élevé, l'activation de l'auto-optimisation est restreinte à l'arrivée d'appareils performants, bloquant l'auto-optimisation d'une application déployée avec un DCQ faible vers des appareils offrant des DCQ moyens. Ainsi, les seuils et les écarts devraient être liés aux fonctions d'appartenance des DCQ, les DCQ faibles ayant des écarts et des seuils de déclenchement plus faibles, par exemple 5 points et 40 points, les DCQ moyens le même écart que dans l'implémentation actuelle et les DCQ élevés des écarts moindres et des seuils plus élevés, par exemple 10 points et 90 points. Une telle implémentation favorisera l'auto-optimisation des applications qui ont été déployées dans des contextes peu favorables alors qu'elle stabilisera les applications bénéficiant de contextes favorables.

4.4 Discussion générale sur les résultats

Nos évaluations et nos tests se sont avérés positifs dans la majorité des cas et ont contribué à démontrer les fonctionnalités du Projet Tyche. Les objectifs que nous nous étions fixés au départ de notre phase de tests et d'évaluations ont été rencontrés pour les cas généraux.

Pour notre premier objectif, les temps de traitement dans toutes les variations des scénarios ont été en deçà de 1 seconde, et ce, en tenant compte du temps induit par l'appel des web services, la latence momentané du réseau, etc. Évidemment, le temps de traitement final est variable selon la taille de l'application à télécharger, à installer et à démarrer. De plus, un nombre d'appareils plus élevé augmenterait les temps de traitement, cependant huit appareils tels que dans notre banc d'essai correspond à un nombre plus élevé que ceux des appartements intelligents existants.

Pour ce qui est de notre objectif de fiabilité, tous les appareils que nous avions posés en hypothèse comme étant l'appareil avec le DCQ le plus élevé, ont reçu les DCQ les plus élevé de leur scénario. À l'exception de quelques itérations où les résultats ont divergés quelque peu de ceux attendus [4], nos hypothèses ont été vérifiées et les DCQ attribués ont suivi l'approximation que nous avions donnée. Ainsi, comme l'attribution des DCQ a respecté celle qu'aurait donné un expert du domaine suite à l'analyse des contextes, notre objectif quant à la représentation des résultats a donc été atteint.

[4] voir précédemment dans les discussions des résultats des scénarios

Les temps de traitement de nos évaluations sont relativement stables pour tous les scénarios et varient de 0,4 à 0,7 seconde, tout dépendant du nombre de données à traiter et des facteurs externes à Tyche. Il y a peu de variation dans les DCQ lorsque les contextes se ressemblent, par exemple, lorsque l'utilisateur se déplace de quelques centimètres ou bien que la saturation d'un appareil varie de quelques pourcentages. De plus, à chaque répétition d'une itération, l'attribution des DCQ a été la même, ce qui est en soit normal puisque les données contextuelles n'ont pas changé d'une répétition à l'autre. Lors de la mise en place finale de tous les scénarios, aucune faute logicielle n'a eu lieu et l'intergiciel est resté fonctionnel pendant plusieurs jours sans qu'une faute ou une baisse dans la qualité de service ne soit observée. À la lumière de ces observations, nous pouvons donc conclure que l'implémentation actuelle du projet Tyche est stable et jusqu'à un certain point fiable.

Pour ce qui est de notre objectif visant une bonne répartition des résultats, l'objectif a été majoritairement atteint. Lors des premiers scénarios, la répartition des DCQ a respecté nos attentes. Les appareils situés dans des zones non adjacentes à celles visées, ayant peu de ressources ou une grande saturation, ont reçu des DCQ faibles, les appareils dans des contextes moyens ont reçu des DCQ moyens et ainsi de suite. Toutefois, lors de la mise en place des scénarios plus complexes tels que le scénario #4, la quantité de données et les interactions entre les parties du FLORE ont mené à des résultats différents de ceux attendus. Toutefois, fait réconfortant, les résultats imprévus n'était pas complètement opposés à nos hypothèses, mais seulement légèrement en dehors de nos prédictions, de l'ordre de 5 à 10 points de DCQ en moyenne.

De plus, comme nous l'avons écrit précédemment, quelques modifications sont nécessaires au FLORE afin qu'il se conforme à nos attentes pour le scénario #5. Le raisonnement sur les préférences des utilisateurs devrait être étendu afin que des préférences négatives envers des périphériques aient plus de poids dans le calcul du DCQ. Il serait également nécessaire d'augmenter l'importance de la mobilité et du champ de vision des utilisateurs afin que les appareils situés hors de la vue d'un utilisateur soient désavantagés par rapport aux autres appareils.

Concernant le scénario #4, les résultats, tant du côté des temps de traitement que de la précision du raisonnement, permettent d'envisager l'utilisation du projet Tyche comme plateforme d'assistance ponctuelle auprès de l'utilisateur. Ainsi, un utilisateur pourrait être assisté, à un temps donné, par une application qui serait alors déployée sur un appareil l'entourant, adapté à ses caractéristiques, ses capacités et ses préférences. Cette fonctionnalité pourrait, par exemple, être utilisée pour aider des personnes avec déficiences cognitives dans la réalisation de leur tâche quotidienne.

Cependant, la coupe est encore loin des lèvres avant d'assister les utilisateurs dans les espaces intelligents à l'aide des mécanismes du projet Tyche. D'autres fonctionnalités vitales devront être développées et intégrées au projet. Dans le contexte d'une assistance, la fiabilité et la stabilité prennent une tout autre dimension et les critères de fiabilité et de stabilité sont beaucoup plus élevés. Ainsi, des tests et des évaluations étendus devront être effectués afin de vérifier la qualité de service lors de diverses situations d'assistance.

Enfin, nous aimerions dans nos prochains travaux évaluer la convivialité (*usability*) des deux outils de gestion, en procédant à une enquête auprès des utilisateurs potentiels. Quoi que la conception de ces outils se soit fait en respectant les grandes lignes de la convivialité d'interfaces, il reste que des améliorations significatives quant à la conception amèneraient à des gains de productivité, à une amélioration de la facilité d'utilisation et réduiraient la courbe d'apprentissage des gestionnaires/aidants. Comparativement à nos précédentes évaluations, une telle enquête nécessiterait l'aide d'experts en convivialité telle que des ergonomes et les autorisations de comités d'éthique, puisque celle-ci serait conduite avec des utilisateurs réels.

Conclusion

À travers cette thèse de doctorat, nous avons introduit la problématique de la gestion logicielle dans les espaces intelligents, présenté notre vision de l'informatique autonome appliquée aux espaces intelligents, proposé une solution d'auto-organisation des logiciels et présenté les résultats des tests et évaluations effectués sur cette solution. Nous avons pu constater, dans l'introduction à la problématique, que peu de travaux se sont penchés sur l'organisation et la configuration des logiciels dans les environnements appliquant l'informatique diffuse, en tenant compte d'un profil élaboré des utilisateurs et d'une vision multi-niveaux de la sensibilité au contexte.

Pour répondre à cette problématique ainsi qu'aux limites des travaux antérieurs, nous avons proposé dans ce document une solution d'organisation autonome des logiciels parmi les composantes des espaces intelligents, le projet Tyche et le FLORE. Celui-ci met à l'avant-plan la notion d'utilisateur face au contexte des milieux dans la construction des organisations logicielles et dans la gestion des milieux. La solution introduit également l'utilisation d'une approche contextuelle macroscopique et microscopique, l'utilisation d'une métrique, le DCQ, pour la quantification des performances des appareils et propose une architecture dynamique pouvant intégrer les autres notions de gestion autonome et de l'informatique diffuse autonome. Enfin, ce document présente les résultats de tests et d'évaluations conduits sur notre implémentation du projet Tyche, ainsi que des discussions sur les résultats et sur la nature de ceux-ci.

Réalisation des objectifs de développement et d'évaluation

Dans les phases initiales de notre travail de thèse, nous avions fixé des objectifs larges visant à concevoir et implémenter une plateforme de gestion des espaces intelligents qui intègrerait les quatre aspects de l'informatique diffuse autonome : l'auto-configuration/organisation, l'auto-réparation, l'auto-protection et l'auto-optimisation. Force est maintenant d'admettre que ces objectifs étaient trop larges pour un seul travail de thèse et surtout une seule personne. Nous

avions alors décidé de reconsidérer nos objectifs de travail et d'aller plutôt en profondeur sur la problématique de l'organisation logicielle plutôt qu'en largeur (autres *selves*). Cette décision nous a ainsi permis de dégager plusieurs innovations notamment en ce qui a trait à l'implémentation du FLORE. Elle nous permet également de mieux voir nos perspectives de travaux futurs, avec l'interconnexion des autres aspects de l'informatique diffuse autonome.

Quoique nous avons déjà discuté de l'atteinte de nos objectifs de développement et d'évaluation dans les Chapitres 3 et 4, nous concluons ici sur la réalisation de ces objectifs. Dans le cadre du projet Tyche, nous avions fixé des objectifs précis quant aux fonctionnalités implémentées et quant aux façons de les réaliser. Nous avons ainsi proposé une architecture dynamique basée sur la plateforme OSGi permettant une extensibilité aisée du projet. L'intergiciel propose des fonctionnalités d'auto-organisation logicielle reposant sur l'utilisation d'une ontologie OWL, des métriques de performance des appareils, l'inclusion du profil de l'utilisateur via les modalités d'interaction et leurs préférences, et même une amélioration proactive de l'organisation logicielle. Enfin, les outils de gestion permettent à plusieurs types d'utilisateurs de gérer les logiciels des environnements de façon rapide et simple. Nous pouvons donc affirmer que nos objectifs ont été atteints quant à l'implémentation de notre solution d'intergiciel de gestion autonome des espaces intelligents.

Les objectifs posés avant l'évaluation et les tests sur le projet Tyche ont également été majoritairement atteints. Les résultats ont démontré que l'intergiciel et le FLORE proposent des temps de traitements, lors d'organisation, en deçà de nos limites fixées. La précision des raisonnements, la fiabilité des évaluations et la répartition des DCQ correspondent aux résultats auxquels nous nous attendions. Hormis l'évaluation des préférences des utilisateurs, les DCQ donnés par le FLORE correspondent aux hypothèses que nous avions fixées lors des scénarios d'évaluation. Sur ce point, nous pouvons également affirmer que la phase d'évaluations et de tests est un succès.

Contributions scientifiques

Vision de l'informatique autonome appliquée aux espaces intelligents pour personnes dépendantes : La première contribution scientifique de notre travail de thèse consiste en la présentation de notre vision de l'informatique autonome appliquée au contexte des espaces intelligents pour personnes dépendantes [30]. Cette vision propose les fonctionnalités de base quant à la gestion autonome des milieux et base l'utilisation des concepts autonomes sur l'implémentation de l'auto-configuration/organisation.

Représentation et utilisation du contexte basée sur une approche macroscopique et microscopique : L'architecture du projet Tyche et l'utilisation même des informations contextuelles implémentent les travaux initiés au laboratoire DOMUS sur la modélisation et la formalisation du contexte macroscopique et microscopique [12]. Cette modélisation du contexte permet une séparation logique des données et du raisonnement entre les entités des milieux, l'agrégation et l'abstraction des données, un dynamisme dans l'ajout et la mise à jour des informations, etc. Cette approche est prometteuse pour une utilisation dans des environnements intelligents ouverts en coopération avec des systèmes orientés services ou agents/acteurs.

Représentation du contexte à l'aide d'ontologie et de description sémantique OWL : La solution proposée base son fonctionnement sur l'utilisation d'une ontologie, recueillant les informations contextuelles macroscopiques des milieux. L'utilisation du langage OWL permet une standardisation des concepts et propriétés des milieux [1], son utilisation sur plusieurs plateformes hétérogènes et une extensibilité accrue. L'utilisation d'une ontologie permet la création de requêtes sur les données contenues par celles-ci et la création de règles d'inférence à l'aide de la description logique à même les ontologies. L'utilisation du langage OWL semble une piste prometteuse pour une utilisation étendue dans les travaux des laboratoires DOMUS et Handicom.

Proposition d'un engin de raisonnement d'organisation logicielle basé sur les ontologies et la logique floue : Nos travaux de recherche proposent l'utilisation d'un engin de raisonnement sur l'organisation logicielle afin de réduire la complexité de la gestion logicielle des espaces intelligents. Cette complexité est réduite par un transfert des tâches décisionnelles et des connaissances des utilisateurs vers l'intergiciel de gestion et son engin de raisonnement des espaces intelligents. Cet engin repose son mécanisme d'auto-organisation sur une métrique de quantification des performances des systèmes, le DCQ, issue en majeure partie d'ensembles de *defuzzification* (logique floue). L'utilisation de la logique floue dans un processus d'organisation logicielle représente une innovation par rapport aux travaux dans ce domaine.

Utilisation des modalités d'interaction des utilisateurs dans l'organisation logicielle des espaces : Le raisonnement sur l'organisation logicielle des espaces intelligents proposé dans nos travaux donne un rôle important aux profils des utilisateurs dans l'évaluation des DCQ et de l'organisation. Ce raisonnement est basé principalement sur les modalités d'interaction des utilisateurs avec les composantes des milieux : appareils, périphériques et dispositifs d'affichage, et avec le contexte de ceux-ci : positions, ressources, zones, etc.

Implémentation d'un prototype fonctionnel et évaluation de celui-ci : La majeure partie de nos travaux de recherche a été dédiée à la conception et l'implémentation du prototype

d'intergiciel de gestion autonome logicielle pour espaces intelligents. Quoiqu'il s'agit d'un prototype, l'intergiciel développé est plus que fonctionnel et les tests et évaluations effectués lors de nos travaux le démontrent. L'évaluation démontre que l'engin de raisonnement d'organisation par la logique floue, le FLORE, répond aux besoins initiaux d'organisation autonome des espaces intelligents.

Perspectives

Tout au long de cette thèse, nous avons introduit à plusieurs reprises des perspectives de travaux futurs. En tant que prototype, le projet Tyche laisse de nombreuses problématiques ouvertes, offrant plusieurs pistes de travail pour la suite de la recherche.

Évidemment, la première perspective à court terme serait la correction des observations que nous avons discutées lors de l'analyse des résultats des tests et évaluations. Ainsi, les règles de raisonnement seront modifiées afin d'attribuer davantage de poids aux préférences des utilisateurs. La modélisation du raisonnement sur la mobilité des utilisateurs et du calcul des champs de vision devra également être remodélisée afin d'abaisser les DCQ des appareils situés hors du champ de vision des utilisateurs, à des distances moyennes (3 à 4 mètres). De plus, nous allons terminer dans les prochains mois l'architecture étendue du projet Tyche intégrant des sous-espaces, leurs noeuds coordonnateurs et appareils. Si le FLORE peut aisément intégrer les notions de sous-espaces, l'auto-optimisation peut quant à elle rencontrer des problèmes, entre autres dans une perspective de surveillance des ressources des appareils des milieux. Ainsi, en plus de développer la surveillance dynamique des ressources, la conception des mécanismes d'équilibre des charges de calcul devra s'intégrer à l'architecture multiniveaux avec des sous-espaces et différents niveaux de contexte.

Lorsque l'analyse des structures et de la topologie de l'appartement DOMUS sera terminée, il sera possible de développer les mécanismes de détection des champs de vision et des chemins avec obstacles. Cet ajout améliora la justesse des DCQ par rapport au contexte réel des milieux. Avec ces ajouts, le prototype sera considérablement amélioré et pourrait être mis dans une situation d'évaluation plus poussée dans un contexte réel, telle que celui des résidences alternatives du Centre de Réadaptation de l'Estrie.

Assistance spontanée et ponctuelle vers les utilisateurs

L'une des perspectives de travail les plus intéressantes concerne l'utilisation de l'auto-organisation comme système de livraison de services lors d'assistance auprès des utilisateurs. Cette livraison sensible au contexte permettrait de déployer des services sur les appareils entourant les utilisateurs dans des situations, tel qu'un service d'aide à la réalisation de tâches. Toutefois, plusieurs développements seront nécessaires afin d'arriver à une solution offrant de telles fonctionnalités. Les modalités d'interaction utilisées par l'auto-organisation devront être étendues afin d'intégrer entre autres, les capacités et les compétences cognitives des utilisateurs.

Afin de gérer les problèmes de concurrences entre les utilisateurs, les applications et les appareils, lors de la livraison des applications d'assistances, devront implémenter une mécanique afin de gérer les cas où, par exemple, un appareil est en cours d'utilisation par un utilisateur X et que l'assistance détermine que cet appareil est la meilleure cible pour une assistance auprès d'un utilisateur Y. Une telle situation provoquerait des problèmes de vie privée si l'application pour l'utilisateur Y affiche des informations sensibles et vice versa pour l'application de l'utilisateur X. Elle pourrait également nuire à l'interaction de l'utilisateur X ou bien provoquer de la confusion chez les deux utilisateurs. Cette mécanique devrait donc inclure la notion d'activité en cours pour les utilisateurs dans l'organisation logicielle, ainsi qu'une hiérarchie dans le rôle des utilisateurs et dans les applications d'assistance. Pour ce faire, il serait nécessaire d'avoir des mécanismes de reconnaissance d'activités [128] et une définition complète de celles-ci dans les ontologies des environnements.

Avec l'implémentation des mécanismes cohérents d'assistance spontanée et ponctuelle, le projet Tyche proposerait alors des outils très intéressants aux développeurs d'applications d'assistance, aux chercheurs et aux aidants professionnels. Le projet Tyche pourrait être associé à des outils tels que des agendas contextualisés, des systèmes de détection de fautes et d'erreurs dans l'exécution des tâches des utilisateurs. Il pourrait également être associé à des systèmes de sûreté des utilisateurs, afin d'amener dynamiquement et de façon autonome des applications d'assistance auprès des utilisateurs.

Sensibilité au contexte macroscopique et microscopique dans les espaces intelligents

Dans une perspective d'une extension du projet Tyche vers des environnements intelligents désenclavés ouverts, tels que des centres commerciaux, des zones urbaines, et des parcs, le modèle d'acquisition des données et la structure de la sensibilité du contexte devront être éten-

dus. Dans ce cas, l'importance des micro-contextes et de la construction des macro-contextes à partir des micro-contexte des milieux est particulièrement importante, puisqu'il n'y a pas nécessairement de structures fixes et fiables telles qu'on les retrouve dans un appartement intelligent : topologie fixe, peu de modifications dans les entités du milieu, peu d'utilisateurs, réseaux de communication fiables, etc. Des travaux au laboratoire DOMUS sont actuellement en cours dont le but est de proposer un modèle de sensibilité contextuelle pour les espaces intelligents ouverts. Nous désirons continuer à nous impliquer dans la conduite de ces travaux et récupérer les résultats dans une perspective d'assistance dynamique auprès des utilisateurs en mobilité.

De plus, nous travaillons actuellement au développement de solutions mobiles pour la découverte des services des espaces, l'extraction dynamique auprès des services, des interfaces utilisateur permettant d'interagir avec ces services (projet de portail de services pour téléphone mobile, Section 3.7). Les résultats de ces travaux pourront être utilisés afin d'implémenter la découverte des services et des entités des environnements ouverts à partir d'appareils mobiles et permettre la construction du macro-contextes et micro-contextes à partir des données récupérées de ces services et entités.

Intégration complète des fonctionnalités de l'informatique diffuse autonome

Évidemment, nous désirons dans nos prochains travaux étendre les fonctionnalités du projet Tyche aux autres aspects de l'informatique diffuse autonome : l'auto-réparation, l'auto-protection et l'auto-optimisation. Ces fonctionnalités s'intègreront à l'auto-organisation développée dans le cadre de ce travail de thèse et permettront aux espaces intelligents de récupérer de façon autonome d'erreurs logicielles et matérielles, de mieux protéger les entités des milieux, utilisateurs compris, contre plusieurs formes de dangers physiques ou virtuels et d'améliorer constamment les performances des systèmes des milieux. Nous avons déjà identifié plusieurs lignes directrices quant à l'implémentation des fonctionnalités, entre autres sur le rôle central de l'auto-organisation/configuration. Le raisonnement par cas [129], déjà mentionné dans les premiers travaux sur l'informatique autonome, semble également une solution à la mise en place de mesures face à des évènements observés dans les milieux. Cette solution pourrait être étendue avec des raisonnements par cas adaptatif, implémentant une forme d'auto-apprentissage avec l'amélioration constante de la banque de cas et de mesures d'intervention.

Ultimement, nous aimerions que cette solution complète de gestion logicielle autonome

pour espaces intelligents ait une fiabilité et une souplesse suffisante qui lui permettrait de passer du concept de prototype à une version fonctionnelle pouvant être déployée dans des habitats réels et non dans des environnements dédiés à la recherche pure tels que l'appartement DOMUS. Le principe de gestion logicielle énoncé dans nos travaux devrait être étendu à la gestion autonome du matériel et des utilisateurs afin d'avoir un impact fort sur la simplification de la gestion et de l'utilisation des espaces intelligents et de l'informatique diffuse en général. Une telle solution faciliterait la mise en place et la maintenance de tels environnements, réduirait de façon significative les coûts reliés à ces milieux et réduirait le temps imparti aux tâches de gestion. Enfin, une solution complète et viable à plusieurs types d'environnement, amènerait, selon nous, à une démocratisation des technologies d'intelligence ambiante. Dans tous les cas, beaucoup de travaux de standardisation du matériel, des périphériques et des systèmes informatisés chez les producteurs de matériels électroniques et de logiciels seront nécessaires. À quand la vision de Weiser appliquée à nos milieux de vie quotidiens ? Seul l'avenir et pourquoi pas, Tyche, le diront.

Annexe A

Extrait de la description sémantique OWL des espaces intelligents

```
<rdf RDF xmlns="http //www domus usherbrooke ca/Environment owl#"
     xml base="http //www domus usherbrooke ca/Environment owl"
     xmlns Environment="http //www domus usherbrooke ca/Environment owl#"
     xmlns rdfs="http //www w3 org/2000/01/rdf-schema#"
     xmlns swrl="http //www w3 org/2003/11/swrl#"
     xmlns protege="http //protege stanford edu/plugins/owl/protege#"
     xmlns owl2xml="http //www w3 org/2006/12/owl2-xml#"
     xmlns xsp="http //www owl-ontologies com/2005/08/07/xsp owl#"
     xmlns owl="http //www w3 org/2002/07/owl#"
     xmlns xsd="http //www w3 org/2001/XMLSchema#"
     xmlns swrlb="http //www w3 org/2003/11/swrlb#"
     xmlns rdf="http //www w3 org/1999/02/22-rdf-syntax-ns#">
    <owl Ontology rdf about=""/>

    <owl Class rdf about="#NonDedicatedDevice">
        <rdfs subClassOf rdf resource="#ElectricalDevice"/>
    </owl Class>

    <owl Class rdf about='#DedicatedDevice">
        <rdfs subClassOf rdf resource="#ElectricalDevice"/>
    </owl Class>

    <owl Class rdf about="#DisplayDevice">
        <rdfs subClassOf rdf resource="#DedicatedDevice"/>
    </owl Class>

    <owl Class rdf about="#PointingDevice">
        <rdfs subClassOf rdf resource="#DedicatedDevice"/>
    </owl Class>

    <Zone rdf about="#Cuisine">
```

```
    <rdf type rdf resource="&owl,Thing"/>
    <hasName rdf datatype="&xsd,string">Cuisine</hasName>
    <hasRoomLimitPosition rdf resource="#CuisineP1"/>
    <hasRoomLimitPosition rdf resource="#CuisineP2"/>
    <hasRoomLimitPosition rdf resource="#CuisineP3"/>
    <hasRoomLimitPosition rdf resource='#CuisineP4"/>
    <hasRoomLimitPosition rdf resource="#CuisineP5"/>
    <hasRoomLimitPosition rdf resource="#CuisineP6"/>
    <parent rdf resource="#DomusAppt"/>
</Zone>

<RoomLimitPosition rdf about="#CuisineP1">
    <rdf type rdf resource="&owl,Thing"/>
    <hasRoomLimitPositionOrder rdf datatype='&xsd,int">1</hasRoomLimitPositionOrder>
    <hasCoordinateX rdf datatype="&xsd,float">2 85</hasCoordinateX>
    <hasCoordinateZ rdf datatype='&xsd,float">7 00</hasCoordinateZ>
</RoomLimitPosition>

<NonDedicatedDevice rdf about="#Cuisine1">
    <rdf type rdf resource="&owl,Thing"/>
    <hasCoordinateY rdf datatype= &xsd,float">
            0 0</hasCoordinateY>
    <hasPermanentMemory rdf datatype="&xsd,float">
            50 0</hasPermanentMemory>
    <hasCoordinateX rdf datatype="&xsd,float">
            5 5</hasCoordinateX>
    <hasCoordinateZ rdf datatype="&xsd,float">
            8 5</hasCoordinateZ>
    <hasRAM rdf datatype="&xsd,float">
            50 0</hasRAM>
    <hasCPU rdf datatype="&xsd,float'>
            50 0</hasCPU>
    <hasPermanentMemorySaturation rdf datatype="&xsd,float">
            0 0</hasPermanentMemorySaturation>
    <hasCPUSaturation rdf datatype="&xsd,float">
            0 0</hasCPUSaturation>
    <hasRAMSaturation rdf datatype="&xsd,float">
            0 0</hasRAMSaturation>
    <hasDescription rdf datatype="&xsd,string"
        >Cuisine1</hasDescription>
    <hasName rdf datatype="&xsd,string">Cuisine1</hasName>
    <hasStatus rdf datatype="&xsd,string">Started</hasStatus>
    <hasUUID rdf datatype="&xsd,float"
        >http //www domus usherbrooke ca/Environment owl#Cuisine1
            </hasUUID>
    <hasServiceAddress rdf datatype="&xsd,string">
            10 44 164 36 9090/ca usherbrooke domus tyche device DeviaceManagerService
            </hasServiceAddress>
    <hasDedicatedDevice rdf resource="#Cuisine1Display"/>
    <hasDedicatedDevice rdf resource="#Cuisine1TactileScreen"/>
    <inTo rdf resource="#Cuisine"/>
</NonDedicatedDevice>
```

```
<DisplayDevice rdf about="#Cuisine1Display">
    <hasHeight rdf datatype="&xsd,float">0 30</hasHeight>
    <hasWidth rdf datatype="&xsd,float">0 30</hasWidth>
    <hasResolutionHeight rdf datatype="&xsd,int">
            1080</hasResolutionHeight>
    <hasDisplayAngle rdf datatype="&xsd,float">
            170 0</hasDisplayAngle>
    <hasRotationY rdf datatype="&xsd,float">
            135 0</hasRotationY>
    <hasResolutionWidth rdf datatype="&xsd,int">
            1920</hasResolutionWidth>
    <hasCoordinateX rdf datatype="&xsd,float">
            5 5</hasCoordinateX>
    <hasCoordinateZ rdf datatype="&xsd,float'>
            8 5</hasCoordinateZ>
    <hasName rdf datatype="&xsd,string">
            Cuisine1Display</hasName>
    <inTo rdf resource="#Cuisine"/>
    <hasType rdf datatype="&xsd,string">Display</hasType>
</DisplayDevice>

<PointingDevice rdf about="#Cuisine1TactileScreen'>
    <hasCoordinateY rdf datatype="&xsd,float">
            0 0</hasCoordinateY>
    <needHandWorkspace rdf datatype="&xsd,float">
            0 0</needHandWorkspace>
    <needHandForce rdf datatype="&xsd,float">
            10 0</needHandForce>
    <hasCoordinateX rdf datatype="&xsd,float">
            5 5</hasCoordinateX>
    <hasCoordinateZ rdf datatype="&xsd,float">
            8 5</hasCoordinateZ>
    <hasName rdf datatype="&xsd,string'>
            Cuisine1TactileScreen</hasName>
    <inTo rdf resource="#Cuisine"/>
            <hasType rdf datatype="&xsd,string">
                    TactileScreen</hasType>
</PointingDevice>

<owl Thing rdf about='#User1">
    <rdf type rdf resource="#User"/>
    <hasRotationY rdf datatype="&xsd,float">0 0</hasRotationY>
    <hasHandWorkspace rdf datatype="&xsd,float">0 2</hasHandWorkspace>
    <hasArmWorkspace rdf datatype="&xsd,float">1 0</hasArmWorkspace>
    <hasCoordinateZ rdf datatype="&xsd,float">2 0</hasCoordinateZ>
    <hasCoordinateX rdf datatype="&xsd,float">2 0</hasCoordinateX>
    <hasHandForce rdf datatype="&xsd,float">2 0</hasHandForce>
    <hasMovingCapacity rdf datatype="&xsd,float">2 0</hasMovingCapacity>
    <hasVisualAcuity rdf datatype="&xsd,int">20</hasVisualAcuity>
    <hasLeftVisionAngle rdf datatype="&xsd,float">80 0</hasLeftVisionAngle>
    <hasRightVisionAngle rdf datatype="&xsd,float">80 0</hasRightVisionAngle>
```

```
        <hasName rdf:datatype="&xsd;string">User1</hasName>
        <hasPreference rdf:resource="#PreferenceUser1Keyboard"/>
        <inTo rdf:resource="#Salon"/>
    </owl:Thing>

    <owl:Thing rdf:about="#PreferenceUser1Keyboard">
        <rdf:type rdf:resource="#Preference"/>
        <hasPreferenceValue rdf:datatype="&xsd;int">1</hasPreferenceValue>
        <hasDedicatedDevice rdf:resource="#Keyboard1"/>
    </owl:Thing>

</rdf:RDF>
```

Annexe B

Extrait du contrôleur de logique floue du FLORE partie coordonnateur

```
FUNCTION_BLOCK DCQUser  // Block definition (there may be more than one block
per file)

VAR_INPUT                                   // Define input variables
        visionAngle    REAL,
        visualAcuity    REAL,
        userMobility    REAL,
        time    REAL,
        preference    REAL,
END_VAR

VAR_OUTPUT                                  // Define output variable
        peripheralOptimality    REAL,
END_VAR

//---------------------------------------------
// Fuzzyfication et de-fuzzy
//---------------------------------------------

FUZZIFY visionAngle
        TERM behind  = 0 0,
        TERM inFront  = 1 0,
END_FUZZIFY

FUZZIFY preference
        TERM dislike  = gauss -1 0  0 25,
        TERM neutral  = gauss 0 0  0 25,
        TERM like     = gauss 1 0  0 25,
END_FUZZIFY

FUZZIFY visualAcuity
```

Annexe B : Extrait du contrôleur de logique floue du FLORE partie coordonnateur

```
        TERM impossible  = gauss 0 0  0 08,
        TERM hard  = gauss 0 33  0 08,
        TERM borderline  = gauss 0 66  0 08,
        TERM optimal  = gauss 1 0  0 08,
END_FUZZIFY

FUZZIFY userMobility
        TERM impossible  = gauss 0  0 22,
        TERM slow  = gauss 0 8  0 18,
        TERM normal  = gauss 1 43  0 1,
        TERM fast  = gauss 2 2  0 285,
END_FUZZIFY

FUZZIFY time
        TERM verySlow  = gauss 30 0  2 0,
        TERM slow  = gauss 20 0  2 0,
        TERM fast  = gauss 10 0  2 0,
        TERM instant  = gauss 0 0  2 0,
END_FUZZIFY

DEFUZZIFY peripheralOptimality
        TERM impossible  = gauss 0 0  6 0,
        TERM notOptimal  = gauss 33 33  6 0,
        TERM subOptimal  = gauss 66 66  6 0,
        TERM optimal  = gauss 100 0  6 0,

        METHOD   COG,           // Use 'Center Of Gravity' defuzzification method
        DEFAULT  = 0,           // Default value is 0 (if no rule activates defuzzifier)
        RANGE  = (0     100),
END_DEFUZZIFY

RULEBLOCK No1
        AND   MIN,                      // Use 'min' for 'and' (also implicit use 'max' for '
            or' to fulfill DeMorgan's Law)
        ACT   MIN,                      // Use 'min' activation method
        ACCU   MAX,                     // Use 'max' accumulation method


RULE 1 0  IF visionAngle IS inFront AND visualAcuity IS optimal AND userMobility IS fast AND
    time IS instant AND preference IS like THEN peripheralOptimality IS optimal WITH 1,

RULE 1 1  IF visionAngle IS inFront AND visualAcuity IS optimal AND userMobility IS fast AND
    time IS instant AND preference IS neutral THEN peripheralOptimality IS optimal,

RULE 1 2  IF visionAngle IS inFront AND visualAcuity IS optimal AND userMobility IS fast AND
    time IS instant AND preference IS dislike THEN peripheralOptimality IS notOptimal,

RULE 2 0  IF visionAngle IS inFront AND visualAcuity IS optimal AND (userMobility IS fast OR
    userMobility IS normal) AND time IS fast AND (preference IS like OR preference IS neutral
    ) THEN peripheralOptimality IS optimal,

RULE 2 1  IF visionAngle IS inFront AND visualAcuity IS optimal AND (userMobility IS fast OR
```

```
    userMobility IS normal) AND time IS fast AND preference IS dislike THEN
    peripheralOptimality IS notOptimal,

RULE 3 0  IF visionAngle IS inFront AND visualAcuity IS optimal AND (userMobility IS NOT fast)
     AND time IS instant AND (preference IS like OR preference IS neutral) THEN
    peripheralOptimality IS optimal,

RULE 3 1  IF visionAngle IS inFront AND visualAcuity IS optimal AND (userMobility IS NOT fast)
     AND time IS instant AND (preference IS like OR preference IS neutral) THEN
    peripheralOptimality IS notOptimal

RULE 4 0  IF visionAngle IS inFront AND visualAcuity IS optimal AND (userMobility IS fast OR
    userMobility IS normal) AND (time IS slow OR time IS verySlow) AND (preference IS like OR
     preference IS neutral) THEN peripheralOptimality IS subOptimal,

RULE 4 1  IF visionAngle IS inFront AND visualAcuity IS optimal AND (userMobility IS fast OR
    userMobility IS normal) AND (time IS slow OR time IS verySlow) AND preference IS dislike
    THEN peripheralOptimality IS notOptimal,

RULE 5 0  IF visionAngle IS inFront AND visualAcuity IS borderline AND userMobility IS
    impossible AND time IS instant AND (preference IS like OR preference IS neutral) THEN
    peripheralOptimality IS subOptimal,

RULE 5 1  IF visionAngle IS inFront AND visualAcuity IS borderline AND userMobility IS
    impossible AND time IS instant AND preference IS dislike THEN peripheralOptimality IS
    notOptimal,

RULE 6 0  IF visionAngle IS inFront AND visualAcuity IS optimal AND userMobility IS slow AND
    time IS fast AND (preference IS like OR preference IS neutral) THEN peripheralOptimality
    IS subOptimal,

RULE 6 1  IF visionAngle IS inFront AND visualAcuity IS optimal AND userMobility IS slow AND
    time IS fast AND preference IS dislike THEN peripheralOptimality IS notOptimal,

RULE 7  IF visionAngle IS inFront AND visualAcuity IS borderline AND (userMobility IS fast OR
    userMobility IS normal) AND (preference IS like OR preference IS neutral) THEN
    peripheralOptimality IS subOptimal,

RULE 8  IF visualAcuity IS borderline AND userMobility IS slow AND (time IS instant OR time IS
     fast) AND (preference IS like OR preference IS neutral) THEN peripheralOptimality IS
    subOptimal

RULE 9  IF visualAcuity IS hard AND (userMobility IS fast OR userMobility IS normal) AND (time
     IS instant OR time IS fast) AND (preference IS like OR preference IS neutral) THEN
    peripheralOptimality IS subOptimal,

RULE 10 0  IF visionAngle IS behind AND visualAcuity IS optimal AND (userMobility IS fast OR
    userMobility IS normal) AND (preference IS like OR preference IS neutral) THEN
    peripheralOptimality IS subOptimal,

RULE 10 1  IF visionAngle IS behind AND visualAcuity IS optimal AND (userMobility IS fast OR
    userMobility IS normal) AND preference IS dislike THEN peripheralOptimality IS notOptimal
```

Annexe B : Extrait du contrôleur de logique floue du FLORE partie coordonnateur

,

RULE 11 0 IF visionAngle IS behind AND visualAcuity IS optimal AND userMobility IS slow AND (time IS instant OR time IS fast) AND (preference IS like OR preference IS neutral) THEN peripheralOptimality IS subOptimal,

RULE 11 1 IF visionAngle IS behind AND visualAcuity IS optimal AND userMobility IS slow AND (time IS instant OR time IS fast) AND preference IS dislike THEN peripheralOptimality IS notOptimal,

RULE 12 0 IF visionAngle IS behind AND visualAcuity IS optimal AND userMobility IS impossible AND time IS instant AND (preference IS like OR preference IS neutral) THEN peripheralOptimality IS subOptimal,

RULE 12 1 IF visionAngle IS behind AND visualAcuity IS optimal AND userMobility IS impossible AND time IS instant AND preference IS dislike THEN peripheralOptimality IS notOptimal,

RULE 13 0 IF visionAngle IS behind AND visualAcuity IS borderline AND userMobility IS fast AND (preference IS like OR preference IS neutral) THEN peripheralOptimality IS subOptimal

,

RULE 13 1 IF visionAngle IS behind AND visualAcuity IS borderline AND userMobility IS fast AND preference IS dislike THEN peripheralOptimality IS notOptimal,

RULE 14 0 IF visionAngle IS behind AND visualAcuity IS borderline AND userMobility IS normal AND (time IS instant OR time IS fast) AND (preference IS like OR preference IS neutral) THEN peripheralOptimality IS subOptimal,

RULE 14 1 IF visionAngle IS behind AND visualAcuity IS borderline AND userMobility IS normal AND (time IS instant OR time IS fast) AND preference IS dislike THEN peripheralOptimality IS notOptimal,

RULE 15 IF visionAngle IS inFront AND (visualAcuity IS optimal OR visualAcuity IS borderline) AND userMobility IS impossible AND (time IS NOT instant) THEN peripheralOptimality IS notOptimal,

RULE 16 IF (visualAcuity IS optimal OR visualAcuity IS borderline) AND userMobility IS slow AND (time IS slow OR time IS verySlow) THEN peripheralOptimality IS notOptimal,

RULE 17 IF visualAcuity IS hard AND (userMobility IS fast OR userMobility IS normal) AND (time IS slow OR time IS verySlow) THEN peripheralOptimality IS notOptimal,

RULE 18 IF visualAcuity IS hard AND userMobility IS slow THEN peripheralOptimality IS notOptimal,

RULE 19 IF visionAngle IS inFront AND visualAcuity IS impossible AND userMobility IS slow THEN peripheralOptimality IS notOptimal,

RULE 20 IF visualAcuity IS impossible AND (userMobility IS fast OR userMobility IS normal) THEN peripheralOptimality IS notOptimal,

RULE 21 IF visionAngle IS behind AND visualAcuity IS optimal AND userMobility IS impossible

```
    AND (time IS NOT instant) THEN peripheralOptimality IS notOptimal,

RULE 22 IF visionAngle IS behind AND visualAcuity IS borderline AND userMobility IS normal
    AND (time IS slow OR time IS verySlow) THEN peripheralOptimality IS notOptimal,

RULE 23 IF visionAngle IS behind AND visualAcuity IS borderline AND userMobility IS
    impossible THEN peripheralOptimality IS notOptimal,

RULE 24 IF visualAcuity IS hard AND userMobility IS impossible AND (time IS instant OR time
    IS fast) THEN peripheralOptimality IS notOptimal,

RULE 25 IF visualAcuity IS hard AND userMobility IS impossible AND (time IS slow OR time IS
    verySlow) THEN peripheralOptimality IS impossible,

RULE 26 IF visionAngle IS inFront AND visualAcuity IS impossible AND userMobility IS
    impossible THEN peripheralOptimality IS impossible,

RULE 27 IF visionAngle IS behind AND visualAcuity IS impossible AND userMobility IS slow THEN
     peripheralOptimality IS impossible,

RULE 28 IF visionAngle IS behind AND visualAcuity IS impossible AND userMobility IS
    impossible AND (time IS NOT verySlow) THEN peripheralOptimality IS impossible,

RULE 29 IF visionAngle IS behind AND visualAcuity IS impossible AND userMobility IS
    impossible AND time IS verySlow THEN peripheralOptimality IS impossible WITH 0,

END_RULEBLOCK

END_FUNCTION_BLOCK
```

Bibliographie

[1] B. Abdulrazak, B. Chikhaoui, C. Gouin-Vallerand, et B. Fraikin. A Standard Ontology for Smart Spaces. *International Journal of Web and Grid Services* **6**(3), 244–268 (2010).

[2] Mark Weiser. The computer for the 21st century. *Scientific American* **265**(3), 66–75 Sept. (1991).

[3] Emile Aarts. Ambient Intelligence : A Multimedia Perspective. *IEEE Multimedia* **11**(1), 12–19 (2004).

[4] S. Giroux, J. Bauchet, H. Pigot, D. Lussier-Desrochers, et Y. Lachappelle. Pervasive behavior tracking for cognitive assistance. *International Conference on Pervasive Technologies Related to Assistive Environments, PETRA 2008* , 86 :1–86 :7 July (2008).

[5] M. Mokhtari, M. Ghorbel, et R. Kadouche. Mobilité et services : Application aux aides technologiques pour les personnes handicapées. *International Symposium On Programming and Systems, ISPS 2005* (2005).

[6] Mark Weiser. Some computer science issues in ubiquitous computing. *Communications of ACM* **36**(7), 75–84 (1993).

[7] Mark Weiser. Turning pervasive computing into mediated spaces. *IBM System Journal* **38**(4), 677–692 (1999).

[8] V. Stanford. Wearable computing goes live in industry. *Pervasive Computing, IEEE* **1**(4), 14–19 (2002).

[9] Stan Kurkovsky. Using Principles of Pervasive Computing to Design M-Commerce Applications. In *ITCC '05 : Proceedings of the International Conference on Information Technology : Coding and Computing (ITCC'05) - Volume II*, 59–64 (IEEE Computer Society, Washington, DC, USA, 2005).

[10] Anind K. Dey. *Providing Architectural Support for Building Context-Aware Applications*. Th de Doctorat, Georgia Institute of Technology, (2000).

[11] N. Ryan, J. Pascoe, et D. Morse. *Computer Applications in Archaeology*, chapter Enhanced Reality Fieldwork : the Context-Aware Archaeological Assistant. (1997).

[12] B. Abdulrazak, P. Roy, C. Gouin-Vallerand, S. Giroux, et Y. Belala. Macro and Micro Context-Awareness for Autonomic Pervasive Computing. In *Proceeding of 12th International Conference on Information Integration and Web-based Applications Services, iiWAS 2010*, Nov. (2010).

[13] L. Rudolph. Project oxygen : Pervasive, human-centric computing - an initial experience. *Proceedings of the 13th International Conference on Advanced Information Systems Engineering, CAiSE 2001* , 1–12 (2001).

[14] S.S. Intille. The goal : smart people, not smart homes. *Proceedings of the International Conference on Smart Homes and Health Telematics, ICOST 2006* , 3–6 (2006).

[15] Cory D. Kidd, Robert Orr, Gregory D. Abowd, Christopher G. Atkeson, Irfan A. Essa, Blair MacIntyre, Elizabeth D. Mynatt, Thad Starner, et Wendy Newstetter. The Aware Home : A Living Laboratory for Ubiquitous Computing Research. , 191–198 (1999).

[16] Anind K. Dey, Gregory D. Abowd, et Daniel Salber. A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *Hum.-Comput. Interact.* **16**, 97–166 December (2001).

[17] Sumi Helal, William Mann, Hicham El-Zabadani, Jeffrey King, Youssef Kaddoura, et Erwin Jansen. The Gator Tech Smart House : A Programmable Pervasive Space. *IEEE Computer* **38**(3), 50–60 (2005).

[18] Mohan Kumar, Behrooz A. Shirazi, Sajal K. Das, Byung Y. Sung, David Levine, et Mukesh Singhal. PICO : A Middleware Framework for Pervasive Computing. *IEEE Pervasive Computing* **2**(3), 72–79 (2003).

[19] Z.Z. Bien, Kwang-Hyun Park, Jin-Woo Jung, et Jun-Hyeong Do. Intention reading is essential in human-friendly interfaces for the elderly and the handicapped. *Industrial Electronics, IEEE Transactions on* **52**(6), 1500–1505 Dec. (2005).

[20] Barry Brumitt, Brian Meyers, John Krumm, Amanda Kern, et Steven Shafer. EasyLiving : Technologies for Intelligent Environments. **1927**, 97–119 (2000).

[21] Candace York. IBM's Advanced PvC Technology Laboratory. Technical report, IBM Research. http ://www.ibm.com/developerworks/wireless/library/wi-pvc/.

[22] J. Hightower, A. LaMarca, et I.E. Smith. Practical Lessons from Place Lab. *IEEE Pervasive Computing* **5**(3), 32–39 (2006).

[23] S. Giroux, H. Pigot, et Mayers A. Indoors pervasive computing and outdoors mobile computing for assisted cognition and telemonitoring. *Computers Helping People with Special Needs* **31**, 953–960 (2004).

[24] J. Bauchet, H. Pigot, et S. Giroux. Cognitive assistance based on ubiquitous computing, mobile computing and cognitive modeling. *International Conference on Aging, Disability and Independence, ICADI 2006* , 228 (2006).

[25] Sylvain Giroux Nadjia Kara Pierre-Yves Groussard, Helene Pigot. User Needs Identification for a Smart Cognitive Assistant based on Participatory Design. In *Proceeding of the 1st International Conference on Ambient Systems, Networks and Technologies*, (2010).

[26] M. Ghorbel, Mounir Mokhtari, et Stéphane Renouard. A Distributed Approach for Assistive Service Provision in Pervasive Environment. In *Proceedings of the 4th international workshop on Wireless mobile applications and services on WLAN hotspots WMASH '06*, 91–100, (2006).

[27] Vanish Talwar, Dejan Milojicic, Qinyi Wu, Calton Pu, Wenchang Yan, et Gueyoung Jung. Approaches for Service Deployment. *IEEE Internet Computing* **9**(2), 70–80 (2005).

[28] J. O. Kephart et D. M. Chess. The vision of autonomic computing. *IEEE Computer* **36**(1), 41–50 (Jan 2003).

[29] C. Gouin-Vallerand, S. Giroux, B. Abdulrazak, et M. Mokhtari. Toward a self-configuration middleware for smart spaces. *International Symposium on Smart Home, SH 2008* , 463–468 Dec. (2008).

[30] Charles Gouin-Vallerand, Bessam Abdulrazak, Sylvain Giroux, et Mounir Mokhtari. Toward autonomic pervasive computing. , 673–676 (2008).

[31] C. Gouin-Vallerand, B. Abdulrazak, S. Giroux, et M. Mokhtari. A self-configuration middleware for smart spaces. *International Journal of Smart Home* **3**(1), 7–16 Jan. (2009).

[32] Wolfgang Trumler, Jan Petzold, Faruk Bagci, et Theo Ungerer. AMUN — Autonomic Middleware for Ubiquitous eNvironments Applied to the Smart Doorplate Project. In *Proceedings of the International Conference on Autonomic Computing*, 274–275, (2004).

[33] A. Ranganathan et R.H. Campbell. Autonomic pervasive computing based on planning. *International Conference on Autonomic Computing* , 80–87 May (2004).

[34] Sumi Helal. Programming Pervasive Spaces. *IEEE Pervasive Computing* **4**(1), 84–87 (2005).

[35] Belkacem Chikhaoui, Yazid Benazzouz, et Bessam Abdulrazak. Towards a universal ontology for smart environments. In *Proceedings of the 11th International Conference on Information Integration and Web-based Applications & Services*, iiWAS '09, 80–87 (ACM, New York, NY, USA, 2009).

[36] C. Gouin-Vallerand, B. Abdulrazak, S. Giroux, et M. Mokhtari. A Software Self-Organizing Middleware for Smart Spaces Based on Fuzzy Logic. In *12th IEEE International Conference on High Performance Computing and Communications (HPCC)*, 138 –145, sept. (2010).

[37] C. Gouin-Vallerand, S. Giroux, B. Abdulrazak, et M. Mokhtari. Tyche project : A context aware middleware for software self-organization in ubiquitous environments. In *8th IEEE International Conference on Ubiquitous Intelligence and Computing (UIC)*, (2011). accepté.

[38] Mahmoud Ghorbel. *Abstract Service Model for Adaptive Provision in Ambient Assistive Living*. Th de Doctorat, Institut National des Télécommunications et Université Pierre et Marie Curie - Paris 6, (2008).

[39] W3C Consortium. OWL Web Ontology Language Overview. http ://www.w3.org/TR/owl-features/, (2004).

[40] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, et Peter F. Patel-Schneider, editors. *The Description Logic Handbook : Theory, Implementation, and Applications*. Cambridge University Press, (2003).

[41] Timothy J. Ross. *Fuzzy Logic with Engineering Applications*. Wiley, Inc., (2004).

[42] OSGi Alliance. OSGi Service Platform Core Specification - Release 4. Technical report, OSGi Alliance, (2005).

[43] World Wide Web Consortium. Web Service Management : Service Life Cycle. (2004).

[44] H. Fayol. *Administration industrielle et générale*. Stratégies et Management. Dunod, (1916).

[45] A. M'hamed. *The Engineering Handbook of Smart Technology for Aging, Disability and Independence*, chapter Safety, Security, Privacy and Trust Issues, 619–630. Computer Engineering Series. John Wiley & Sons (2008).

[46] J. L. Ruiz, J. C. Duenas, F. Usero, et C. Díaz. Deployment in dynamic environments. *Conférence Francophone Sur Le Déploiement Et La (Re) Configuration De Logiciels* , 85–98 (2004).

[47] P. Deutsch. Eight Fallacies of distributed computing. http ://michael.toren.net/ mirrors/eight-fallacies-of-distributed-computing/, (2004).

[48] B. Randell. Facing up to Faults (Turing Memorial Lecture). *Computer Journal* **43**(2), 95–106 (2000).

[49] Shiva Chetan, Anand Ranganathan, et R. Campbell. Towards fault tolerance pervasive computing. *Technology and Society Magazine, IEEE* **24**(1), 38–44 Spring (2005).

[50] Jim Gray. Notes on Data Base Operating Systems. In *Operating Systems, An Advanced Course*, 393–481 (Springer-Verlag, London, UK, 1978).

[51] *An introduction to multi-agent systems*. Wiley, (2009).

[52] C. Becker, M. Handte, G. Schiele, et K. Rothermel. PCOM - a component system for pervasive computing. In *Pervasive Computing and Communications, 2004. PerCom 2004. Proceedings of the Second IEEE Annual Conference on*, 67 – 76, march (2004).

[53] C. Becker, G. Schiele, H. Gubbels, et K. Rothermel. BASE - a micro-broker-based middleware for pervasive computing. In *Pervasive Computing and Communications, 2003. (PerCom 2003). Proceedings of the First IEEE International Conference on*, 443 – 451, march (2003).

[54] J. M. Paluska J. Waterman C. Terman U. Saif, H. Pham et S. Ward. A case for goal-oriented programming semantics. In *In Workshop on System Support for Ubiquitous Computing (UbiSys'03), 5th International Conference on Ubiquitous Computing (UbiComp 2003)*, October 12 (2003).

[55] Steve Shafer, John Krumm, Barry Brumitt, Brian Meyers, Mary Czerwinski, et Daniel Robbins. The new EasyLiving Project at Microsoft Research. In *Proc. Joint DARPA/-NIST Smart Spaces Workshop*, 30–31, (1998).

[56] Kyu Cheol Cho, Chang Hyeon Noh, et Jong Sik Lee. Ontology-Based Intelligent Agent for Grid Resource Management. In *Proceedings of the 1st International Conference on Computational Collective Intelligence. Semantic Web, Social Networks and Multiagent Systems*, ICCCI '09, 553–564 (Springer-Verlag, Berlin, Heidelberg, 2009).

[57] A. Ranganathan, C. Shankar, et R.H. Campbell. Application polymorphism for autonomic ubiquitous computing. *Multiagent Grid Syst.* **1**(2), 109–129 (2005).

[58] Rachid Kadouche, Bessam Abdulrazak, Mounir Mokhtari, Sylvain Giroux, et Helene Pigot. A semantic approach for accessible services delivery in a smart environment. *Int. J. Web Grid Serv.* **5**, 192–218 August (2009).

[59] D. Blackman. Debian Package Management, Part 1 : A User's Guide. *Linux Journal* , 1–12 (2000).

[60] R. Ferri. The OSCAR revolution. *Linux Journal* **2002**(98), 5 (2002).

[61] J. Piesing. The DVB Multimedia Home Platform (MHP) and Related Specifications. *Proceedings of the IEEE* **94**(1), 237–247 Jan. (2006).

[62] J.O. Kephart. Research challenges of autonomic computing. *Software Engineering, 2005. ICSE 2005. Proceedings. 27th International Conference on* , 15–22 May (2005).

[63] Sherif A. Gurguis et Amir Zeid. Towards autonomic web services : achieving self-healing using web services. In *DEAS '05 : Proceedings of the 2005 workshop on Design and evolution of autonomic application software*, 1–5 (ACM, New York, NY, USA, 2005).

[64] James Z. Wang et Matti A. Vanninen. A Novel Self-Configuration Mechanism for Heterogeneous P2P Networks. In *IAT '04 : Proceedings of the Intelligent Agent Technology, IEEE/WIC/ACM International Conference*, 281–287 (IEEE Computer Society, Washington, DC, USA, 2004).

[65] Rajarshi Das, Jeffrey O. Kephart, Charles Lefurgy, Gerald Tesauro, David W. Levine, et Hoi Chan. Autonomic multi-agent management of power and performance in data centers. In *AAMAS '08 : Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems*, 107–114 (International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, 2008).

[66] David M. Chess. Security in autonomic computing. *SIGARCH Comput. Archit. News* **33**(1), 2–5 (2005).

[67] Jeffrey O. Kephart et Rajarshi Das. Achieving Self-Management via Utility Functions. *IEEE Internet Computing* **11**(1), 40–48 (2007).

[68] Omer F. Rana et Jeffrey O. Kephart. Building Effective Multivendor Autonomic Computing Systems. *IEEE Distributed Systems Online* **7**(9), 3 (2006).

[69] Gang Huang, Tiancheng Liu, Hong Mei, Zizhan Zheng, Zhao Liu, et Gang Fan. Towards autonomic computing middleware via reflection. 135–140 vol.1, Sept. (2004).

[70] Weishan Zhang et K.M. Hansen. Semantic Web Based Self-Management for a Pervasive Service Middleware. In *Self-Adaptive and Self-Organizing Systems, 2008. SASO '08. Second IEEE International Conference on*, 245–254, Oct. (2008).

[71] A.R. Haydarlou, M.A. Oey, B.J. Overeinder, et F.M.T. Brazier. Using Semantic Web Technology for Self-Management of Distributed Object-Oriented Systems. In *Web Intelligence, 2006. WI 2006. IEEE/WIC/ACM International Conference on*, 489–493, Dec. (2006).

[72] Wolfgang Trumler, Robert Klaus, et Theo Ungerer. Self-configuration Via Cooperative Social Behavior. In *Third International Conference on Autonomic and Trusted Computing 2006, ATC*, volume 4158 of *Lecture Notes in Computer Science*, 90–99. Springer, (2006).

[73] Wolfgang Trumler, Andreas Pietzowski, Benjamin Satzger, et Theo Ungerer. Adaptive Self-optimization in Distributed Dynamic Environments. In *First International Conference on Self-Adaptive and Self-Organizing Systems, SASO 2007*, 320–323 (IEEE Computer Society, Washington, DC, USA, 2007).

[74] A.A. Syed, J. Lukkien, et R. Frunza. An architecture for self-organization in pervasive systems. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2010*, 1548 –1553, (2010).

[75] Martin Feeney et Richard Frisby. Autonomic Management of Smart Spaces. *International Workshop on Managing Ubiquitous Communications And Services, MUCS 2006* (2006).

[76] IBM. Automic Computing Toolkit. http ://www.ibm.com/developerworks/autonomic/overview.html, (2009).

[77] B. Abdulrazak et A. Helal. Enabling a Plug-and-play integration of smart environments. *Information and Communication Technologies, 2006. ICTTA '06. 2nd* **1**, 820–825 April (2006).

[78] Mounis Khatib, Khaled Masmoudi, et Hossam Afifi. An On-demand Key Establishment Protocol for MANETs. In *AINA '06 : Proceedings of the 20th International Conference on Advanced Information Networking and Applications*, 924 (IEEE Computer Society, Washington, DC, USA, 2006).

[79] Mhamed Abdallah, Cecilia Fred, et Arab Farah. An Authentication Architecture Dedicated to Dependent People in Smart Environments. In *ICOST*, Takeshi Okadome, Tatsuya Yamazaki, et Mounir Makhtari, editors, volume 4541 of *Lecture Notes in Computer Science*, 90–98. Springer, (2007).

[80] Pierre Busnel, Paul El Khoury, Keqin Li, Ayda Saidane, et Nicola Zannone. S&D Pattern Deployment at Organizational Level : A Prototype for Remote Healthcare System. volume 244, 27–39, August (2009).

[81] Michael W. Shapiro. Self-Healing in Modern Operating Systems. *Queue* **2**, 66–75 December (2004).

[82] Luciano Baresi, Sam Guinea, et Liliana Pasquale. Self-healing BPEL processes with Dynamo and the JBoss rule engine. In *International workshop on Engineering of software services for pervasive environments : in conjunction with the 6th ESEC/FSE joint meeting*, ESSPE '07, 11–20 (ACM, New York, NY, USA, 2007).

[83] S. Fox et D. B. Leake. Introspective reasoning for index refinement in case-based reasoning. *Experimental Theoretical Artificial Intelligence* **13**(1), 63–88 (2001).

[84] C. Gouin-Vallerand et S. Giroux. Managing and Deployment of Applications with OSGi in the Context of Smart Home. *Third IEEE International Conference on Wireless and Mobile Computing, Networking and Communications, WiMOB 2007* , 70–78 Oct. (2007).

[85] Jena - A Semantic Web framework for Java - website. http ://jena.sourceforge.net/.

[86] Mats Neovius, Kaisa Sere, Lu Yan, et M. Satpathy. A Formal Model of Context-Awareness and Context-Dependency. *IEEE International Conference on Software Engineering and Formal Methods* , 177–185 (2006).

[87] Stefan Gessler, Miquel Martin, et Stefan Weiss. Context Awareness in Future Life Scenarios : Impact on Service Provisioning Platforms. *IEEE/IPSJ International Symposium on Applications and the Internet Workshops* , 144–147 (2005).

[88] A. Agostini, C. Bettini, et D. Riboni. Loosely coupling ontological reasoning with an efficient middleware for context-awareness. In *Mobile and Ubiquitous Systems : Networking and Services, 2005. MobiQuitous 2005. The Second Annual International Conference on*, 175 – 182, july (2005).

[89] Jit Biswas, Andrei Tolstikov, Maniyeri Jayachandran, Victor Foo, Aung Wai, Clifton Phua, Weimin Huang, Louis Shue, Kavitha Gopalakrishnan, Jer-En Lee, et Philip Yap. Health and wellness monitoring through wearable and ambient sensors : exemplars from home-based care of elderly with mild dementia. *Annals of Telecommunications* **65**, 505–521 (2010).

[90] Patrice Roy, Bessam Abdulrazak, et Yacine Belala. Approaching context-awareness for open intelligent space. In *Proceedings of the 6th International Conference on Advances in Mobile Computing and Multimedia*, MoMM '08, 422–426 (ACM, New York, NY, USA, 2008).

[91] Davy Preuveneers, Jan Van den Bergh, Dennis Wagelaar, Andy Georges, Peter Rigole, Tim Clerckx, Yolande Berbers, Karin Coninx, Viviane Jonckers, et Koen De Bosschere. Towards an Extensible Context Ontology for Ambient Intelligence. In *Ambient Intelligence,* Panos Markopoulos, Berry Eggen, Emile Aarts, et James L. Crowley, editors, volume 3295 of *Lecture Notes in Computer Science*, 148–159. Springer Berlin / Heidelberg (2004).

[92] Tao Gu, Xiao Hang Wang, Hung Keng Pung, et Da Qing Zhang. An Ontology-based Context Model in Intelligent Environments. In *In proceedings of communication networks and distributed systems modeling and simulation conference*, 270–275, (2004).

[93] Xiaohang Wang, Jin Song Dong, ChungYau Chin, SankaRavipriya Hettiarachchi, et Daqing Zhang. Semantic Space : An Infrastructure for Smart Spaces. *IEEE Pervasive Computing* **3**, 32–39 (2004).

[94] E.M. Maximilien et M.P. Singh. A framework and ontology for dynamic Web services selection. *Internet Computing, IEEE* **8**(5), 84 – 93 sept.-oct. (2004).

[95] Lorenzo Sommaruga, Antonio Perri, et Francesco Furfari. DomoML-env : an ontology for Human Home Interaction. In *Proceedings of SWAP 2005, the 2nd Italian Semantic Web Workshop, Trento, Italy, December 14-16, 2005, CEUR Workshop Proceedings*, (2005).

[96] Harry Chen, Filip Perich, Tim Finin, et Anupam Joshi. SOUPA : Standard Ontology for Ubiquitous and Pervasive Applications. In *In International Conference on Mobile and Ubiquitous Systems : Networking and Services*, 258–267, (2004).

[97] Dario Bonino et Fulvio Corno. DogOnt - Ontology Modeling for Intelligent Domotic Environments. In *The Semantic Web - ISWC 2008*, Amit Sheth, Steffen Staab, Mike Dean, Massimo Paolucci, Diana Maynard, Timothy Finin, et Krishnaprasad Thirunarayan, editors, volume 5318 of *Lecture Notes in Computer Science*, 790–803. Springer Berlin / Heidelberg (2008).

[98] Manuel Román, Christopher Hess, Renato Cerqueira, Roy H. Campbell, et Klara Nahrstedt. Gaia : A Middleware Infrastructure to Enable Active Spaces. *IEEE Pervasive Computing* **1**, 74–83 (2002).

[99] Siyuan Ma, Xiaoshe Dong, Yiduo Mei, Zhao Wang, et Zhengdong Zhu. Taxonomy and an Ontology for Grid Metrics. In *ChinaGrid Annual Conference, 2008. ChinaGrid '08. The Third*, 300 –307, (2008).

[100] Jérémy Bauchet. *Archipel : un framework objet pour une approche ubiquitaire de l'assistance cognitive*. Th de Doctorat, Université de Sherbrooke, (2008).

[101] George J. Klir et Bo Yuan. *Fuzzy sets and fuzzy logic : theory and applications*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, (1995).

[102] Lawrence R. Rabiner. Readings in speech recognition. 267–296 (Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1990).

[103] Stuart J. Russell et Peter Norvig. *Artificial Intelligence : A Modern Approach*. Pearson Education, 2 edition, (2003).

[104] Lotfi A. Zadeh. Fuzzy Sets. *Information and Control* **8**(3), 338–353 (1965).

[105] Hani Hagras. Type-2 fuzzy logic controllers : a way forward for fuzzy systems in real world environments. In *Proceedings of the 2008 IEEE world conference on Computational intelligence : research frontiers*, WCCI'08, 181–200 (Springer-Verlag, Berlin, Heidelberg, 2008).

[106] Chih-Knan Chiang, Hung-Yuan Chung, et Jin-Jye Lin. A self-learning fuzzy logic controller using genetic algorithms with reinforcements. *Fuzzy Systems, IEEE Transactions on* **5**(3), 460 –467 August (1997).

[107] Renaud Sirdey, Jacques Carlier, Hervé Kerivin, et Dritan Nace. On a resource-constrained scheduling problem with application to distributed systems reconfiguration. *European Journal of Operational Research* **183**(2), 546 – 563 (2007).

[108] J. D. Ullman. NP-complete scheduling problems. *J. Comput. Syst. Sci.* **10**, 384–393 June (1975).

[109] Eric Beaudry, Froduald Kabanza, et Francois Michaud. Planning for Concurrent Action Executions Under Action Duration Uncertainty Using Dynamically Generated Bayesian Networks. In *Internation Conference on Automated Planning and Scheduling (ICAPS)*, 10–17, (2010).

[110] Zeljko Obrenovic et Dusan Starcevic. Modeling Multimodal Human-Computer Interaction. *ACM Computer Journal* **37**, 65–72 September (2004).

[111] T. Muraoka et H. Ikeda. Selection of display devices used at man-machine interfaces based on human factors. *Industrial Electronics, IEEE Transactions on* **51**(2), 501 – 506 (2004).

[112] N. Carrey. Establishing Pedestrian Walking Speeds. Technical report, Portland State University - ITE Student Chapter, (2005).

[113] Gabor Abellan Van Kan, Y. Rolland, S. Andrieu, J. Bauer, O. Beauchet, M. Bonnefoy, M. Cesari, L.M. Donini, S. Gillette-Guyonnet, M. Inzitari, F. Nourhashemi, G. Onder, P. Ritz, A. Salva, M. Visser, et B. Vellas. Gait speed at usual pace as a predictor of adverse outcomes in community-dwelling older people an International Academy on Nutrition and Aging (IANA) Task Force. *The Journal of Nutrition, Health amp ; Aging* **13**, 881–889 (2009).

[114] Maurits Clemens Kaptein, Clifford Nass, et Panos Markopoulos. Powerful and consistent analysis of likert-type ratingscales. In *Proceedings of the 28th international conference on Human factors in computing systems*, CHI '10, 2391–2394 (ACM, New York, NY, USA, 2010).

[115] Jakob Nielsen et Jonathan Levy. Measuring usability : preference vs. performance. *Commun. ACM* **37**, 66–75 April (1994).

[116] Benjamin Satzger, Florian Mutschelknaus, Faruk Bagci, Florian Kluge, et Theo Ungerer. Towards Trustworthy Self-optimization for Distributed Systems. In *Software Technologies for Embedded and Ubiquitous Systems,* Sunggu Lee et Priya Narasimhan, editors, volume 5860 of *Lecture Notes in Computer Science*, 58–68. Springer Berlin / Heidelberg (2009).

[117] Hermann Krallmann, Christian Schröpfer, Vladimir Stantchev, et Philipp Offermann. Enabling Autonomous Self-Optimisation in Service-Oriented Systems. In *Autonomous Systems – Self-Organization, Management, and Control,* Bernd Mahr et Sheng Huanye, editors, 127–134. Springer Netherlands (2008).

[118] Céline Descheneaux et Hélène Pigot. Interactive Calendar to Help Maintain Social Interactions for Elderly People and People with Mild Cognitive Impairments. In *Proceedings of the 7th International Conference on Smart Homes and Health Telematics : Ambient Assistive Health and Wellness Management in the Heart of the City*, ICOST '09, 117–124 (Springer-Verlag, Berlin, Heidelberg, 2009).

[119] K.B. Lamine et F. Kabanza. History checking of temporal fuzzy logic formulas for monitoring behavior-based mobile robots. In *Tools with Artificial Intelligence, 2000. ICTAI 2000. Proceedings. 12th IEEE International Conference on*, 312 –319, (2000).

[120] Seong ick Moon, K.H. Lee, et Doheon Lee. Fuzzy branching temporal logic. *Systems, Man, and Cybernetics, Part B : Cybernetics, IEEE Transactions on* **34**(2), 1045 – 1055 (2004).

[121] M. F. Folstein, S. E. Folstein, et P. R. McHugh. "Mini-mental state". A practical method for grading the cognitive state of patients for the clinician. *Journal of Psychiatric Research* **12**(3), 189–198 (1975).

[122] I. Gélinas, L. Gauthier, M. McIntyre, et S. Gauthier. Development of a functional measure for persons with Alzheimers disease : the disability assessment for dementia. *The American journal of occupational therapy official publication of the American Occupational Therapy Association* **53**(5), 471–481 (1999).

[123] Charles L. Forgy. Expert systems. chapter Rete : a fast algorithm for the many pattern/many object pattern match problem, 324–341. IEEE Computer Society Press, Los Alamitos, CA, USA (1990).

[124] JBoss Community. Drools website. http ://www.jboss.org/drools/, (2009).

[125] Stuart K. Card, George G. Robertson, et Jock D. Mackinlay. The information visualizer, an information workspace. In *Proceedings of the SIGCHI conference on Human factors in computing systems : Reaching through technology*, CHI '91, 181–186 (ACM, New York, NY, USA, 1991).

[126] Sumi Helal et Chao Chen. The Gator Tech Smart House : enabling technologies and lessons learned. In *Proceedings of the 3rd International Convention on Rehabilitation Engineering & Assistive Technology*, i-CREATe '09, 13 :1–13 :4 (ACM, New York, NY, USA, 2009).

[127] Julie A. Kientz, Shwetak N. Patel, Brian Jones, Ed Price, Elizabeth D. Mynatt, et Gregory D. Abowd. The Georgia Tech aware home. In *CHI '08 extended abstracts on Human factors in computing systems*, CHI EA '08, 3675–3680 (ACM, New York, NY, USA, 2008).

[128] Patrice Roy, Bruno Bouchard, Abdenour Bouzouane, et Sylvain Giroux. A Hybrid Plan Recognition Model for Alzheimer's Patients : Interleaved-Erroneous Dilemma. In *Proceedings of the 2007 IEEE/WIC/ACM International Conference on Intelligent Agent Technology*, IAT '07, 131–137 (IEEE Computer Society, Washington, DC, USA, 2007).

[129] M.J. Khan, M.M. Awais, et S. Shamail. Enabling Self-Configuration in Autonomic Systems Using Case-Based Reasoning with Improved Efficiency. In *Autonomic and Autonomous Systems, 2008. ICAS 2008. Fourth International Conference on*, 112–117, March (2008).